МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Запорізький національний технічний університет

ВІДДАЛЕНИЙ ТА ВІРТУАЛЬНИЙ ІНСТРУМЕНТАРІЙ В ІНЖИНІРИНГУ

Монографія

За загальною редакцією
др. Карстена Хенке

Запоріжжя
ДИКЕ ПОЛЕ
2015

УДК 004.41
ББК 32.973
В-42

Автори: **А. В. Пархоменко, Г. В. Табунщик, М. О. Поляков, О. М. Гладкова, Т. І. Каплієнко, Т. Ю. Ларіонова.**

Рекомендовано до друку Вченою радою Запорізького національного технічного університету (протокол № 12 від 03 червня 2015 року)

Р е ц е н з е н т и:

ГОМЕНЮК С. І., доктор технічних наук, професор, декан математичного факультету Запорізького національного університету;
ТАРАСОВ О. Ф., доктор технічних наук, професор, завідувач кафедри комп'ютерних інформаційних технологій Донбаської державної машинобудівної академії;
ТЕСЛЮК В. М., доктор технічних наук, професор, професор кафедри САПР Національного університету «Львівська політехніка».

В-42 **Віддалений** та віртуальний інструментарій в інжинірингу: монографія /за заг. ред. Карстена Хенке. – Запоріжжя: Дике Поле, 2015. – 250 с.
ISBN 978–966–2752–74–8

Книга містить основні концепції використання віртуальних, керованих дистанційно пристроїв, а також розподілених віддалених лабораторій. Розглянуті можливості їх використання при проектуванні вбудованих систем, для розробки систем керування складними електротехнічними установками і комплексами та для оцінювання якості вбудованих систем. Видання може бути корисним для фахівців в галузі проектування вбудованих систем, спеціалістів з електромеханіки, студентів та аспірантів.

УДК 004.41

Видання здійснено за підтримки міжнародного проекту ICo-op «Промислове співробітництво та креативна інженерна освіта на основі дистанційного інженерного та віртуального інструментарію» (530278-TEMPUS-1-2012-1-DE-TEMPUS-JPHES) за програмою TEMPUS Європейської комісії.
Зміст даного матеріалу відображає думку авторів та Європейська комісія не несе відповідальності за використання інформації, що міститься в монографії.

ISBN 978–966–2752–74–8

ЗМІСТ

PREFACE

One of the aims of the international Tempus project 530278-TEMPUS-1-2012-1-DE-TEMPUS-JPHES “Industrial Cooperation and Creative Engineering Education based on Remote Engineering and Virtual Instrumentation” [ICo-op] (www.ico-op.eu) is to increase university enterprise collaboration. As basis for such collaboration by the Integrated Communication Systems Group at the Ilmenau University of Technology, Germany, the hybrid interactive online laboratory GOLDi (Grid of Online Lab Devices, www.goldi-labs.net) was provided, which gives the students the possibility to work on real physical systems without the need to stand in line at a lab or the need to take care of opening hours.

The range of functionality of laboratory is very wide and can be applied in different fields of engineering. This was proved by the colleagues in Zaporizhzhya National Technical University who recommend the usage of the GOLDi for design of the embedded systems, for intellectual controller systems design and for the processes of verification of the embedded systems.

Developed courses allow the university to find new partners and we hope they made sustainable partnerships with such enterprise as «Motor Sich», «Ukrtelecom», «Zaporizhtransformator» and others companies of Zaporizhzhya region. During the first pilot teaching near 40 representatives from different enterprises of the region had an opportunity to work with the laboratory and improve their skills. This allows spreading the ideas of the long life learning approach in Ukraine and we hope it will help to make new basis for further sustainable collaboration.

Dr. Ing. Karsten Henke
Coordinator of the ICo-op project,
Ilmenau University of Technology,
Germany

ПЕРЕДМОВА

Головною метою міжнародного проекту Темпус 530278-TEMPUS-1-2012-1-DE-TEMPUS-JPHES «Промислове співробітництво і творча технічна освіта, засновані на дистанційному інжинірингу та віртуальному інструментарію» було розширення можливостей співпраці між університетами та роботодавцями. Фундаментом співпраці було обрано гібридну інтерактивну лабораторію GOLDi (Grid of Online Lab Devices Ilmenau, www.goldi-labs.net), що була розроблена групою інтегрованих комунікаційних систем Університету Технологій Ільменау, Німеччина, що дозволяє студентам працювати з реальними фізичними системами без очікування лабораторного часу.

Функціональність лабораторії дозволяє вирішувати широкий спектр завдань у різних технічних галузях, що було підтверджено колегами Запорізького національного технічного університету, які рекомендували використання лабораторії GOLDi для завдань проектування вбудованих систем та інтелектуальних мікроконтролерних систем керування, а також для процесу верифікації вбудованих систем.

Розроблені навчальні курси дозволили університету знайти нових партнерів та, будемо сподіватися, організувати стійке партнерство з такими підприємствами як «Мотор-Січ», «Укртелеком», «Запоріжтрансформатор» та іншими компаніями запорізького регіону. Під час першого пілотного навчання близько 40 представників підприємств отримали можливість працювати з лабораторією та покращити їх практичні навички. Ми вважаємо, що це дозволяє поширити ідею навчання на протязі всього життя в Україні і сподіваємося стане новим фундаментом для подальшої сталої співпраці.

Доктор техн. наук Карстен Хенке
Координатор проекту ICo-op,
Університет Технологій Ільменау,
Німеччина

ЧАСТИНА 1. ТЕХНОЛОГІЇ ТА СИСТЕМИ ВІРТУАЛЬНОЇ ТА ВІДДАЛЕНОЇ ІНЖЕНЕРІЇ

А.В. Пархоменко,О.М. Гладкова

ВСТУП ДО ЧАСТИНИ 1

Швидкий розвиток Інтернет-технологій та їх зростаюча популярність дуже суттєво впливають на розвиток техніки. Спільне використання віртуальних, а також керованих дистанційно пристроїв, а також розподілених віддалених лабораторій є актуальним завданням сьогодні, у зв'язку з:

- зростаючою складністю інженерних завдань, що вимагають розв'язку в короткі терміни реалізації проекту;
- використанням в процесі проектування все більшої кількості спеціалізованого дорогого обладнання, а також програмних засобів;
- недоступністю високотехнологічного обладнання для малих і середніх підприємств;
- необхідністю висококваліфікованого персоналу для управління сучасним обладнанням;
- вимогами глобалізації та розподілу праці.

Сьогодні віддалені лабораторії успішно розроблені і впроваджені в усьому світі. Однак, можна зауважити, що переважна більшість вже реалізованих експериментів та обладнання з віддаленим доступом використовуються як суто освітні ресурси, а можливості для професійного дослідника або проектувальника не повністю вивчені.

З іншого боку, сучасний ринок вбудованих систем (ВС) безперервно розвивається і потребує створення все більш складних систем за все коротші терміни. У цих умовах розробка систем «з нуля» є просто неефективною. Тому, одним з ключових напрямків у підвищенні ефективності проектування ВС, є накопичення технічних рішень з метою їх повторного використання. Це (повторне використання) знань може бути проілюстровано та випробувано за допомогою віддалених лабораторій. Існуюче розмаїття описів вже розроблених компонентів (апаратних блоків, програм, реалізацій

алгоритмів і т.д.) перешкоджає їх повторному використанню, однак застосування віддаленого експерименту дозволить проектувальникам одержати інформацію про готові апаратно-програмні платформи і компоненти для прийняття рішень щодо реалізації вбудованих систем.

Перший розділ присвячено новим підходам до проектної та виробничої діяльності на основі віртуального інжинірингу та віддаленого експерименту. Наведено аналіз структурних та функціональних особливостей деяких відомих віддалених лабораторій.

У другому розділі висвітлені дослідження ринку ВС. Також виконано аналіз вимог, які повинні бути розглянуті в ході проектування вбудованих систем. Описані особливості структурування вимог і розроблена модель вимог. Запропоновано методику створення вимог до ВС, з урахуванням процесів їх визначення та аналізу. Описані існуючі підходи до реалізації апаратного та програмного забезпечення вбудованих систем.

Третій розділ представляє питання проектування вбудованих систем за допомогою віртуального і віддаленого інструментарію: Інтегроване середовище розробки ВС; практичні завдання проектування вбудованих систем з використанням віддаленого експерименту; можливості використання віддаленої лабораторії RELDES (розробленої в Запорізькому національному технічному університеті, Україна) та гібридної лабораторії GOLDI (розробленої в Технічному університеті Ільменау, Німеччина) для проектування ВС.

The mushroom growth of Internet technologies and its increasing popularity has had an enormous impact on engineering. The shared use of virtual and remote controlled devices as well as distributed remote and virtual laboratories is actual task today, due to:

- the growing complexity of engineering tasks;
- more and more specialized and expensive equipment as well as software tools and simulators;
- the necessity of use of expensive equipment and software tools/ simulators in short time projects;
- the application of high tech equipment also in SMEs (small and medium-sized enterprises);
- the need of high qualified staff to control recent equipment;
- the demands of globalization and division of labor

Today Remote Laboratories are successfully developed and implemented worldwide. An observation is that the vast majority of already implemented experiments and installations with remote access are used as pure educational resources and that the possibilities for the professional researcher or designer are not fully explored.

On the other hand the Embedded Systems market is permanently evolving and requires the creation of more complex systems in an increasingly shorter period of time. Under these conditions, the development of systems «from scratch» is not effective. Therefore, one of the key areas to improve the efficiency of Embedded Systems design is the accumulation of technical solutions for reuse. This (reuse) of knowledge can be illustrated and trained in virtual and remote laboratories. The disunity of descriptions of already developed components (hardware units, programs, algorithms implementations, etc.) hinders their reuse, but the application of virtual and remote experiments will allow the designer to obtain information about hardware and software platforms and components for making decisions on the Embedded Systems realization.

The first chapter presents new approaches to design and production activity based on virtual engineering and remote experiment. The analysis of the structural and functional features of some well-known remote laboratories is shown.

At the second chapter ES market research presented. Also the requirements analysis that must be considered during embedded systems design has been done. Peculiarities of the requirements structuring and developed model of requirements are described. Methodology of creating requirements, taking into account the processes of their definition and analysis proposes. The existing approaches to embedded systems hardware/software realization are described.

The third chapter presents issues of embedded systems design using virtual and remote tools: ES Integrated Development Environment; ES Practical Design Using Remote Experiment; remote laboratory RELDES (ZNTU, Ukraine) and hybrid laboratory GOLDI (TU Ilmenau, Germany) using for ES design.

1 СУЧАСНІ ПІДХОДИ ДО ПРОЕКТНОЇ ТА ВИРОБНИЧОЇ ДІЯЛЬНОСТІ НА ОСНОВІ ВІРТУАЛЬНОЇ ІНЖЕНЕРІЇ ТА ВІДДАЛЕНОГО ЕКСПЕРИМЕНТУ

1.1 Технології віртуальної інженерії

Віртуальна інженерія надає засоби імітації різних видів інженерної діяльності на всіх етапах розробки та виробництва виробів: проектування та моделювання; імітація машинної обробки; імітація збирання; управління виробничими лініями; візуальний контроль та оцінювання. Дослідження віртуального прототипу дозволяє: скоротити час розробки; дослідити більше альтернативних варіантів конструкції; не створювати фізичні прототипи, які багато коштують; виконати дослідження прототипу, які є неможливими в реальних умовах на макеті; не проводити фізичні експерименти, що потребують значного часу.

Віртуальне виробництво – це інтегроване виробниче середовище, що може бути класифіковане як:

- проектно-орієнтоване віртуальне виробництво, або імітаційне середовище для проектування продукту та оцінки можливостей його виробництва.
- виробничо-орієнтоване віртуальне виробництво, або імітаційне середовище для планування виробничих процесів та виробництва;
- управлінсько-орієнтоване віртуальне виробництво, або імітаційне середовище для моделювання функціонування виробничих цехів.

Основними компонентами віртуального виробництва є віртуальне проектування, цифрова імітація, віртуальне прототипування та віртуальний завод.

Віртуальне проектування надає можливість створювати компоненти, модифікувати їх, управляти різними пристроями і взаємодіяти з віртуальними об’єктами в віртуальному середовищі. Конструктор може бачити стереоскопічне зображення віртуальних об’єктів і чути просторовий реалістичний звук. Ці зображення та звук вини-

кають, коли рука конструктора рухає віртуальною рукою і пальцем. Дотик до віртуального об'єкту відчувається конструктором у вигляді зворотного зв'язку. Конструктор отримує можливість ефективно втілити в проекті свої ідеї, а також перевірити функціональну поведінку конструкції. Даний підхід дозволяє також врахувати точку зору потенційного користувача продукції, оскільки вже на ранніх стадіях проектування можуть бути повною мірою оцінені такі якості, як доступність і керованість. З'являється також можливість врахувати при проектуванні складний і важко формалізований досвід експертів в збиранні або маніпулюванні деталями. При цьому, система віртуального проектування допоможе зрозуміти положення користувача, його взаємодію з об'єктами і послідовність операцій збирання.

Цифрова імітація спрямована на оцінювання роботи виробів без використання фізичних прототипів, а також перевірку технологічних процесів. За допомогою імітації користувач може спрогнозувати зіткнення між інструментом і пристосуванням або деталлю, перевірити траєкторію переміщення інструменту верстата з ЧПУ, щупа координатно-вимірювальної машини або руки робота. Візуалізація допомагає інженерам краще зрозуміти систему, оскільки дозволяє легко усвідомити ідею конструкції і заздалегідь перевірити її експлуатаційні якості. В даний час для цієї мети використовується головним чином кінематична імітація твердих тіл. Імітація моделей більш високого рівня рідин, людських істот і складних середовищ вимагає складного моделювання фізичних ефектів, включаючи ефекти динаміки, вібрації, акустики і деформації. Тим не менш, складні імітації з використанням віртуальних прототипів можуть здійснити перевірку робочих характеристик системи швидше і з меншими витратами.

Віртуальне прототипування передбачає побудову комп'ютерного прототипу проектованого виробу, геометрія та фізична поведінка якого відповідають реальному продукту. Системи віртуального прототипування дозволяють перевіряти можливість створення агрегатів в наявних виробничих умовах шляхом візуалізації процесу збирання. Збирання віртуального прототипу дозволяє виявити конструктивні прорахунки, відпрацювати послідовність та оптимальну траєкторію руху деталей і, в разі необхідності, внести зміни в проект. Передові системи надають можливість проведення структурного та функціонального аналізу віртуального прототипу, а також кінематичної та динамічної імітації його роботи.

Віртуальний завод – це модель виробничої системи, що імітує виробничі ділянки, процеси, складські системи, а також автоматизоване заводське обладнання – верстати, роботи і конвеєри. Побудована модель виробництва дозволяє проаналізувати витрати на робочу силу, експлуатаційні витрати, витрати на обробку і тривалості виробничого циклу. Це дозволяє використовувати віртуальний завод для планування виробництва, оцінки проектів виробничих систем і порівняння альтернативних способів виробництва. Якщо за допомогою віртуального заводу зімітувати весь ланцюжок поставок, це дозволить оцінювати та оптимізувати весь процес управління ресурсами і виробництвом.

Таким чином, сучасні технології віртуальної інженерії дозволяють:

- оцінити можливість виробництва різних варіантів конструкції (включаючи оцінку якості збирання або експлуатаційних характеристик проектованих виробів);
- оптимізувати виробничий процес (методом цифрової імітації);
- легко налаштувати продукт під вимоги замовника;
- ефективно накопичити широку базу знань;
- забезпечити основу для колективної розробки проектів [48] .

1.2 Віртуальні та віддалені лабораторії

Віртуальна лабораторія представляє собою програмно-апаратний комплекс, що дозволяє проводити дослідження без безпосереднього контакту з реальною установкою або за умови її повної відсутності. У першому випадку ми маємо справу з так званою лабораторною установкою з віддаленим доступом, до складу якої входить реальна лабораторія, програмно-апаратне забезпечення для управління установкою і оцифровки отриманих даних, а також засоби комунікації. У другому випадку всі процеси моделюються за допомогою комп'ютера.

Отже, під Віртуальними лабораторіями розуміється два типи програмно-апаратних комплексів:

- лабораторна установка з віддаленим доступом назвемо такий комплекс дистанційна лабораторія (або лабораторія віддаленого доступу);

- програмне забезпечення, що дозволяє моделювати лабораторні досліди віртуальна лабораторія (у вузькому сенсі) .
- Основними перевагами віртуальних лабораторій є наступні:
- відсутність необхідності придбання дорогого устаткування і реактивів;
- можливість моделювання процесів, протікання яких принципово неможливо в лабораторних умовах;
- можливість спостереження того, що відбувається в іншому масштабі часу;
- безпека у випадках, коли здійснюється робота, наприклад, з високими напруженнями або хімічними речовинами.
- можливість швидкого проведення серії дослідів з різними значеннями вхідних параметрів;
- економія часу і ресурсів для представлення результатів в електронному форматі;
- можливості використання віртуальної лабораторії в дистанційному навчанні [66] .

На сьогоднішній день у світовому досвіді створення та експлуатації лабораторій віддаленого доступу спостерігається явна тенденція «інформаційного об'єднання» існуючих Інтернет-ресурсів доступу до лабораторних експериментів. Необхідність цього обумовлена двома чинниками:

- надто велика кількість вже реалізованих проектів по віддаленому доступу для проведення експериментів з самих різних галузей науки робить дуже складною задачу пошуку та класифікації необхідних лабораторій;
- фінансово-економічний фактор: можливість отримання єдиних грантів на виконання проектів лабораторій віддаленого доступу для великої кількості груп виконавців, які потім розміщують посилання на веб-ресурси своїх віддалених лабораторій на одному і тому ж (головному) веб-сайті [58] .

Нижче наведено огляд базових можливостей відомих віртуальних та віддалених лабораторій для освіти.

Гібридна online лабораторія GOLDI (Grid of Online Lab Devices Ilmenau) (http://www.tu-ilmenau.de/goldi/) – розроблена на кафедрі

Інтегрованих комунікаційних систем Технічного університету Ільменау, Німеччина (http://ih7.theoinf.tu-ilmenau.de/applets/index.htm). Вона надає повний набір інструментів, що підтримує всі проектні кроки для комплексних задач управління і регулювання (наприклад, в області автоматичного управління, робототехніки, дистанційного керування). Мета GOLDI системи показати сучасні технічні способи і проблеми дистанційного управління та дистанційного спостереження реальних процесів, що пов'язані з інтегрованим та інтерактивним використанням сучасного Інтернету та Інтернет-технологій, таких як HTML5, JavaScript, і т.і. Вона пропонує різні функції, такі як візуалізація та анімація, що дозволяють спостерігати і тестувати всі властивості проекту. У поєднанні з методами формального проектування, симуляції і прототипування використовується для створення основи для проектування надійної системи.

Щоб перевірити функціональність всієї конструкції, деякі спеціальні функції симуляції та валідації включені в якості невід'ємної частини GOLDI системи, а саме:

- використання імітаційних моделей фізичної системи для візуалізації прототипів,
- послідовне і паралельне виконання цих прототипів,
- візуалізація процесу моделювання з інструментами, також використовується для специфікації.
- функції для генерації тестових зразків і
- генерація коду для синтезу апаратного і програмного забезпечення.

GOLDI пропонує Web-орієнтоване середовище, яке підтримує вище зазначені функції для генерації та реалізації проекту, використовуючи імітаційні моделі (Рис.2.1). Будь-коли студенти мають можливість скоригувати свої алгоритми в разі помилки. Таким чином, крок за кроком вони можуть домогтися безпомилкового рішення (перевірений алгоритм управління) .

Ця online лабораторія використовується для проведення практичних занять, а також дає практичний досвід для розробки вбудованої електроніки. Робиться це шляхом дистанційного навчання через Internet і володіє такою перевагою, що курси можуть бути запропоновані на міжнародному рівні в усьому світі. Крім того, для всіх студентів, лабораторія пропонує розширити години роботи (24 години,

7 днів на тиждень), у порівнянні зі стандартними годинами роботи звичайної лабораторії. Крім пропонованих переваг для студентів, також, зменшує витрати на академічному викладанні і покращує якість, пропонуючи більше можливостей практичного навчання.

GOLDI пропонує три різні специфікації та механізми контролю застосування фізичних систем (електромеханічних моделей, наприклад, ліфт, гнучкий виробничий модуль, висотний одно об'ємний склад) в лабораторії (Рис.1.1) .

WebLab-Deusto дистанційна лабораторія Інженерного факультету університету Деусто (Іспанія, Більбао) (https://www.weblab.deusto.es/). Вона може бути завантажена і розгорнута для обслуговування нових віддалених експериментів в різних середовищах і операційних системах (Рис.1.2) .

Основні особливості системи безпечність, масштабованість, а також використання SSl як засобу взаємодії, перевірка експерименту в шарах управління, і відстеження використання системи та повідомлень, що надсилаються студентами. Система також підтримує різні схеми аутентифікації, такі як регулярні бази даних, LDAP, підтримка заочного навчання студентів OpenID. Вона достатньо розширювана, щоб бути інтегрованою з такими платформами, як Facebook,.LRN, Joomla і Moodle, а також зовнішніми експериментами як, наприклад, VISIR.

На цей час експеримент може бути розроблений з використанням двох різних підходів: керовані та некеровані експерименти.

WebLab-Deusto об'єднує свої ресурси для розробки такого сценарію, де студентам обладнання для експериментів доступно в університеті в електронному вигляді. WebLab-Deusto пропонує сім видів різних експериментів: Xilinx програмовані логічні пристрої; Мікросхема на PIC мікроконтролерах; Програмований рухливий робот; Електроніка та приладобудування; Логічні та підключення користувальницького C ++ додатки до осцилограф через GPIB; Weblab-Субмарина: підтримка справжнього віддаленого акваріума.

WEBENCH® Design Center – віртуальна лабораторія компанії Texas Instruments. (http://www.ti.com/lsds/ti/analog/webench/overview.page) .

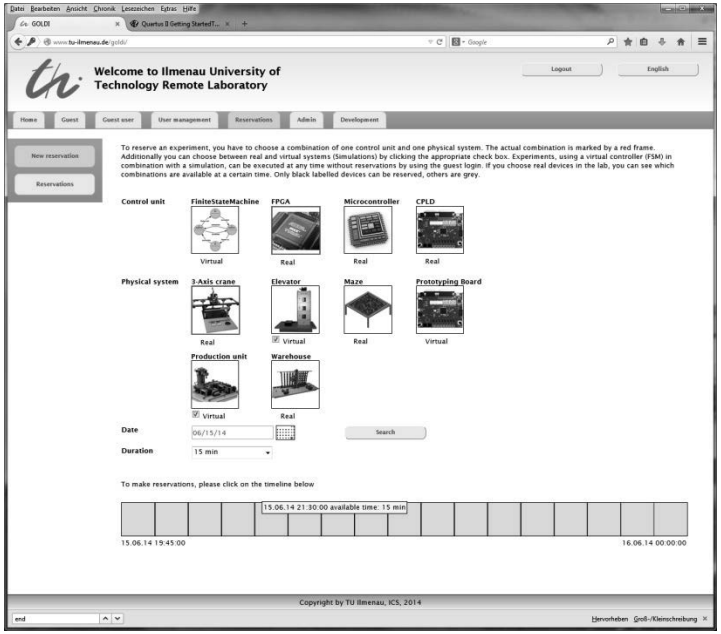


Рисунок 1.1. Приклад веб-інтерфейсу лабораторії GOLDI

Засоби проектування WEBENCH представляють собою ефективні програмно реалізовані алгоритми і графічні інтерфейси, які забезпечують швидку роботу додатків з управління електроживленням, розробку драйверів для живлення світлодіодів, і додатків, що працюють з різними сенсорами (вимірювальними пристроями). Ці прості у використанні інструменти допомагають створювати, оптимізувати і моделювати проекти, які відповідають унікальним технічним вимогам розробки. Це дозволяє користувачеві порівняти поточну вартість на рівні системи і логістичного ланцюжка перед проектуванням (Рис.1.3) .

Labshare національна некомерційна мережа лабораторій віддаленого доступу Австралії (http://www.labshare.edu.au/). Включає в себе 5 технічних вищих навчальних закладів. Експериментальні стенди надають 10 експериментів (Рис.1.4):

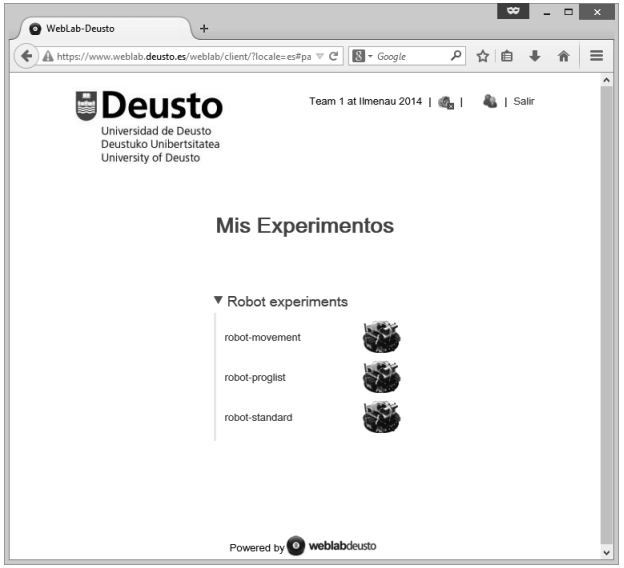


Рисунок 1.2. Приклад веб-інтерфейсу лабораторії Деусто

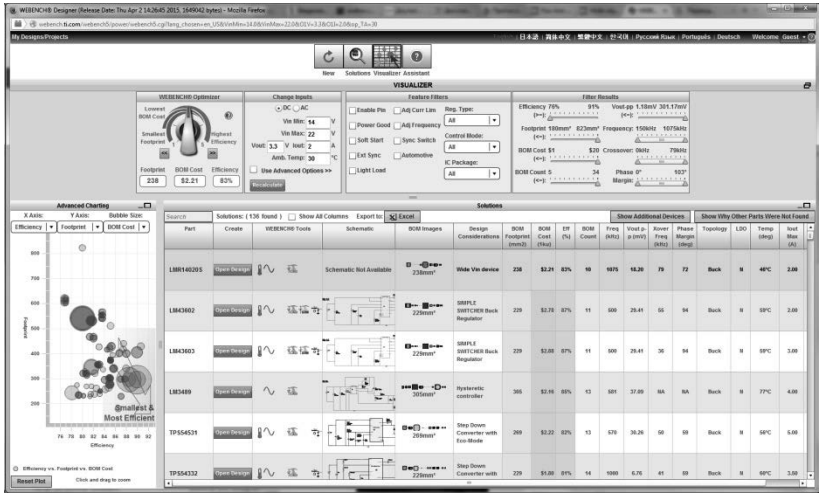


Рисунок 1.3. Приклад веб-інтерфейсу лабораторії WEBENCH® Design Center

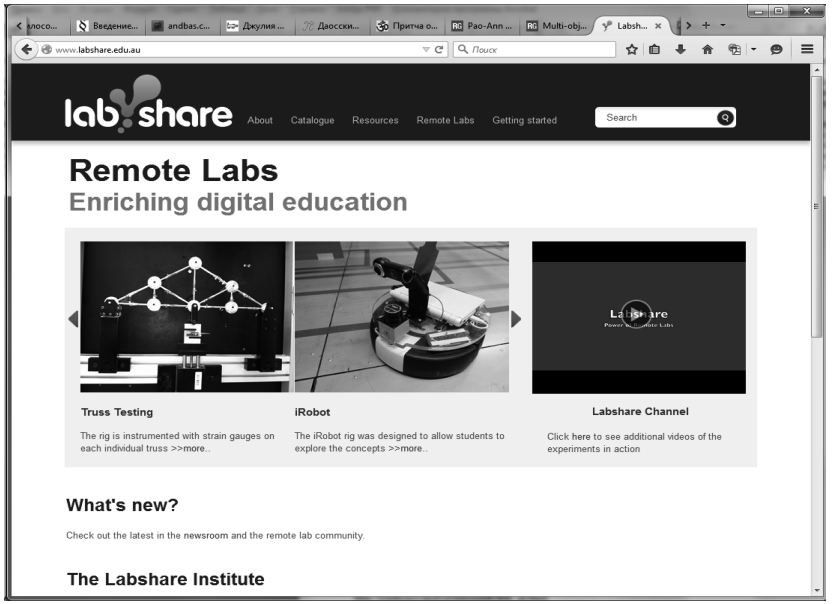


Рисунок 1.4. Приклад веб-інтерфейсу лабораторії Labshare

- контроль рідини в зв'язних баках;
- інженерна геологія (дослідження зразків порід і матеріалів);
- установка гідроелектроенергії, що демонструє основний принцип перетворення кінетичної енергії води, що тече, в електричну енергію, за допомогою обертання турбіни (колеса Пельтона), підключеного до електричного генератору;
- установка iRobot для дослідження поняття дистанційного управління роботом, точності датчиків, локалізації (адаптації до місця) і упорядкування схеми пересування;
- зразок структурної балки, для дослідження сили розтягування і стискання в структурах, таких як бетонна підлога або авіаційні крила;
- установка, що дозволяє з'єднати реальні електронні компоненти в ланцюг, який далі може бути протестований з використанням контрольно-вимірювальних приладів з реальними даними;

- вивчення основ автоматизації ПЛК (програмований логічний контролер) шляхом програмування промислових ПЛК з використанням релейної логіки;
- розширений аналіз поведінки будівлі під час землетрусу;
- вивчення наслідків зміни джерела енергії та струму намагнічування на генеровану потужність, напругу і струм;
- різниця між турбулентним і ламінарним потоками в трубах.

e-Laboratory Project дистанційна лабораторія Чехії та Словаччини (http://www.ises.info/index.php/en/) (Рис.1.5). Дистанційні експерименти: контроль рівня води; метеостанція; електромагнітна індукція; природні і керовані коливання; перетворення сонячної енергії; дифракція мікрооб'єктів; принцип невизначеності Гейзенберга; фотоефект; поляризація світла; радіоактивність.

iLabs онлайн лабораторія, створена в Массачусетському технологічному інституті для проведення лабораторних практикумів в інженерній освіті (https://wikis.mit.edu/confluence/display/ILAB2/Home; jsessionid=F02FCB44122201A3DFB256BE892B9CDA) .

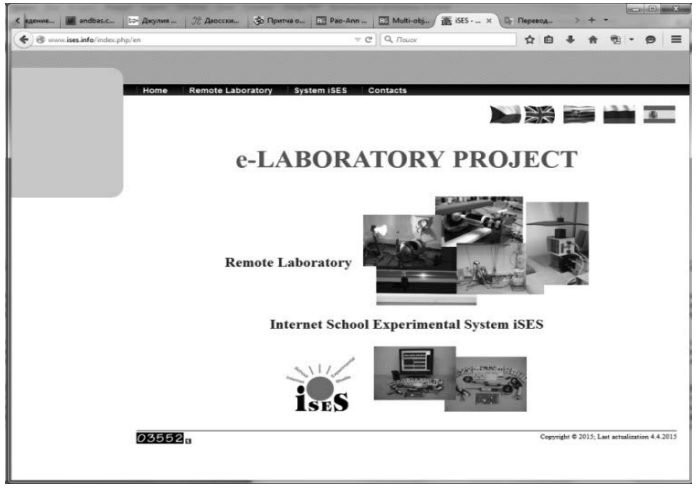


Рисунок 1.5. Приклад веб-інтерфейсу лабораторії e-Laboratory Project

Налічує близько 20 експериментів у галузі мікроелектроніки, хімічної промисловості, полімер кристалізації, проектування споруд та обробки сигналів (Рис.1.6) .

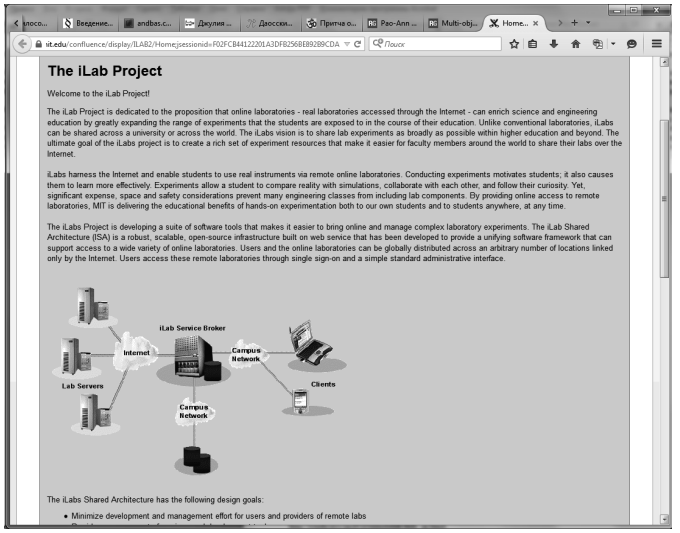


Рисунок 1.7. Приклад веб-інтерфейсу лабораторії Remote-LABGymKT

Remote-LABGymKT дистанційна лабораторія Чехії. (http://remotelab.fyzika.net/vzdalene-experimenty.php?lng=en#DERIL Реалізовано 6 експериментів в галузі фізики:

- залежність опору металів і напівпровідників;
- визначення горизонтальної складової магнітного поля Землі за допомогою котушок Гельмгольца;
- дистанційне керування рукою роботом-маніпулятором;
- визначення вольтамперних характеристик світлодіодів (приблизне визначення постійної Планка) (Рис.1.7) .

Remotely controlled laboratory Чехія, Палацький університет, кафедра експериментальної фізики (http://ictphysics.upol.cz/remotelab/rlab_en.html). Реалізовано 5 експериментів (Рис.1.8):

- вольт-амперні характеристики 6 різних лампочок;
- визначення прискорення сили тяжіння;
- вивчення потоку води в системі труб;
- метеостанція;
- моніторинг радіоактивного фону.

OpenLabs Технологічний інститут Блекинге, Швеція (http://openlabs.bth.se/electronics/index.php/en). Лабораторія надає ресурси, необхідні для експериментування в електроніці. Надає основне обладнання, таке як осцилограф, мультиметр, функціональний генератор і блок живлення. З цими та множиною електронних компонентів можна побудувати схеми на віртуальній платі. Жоден з вимірів не моделюється. Ланцюги, які створюють будуть сформовані та виміряні, і реальні результати вимірювань будуть відображатися (Рис.1.9) .

Як свідчить наведений вище загальний огляд функціональних можливостей відомих віддалених та віртуальних лабораторій – більшість з них використовуються для суто освітніх цілей, а можливості використання для задач реального проектування та виробництва повністю не досліджені.

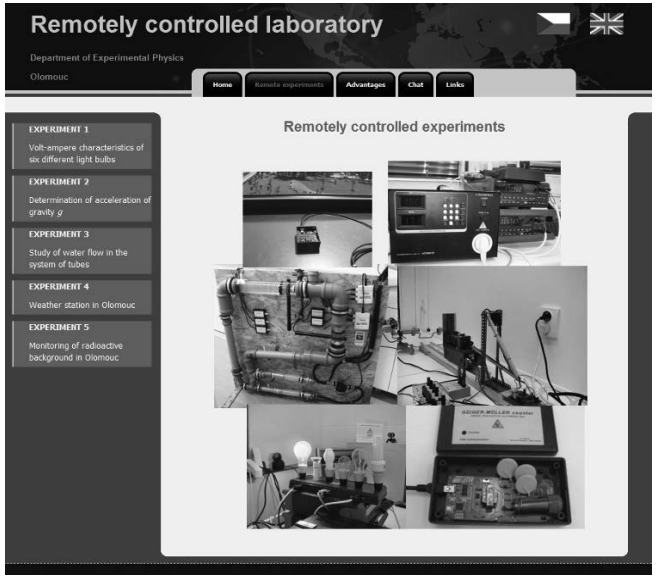


Рисунок 1.8. Приклад веб-інтерфейсу лабораторії Remotely controlled laboratory

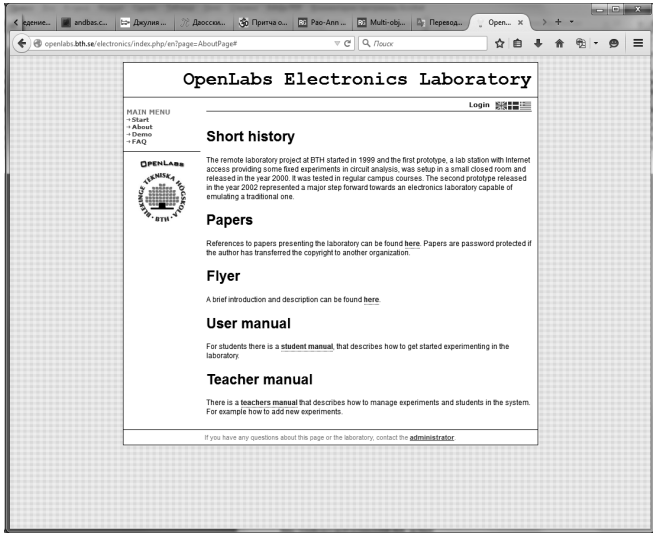


Рисунок 1.9. Приклад веб-інтерфейсу лабораторії OpenLabs

2 ВБУДОВАНІ СИСТЕМИ

2.1 Особливості та ринок вбудованих систем

Вбудована система (ВС) (англ. Embedded System) визначається сьогодні як спеціалізована обчислювальна система, яка в силу розв'язуваної задачі безпосередньо взаємодіє з фізичними об'єктами і процесами. До складу простої ВС входять:

- мікропроцесорний модуль з пам'яттю;
- периферійна система (датчики, виконавчі елементи і контролери введення-виведення для зв'язку з об'єктом контролю/управління, пристрої людино-машинного інтерфейсу за необхідністю);
- система електроживлення;
- конструктив, що об'єднує (шасі, корпус);
- програмне забезпечення, що керує (ПЗ).

Основними особливостями ВС вважаються:

- робота в реальному масштабі часу (майже завжди);
- різноманітні, часто складні, умови експлуатації;
- автономність роботи (відсутність оператора, обмеження електроживлення);
- високі вимоги щодо надійності і безпеки функціонування;
- обмежені ресурси;
- критичні застосування, пов'язані зі здоров'ям і життям людини [57] .

Вбудовані системи є особливим класом систем, який можна віднести як до автоматизованих, так і до інформаційно-керуючих систем, оскільки вбудована система підтримує (вирішує) ряд автоматизованих завдань і за визначенням є спеціалізованою системою, вбудованою в пристрій, яким вона буде керувати.

ВС відносяться до категорії систем з переважно програмною реалізацією (Software-Intensive або Software-Dominated Systems). Це означає, що велика частина функціональності системи реалізується програмним способом. Програмованість і конфігурованість пронизують всі рівні і компоненти ВС у все більшій мірі. Складність

і питома вага програмної складової у ВС стрімко зростає. Термін «вбудоване програмне забезпечення» (Embedded Software) підкреслює особливі властивості такого ПЗ і технологій його створення. Елементну базу ВС складають електронні, оптичні, механічні та інші фізичні компоненти (елементи, модулі, блоки), з яких складається фізична реалізація ВС. Сьогодні в переліку таких компонентів складні мікросхеми процесорів (мікропроцесори), контролерів, акселераторів, системні плати обчислювачів. В свою чергу, до складу таких елементів входять програмні засоби (завантажувачі, стеки протоколів та інші), що розміщуються у вбудованих блоках постійної пам'яті. Таким чином, навіть традиційне уявлення обчислювальної елементної бази виходить далеко за межі опису тільки конструкції і схемотехніки, торкаючись все більше питань системотехніки, програмування, архітектури [57] .

Світовий ринок ВС постійно зростає (рис.2.1) [10,11,12]. З кожним роком ці системи стають дедалі складнішими та вимагають більш високого рівня надійності та стійкості.

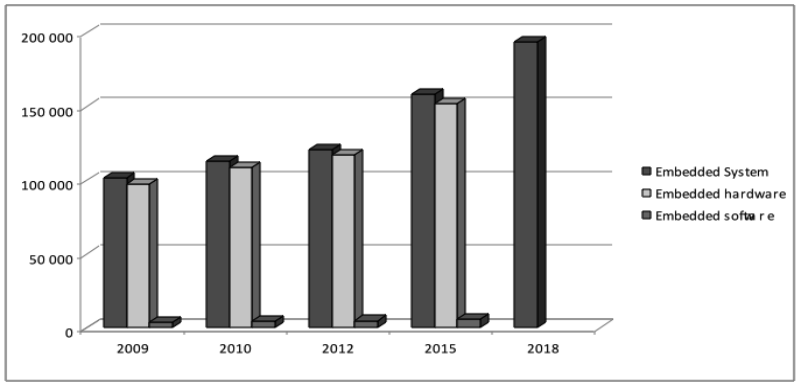


Рисунок 2.1. Аналіз ринку вбудованих технологій

За галузями застосування ринок вбудованих систем може бути сегментовано наступним чином: автомобільний сегмент, телекомунікація, медицина, промисловість, побутова техніка, військова і аерокосмічна техніка та інші. Однією з найскладніших і трудомістких категорій ВС є системи управління різними типами рухомих об'єктів (повітряними, водними, сухопутними). Цікавим як для ді-

тей так і для дорослих є клас ігрових рухомих об'єктів, а саме: безпілотні літаки, гелікоптери, квадрокоптери, дистанційно-керовані автомобілі та катери.

Вбудовані системи для розробників обчислювальної техніки є одним з найбільш складних об'єктів проектування [56]. Навіть поверхневий аналіз типових вимог та обмежень, які необхідно враховувати при створенні ВС, підтверджує це:

- мінімальне власне енергоспоживання (можливо автономне живлення);
- мінімальні власні габарити і вага;
- міцність і жорсткість конструкції;
- забезпечення теплових режимів;
- радіаційна і електромагнітна стійкість (можливо працездатність у вакуумі);
- гарантований час напрацювання на відмову;
- термін доступності рішення на ринку і т.і. [40] .

Визначення, аналіз і правильний розподіл вимог між програмними і апаратними компонентами архітектури на етапі високорівневого проектування є найважливішим фактором успіху при реалізації ВС.

Однак, як зазначають фахівці, значний розрив, що існує між вимогами до ВС і ефективністю апаратно-програмного проектування, необхідними об'ємами верифікації та тестування пристроїв буде найближчим часом продовжувати зростати [56]. В основному, проблема полягає в неспроможності традиційних підходів до проектування ВС на тлі сучасних вимог до таких систем. Одним з перспективних з точки зору побудови сучасних методик проектування ВС напрямів, є комплексне врахування вимог технічного завдання та обмежень в рамках всього проекту.

Перед розробником вбудованої системи стоїть задача реалізації її повного життєвого циклу (ЖЦ), починаючи від складання технічної документації на розробку і закінчуючи комплексними випробуваннями в складі виробу, а, можливо, і супроводом при виробництві. На сьогоднішній день в Україні діють національні стандарти з управління життєвим циклом систем і програмних засобів. Але, питання формування проектної документації, яка містить вимоги до створюваної системи, а також її склад та зміст залишаються відкритими. Практичне застосування існуючих міжнародних стандартів в Україні вимагає попередньої роботи з їх адаптації та впровадження.

2.2 Аналіз вимог до вбудованих систем та створення проектної документації

2.2.1 Аналіз типів вимог для вбудованих систем

Робота з вимогами є дуже важливою при створенні проекту вбудованої системи і досягнення поставлених цілей з його реалізації. Однак, існуючі на сьогоднішній день стандарти, а також роботи в галузі створення і аналізу вимог описують лише окремі аспекти цієї задачі і не враховують всі особливості проектування вбудованих систем.

Проектування системи, згідно з [47], включає в себе наступні технічні процеси роботи з вимогами:

- процес визначення вимог правовласника;
- процес аналізу вимог.

Результатом процесу визначення вимог правовласника є перелік всіх зацікавлених осіб, а також їх вимоги до системи, які є основою для подальшого аналізу і виявлення функціональних і не функціональних вимог до системи.

Результатом процесу аналізу вимог є повне перетворення вимог правовласника (зацікавлених осіб) в технічне бачення розроблюваної системи, тобто чітко визначену класифікацію вимог до системи. Процес аналізу вимог суттєво впливає на архітектурне проектування, а також на засоби його реалізації.

Згідно стандарту [43], під поняттям «Вимоги до системи» розуміється така сукупність:

- вимоги до системи в цілому (бізнес вимоги, зовнішній інтерфейс);
- вимоги до функцій (задач), що виконуються системою (функціональні вимоги, атрибути якості);
- вимоги до видів забезпечення (обмеження) .

До першого виду вимог відноситься також опис структури системи, куди входить перелік її компонентів. Для складних проектів,

що включають безліч компонентів (підсистем), необхідно провести розподіл вимог високого рівня за цими компонентами [32] .

У стандарті [44] дається таке визначення вбудованої системи – це набір апаратних і програмних компонентів, створений для виконання певної функції або низки функцій. Оскільки розробка програмного забезпечення (ПЗ) є найважливішою складовою проектування ВС, потрібно застосовувати комплексний підхід на основі процесів системної та програмної інженерії, і визначити як системні (високорівневі вимоги до всього проекту), так і програмні вимоги (тобто вимоги, розподілені між програмними компонентами даного проекту) .

Першим кроком до об'єднаного комплексу стандартів, що описують процеси життєвого циклу (ЖЦ) систем і програмних засобів, є міжнародний стандарт [42]. Він встановлює строгий зв'язок між системою і використовуваними в ній програмними засобами, трактуючи програмний засіб як складову частину системи, за допомогою виділення вимог до програмних засобів з вимог до системи. У стандарті наведено різницю між аналізом системних вимог і аналізом вимог до програмного продукту, оскільки в загальному випадку побудова системної архітектури визначає системні вимоги для різних складових частин системи, а аналіз вимог до програмних засобів зумовлює вимоги до них, виходячи із системних вимог, призначених кожній складовій програмній частині.

Крім перерахованих вище у стандарті [47] процесів роботи з вимогами системи, у стандарті [42] виділяються також спеціальні процеси програмних засобів, призначені для створення конкретного програмного елементу системи, зокрема процес аналізу вимог до ПЗ. В ході цього процесу визначаються вимоги до програмних елементів системи та їх інтерфейсів, встановлюється вплив вимог до ПЗ на середовище функціонування, а також сумісність та простежуваність між вимогами до ПЗ і вимогами до системи.

В [44] підкреслюється, що вхідними даними для процесу визначення вимог до ПЗ є системні вимоги, опис апаратного інтерфейсу і архітектури системи (якщо вони не включені в системні вимоги). А цілі процесу проектування ПЗ полягають у тому, щоб розробити архітектуру ПЗ і вимоги нижнього рівня на основі вимог верхнього рівня до ПЗ. Звідси випливає, що розробка самої архітектури ПЗ відбувається після опису апаратної частини проекту та її архітектури.

В роботі Леффінгуелла [19] детально розглядається процес управління вимогами і методи для розробки програмних додатків. Модель вимог Леффінгуелла складається з 3-х кластерів (рівнів):

- потреби – вимоги користувача/зацікавлених осіб;
- функції – обслуговування, що надається системою для задоволення вимог зацікавлених осіб;
- програмні вимоги – конкретизовані вимоги до програмного забезпечення.

Кластер «потреби» стосується області розуміння проблеми користувачів (замовників) для подальшої побудови системи, що задовольняє їх потребам. Наступні два кластери «функції» і «програмні вимоги» стосуються області вирішення проблем користувача.

Як наголошується в [51], в різних методологіях розробки ПЗ, застосовується підхід, який базується на визначенні груп вимог до програмного продукту. Такий підхід зазвичай включає групи (типи, категорії) вимог, наприклад: системні, програмні, функціональні, нефункціональні (зокрема, атрибути якості) і т.і.

Класичний приклад високорівневого структурування груп вимог, як вимог до програмного продукту, представлений у книзі К. Вігерса [32]. Модель вимог, запропонована Вігерсом для проектування ПЗ складається з трьох рівнів вимог:

- бізнес – навіщо розробляється система;
- користувача – описують користувачів системи, а також їх роботу з нею;
- функціональні – визначають функціональність (поведінку) програмної системи.

Модель передбачає два типи вимог:

- функціональні – що система повинна робити;
- нефункціональні – які умови при роботі повинні виконуватися.

У розглянутій моделі вимог під зовнішнім інтерфейсом розуміють «інтерфейс користувача», тобто взаємодію користувача з ПЗ. Для розробки ВС в стандарті [44] надається більш розширене визначення інтерфейсу це взаємозв'язок між двома або більше об'єктами, які спільно використовують і забезпечують дані або обмінюються ними. В якості таких об'єктів можуть виступати: елемент конфігурації ПЗ (ЕКПЗ) сукупність компонентів ПЗ, а також елемент кон-

фігурації апаратури (ЕКА) сукупність компонентів апаратних засобів (АЗ). Відповідно, варіанти інтерфейсів можуть бути наступні: ЕКПЗ/ЕКПЗ, ЕКПЗ/ЕКА, ЕКПЗ/користувач, ЕКА/користувач або модуль ПЗ/модуль ПЗ.

Таким чином, моделі вимог до ПЗ, представлені в роботах [32,19] можуть розглядатися в якості основи для розробки моделі вимог, котра враховує специфічні особливості і різні аспекти проектування ВС. Однак, вони повинні пройти детальне доопрацювання з метою врахування інших типів вимог, а також процесів роботи з вимогами до системи, представлених у стандартах [42–44] .

2.2.2 Структура та особливості моделі вимог для вбудованих систем

Ґрунтуючись на виконаному аналізі, запропонована модель вимог для проектування ВС, схематичне представлення якої наведено на рис. 2.2. Вона, як і моделі вимог Вігерса і Леффінгуелла, має 3 рівня, проте на відміну від моделі Вігерса, не використовує понять функціонального типу та функціонального рівня вимог. Оскільки ВС являє собою сукупність апаратних засобів і програмного забезпечення, то запропонована схема показує розподіл рівнів моделі на вимоги до програмних і апаратних компонентів системи.

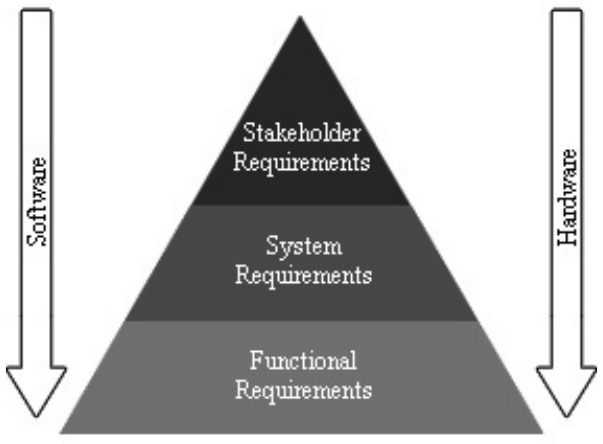


Рисунок 2.2. Модель вимог для ВС

У запропонованій моделі вимог виділені наступні рівні:

- вимоги зацікавлених осіб – включають вимоги всіх зацікавлених осіб проекту, наприклад, таких як, клієнт і кінцевий користувач продукту. Можуть бути представлені: бізнес-вимоги, варіанти використання, атрибути якості;
- системні – вимоги до системи, що включають в себе комбінацію взаємопов'язаних компонентів (апаратні засоби, ПЗ, користувачі та ін.). Можуть бути представлені: системні вимоги (до системи в цілому), обмеження, інтерфейси, вимоги до умов експлуатації;
- функціональні – функції (функціональні характеристики), розподілені між програмними і апаратними компонентами системи.

У таблиці 2.1 представлені відмінні особливості запропонованої моделі вимог у порівнянні з розглянутими вище моделями Вігерса і Леффінгуелла.

Таблиця 2.1. Відмінні особливості моделей вимог

Запропонована модель	**Модель Вігерса**	**Модель Леффінгуелла**
3 рівня вимог: - вимоги зацікавлених осіб; - системні вимоги; - функціональні вимоги.	3 рівня вимог: - бізнес-вимоги - вимоги користувачів; - функціональні.	3 рівня вимог: - потребі; - функції; - програмні вимоги.
2 типи вимог: - апаратні; - програмні.	2 типи вимог: - функціональні; - нефункціональні.	
Інтерфейси: користувача, ПЗ/ПЗ, АЗ/ПЗ, АЗ/АЗ, ПЗ (АЗ) /користувач.	Інтерфейси: користувача, ПЗ/ПЗ, ПЗ/ користувач.	Додаткова категорія: інтерфейс користувача.

Можна підкреслити такі відмінності запропонованої моделі від моделі Вігерса:

- на верхньому рівні об'єднуються бізнес-вимоги і вимоги користувачів;

- на введеному другому рівні розташовуються високорівневі вимоги до розроблюваної вбудованої системі, а також визначаються вимоги до даних, інтерфейсів та системні обмеження;
- на нижньому рівні, враховуючи особливості ВС, відбувається декомпозиція функціональних вимог на функціональні вимоги до програмної і апаратної складових системи.

Таким чином, на основі моделей вимог Вігерса і Леффінгуелла було розроблено модель вимог для ВС. У моделі реалізовані 3 рівня і 2 типи вимог для системи в цілому, а також для її програмної і апаратної складових.

Для розробленої моделі вимог пропонується методика створення вимог для ВС (рис. 2.3), що дозволяє організувати поетапне виявлення, аналіз, документування та перевірку вимог з урахуванням особливостей ВС та включає таку послідовність етапів:

1. Визначення бізнес-вимог: виявлення концепції створюваного продукту.
2. Визначення користувачів, виділення їх у групи і фіксування їх вимог.
3. Визначення варіантів використання (сценаріїв) продукту за допомогою опису робочих процесів користувачів.
4. Розробка варіантів використання: фіксування варіантів використання та ретельне вивчення послідовності взаємодії системи і зовнішньої діючої особи (Приклад ефективного способу формулювання та документування такого виду вимог представлений нижче) .
5. Виділення атрибутів якості: виявлення якісних характеристик виконання системою конкретних дій.
6. Визначення: вимог до системи, враховуючи все оточення, обмежень, вимог до умов експлуатації, інтерфейсів.
7. Перевірка розроблених специфікацій вимог на наявність помилок, повноту, недвозначність, несуперечливість та ін. Бажано, щоб у перевірці специфікації брали участь усі зацікавлені особи.

Після чого слідують етапи визначення функціональних вимог до АЗ (8) і ПЗ (9), що має на увазі більш детальне виявлення та перетворення вимог користувача в форму, корисну для розробників. Фіксування функціональних вимог до програмного забезпечення

в специфікації вимог до ПЗ може виконуватися за допомогою подання очікуваної поведінки системи у формі «подія-реакція» (приклад представлений нижче). Дана форма є зручною для подальшої розробки варіантів тестування.

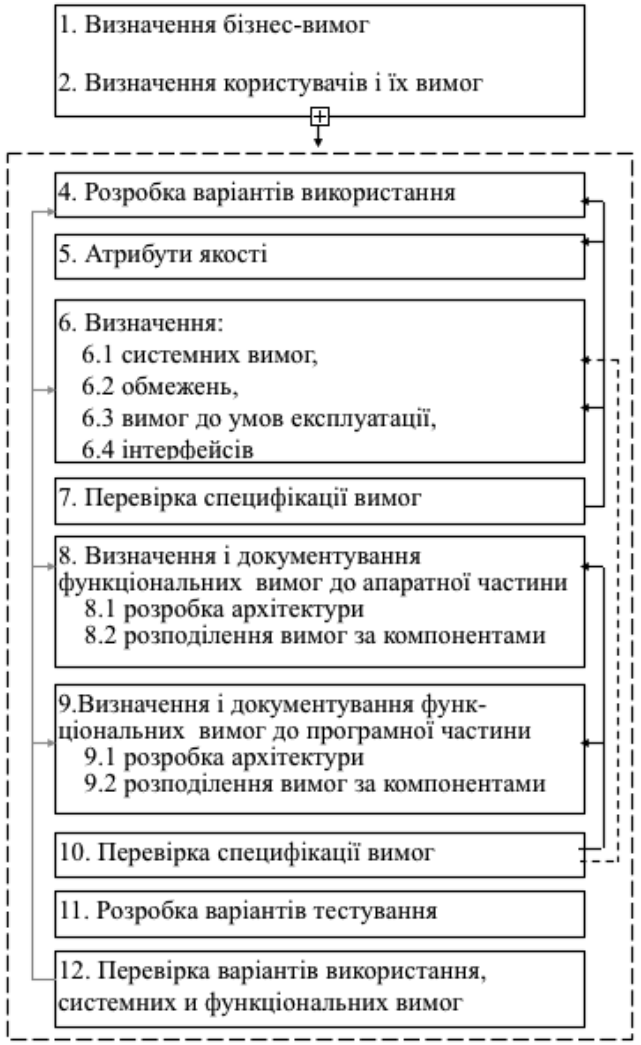


Рисунок 2.3. Методика створення вимог

10. Перевірка розроблених специфікацій вимог.
11. Розробка варіантів тестування: опис станів і переходів між станами, а також умов, за яких відбуваються переходи зі стану в стан.
12. Перевірка розроблених специфікацій системних вимог, функціональних вимог і варіантів використання на наявність помилок, повноту, суперечливість та ін.

Створення вимог являє собою ітеративний процес. Перші 3 етапи зазвичай виконуються один раз (але при необхідності вони переглядаються і змінюються), інші етапи повторюються для кожної нової версії продукту.

Одним з важливих етапів контролю створюваних вимоги є трасування (взаємозв'язок). Щоб забезпечити трасування, кожна вимога повинна бути унікальною і ідентифікованою, щоб мати можливість посилатися на неї. Найкраще керувати трасуванням, використовуючи інструмент управління вимогами. Можна виділити такі інструменти управління вимогами як: IBM Rational RequisitePro, Borland CaliberRM, DOORS. Для Rational RequisitePro можна відзначити ряд переваг:

- підтримка Word для створення проектної документації;
- наявність шаблонів для розробки проектної документації;
- наявність зручного навігатору між Microsoft Word і потужною інфраструктурою бази даних;
- можливість інтеграції із засобами об'єктно-орієнтованого моделювання;
- можливість простеження взаємозв'язку вимог різних рівнів за допомогою Матриці трасування.

Створювана матриця трасування в IBM Rational RequisitePro дозволяє при модифікації одних вимог легко визначити, яких вимог торкається ця зміна і які необхідно перевірити [35]. Трасування створюваних вимог відповідно до схеми розробленої моделі вимог в RequisitePro виглядає таким чином (рис. 2.4) .

На верхньому рівні знаходяться Потреби зацікавлених осіб (ідентифікатор даного виду вимог STRQ), а також Системні вимоги (SYST), сформульовані з Потреб зацікавлених осіб.

Функціональна вимога – це надана системою функціональність, зазвичай формулюється бізнес-аналітиком; призначення вимоги –

задовольнити потреби замовника. Функціональні вимоги (FEAT) розподіляються за різними підсистеми проекту на основі Системних вимог: FEAT_HW – функціональні вимоги до АЗ; FEAT_SW функціональні вимоги до ПЗ.

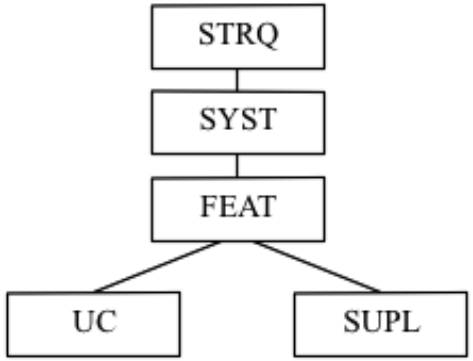


Рисунок 2.4. Трасування вимог в RequisitePro

Сценарій Використання (UC) – це опис поведінки системи в термінах послідовності дій.

Додаткова вимога (SUPL) – це інша вимога (зазвичай нефункціональна), яка не може бути охоплена сценаріями використання [21] .

Запропонована модель вимог, а також методика створення вимог були практично застосовані при реалізації проекту створення ВС керування рухомою платформою [22]. Описані нижче приклади являють собою взаємопов'язану послідовність вимог, що стосуються окремої частини системи.

Приклад 1. Вимоги користувача.

STRQ7: Зміна максимально допустимої швидкості рухомої платформи в залежності від категорії користувача (дорослі, діти), а також рівня навичок управління.

Приклад 2. Варіант використання.

UC3: Переключення швидкості рухомої платформи.

Дійові особи: Оператор.

Вихідні умови: оператор увімкнув Центральний пульт управління (ЦПУ) і виконав попередні налаштування заїзду рухомої платформи.

Нормальний напрям:

1. Оператор перемикає швидкісний режим рухомої платформи.
2. Система відправляє команду на рухому платформу про переключення швидкості.
3. Рухома платформа приймає команду і обробляє її.
4. На дисплеї з'являється повідомлення про переключення швидкості рухомої платформи в обраний режим.

Альтернативний напрям: немає.

Винятки: команда не була відправлена система видає номер помилки на дисплей.

Приклад 3. Системні вимоги.

SYST4: Система повинна переключати (збільшувати і зменшувати) максимальну швидкість радіокерованої рухомої платформи.

При переключенні швидкостей, система повинна виводити повідомлення на дисплей про швидкісний режим, який був обраний.

Приклад 4. Обмеження.

SUPL2: Наявність двох режимів завдання максимальної швидкості радіокерованої рухомої платформи:

режим Kids – 6 км/год.

режим Normal – 20 км/год.

В даному випадку вимога розбивається на функціональні вимоги до апаратного і програмного забезпечення.

Приклад 5. Вимоги до апаратного забезпечення.

FEAT_HW7: Двигун, встановлений на рухомій платформі, повинен як мінімум забезпечувати швидкість 20 км/год.

Функціональна вимога для ПЗ визначається, виходячи із заданих системних вимог і апаратної складової. Оскільки в нашому проекті на рухомій платформі був встановлений двигун, що забезпечує максимальну швидкість 60 км/год., то функціональна вимога до ПЗ буде виглядати, як наведено у наступному прикладі.

Приклад 6. Вимоги до ПЗ.

Подія: Встановлення швидкості на Центральному пульті управління в режим Kids.

Реакція: Встановлення 10% швидкості від максимальної швидкості платформи. На дисплеї ЦПУ з'являється напис, який підтверджує, що швидкість платформи була змінена «Speed Mode: Kids».

Подія: Встановлення швидкості на Центральному пульті управління в режим Normal.

Реакція: Встановлення 30% швидкості від максимальної швидкості платформи. На дисплеї ЦПУ з'являється напис, який підтверджує, що швидкість платформи була змінена «Speed Mode: Normal».

Також прикладами вимог зацікавлених осіб до створюваної ВС керування рухомою платформою є наступні:

- STRQ1: об'єкт (платформа) повинен бути дистанційно керованим;
- STRQ3: створити ефект присутності користувача в рухомій платформі;
- STRQ10: забезпечити збирання і зберігання станів системи за весь час роботи.

Дані вимоги в подальшому були сформульовані в високорівневі системні вимоги:

- SYST3: передача даних від пульта управління до об'єкта повинна здійснюватися з використанням радіоканалу на відстані не більше 40 м;
- SYST4: передавати відео-зображення по радіоканалу від камери, встановленої на рухомій платформі на відео-окуляри користувача;
- SYST12: організувати зберігання даних про поточний стан системи на флеш-пам'ять, встановлену в пульті управління (дану вимогу можна трасувати в функціональну вимогу до ПЗ: програма повинна зберігати одержувані дані з ЦПУ в форматі txt з розділювачами) .

Для успішної реалізації проекту за допомогою системи управління вимогами IBM Rational RequisitePro була виконана розробка необхідної проектної документації:

- Glossary – всі терміни проекту;
- Vision – повний опис системи і функцій системного рівня (системні вимоги);
- Use Case Specific – сценарії використання, алгоритми;
- Functional Specification – функціональні вимоги;
- Supplementary Specification – функціональні вимоги, не пов'язані зі сценаріями використання або нефункціональні (додаткові) вимоги.

За підсумками роботи з вимогами за допомогою RequisitePro, було розроблено набір документів, що включає в себе всі вимоги

до розроблюваної ВС керування рухомою платформою. Після того, як були сформульовані вимоги зацікавлених осіб та системні вимоги, була успішно розроблена архітектура системи, виконаний аналіз вимог до АЗ і ПЗ і розроблена архітектура ПЗ.

Практичне застосування запропонованої моделі та методики дозволило виконати ефективну розробку вимог і успішно реалізувати проект створення ВС керування рухомою платформою.

2.3 Базова концепція проектування вбудованих систем з використанням віддаленого та віртуального інструментарію

На даний час експерти [1,31,33,56] представляють і аналізують різні високорівневі методології проектування вбудованих систем:

- об'єктно-орієнтоване проектування;
- паралельне апаратно-програмне проектування;
- платформно-орієнтоване проектування;
- акторно-орієнтоване проектування;
- багатомовне проектування;
- аспектне проектування.

Паралельне апаратно-програмне проектування (Hardware-Software CoDesign) розглядається як один з перспективних підходів до проектування ВС. Дана методологія має як переваги, так і недоліки. З одного боку, це ускладнення процесу проектування вбудованих систем, а з іншого боку, значне поліпшення характеристик кінцевого продукту в порівнянні з альтернативними версіями проектних рішень.

В цілому, процес проектування вбудованих систем виглядає, як наведено на рис.2.5. Процес проектування апаратного та програмного забезпечення розподіляється певним чином у часі (рис.2.6). Сформована практика проектування вбудованих систем показує, що спочатку приймаються апаратні рішення. Далі, на цій основі здійснюється програмна надбудова. Це в свою чергу може призвести до великого ризику затримки всього циклу проектування виробів [31] .

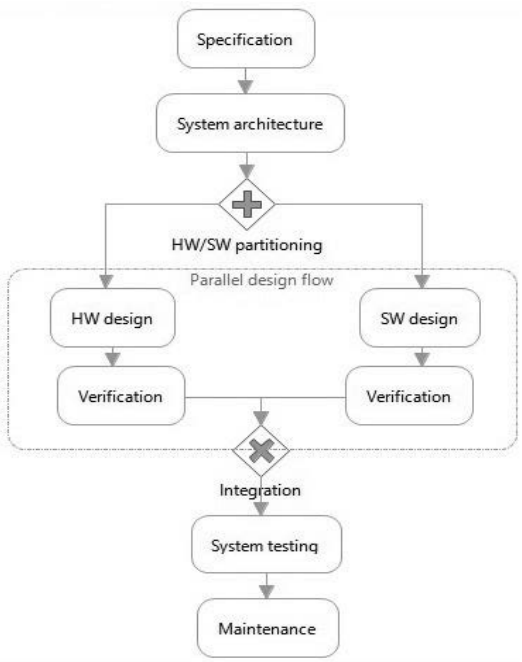


Рисунок 2.5. Класичний процес проектування вбудованих систем

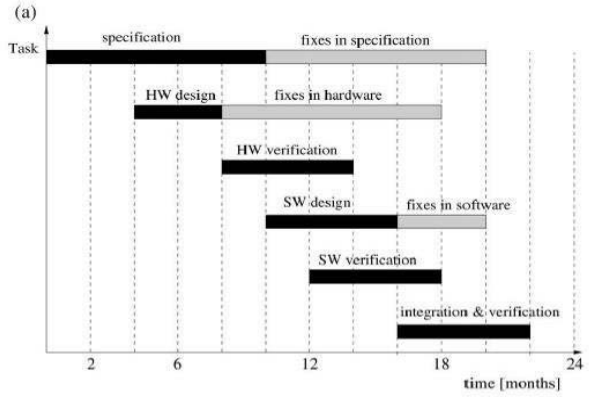


Рисунок 2.6. Часова діаграма класичного процесу проектування вбудованих систем

Після реалізації апаратних і програмних компонентів, вони повинні бути окремо перевірені, інтегровані, і далі вся система тестується в цілому. На етапі інтеграції апаратного та програмного забезпечення часто також виникають проблеми, оскільки для дотримання термінів системної інтеграції, часто доводиться розпочинати роботу з неповним набором готових рішень.

Оскільки ринок ВС постійно розвивається і вимагає створення більш складних систем за більш короткий час, розробка систем «з нуля» не є ефективною. Саме тому, в області проектування вбудованих систем популярно використовувати готові рішення, що дозволяють прискорити розробку продукту і таким чином скоротити час виходу на ринок.

Сьогодні виробники пропонують багато готових апаратно-програмних платформ, кожна з яких має свій форм-фактор і функціональність, а її вибір залежить від розв'язуваної задачі. При цьому, проектувальник ВС повинен знати можливості існуючих на ринку готових платформ і вміти прийняти відповідальне проектне рішення щодо застосування певної апаратно-програмної платформи. На жаль, інформація, яка пропонується на сайтах не завжди дозволяє зробити правильний обґрунтований вибір і оптимально реалізувати проект. Придбання малими та середніми підприємствами величезної кількості різних платформ для аналізу їх можливостей, незважаючи на порівняно невисокі ціни, є неприйнятним [54] .

У зв'язку з швидким розвитком популярності готових апаратно-програмних платформ, деякі компанії (напр. Autodesk) пропонують програми для віртуальної імітації їх роботи. Наприклад, можна виділити такі симулятори для Arduino як: 123D Circuits, Virtualbreadbord, Simulino, Virtronics, Simuino, Arduino Simulator та ін. [4,13,28,29,37]. Однак, симуляція не замінює реальної роботи з апаратною частиною і з програмним забезпеченням. Фактично, замість реального фізичного процесу симулятор дозволяє вивчити всього лише його математичну модель. Багато програм-симуляторів є платними, вимагають часу на їх вивчення, мають обмежені функціональні можливості і неповну елементну базу. Існують також потужні пакети схемотехнічного проектування, наприклад ISIS PROTEUS, Altium Designer та інші. Однак, застосування їх повного функціоналу не завжди потрібно, а вартість досить висока.

Таким чином, одним з ключових напрямків щодо підвищення ефективності проектування ВС, є накопичення технічних рішень для повторного використання. Різноманітність та роз'єднаність описів вже розроблених компонентів (апаратних блоків, алгоритмів, програмних реалізацій і т.д.) перешкоджає їх повторному використанню, але застосування віддалених експериментів дозволить дизайнерові отримати інформацію для прийняття рішень щодо реалізації ВС.

Тому, створення спеціалізованої лабораторії для дослідження готових апаратно-програмних платформ на основі віддаленого експерименту є актуальним завданням. Застосування віддаленої лабораторії вносить зміни в процес проектування вбудованих систем (Рис.2.7) .

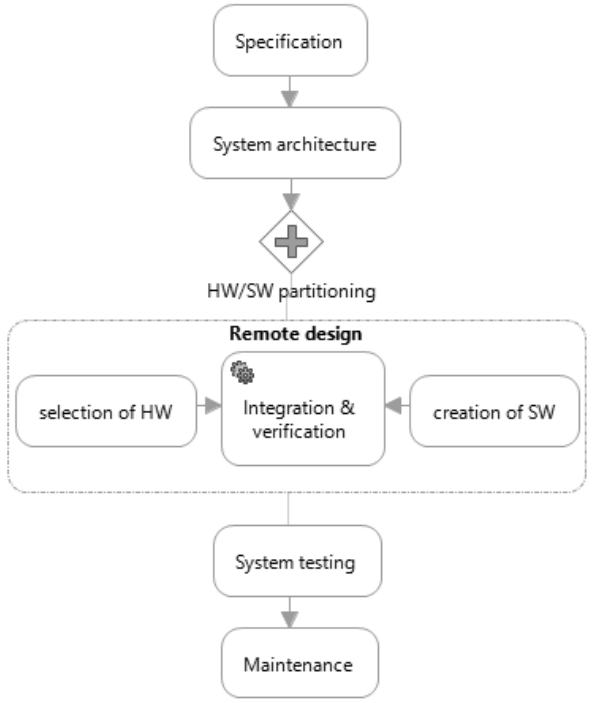


Рисунок 2.7. Процес проектування ВС з використанням віддаленого експерименту

За допомогою віддаленої лабораторії проектувальник може:

- отримувати та аналізувати інформацію про технічні характеристики готових апаратно-програмних платформ;
- виконувати вибір готових апаратних рішень;
- виконувати розробку та верифікацію програмного забезпечення;
- виконувати інтеграцію апаратного та програмного забезпечення ВС;
- тестувати на сумісність і працездатність апаратні і програмні компоненти проектованої системи;
- спостерігати проведення експерименту на реальному обладнанні.

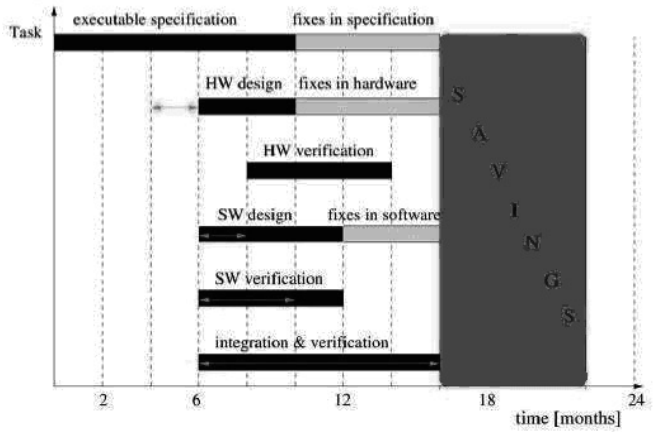


Рисунок 2.8. Часова діаграма процесу проектування вбудованих систем з використанням віддаленого експерименту

Використання віддаленої лабораторії дозволить проектувальнику швидко оцінити можливості готових платформ, прийняти рішення щодо їх інтеграції в проект та, таким чином, знизити ризики та часові витрати на проектування [23,24] .

2.4 Підходи до реалізації апаратного забезпечення вбудованих систем

Апаратна частина сучасних вбудованих систем може бути реалізована з використанням різноманітних технологій: промислові

комп'ютери, програмовані логічні контролери та контролери автоматизації, мобільні та інтернет-пристрої, контролерні і сенсорні мережі, мікроконтролери, сигнальні процесори, програмовані логічні інтегральні схеми (ПЛІС), замовні надвеликі інтегральні схеми, одноплатні комп'ютери. Всі технології для розробки вбудованих систем мають як сильні так і слабкі сторони (Рис.2.9). У більшості випадків, вибір залежить від складного поєднання чинників, і жодна з цих технологій не буде виступати в якості ідеального рішення [57] .

Необхідно відзначити, що розробники з метою популяризації та просування своїх виробів, створили відлагоджувальні плати (платформи). Найбільш відомими на сьогодні є: Arduino, Freeduino, Raspberry Pi, Cyclone Altera, Mbed та ін. (Рис. 2.10). Ці платформи оснащені всіма необхідними апаратними пристроями, за необхідністю до них можна підключити додаткові модулі, які призначені для розширення функціоналу базової плати.

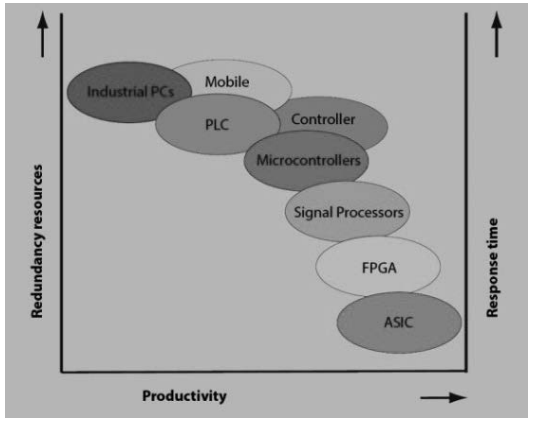


Рисунок 2.9. Технології реалізації вбудованих систем

Arduino торгова марка апаратно-програмних засобів для побудови простих систем автоматики і робототехніки, орієнтована на непрофесійних користувачів. Програмна частина складається з безкоштовною програмної оболонки (IDE) для написання програм, їх компіляції та програмування апаратури. Апаратна частина являє собою набір змонтованих друкованих плат, що продаються як офіційним виробником, так і сторонніми виробниками. Повністю відкрита архітектура системи дозволяє вільно копіювати або доповню-

вати лінійку продукції Arduino. Arduino може використовуватися як для створення автономних об'єктів автоматики, так і підключатися до програмного забезпечення на комп'ютері через стандартні дротові і бездротові інтерфейси.

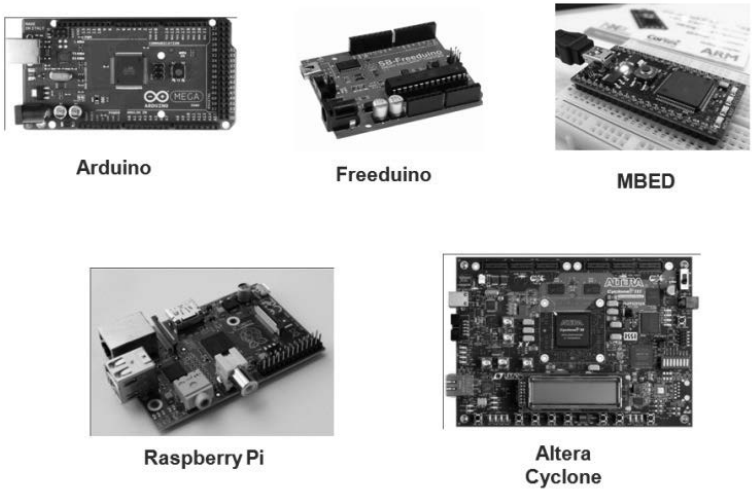


Рисунок 2.10. Приклади готових платформ

У концепцію Ардуіно не входить корпусний або монтажний конструктив. Розробник вибирає метод установки і механічного захисту плат самостійно. Сторонніми виробниками випускаються набори робототехничної електромеханіки, орієнтованої на роботу спільно з платами Ардуіно.

У лінійці пристроїв Arduino в основному застосовуються мікроконтролери Atmel AVR ATmega328, ATmega168, ATmega2560, ATmega32U4 з частотою тактирування 16 або 8 МГц. У старих виробах застосовувалися ATmega8, ATmega1280 та інші. Є також плата на процесорі ARM Cortex M3 AT91SAM3X8E з частотою тактування 84 МГц. У мікроконтролер попередньо прошивається завантажувач (BootLoader), тому зовнішній програматор не потрібен. Проте більшість плат має штирьовий рознім для внутрисхемного програмування [5] .

Freeduino є спільним проектом з відкритим вихідним кодом, спрямованим на реплікацію і виробництво Arduino-сумісних апаратних платформ. Схеми в форматах Freeduino Eagle SCH, BRD і Gerber дозволяють користувачам створювати плати, які на 100% функціонально, електрично і фізично сумісні з Arduino.

У той час, як Arduino є захищеною торговою маркою, Freeduino поставляється з безкоштовною і необмеженою ліцензією на використання імені Freeduino і є доступним для будь-якого використання.

Серед вже створених платформ існують аналоги таких варіантів Arduino як: Mega 2560, UNO, Duemilanove, Nano, Diecimila, Diecimila. Крім цього існують аналоги безлічі плат розширення, або так звані шилди.

Технічні характеристики, в цілому, є аналогічними платам Arduino, а якість складання може варіюватися залежно від виробників і проектувальників тих чи інших варіантів Freeduino. Серед явних переваг слід відзначити низькі ціни в порівнянні з оригінальними Arduino.

Freeduino можна використовувати практично у всіх галузях електроніки, де потрібно керувати системою за заданим алгоритмом з можливістю реагування на зовнішні сигнали.

За допомогою Freeduino можливо легко виготовити системи управління різними електронними пристроями: світломузика, сигналізація, кроковий двигун, рідкокристалічна панель, а також можливе застосування для вимірювання фізичних величин спільно з датчиками рівня, руху, ваги та багато іншого [14,46] .

Raspberry Pi являє собою компактний одноплатний комп'ютер. Він оснащений рознімами для цифрового і аналогового виводу відеосигналу і USB рознімами для підключення клавіатури і інших периферійних пристроїв. Ця багатоцільова платформа адаптована під програмування на мовах Python і Scratch. Вона здатна впоратися із завданнями звичайного настільного комп'ютера. Також Raspberry Pi оснащена мережевим інтерфейсом і виводами загального призначення, що дозволяє користувачеві реалізувати комунікацію платформи з всілякими пристроями.

Основним завданням розробників платформи було створення пристрою для освітніх цілей, що дозволяє отримати поглиблені знання про принципи роботи складної обчислювальної техніки. Платформа має підтримку декількох дистрибутивів ОС Linux, а так само RISC OS операційної системи для обчислювальних машин з архітектурою RISC.

Платформа має широкий спектр потенційних застосувань. Так, наприклад, вона може бути використана в якості малопотужного домашнього виділеного сервера. Іншим прикладом може служити плата розширення для смартфонів, що реалізує тепловізор. Також платформа знаходить застосування в напрямку «розумний дім», що активно розвивається сьогодні [52,59,61,63] .

Altera Cyclone® це ПЛІС, спроектована з розрахунком на низьку вартість. Завдяки вбудованій пам'яті, зовнішнім інтерфейсам пам'яті і схемі управління тактовою частотою, Cyclone є оптимальним рішенням для великої кількості завдань, в яких низька вартість є пріоритетною.

Cyclone забезпечують дешеву альтернативу інтегральним схемам спеціального призначення (ASIC), які зараз широко розповсюджені у вбудованих системах. Завдяки низьким ціна Cyclone дає можливість використовувати ПЛІС в масовому виробництві. ПЛІС широко використовується для побудови різних за складністю і за можливостями цифрових пристроїв. Це застосування, де необхідна велика кількість портів вводу-виводу (більш ніж 1000 виводів), цифрова обробка сигналу, цифрова відеоаудіоапаратура, високошвидкісна передача даних, криптографія, проектування та прототипування ASIC, мости (комутатори) між системами з різною логікою і напругою живлення, реалізація нейрочипів, моделювання квантових обчислень та ін. [53] .

Mbed являє собою платформу для розробки інтелектуальних пристроїв, яка заснована на 32-розрядній ARM Cortex-M архітектурі. Вона призначена для забезпечення високої продуктивності, швидкого створення прототипів і розробки продукції, з акцентом на пристрої, що працюють з постійним підключенням до Інтернету.

Інструментарій для програмування, що поставляється з Mbed, є чисто онлайновий. Код програм пишеться і компілюється в IDE в Інтернеті з AJAX-компонентами. Інтегрований з онлайн інструментами формами підтримки мережевої спільноти, обміну кодом та документацією.

Mbed поставляється з набором високорівневих об'єктно-орієнтованих бібліотек, які реалізують взаємодію з більшістю інтерфейсів вводу-виводу, наявних на платі. Контролери Mbed створені для управління пристроями, що працюють в режимі онлайн. Прикладом таких пристроїв може служити «розумна» побутова техніка, а також системи типу «розумний дім» [20] .

Зазвичай, вибір технології реалізації вбудованої системи обумовлюється функціоналом системи та вимогами до неї. Наприклад, при реалізації системи управління рухомими об'єктами (Рис. 2.11) використання мікроконтролерів (МК) дає ряд переваг: компактність, низька собівартість та мінімальне енергоспоживання, що дуже важливо для пристроїв батарейного живлення, зокрема, якими і є рухомі об'єкти. До того ж, як вважають фахівці, системи виконані на базі мікроконтролера, є більш гнучкими, оскільки керуюча програма може бути багаторазово опрацьована без змін апаратної частини пристрою [7,9,36,39,41,49] .

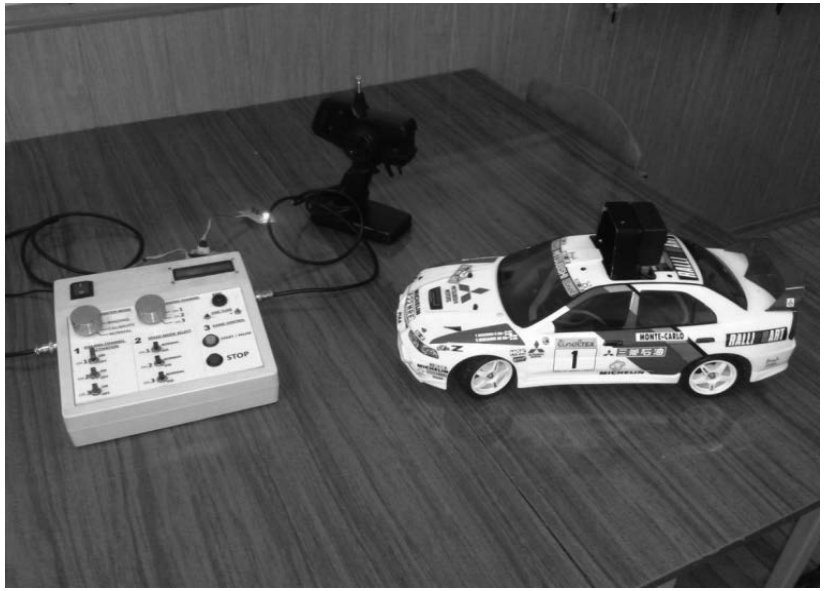


Рисунок 2.11. Система управління рухомим об'єктом

На сьогоднішній день існує класичний підхід до створення мікроконтролерних систем (МКС), представлений на рис. 2.12, якого дотримується переважна кількість розробників [60]. Цей підхід є трудомістким, вимагає багато часу на планування системи та високого рівня знань зі схемотехніки, електротехніки, програмування

МК та ін. (вивчення особливостей роботи використовуваних пристроїв, створення проектів монтажних плат, відлагодження програм та ін.) .

Використання готових платформ вносить свої зміни в етапи класичного підходу до створення вбудованих систем: По-перше, спрощується етап проектування, завдяки використанню готових конструктивних блоків. Крім того, спеціалізоване програмне забезпечення, яке постачається в комплекті з платами, високорівнева мова програмування і безліч стандартних функцій та бібліотек, спрощує написання та відлагодження програм [25] .

Але, необхідно зауважити, що виведення на широкий ринок готових платформ поставило перед їх розробниками задачу універсалізації доступу та роботи з цими пристроями. Ціною шаблонного принципу побудови стали проблеми застосування таких платформ. Так, для створення складних систем доводиться об'єднувати декілька платформ, що призводить до значної вартості, об'єму і маси та є неприйнятним для вбудованих пристроїв. З іншого боку, для організації роботи простих пристроїв (наприклад, світлофор) немає сенсу використовувати готову платформу, оскільки це буде коштувати дорожче, а повний функціонал обраної платформи не буде використовуватись.

Тому, іноді набагато простіше використовувати кілька МК з самостійно розробленими платами так, щоб кожен модуль виконував конкретні функції. Такий метод вимагає більшого часу планування системи та рівня знань, але позбавлений недоліків використання готових платформ і більше того, створює переваги, такі як: зниження енергоспоживання, відносна легкість налагодження і збільшення терміну експлуатації, що досягається за рахунок зниження кількості використовуваних елементів і рознімів.

Таким чином, для правильного вибору варіанту апаратної реалізації необхідно враховувати ряд факторів, основними з яких є вимоги до функціоналу і конструкції пристроїв.

Рисунок 2.12. Етапи проектування ВС на основі МК

2.5 Підходи до реалізації програмного забезпечення вбудованих систем

Успіх просування продукції на ринку суттєво залежить від часу, витраченого на розробку, у тому числі ПЗ. Час проектування ПЗ, в свою чергу, залежить від використовуваних підходів.

Так, наприклад, на платформі Raspberry Pi виконується ОС сімейства Linux або RISC OS, отже програмне забезпечення в разі використання такої платформи це скрипти та/або бінарні виконувані файли для відповідної операційної системи. У випадку створення мікроконтролерної системи, програмне забезпечення представляється набором машинних команд даного мікроконтролера на мові асемблера або С. Для програмно-апаратних платформ на основі мі-

кроконтролерів розробниками пропонуються набори бібліотек і середовища розробки, що суттєво спрощують процес програмування. Системи, що засновані на ПЛІС, мають два рівні програмування: на мові опису апаратури (наприклад VHDL), а також на мові програмування змодельованої мікропроцесорної системи (наприклад С) .

Аналіз використовуваних мов для програмування вбудованих систем свідчить, що у більшості випадків використовується сімейство С мов. Приблизно третя частина розробників вбудованих систем все ще мусить використовувати мову Асемблеру для невеликої частини низькорівневих процедур для деяких пристроїв. Результат цього аналізу наведено на рисунку 2.13.

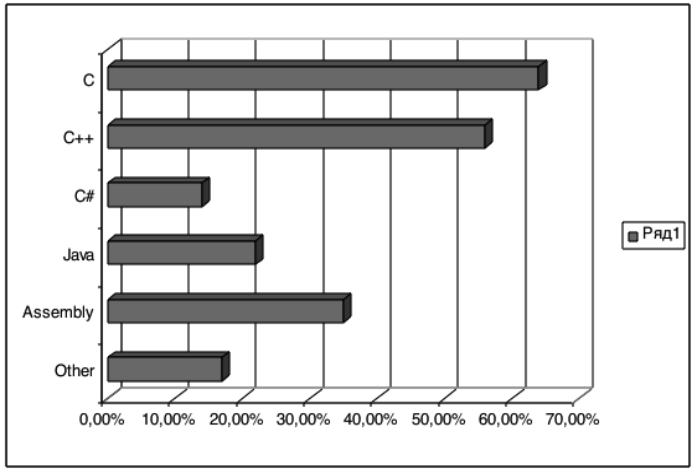


Рисунок 2.13. Мови програмування вбудованих систем

В загальному випадку, програмна частина вбудованих систем може бути реалізована з використанням операційних систем реального часу (Windows CE, QNX, VxWorks, FreeRTOS, OnTime RTOS та ін..) або шляхом організації програмного циклу (використання підходу «суперцикл» або «switch-case») .

Оскільки більша частина ВС, які розроблюються сьогодні, є системами реального часу, це накладає додаткові обмеження на розробку програмного забезпечення вбудованої системи. Провівши аналіз визначень «системи реального часу», які наводяться в різних роботах, можна підвести підсумок, що система реального часу повинна передбачати: здатність одночасно обробляти численні зовнішні події; своєчасність виконання обробки даних; передбачувану поведінку системи на зовнішні вхідні параметри; обмежений час реакції на вхідні сигнали.

В результаті пошуку методів проектування ПО для вбудованих систем, було виявлено декілька підходів. Один з них використання операційних систем реального часу (ОСРЧ), другий організація програми з використанням IDE [45,65] .

Аналіз робіт з даної тематики показав, що ОСРВ мають свої переваги і недоліки [38,50] .

ОСРЧ це планувальник паралельних асинхронних завдань в «реальному часі». Порядок, в якому виконуються завдання, визначається Планувальником. Він самостійно забезпечує те, щоб задача з високим пріоритетом виконувалася перш за все, а з низьким тільки лише тоді, коли залишається час. Без ОСРВ потрібно самостійно реалізовувати затримки і таймери необхідні для виконання програми з безліччю кінцевих автоматів, що є трудомістким процесом.

Операційні системи реального часу мають модульну структуру і основне ядро, яке не залежить від особливостей мікроконтролера, що дозволяє переносити їх на різні мікроконтролери. При написанні програми мовою Асемблера, це складно зробити, що пов’язано не тільки з різною системою команд, але і з наявністю у мікроконтролерів різних функціональних блоків. Ще одна перевага ОСРЧ: якщо раптом буде допущена помилка в коді, то «впаде» тільки те завдання, в якому буде присутня помилка (але можливо також, що і всі завдання з нижчим пріоритетом теж), забезпечуючи захист від перевантаження.

Однак, застосування ОСРЧ не вирішує всіх проблем, що виникають при розробці ПЗ. По-перше, застосування ОСРЧ є обмеженим порівняно потужними мікроконтролерами, оскільки ОСРЧ для своєї роботи вимагає ресурси мікроконтролера: пам’ять і системний час, яких і так небагато (мінімальні вимоги ОСРЧ до МК: 512 байт ОЗУ і 4–16 Кб пам’яті програми). І замість того, щоб пустити їх на завдання, доводиться віддавати Планувальнику. По-друге, ОСРЧ застосовують там, де необхідно організувати виконання і взаємодію декількох програмних потоків. Розробка додатків, що складаються з безлічі асинхронних потоків, що виконуються найчастіше з різними

пріоритетами, вимагає від програміста застосування складних засобів організації взаємодії потоків. Плюс, необхідність додаткових знань і навичок для не дуже простої організації завдань: м'ютекси, семафори, пріоритети і т.п.

Таким чином, ОСРЧ є необхідним і корисним засобом при створенні складних багатозадачних систем реального часу, сприяє спрощенню етапу розробки і скороченню часу створення ПЗ. Але, все це за умови можливості виділення додаткових ресурсів МК для забезпечення роботи ОСРЧ, а також наявності досвіду застосування ОСРЧ.

При використанні спеціалізованих IDE рішення задачі проектування ПЗ зводиться до вибору варіанта організації програмного циклу: «суперцикл» або switch-технологія.

Програма за зразком «суперцикл» виглядає, як нескінченний цикл, в якому МК послідовно робить все, що він повинен робити.

МК може працювати з ВС, яка має зовнішню периферію, двома способами: шляхом опитування або використовуючи переривання. У першому випадку ми періодично будемо опитувати, чи немає даних, доки не отримаємо їх. У другому випадку налаштовуємо МК так, щоб він генерував переривання в той момент, коли з'являються нові дані. Особливість роботи з перериваннями така, що обробник переривання (код, який буде викликаний безпосередньо в той момент, коли переривання відбувається) повинен бути якомога коротшим.

Якщо, наприклад, необхідно організувати програмний захист від перевантаження, який має спрацьовувати якнайшвидше після настання події перевантаження (інакше все згорить), то загальний цикл не дозволить цього зробити, тому, що всі події в ньому виконуються по порядку. І гарантований час реакції на подію ніяк не може бути меншим, ніж час виконання однієї ітерації циклу. І якщо десь в цьому циклі буде помилка тоді перестане працювати вся система.

Така структура зручна для виконання дуже невеликої кількості завдань. У разі необхідності системі опитувати десяток датчиків, реагувати на кілька критичних подій, обробляти команди, послідовний загальний цикл просто не дозволить цього зробити, тому що для кожної з подій є свої обмеження в часі реакції і частоті опитування або управління.

Другим варіантом програмної реалізації є switch-технологія (базова конструкція автомати). Switch-технологія називається також «автоматне програмування». Вона заснована на уявленні програми у вигляді графа переходів скінченного автомата. Switch-технологія отримала назву від оператора switch мови С, який є основною структурою. У цьому випадку організація задач хоч і не проста, але зводиться до визначення: де, яку функцію запускати. Після виконання завдання програма повертається в суперцикл.

Застосування switch-технології в ряді випадків знімає необхідність використання ОСРВ (проте не виключає) для складних і багатозадачних систем. По-перше, взаємодія між автоматами здійснюється простішим чином, ніж між потоками ОСРВ, по-друге, застосування автоматів робить поведінку програми абсолютно детермінованою, а взаємодія потоків в ОСРВ вимагає додаткових (і немалих) зусиль для усунення колізій.

Перша перевага Switch-технології: програми, побудовані за пропонованою технологією, легко документувати. Вона являє собою сукупність скінченних автоматів, які можуть бути виражені в наочній графічній формі зі зв'язками і внутрішніми структурами автоматів. Перевага друга: можливість повторного використання коду. Автомат в пропонованому стилі програмування є свого роду «цеглинкою», автономною одиницею програми. Його зв'язки з іншими автоматами зведені до мінімуму і уніфіковані. Його можна розробити окремо, а потім застосовувати в різних програмних проектах. Звідси випливає і третя перевага: програма легко піддається модифікації, оскільки кількість зв'язків між автоматами мінімальна [65] .

Таким чином, якщо система дуже проста і невимоглива до часу реакції, її простіше зробити за зразком «суперцикл». Якщо ж система складна, поєднує в собі багато різних дій і реакцій, які до того ж критичні до часу, тоді більш доцільним є використання технології switch-case або операційних систем реального часу.

Кожен з цих підходів має свої переваги і недоліки, і вибір того чи іншого підходу залежить від завдань проекту, а також, умінь і навичок розробника.

3 ПРОЕКТУВАННЯ ВБУДОВАНИХ СИСТЕМ З ВИКОРИСТАННЯМ ВІРТУАЛЬНОГО ТА ВІДДАЛЕНОГО ІНСТРУМЕНТАРІЮ

3.1 Інтегроване середовище розробки вбудованих систем

При проектуванні ВС важливу частину розробки займає процес моделювання та відлагодження майбутнього електронного пристрою за допомогою ECAD-системи (Electronic Computer-Aided Design). В цій галузі сьогодні пропонується багато рішень: Altium Designer, OrCAD, Proteus VSM, Eagle та ін..

Altium Designer комплексна система проектування електронних пристроїв на базі друкованих плат, яка дозволяє розробнику створювати проекти, починаючи з принципової схеми та VHDL-опису ПЛІС, проводити моделювання отриманих схем і VHDL-кодів, готувати файли для виробництва, а концепція Live Design, так зване живе проектування, дозволяє завершити проект його налагодженням на платі NanoBoard. Контроль цілісності проекту дозволяє відслідковувати зміни в частинах проекту і синхронізувати їх. Використовуючи плату налагодження NanoBoard можливо налагоджувати ПЛІС-проекти на етапі створення принципової схеми [2,3,55,62,64] (рис.3.1) .

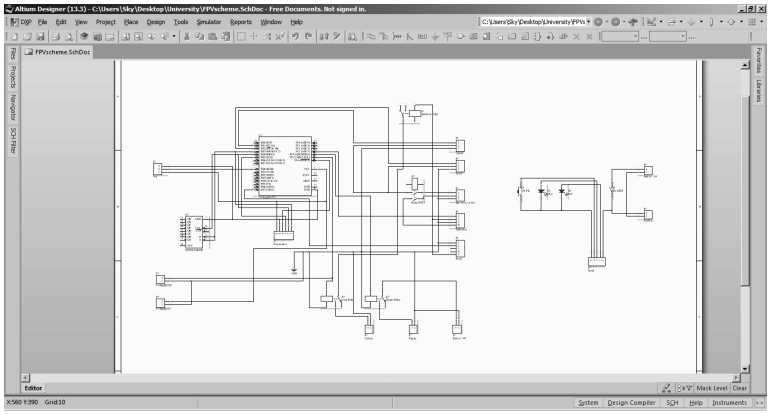


Рисунок 3.1. Приклад проекту в Altium Designer

PROTEUS VSM – програма-симулятор мікроконтролерних систем та електронних схем. Підтримує МК: PIC, 8051, Atmel, HC11, ARM7/LPC2000 та інші. Дозволяє достовірно моделювати і налагоджувати досить складні пристрої, в яких може міститися кілька МК одночасно і навіть різних родин в одному пристрої. Пакет Proteus складається з двох частин: ISIS програма синтезу та моделювання безпосередньо електронних схем і ARES програма розробки друкованих плат [26] (Рис.3.2) .

Моделювання роботи програмованих пристроїв мікроконтролерів, мікропроцесорів, DSP та ін., здійснюється шляхом завантаження в програму файлу з *.hex розширенням. Hex – файл створюється спеціальними інструментами (наприклад, AVR Studio, WinAVR, Atmel Studio). Після розробки електронних моделей модулів системи та програм для МК, проводиться моделювання роботи системи в цілому, шляхом об’єднання в один проект Proteus всіх модулів системи та завантаження *.hex-файлів. Це дозволяє виявити помилки, які були допущені або у програмі, або у самій схемі, а також відстежити й налагодити алгоритм роботи МК. Після цього, hex-файли прошиваються у програмну пам’ять мікроконтролерів і тестуються вже на прототипі пристрою (рис.3.3) .

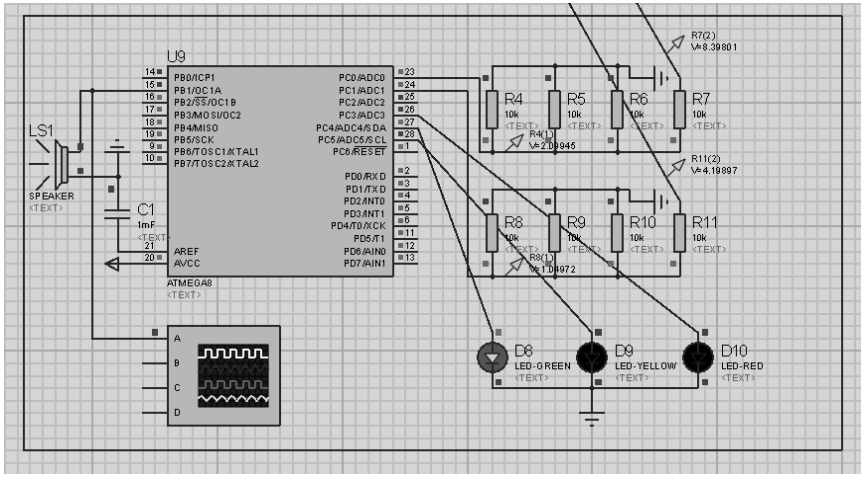


Рисунок 3.2. Приклад проекту в Proteus VSM

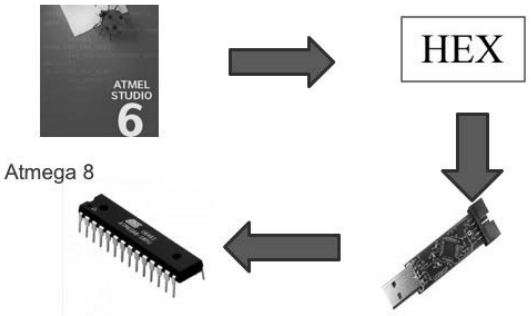


Рисунок 3.3. Основні етапи проекту

Atmel® Studio інтегроване середовище розробки для розробки 8- і 32- бітних додатків від компанії Atmel. Atmel Studio надає повне і просте у використанні середовище для написання та налагодження додатків, на мові /C++ або на асемблері. Містить асемблер і симулятор, що дозволяє відстежити виконання програми. Atmel Studio містить в собі менеджер проектів, редактор вихідного коду, інструменти віртуальної симуляції та внутрішньо схемного налагодження [6] (рис.3.4) .

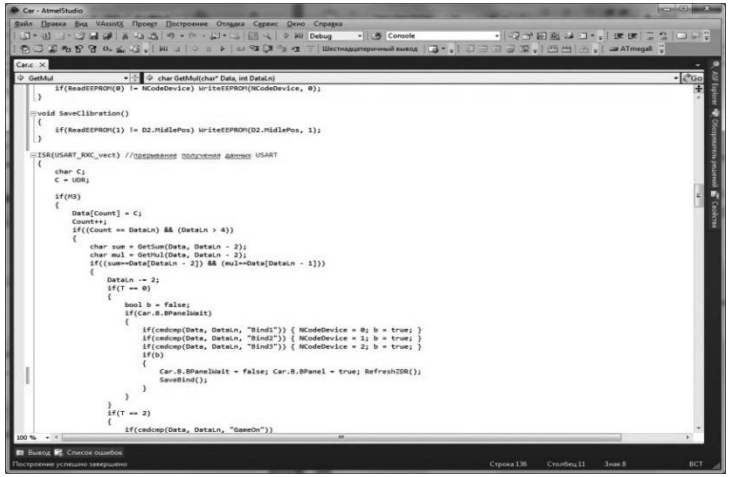


Рисунок 3.4. Робоче середовище Atmel Studio

Arduino це зручна платформа швидкої розробки електронних пристроїв для новачків і професіоналів. Платформа користується величезною популярністю в усьому світі завдяки зручності і простоті мови програмування, а також відкритій архітектурі і програмному коду. Пристрій програмується через USB без використання програматорів. Мікроконтролер на платі програмується за допомогою мови Arduino (заснованій на мові Wiring) і середовища розробки Arduino (засноване на середовищі Processing). Проекти пристроїв, засновані на Arduino, можуть працювати самостійно, або ж взаємодіяти з програмним забезпеченням на комп'ютері (Flash, Processing, MaxMSP). Плати можуть бути зібрані користувачем самостійно або придбані готові. Програмне забезпечення доступне для безкоштовного скачування [5] .

Віртуальний прототип рухомої платформи може бути спроектований та досліджений в середовищі будь-якої MCAD-системи. *PTC Creo* це масштабоване, функціонально сумісне програмне забезпечення для автоматизованого конструювання виробів. Дозволяє командам створювати, аналізувати, переглядати і максимально використовувати проекти виробів, засобами дво- та тривимірного проектування, параметричного і прямого моделювання [8,30,55] (рис.3.5) .

Рисунок 3.5. Приклад проекту рухомої платформи в середовищі Creo

3.2 Проектування вбудованих систем з використанням віддаленого експерименту

3.2.1 Спеціалізована віддалена лабораторія RELDES

Використання віддаленої лабораторії для комплексного апаратно-програмного проектування вбудованих систем є актуальним завданням, оскільки існуючі в цій області проблеми вимагають якісно нових методик, технологій і засобів проектування ВС.

Спеціалізована віддалена лабораторія проектування вбудованих систем RELDES (REmote Laboratory for Embedded Systems Design) орієнтована на допомогу розробнику у швидкому та ефективному виборі типових проектних рішень на основі готових апаратно-програмних платформ.

Лабораторія забезпечує віддалений доступ до експериментального обладнання на базі платформи Arduino. Як вже зазначалося, Arduino є потужним мікроконтролером з великим об'ємом пам'яті, що дозволяє створювати складні електронні пристрої. Крім того, апаратно-обчислювальна платформа Arduino є однією з найбільш популярних і простих у вивченні. Порівняно з іншими схожими платформами Arduino має ряд переваг:

- проект розроблявся і розвивається як проект з відкритим кодом, який працює як мережевий проект/спільнота, дозволяючи учасникам обмінюватися досвідом і готовими прикладними напрацюваннями, додатково прискорюючи процес розробки і налагодження;
- низька вартість самого мікроконтролера і розширень до нього;
- простота і кросплатформність середовища програмування (ОС Windows, Macintosh OSX і Linux: 32 / 64bit) .

Лабораторія дозволяє користувачам: отримувати і аналізувати інформацію про специфікації готових апаратних і програмних компонентів для їх повторного використання, виконувати вибір апаратних платформ, розробку програмного коду, тестування на сумісність та працездатність апаратної і програмної частин проектованої системи. Віддалене тестування компонентів дозволяє швидко оцінити можливість їх застосування та прийняти рішення про їх інтеграцію в проект, тим самим знижуючи ризики і часові витрати на проектування.

Набір експериментів залежить від специфіки проектованої ВС. На цей час, в якості типових рішень для проектування системи управління рухомими об'єктами пропонуються експерименти на основі серводвигунів, рідкокристалічного дисплею, світлодіодного світлофора, ультразвукового датчика відстані та ін.

Запропонований набір експериментів може бути розширений для інших програмно-апаратних платформ, а також завдань, пов'язаних з реалізацією різних класів ВС.

Стенд з експериментами з'єднується з сервером лабораторії через послідовний інтерфейс. Як сервер віддаленої лабораторії виступає комп'ютер з ОС Linux Debian. Програма-сервер надає користувачеві доступ до програмування експерименту і забезпечує перегляд його виконання через веб-камеру. Потокове відео реалізовано за допомогою ffmpeg. Сервер обробляє запити від веб-клієнта та залежно від отриманих даних, здійснює наступні дії:

- компілює отриманий вихідний код;
- повертає результати компіляції;
- завантажує об'єктний код в контролер (за умови, що контролер вільний);
- ставить клієнта в чергу (якщо експеримент зайнятий) .

Спілкування клієнта з сервером здійснюється через розроблений веб-інтерфейс, що реалізований з використанням HTML + CSS + JS. Сервіс запущений на сервері Apache з використанням реляційної бази даних MySQL. Для доступу до віддаленої лабораторії не потрібно встановлювати додатково ніяке інше програмне забезпечення, необхідний тільки актуальний веб-браузер (рис. 3.6, 3.7, 3.8) .

Лабораторія забезпечує користувача інформацією щодо програмних та апаратних рішень для подальшого використання, дозволяє отримати навички програмування вбудованих систем. Сценарій використання лабораторії передбачає як використання готових шаблонів програм для стендових пристроїв, так розробку програм користувачами з нуля.

Переваги нашої лабораторії:

- кросплатформність (використовуйте будь які пристрої та ОС);

- доступність (використовуйте наше обладнання замість витрачання грошей);
- конфігурованість (пишіть свій власний код або використовуйте шаблони) .

Практичне застосування розробленої лабораторії дозволить проектувальнику прийняти правильне проектне рішення про доцільність застосування тієї чи іншої готової платформи, а також щодо можливих варіантів розвитку проекту [23,24,27,54] .

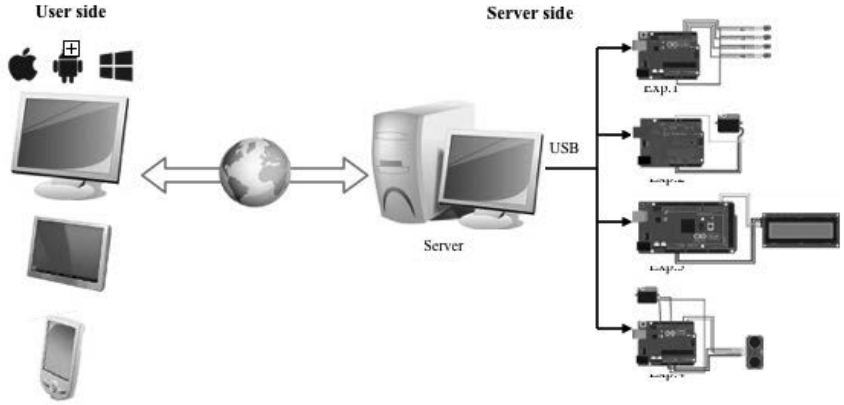


Рисунок 3.6. Архітектура віддаленої лабораторії RELDES

Рисунок 3.7. Загальний інтерфейс віддаленої лабораторії RELDES

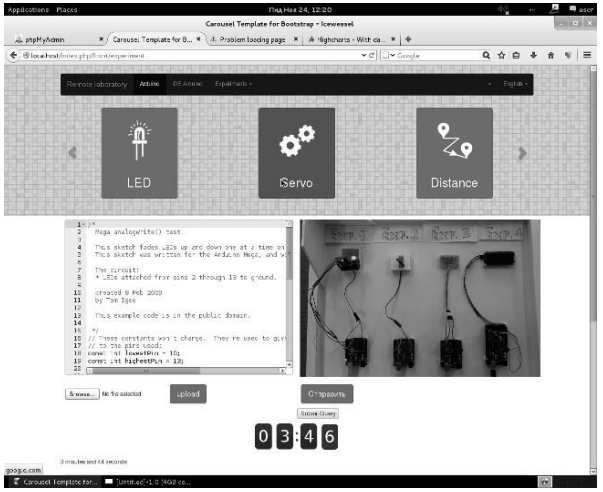


Рисунок 3.8. Інтерфейс сторінки експериментів лабораторії RELDES

3.2.2 Опис набору експериментів лабораторії RELDES

Експеримент 1. Світлофор. В даному експерименті проводиться керування набором світлодіодів. Управління здійснюється через цифрові виводи плати Arduino. Діоди підключені через струмообмежуючі резистори і мають спільне заземлення (рис. 3.9, 3.10). Сценарії експерименту: Почергове включення, Стрічка, що біжить, Плавна зміна яскравості.

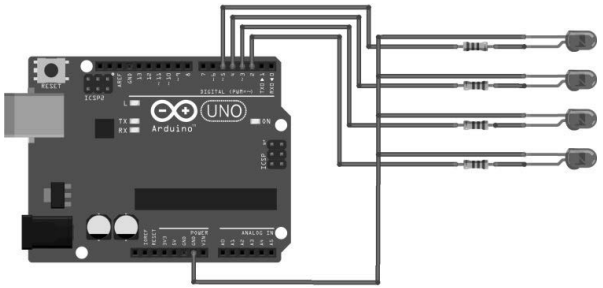


Рисунок 3.9. Модель експерименту Світлофор

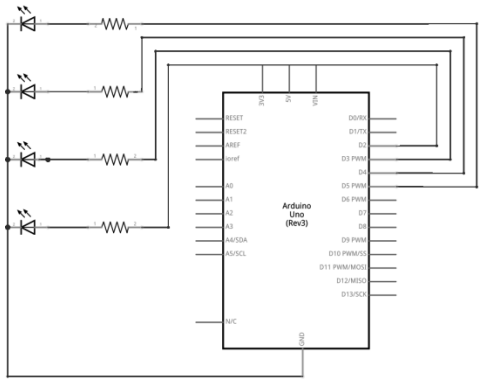


Рисунок 3.10. Схема реалізації експерименту Світлофор

Приклади реалізації експерименту «Світлофор» наведені нижче. Програмний код складається з функцій ***setup*** і ***loop***. Перша виконується один раз при включенні плати. В даному випадку в ній здійснюється установка виводів з 2 по 13 в режим виходу за допомогою виклику ***pinMode***, параметрами якого є номер виводу і його режим. Друга функція ***loop*** виконується в нескінченному циклі весь інший час роботи плати. Для зміни яскравості діодів викликається функція ***analogWrite,*** аргументами якої є номер виводу та напруга. Напруга змінюється в діапазоні 0–5В. Значення напруги не використовується безпосередньо як параметр ***analogWrite***, а число від 0 до 255, відповідне напрузі на проміжку 0–5В. Зміна напруги відбувається за допомогою широтно-імпульсної модуляції. Також в програмі використовується виклик функції ***delay***, яка призупиняє роботу плати на вказану кількість часу (мс). В даному випадку це потрібно для того, щоб робота виконувалася досить повільно, щоб можна було помітити зміну яскравості.

Сценарій 1.1. В цьому прикладі в циклі здійснюється послідовне включення та виключення кожного діода з затримкою в 700мс.

```
const int lowestPin = 2;
const int highestPin = 5;
void setup () {
for (int thisPin =lowestPin; thisPin <= highestPin;
thisPin++) {
pinMode (thisPin, OUTPUT);
```



```
}
}
void loop () {
for (int thisPin =lowestPin; thisPin <= highestPin;
thisPin++) {
analogWrite (thisPin, 255);
delay (700);
analogWrite (thisPin, 0);
}
}
```

Сценарій 1.2. В цьому прикладі в функції loop виконується почергове включення всіх діодів в першому циклі та їх послідовне виключення у другому. Ці цикли чергуються.

```
const int lowestPin = 2;
const int highestPin = 5;
void setup () {
for (int thisPin =lowestPin; thisPin <= highestPin;
thisPin++) {
pinMode (thisPin, OUTPUT);
}
}
void loop () {
for (int thisPin =lowestPin; thisPin <= highestPin;
thisPin++) {
analogWrite (thisPin, 255);
delay (300);
}
for (int thisPin =lowestPin; thisPin <= highestPin;
thisPin++) {
analogWrite (thisPin, 0);
delay (300);
}
}
```

Сценарій 1.3. Тут почергово для кожного діоду здійснюється плавна зміна яскравості від нуля до максимуму та зворотньо. Зміна яскравості здійснюється у вкладених циклах.

```
const int lowestPin = 2;
const int highestPin = 5;
void setup () {
for (int thisPin =lowestPin; thisPin <= highestPin;
thisPin++) {
pinMode (thisPin, OUTPUT);
}
```

```
}
void loop () {
for (int thisPin =lowestPin; thisPin <= highestPin;
thisPin++) {
for (int brightness = 0; brightness < 255; brightness++)
{
analogWrite (thisPin, brightness);
delay (2);
}
for (int brightness = 255; brightness >= 0; brightness—)
{
analogWrite (thisPin, brightness);
delay (2);
}
delay (100);
}
}
```

Експеримент 2. Серводвигун. У даному експерименті проводиться керування серводвигуном. Управління проводиться послідовним кодом через цифровий вивід. Серводвигун має три виводи: живлення, земля і сигнальний (рис.3.11, 3.12). Сценарії експерименту: Поворот на дві позиції, Плавне обертання.

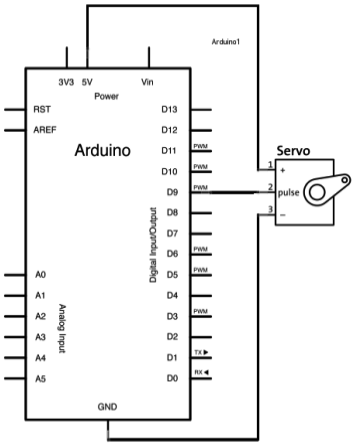


Рисунок 3.11. Модель експерименту Серводвигун

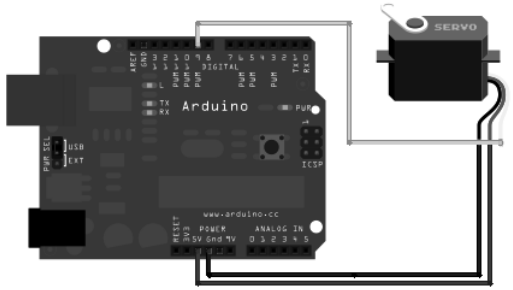


Рисунок 3.12. Схема реалізації експерименту Серводвигун

Приклади реалізації експерименту «Серводвигун» наведені нижче. В програмі підключена бібліотека ***Servo.h,*** що відповідає за управління серводвигуном. Вона надає можливість користуватися класом ***Servo***, через який і здійснюється керування. На початку програми створюється об'єкт цього класу. Програмний код складається з функцій ***setup*** і ***loop***. Перша виконується один раз при включенні плати. В ній здійснюється установка керуючого виходу для серводвигуна на вивід 9 за допомогою виклику методу **attach** об'єкта **servo**. Друга функція ***loop*** виконується в нескінченному циклі весь інший час роботи плати. В даній функції викликаються команди управління серводвигуном, що задають його позицію в градусах (метод ***write***). Функція ***delay*** використовується для призупинення роботи плати на вказану кількість мілісекунд. У даному випадку це необхідно, оскільки швидкість роботи серводвигуна механічно обмежена, і він не здатний змінювати позицію настільки швидко, наскільки контролер здатний це запитувати. Слід зазначити, що діапазон роботи серводвигунів може відрізняться в залежності від двигуна і не завжди дорівнює 0–180.

Сценарій 2.1. Обертання на дві позиції. Тут серводвигуну почергово задаються 2 позиції. Між цим здійснюється затримка в 2 секунди, щоб двигун встиг здійснити обертання до отримання наступної команди.

```
#include <Servo.h>
Servo myservo;
int pos = 0;
```

```
void setup ()
{
myservo.attach (9);
}
void loop ()
{
      if (0 == pos) pos = 120;
else pos = 0;
myservo.write (pos);
delay (2000);
}
```

Сценарій 2.2. Плавне обертання. Тут здійснюється плавне обертання двигуна. Для цього в циклі кожні 15мілісекунд викликається функція зміни позиції двигуна з різницею в 1 градус.

```
#include <Servo.h>
Servo myservo;
int pos = 0;
void setup ()
{
myservo.attach (9);
}
void loop ()
{
for (pos = 0; pos < 180; pos += 1)
{
myservo.write (pos);
delay (15);
}
for (pos = 180; pos>=1; pos-=1)
{
myservo.write (pos);
delay (15);
}
}
```

Експеримент 3. Рідкокристаличний дисплей. У даному експерименті проводиться керування рідкокристалічним дисплеєм. Управління зображенням дисплея здійснюється через цифрові виводи за протоколом послідовної передачі даних I2C (рис.3.13, 3.14). Сценарії експерименту: Hello, world!, Динамічний рядок.

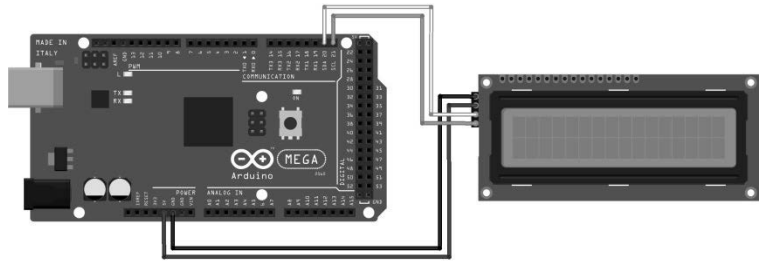


Рисунок 3.13. Модель експерименту Рідкокристалічний дисплей

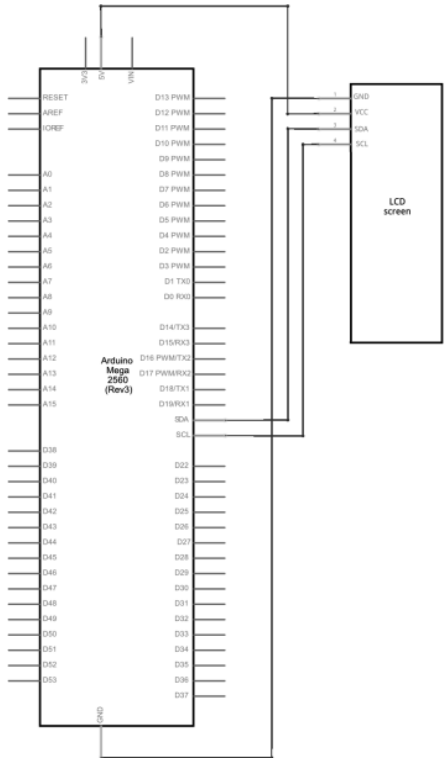


Рисунок 3.14. Схема реалізації експерименту Рідкокристалічний дисплей

Нижче наведено різні приклади реалізації експерименту Рідкокристаличний дисплей. У першу чергу слід відзначити, що в програмі підключені бібліотеки ***LiquidCrystal_I2C.h*** (відповідає за управління РК-дисплеєм) та ***Wire.h*** (реалізує I2C протокол). Вони надають можливість користуватися класом ***LiqudCrystal_I2C,*** через який і проводиться керування. Можна побачити як на початку програми створюється об'єкт цього класу. В даному випадку параметри відповідають вбудованому в екран контролеру та змінювати їх не рекомендується.

Програмний код складається з функцій ***setup*** і ***loop***. Перша виконується один раз при включенні плати. У даному випадку в ній здійснюється установка параметрів роботи РК-дисплею за допомогою виклику ***begin*** с параметрами, які дають зрозуміти кількість стовпців і рядків символів. Для увімкнення/вимкнення вбудованої підсвітки дисплея використовуються функції ***backlight/noBacklight.*** Друга функція ***loop*** виконується в нескінченному циклі весь інший час роботи плати. У даному випадку в ній здійснюється установка курсору в початок другого рядка (нумерація здійснюється з нуля) за допомогою виклику ***setCursor*** та друк часу роботи плати в секундах. Функція ***print*** викликається для виводу на екран вказаного набору символів. Виклик ***clear*** дозволяє очистити вміст екрану, а функція ***delay*** призупиняє роботу плати на вказаний в мілісекундах час.

Сценарій 3.1. Hello, world! В даному прикладі в перший рядок екрану виводиться повідомлення «Hello, world!», а після секундної затримки у другий рядок виводиться «I am display». Через мить після цього екран очищується. Дії повторюються циклічно.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd (0x27, 2, 1, 0, 4, 5, 6, 7, 3,
POSITIVE);
void setup () {
lcd.begin (16,2);
lcd.backlight ();
}
void loop () {
lcd.setCursor (0,0);
lcd.print («Hello, world!»);
delay (1000);
```



```
lcd.setCursor (0,1);
lcd.print («I am display»);
delay (1000);
lcd.clear ();
delay (1000);
}
```

Сценарій 3.2. Динамічний рядок. В даному прикладі реалізовано рядок, що рухається. В кожній ітерації циклу здійснюється зсув початкової позиції курсору, з якої на наступній ітерації буде виконуватися виведення тексту.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd (0x27, 2, 1, 0, 4, 5, 6, 7, 3,
POSITIVE);
int col_offset = 0;
int row_offset = 0;
void setup () {
lcd.begin (16,2);
lcd.backlight ();
}
void loop () {
lcd.setCursor (col_offset, row_offset);
lcd.print («Running text»);
delay (100);
lcd.clear ();
col_offset = (col_offset + 1)% 16;
if (15 == col_offset) row_offset = (row_offset + 1)%
2;
}
```

Експеримент 4. Ультразвуковий датчик відстані. У даному експерименті проводиться зчитування показань ультразвукового датчика відстані. Датчик має три виводи: живлення, земля і сигнальний. Сигнальний вивід підключений на цифровий вивід плати, дані надходять в послідовному коді (Рис.3.15, 3.16). Сценарії експерименту: Вимірювання дистанції, Датчик відстані з серводвигуном.

Нижче наведено приклади реалізації експерименту «Датчик відстані». Передача показників датчика відстані на комп'ютер користувача здійснюється через послідовний інтерфейс. Для роботи з ним використовується клас ***Serial***, через який і відбувається вся взаємодія з інтерфейсом. Програмний код складається з функцій ***setup*** і ***loop***. Перша виконується один раз при включенні плати.

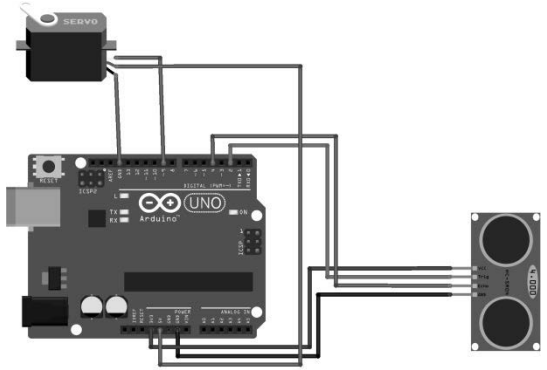


Рисунок 3.15. Модель експерименту Ультразвуковий датчик відстані

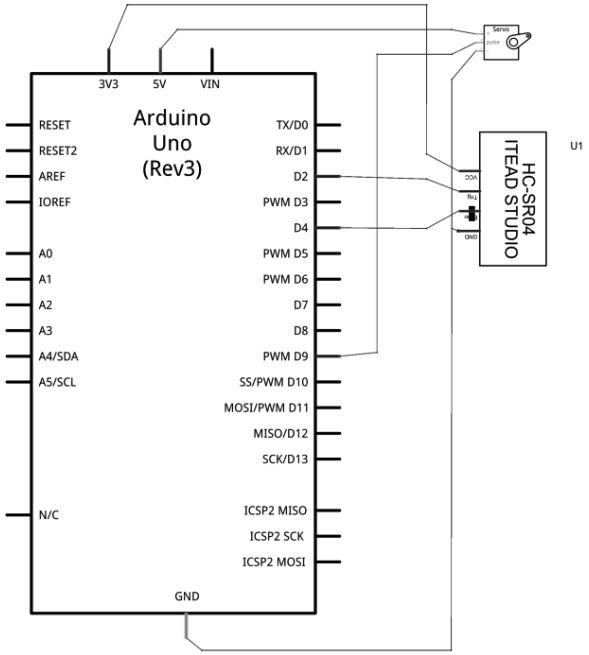


Рисунок 3.16. Схема реалізації експерименту
Ультразвуковий датчик відстані

У даному випадку в ній здійснюється ініціалізація послідовного інтерфейсу з частотою 9600 за допомогою виклику методу ***begin***, параметром якого служить частота передачі даних. Здійснюється встановлення режимів для виводів датчика функцією ***pinMode***, параметрами якої служать номер виводу та режим. Також перед описом функції ***setup*** оголошують змінні, що зберігають номери виводів, до яких буде підключений датчик відстані. Друга функція ***loop*** виконується в нескінченному циклі весь інший час роботи плати. Робота з датчиком відстані здійснюється в два етапи, що постійно чергуються: запит на вимірювання і зчитування показників. Оскільки датчик є ультразвуковим, показники, які повертаються ним, являють собою час, за який ультразвукової сигнал пройшов до об'єкта і назад. Через це, після зчитування показників слід перетворити значення, щоб обчислити відстань. Для запиту на датчик протягом більше 2мкс подається логічна одиниця. Перед та після неї передається логічний нуль. Це потрібно для того, щоб впевнитися, що датчик правильно зчитав команду. Дані операції здійснюються за допомогою викликів ***digitalWrite***, параметрами якого є номер виводу і значення (LOW або HIGH). Між викликами здійснюється затримка за допомогою функції ***delayMicroseconds.*** Для читання показників датчика зчитується тривалість вхідного сигналу. Це проводиться за допомогою виклику функції ***pulseIn***, параметрами якої є номер входу і сигнал, тривалість якого слід дізнатися (логічний 0 або 1). Ця функція повертає тривалість сигналу в мкс. Отримане значення заноситься в змінну ***duration.*** Після отримання цього значення, за допомогою функцій ***microsecondsToCentimeters*** і ***microsecondsToInches*** здійснюється арифметичне перетворення значення часу у відстань в см і дюймах відповідно. По завершенню цих операцій здійснюється пересилка отриманої інформації через серійний інтерфейс ***Serial*** за допомогою виклику методу ***print***.

Сценарій 4.1. Вимірювання відстані. Датчик відстані може обертатися за допомогою серводвигуна. Це дозволяє вимірювати дистанцію до різних об'єктів. На стенді на різній відстані від датчика розташовані перегородки. Користувачеві пропонується проводити повороти двигуна з подальшим зняттям показань датчика відстані.

Положення двигуна задається викликом write, параметром якого є кут повороту. В даному випадку значення кутів зберігаються в масиві і по черзі задаються двигуну. Команда на вимірювання відстані

подається датчику за допомогою імпульсу тривалістю 10мкс. Для цього послідовно проводиться запис значень (0 і 1) за допомогою виклику DigitalWrite у відповідний висновок. Після вимірювання відстані проводиться читання показань датчика (час проходу ультразвукового сигналу) за допомогою виклику pulseIn. Виведення даних користувачеві проводиться через послідовний інтерфейс за допомогою функції print.

```
const int trigPin = 2;
const int sigPin = 4;
void setup () {
Serial.begin (9600);
pinMode (trigPin, OUTPUT);
pinMode (sigPin, INPUT);
}
void loop ()
{
long duration, inches, cm;
digitalWrite (trigPin, LOW);
delayMicroseconds (2);
digitalWrite (trigPin, HIGH);
delayMicroseconds (5);
digitalWrite (trigPin, LOW);
duration = pulseIn (sigPin, HIGH);
inches = microsecondsToInches (duration);
cm = microsecondsToCentimeters (duration);
Serial.print (inches);
Serial.print («in, «);
Serial.print (cm);
Serial.print («cm»);
Serial.println ();
delay (100);
}
long microsecondsToInches (long microseconds)
{
return microseconds / 74 / 2;
}
long microsecondsToCentimeters (long microseconds)
{
return microseconds / 29 / 2;
}
```

Сценарій 4.2. Датчик відстані з серводвигуном.

```
#include <Servo.h>
Servo myservo;
```



```
int positions [] = {0, 30, 60, 90, 120, 150, 180};
int pos_num = 7;
int pos = 0;
const int trigPin = 2;
const int echoPin = 4;
void setup ()
{
Serial.begin (9600);
myservo.attach (9);
pinMode (trigPin, OUTPUT);
pinMode (echoPin, INPUT);
}
void loop ()
{
myservo.write (positions [pos]);
pos++;
if (pos == pos_num) pos = 0;
digitalWrite (trigPin, LOW);
delayMicroseconds (2);
digitalWrite (trigPin, HIGH);
delayMicroseconds (10);
digitalWrite (trigPin, LOW);
Serial.print (pulseIn (echoPin, HIGH) / 29 / 2);
Serial.print («cm»);
Serial.println ();
delay (500);
}
```

На рис. 3.17 ви можете побачить команду розробників віддаленої лабораторії RELDES.

На рисунку 3.18 наведено діаграму використання лабораторії.

Щоб отримати доступ до експериментів, користувач має зареєструватися. Користувачі можуть завантажувати власний програмний код в мікроконтролер на платі Arduino через веб-клієнт, або використовувати запропоновані згідно з різними сценаріями проведення експериментів готові програми. Сервер лабораторії компілює отриманий початковий код та завантажує в мікроконтролер на платі Arduino. Якщо програма містить помилки, сервер генерує і повертає результати компіляції. Таким чином, користувач має можливість створювати і редагувати програмний код безпосередньо в веб-інтерфейсі клієнта без необхідності використання інструментів розробки на власному ПК.

Рисунок 3.17. Команда розробників віддаленої лабораторії RELDES

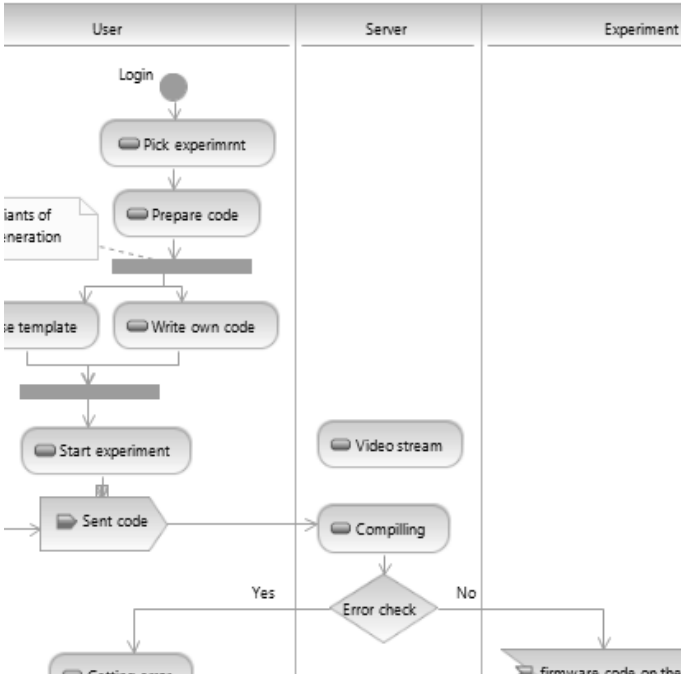


Рисунок 3.18. Діаграма використання лабораторії RELDES

3.2.3 Гібридна лабораторія GOLDI

Експеримент. Плата швидкого прототипування для онлайн-лабораторії [15–18,34]. Зовнішній вигляд плати наведено на рис. 3.19. Режими роботи:

1) Швидке прототипування цифрової системи через веб:

- розробка (текстова, друкована плата, автоматний граф);
- програмування ПЛІС через інфраструктуру лабораторії;
- тестування проекту за допомогою RIA (Rich Internet Application) (так званий насичений Інтернет-додаток) та плати швидкого прототипування через веб.

2) Веб-валідація цифрової системи:

- ідентифікація функцій проекту (чорний ящик);
- пошук несправностей проекту за допомогою маніпуляцій над виводами або завантаження таблиці істинності або тестового вектора;
- програмування ПЛІС не доступно студентам.

Основні компоненти плати швидкого прототипування: MAX V 5M1270Z (ПЛІС Altera); Клавіші управління (8 перемикачів, 8 кнопок, 2 енкодера); світлодіодні індикатори (4 семисегметних дисплея, діодна панель); інші компоненти (кварцовий генератор імпульсів 10МГц, синтезатор частот, п’єзоелектричний генератор імпульсів, зовнішній енкодер, UART (через USB), SUB-D роз’єм на 25 виводів).

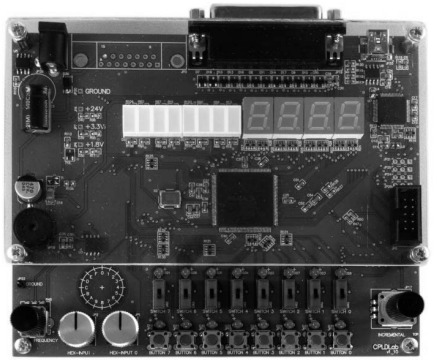


Рисунок 3.19. Плата швидкого прототипування

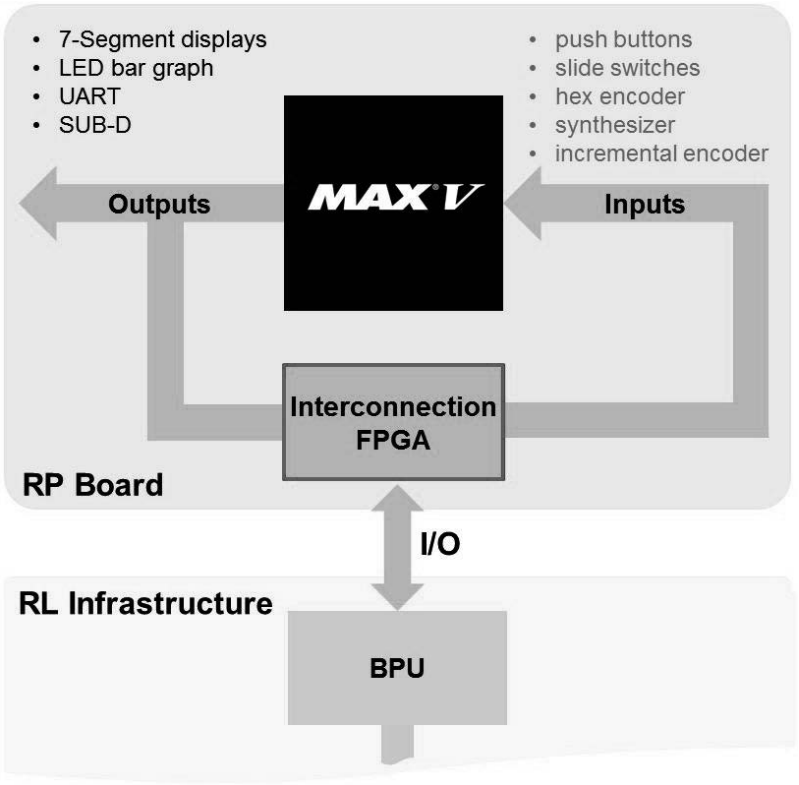


Рисунок 3.20. Схема взаємодії з інфраструктурою лабораторії

Схема взаємодії з інфраструктурою віддаленої лабораторії показана на рис. 3.20:

- взаємодія з віддаленою лабораторією через проміжну ПЛІС;
- всі виводи плати видалені з друкованої плати і замінені прямими з'єднаннями з виводами проміжної ПЛІС:
- значення входів встановлюються згідно даним користувача;
- значення виходів зчитуються безпосередньо;
- проміжна ПЛІС з'єднана з BPU (т.зв. модуль передбачення переходів), що входить в інфраструктуру віддаленої лабораторії.

Користувальницький веб-інтерфейс, реалізований на HTML5 RIA, наведено на рис. 3.21.

Рисунок 3.21. Користувальницький RIA веб-інтерфейс

Основні компоненти інтерфейсу:

- завантаження створеного коду ПЛІС для автоматичного програмування у віддаленій лабораторії;
- візуальна модель плати швидкого прототипування для віртуального управління входами і перегляду виходів плати;
- зображення веб-камери віддаленої лабораторії для безпосереднього перегляду результатів дій користувача.

- стандартні засоби проектування: реалізація контрольних задач (наприклад через Altera IDE. Студенти мають можливість використовувати користувальницькі бібліотеки ядра);
- завантаження зібраного проекту на плату;
- веб-програмування плати через інфраструктуру лабораторії.

Етапи проектування цифрової системи показані на рисунку 3.22. Діаграма показує наступні основні блоки:

- стандартні засоби проектування: реалізація контрольних завдань (наприклад через Altera IDE. Студенти мають можливість використовувати користувальницькі бібліотеки ядра);
- завантаження зібраного проекту на плату;
- веб-програмування плати через інфраструктуру лабораторії.

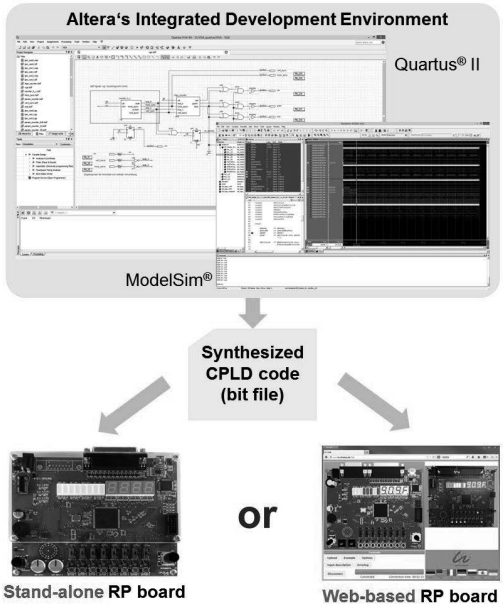


Рисунок 3.22. Етапи проектування цифрової системи

Текстовий метод розробки представлений логічними виразами, таблицями істинності або мовами апаратної архітектури (AHDL, VHDL, Verilog) (рис. 3.23) .

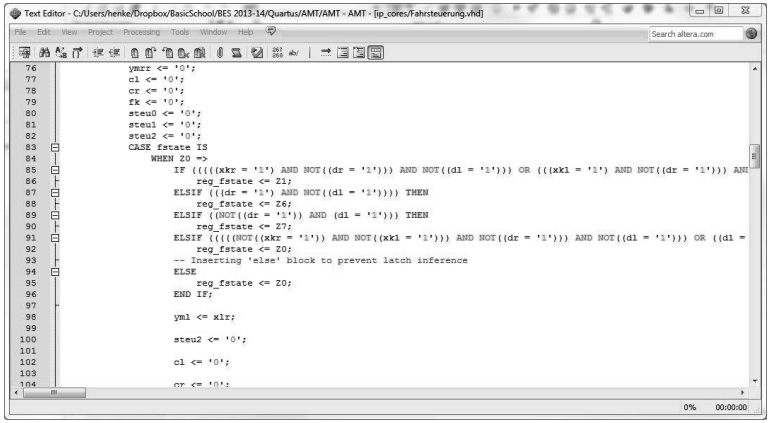


Рисунок 3.23. Текстовий метод розробки

Графічний метод розробки представлений діаграмами блоків та/або діаграмами схем (рис. 3.24) .

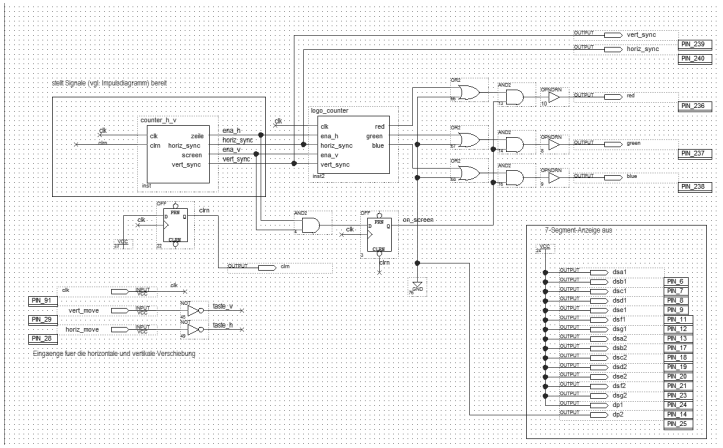


Рисунок 3.24. Графічний метод розробки

Інтегрований FSM редактор представлений автоматним графом (FSM) (рис. 3.25) .

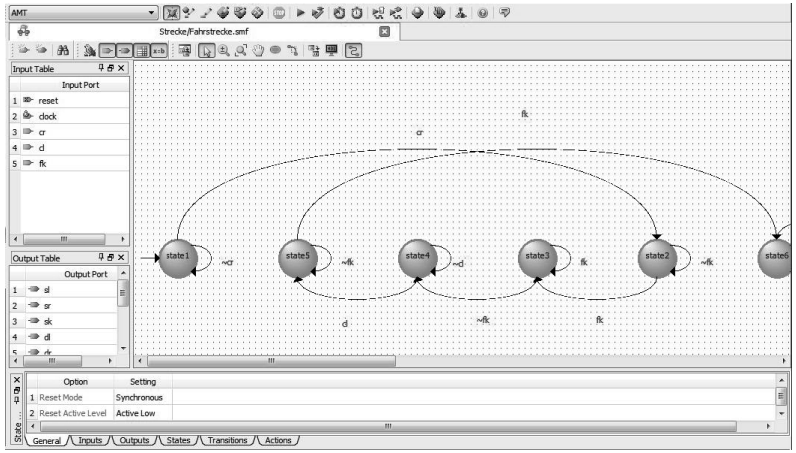


Рисунок 3.25. Інтегрований FSM редактор

Симуляція проекту за допомогою різноманітних інструментів (наприклад, Qsim або ModelSim) зображена на рисунку 3.26.

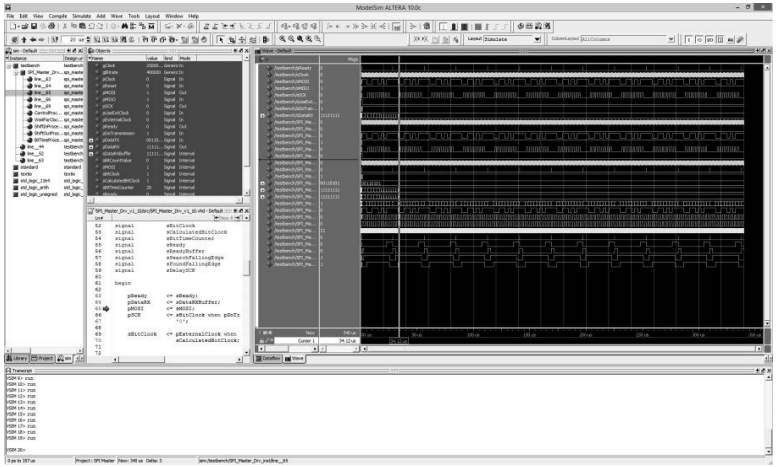


Рисунок 3.26. Симуляція проекту

ЛІТЕРАТУРА ДО ЧАСТИНИ 1

1. Abdurohman, M., Kuspriyanto, Sutikno, S., Sasongko, A., «The New Embedded System Design Methodology For Improving Design Process Performance», International Journal of Computer Science and Information Security, Vol. 8, No. 1, pp. 35–43, 2010.
2. Altium Designer (Protel) – сквозная система проектирования печатных плат [Електронний ресурс] / Eurointech.– Режим доступу: http://www.eurointech.ru/protel
3. Altium Designer [Електронний ресурс] / CSoft.– Режим доступу: http://www.csoft.ru/catalog/soft/altium-designer/altium-designer-10.html
4. Arduino Simulator [Электронный ресурс]: Режим доступа: www/ URL: http://schogini.in/app-websites/arduino-simulator/
5. Arduino [Электронный ресурс]: Режим доступа: www/ URL: https://ru.wikipedia.org/wiki/Arduino
6. Atmel Studio [Електронний ресурс] / Atmel.– Режим доступу: www/ URL: http://www.atmel.com/tools/atmelstudio.aspx

7. A tradeoff between microcontroller, DSP, FPGA and ASIC technologies [Available electronically] / EE Times. Access mode: www/ URL: http://www.eetimes.com/document.asp?doc_id=1275272
8. Creo Parametric. [Електронний ресурс] .– Режим доступу:. http://www.pro-technologies.ru/product/Creo_Parametric/ – Загл. с экр.
9. Difference Between FPGA and Microcontroller [Available electronically] / DifferenceBetween.net.– Access mode: www/ URL: http://www.differencebetween.net/technology/difference-between-fpga-and-microcontroller/
10. Ebert, C. Software: Facts, Figures, and Future/ Christof Ebert, Capers Jones//Computer, vol. 42, no. 4, pp. 42–52, April 2009, doi:10.1109/MC.2009.118
11. Embedded Systems: Technologies and Markets [Електронний ресурс] / BCC Research.– Режим доступу: www/ URL: http://www.bccresearch.com/market-research/information-technology/embedded-systems-technologies-markets-ift016d.html
12. Embedded System Market – Global Industry Analysis, Size, Share, Growth, Trends and Forecast, 2012–2018 [Електронний ресурс] / Transparency Market Research.– Режим доступу: www/ URL: http://www.transparencymarketresearch.com/embedded-system.html
13. Electronics from beginner to pro. http://123d.circuits.io/
14. Freeduino [Электронный ресурс]: Режим доступа: www/ URL: http://www.freeduino.org/freeduino_open_designs.html
15. Henke, K. Fields of Applications for Hybrid Online Labs / Karsten Henke, Steffen Ostendorff, Heinz-Dietrich Wuttke, Tobias Vietzke, Christian Lutze // International Journal of Online Engineering (iJOE), Vol 9 (2013) – Access mode: http://online-journals.org/i-joe/article/view/2542
16. Henke, K. Using Interactive Hybrid Online Labs for Rapid Prototyping of Digital Systems / K. Henke, G. Tabunshchyk, H. D. Wuttke, T. Vietzke, St. Ostendorff // Remote Engineering & Virtual Instrumentation REV2014, Porto, Portugal, February 2014, pp.61–66

17. Henke, K. A Grid Concept for Reliable, Flexible and Robust Remote Engineering Laboratories/ Karsten Henke, Steffen Ostendorff, Heinz- Dietrich Wuttke, Stefan Vogel // Remote Engineering & Virtual Instrumentation REV2012, Bilbao, Spain, July 2012
18. Henke K. Web-based Rapid Prototyping of Digital Systems/ Karsten Henke, Silvia Krug // International Conference on Interactive Computer-Aided Blended Learning, Antigua, Guatemala, November 2011, pp. 22–27
19. Leffingwell D. Managing Software Requirements: A Use Case Approach / Dean Leffingwell, Don Wirding – Addison-Wesley, 2003.– 402 page.– ISBN 0–321–12247-X.
20. Mbed [Электронный ресурс]: Режим доступа: www/ URL: http://en.wikipedia.org/wiki/Mbed
21. Parkhomenko A. Complex requirements analysis for the high-level design of Embedded Systems / A. Parkhomenko, O. Gladkova // Вісник НУ «Львівська політехніка» – Комп'ютерні системи проектування. Теорія і практика.– Львів: Львівська політехніка, № 808 (2015) .– С.3–10
22. Parkhomenko A. V. Virtual Tools and Collaborative Working Environment in Embedded System Design / A. V. Parkhomenko, O. N. Gladkova // Proceedings of XI International Conference on Remote Engineering and Virtual Instrumentation (REV2014) (26–28 February, 2014, Porto, Portugal) Porto: Polytechnic, 2014. P. 91–93. Simuino [Электронный ресурс]: Режим доступа: www/ URL: https://code.google.com/p/simuino/
23. Parkhomenko A. Development and Application of Remote Laboratory for Embedded Systems Design / A. Parkhomenko, O. Gladkova, E. Ivanov, A. Sokolyanskii, S. Kurson// JOE – Volume 11, Issue 3, 2015.– PP.27–31
24. Parkhomenko, A. Internet-based Technologies for Design of Embedded Systems / A. Parkhomenko, O. Gladkova, E. Ivanov, A. Sokolyanskii, S. Kurson// Proceedings of XIII International Conference «The Experience of Designing and application of CAD Systems in Microelectronics (CADSM 2015)», (24–27 February, 2015, Lviv-Polyana, Ukraine) Lviv: Lviv Polytechnic, 2015.P. 167–171.

25. Parkhomenko, A. Analysis and application of existent approaches in microcontroller system designing / Anzhelika Parkhomenko, Olga Gladkova // Proceedings of IX-th International Conference «PERSPECTIVE TECHNOLOGIES AND METHODS IN MEMS DESIGN (MEMSTECH 2013)», Lviv: Lviv Polytechnic, 2013. P.268–270.
26. Proteus VSM. Пошаговая отладка [Електронний ресурс] – Режим доступу: http://we.easyelectronics.ru/CADSoft/proteus-vsm-poshagovaya-otladka.html – Загл. с экр.
27. RELDES [Електронний ресурс] .– Режим доступу: http://youtu.be/u2anq – UYFg
28. Simulator for Arduino [Электронный ресурс]: Режим доступа: www/ URL: http://www.virtronics.com.au/Simulator-for-Arduino.html
29. Simuino [Электронный ресурс]: Режим доступа: www/ URL: https://code.google.com/p/simuino/
30. Shih, Randy H. Parametric Modeling with Creo Parametric 1.0 / Randy H. Shih.– SDC Publisher: Stepher Schroff, 2011.– 432 pp.
31. Teich, J., «Hardware/Software Codesign: The Past, the Present, and Predicting the Future», Proceedings of the IEEE, Vol.100, pp. 1411–1429, Germany, May 13, 2012.
32. Wiegers K. E. Software Requirements. / Karl E. Wiegers.– 2nd Edition.– Microsoft Press, 2003.– 544 p. ISBN:978–0–7356–1879–4.
33. Wolf H., Wayne, «Hardware-Software Co-Design of Embedded Systems», Proceedings of the IEEE, Vol.82, NO. 7, pp. 967–989, Germany, July 1994.
34. Wuttke H- D. Remote and Virtual Laboratories in Problem-Based Learning Scenarios / Heinz- Dietrich Wuttke, Raimund Ubar, Karsten Henke//, IEEE International Symposium on Multimedia, Taichung, Taiwan, December 2010, pp.377–382
35. Zielczynski P. Requirement Management Using IBM Rational RequisitePro. / Peter Zielczynski – IBM Press, 2007.– 360 pages.– ISBN 0–321–38300–1.
36. Белов, А. В. Создаем устройства на микроконтроллерах / А. В. Белов.– СПб.: Наука и Техника, 2007.– 304с.

37. Блог розробника Emulino [Электронный ресурс]: Режим доступа: www/ URL: http://hewgill.com/journal/entries/507-emulino-arduino-cpu-emulator
38. Бурмистров А. В., Филин И. В. Операционные системы реального времени для микроконтроллерных систем // Успехи современного естествознания.– 2012.– № 6.– стр. 87–88
39. Васильев, А. Е. Микроконтроллеры. Разработка встраиваемых приложений / А. Е. Васильев.– СПб.: БХВ-Петербург, 2008.– 304с.
40. Встраиваемые системы [Электронный ресурс] / Родник.– Режим доступа: www/ URL: http://www.rodnik.ru/product/spa/embedded-solutions/
41. Высокопроизводительные 8-разрядные RISC микроконтроллеры семейства AVR [Електронний ресурс] / Рынок микроэлектроники.– Режим доступу: www/ URL: http://www.gaw.ru/html.cgi/txt/ic/Atmel/micros/ avr/about.htm
42. ГОСТ Р ИСО/МЭС 12207–2010 Information technology. System and software engineering. Software life cycle processes (ISO/IEC 12207:2008) .– Введ. 01.03.2012.– Москва: Стандартинформ, 2011–104с.
43. ГОСТ 34.602–89. Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы.– Введ. 01.01.1990.– М.: ИПК издательство стандартов, 2004.– 11 с.
44. ГОСТ Р 41904–2002 Программное обеспечение встроенных систем. Общие требования к разработке и документированию.– Введ. 24.07.2002.– ИПК Издательство стандартов, 2002.– 62с
45. Два подхода к реализации ПО для embedded [Електронний ресурс] / Программинг микроконтроллеров.– Режим доступу:: www/ URL: http://habrahabr.ru/post/148805/
46. Дестрибьютор Freeduino в России [Электронный ресурс]: Режим доступа: www/ URL: http://freeduino.ru/arduino/products.html
47. ДСТУ ISO/IEC 14288:2004. Інформаційні технології. Процеси життєвого циклу системи (ISO/IEC 14288:2002, IDT) .– Введ. 01.07.2007.– К.: Держстандарт України, 2004.– 48 с.

48. Кунву, Ли. Основы САПР (CAD/CAM/CAE) / Кунву Ли.– СПб.: Питер, 2004.– 560 с.: ил.
49. Ламбер Е. 8-разрядные микроконтроллеры AVR корпорации Atmel: новинки тенденции развития / Елена Ламбер // Компоненты и технологии, № 6.– 2009.– С. 62–65
50. Операционные системы реального времени [Електронний ресурс] / CIT Forum.– Режим доступу: www/ URL: http://citforum.ru/operating_systems/rtos/1.shtml
51. Основы программной инженерии [Электронный ресурс] / SWEBOK.– Режим доступа: www/ URL: http://swebok.sorlik.ru
52. Официальный сайт Raspberry Pi [Электронный ресурс]: Режим доступа: www/ URL: www.raspberrypi.org
53. Официальный сайт Altera Cyclone [Электронный ресурс]: Режим доступа: www/ URL: http://www.altera.com/devices/fpga/cyclone/cyc-index.jsp
54. Пархоменко А. Інтерактивна віддалена лабораторія дослідження апаратно-програмних платформ / А. Пархоменко, О. Гладкова // Тези доповіді Міжнародної науково-практичної конференції «Інтернет-Освіта-Наука-2014» «ІОН-2014» (14–17 жовтня 2014, Вінниця), 2014. С. 248–250
55. Пархоменко, А. В. Автоматизоване проектування електронних засобів в середовищах Creo та Altium Designer / А.В.Пархоменко, А. В. Притула, В. М. Крищук.– Запоріжжя: Дике поле, 2013.– 240 с.
56. Платунов А. Е. Высокоуровневое проектирование встраиваемых систем. Часть 1: учеб. пособие / А. Е. Платунов, Н. П. Постников – СПб.: НИУ ИТМО, 2011.– 121 с.
57. Платунов А. Встраиваемые системы управления. / Платунов А. // CONTROL ENGINEERING РОССИЯ. 2013.– № 1 (43) .– С. 16–24
58. Постников Е. Б. Обзор мирового опыта создания и эксплуатации лабораторий удаленного доступа. [Электронный ресурс]: Режим доступа: www/ URL: http://www.efmsb.ru/download/Mirovoy_opit_sozdaniya_i_ekspluatacii_laboratoriy_udalennogo_dostupa.pdf

59. Проект тепловизора [Электронный ресурс]: Режим доступа: www/ URL: https://www.tindie.com/products/PureEngineering/flir-lepton-thermal-camera-breakout-2/

60. Разработка микропроцессорной системы на основе микроконтроллеров [Электронный ресурс] / Интернет университет.– Режим доступа: www/ URL: http://www.intuit.ru/department/hardware/mpbasics/11/

61. Русскоязычный ресурс Raspberry Pi [Электронный ресурс]: Режим доступа: www/ URL: http://raspberrypi.su/description.html

62. Сабунин, А. Е. Altium Designer. Новые решения в проектировании. /А.Е Сабунин– Солон Пресс, 2009.– 432с.

63. Системы для умного дома с применением Rasperry Pi [Электронный ресурс]: Режим доступа: www/ URL: http://habrahabr.ru/post/237597/

64. Суходольский, В. Ю. Altium Designer. Проектирование функциональных узлов РЭС на печатных платах / В. Ю Суходольский.– СПб.: БХВ-Петербург, 2009.– 480 с.

65. Татарчевский В. Применение Switch-технологии при разработке прикладного программного обеспечения для микроконтроллеров. Часть 1 // Компоненты и технологии.– 2006.– № 11

66. Трухин А. В. «Об использовании виртуальных лабораторий в образовании» // Открытое и дистанционное образование.– 2002.– № 4 (8) .

ЧАСТИНА 2. СИСТЕМИ КЕРУВАННЯ ЕЛЕКТРИЧНИМИ МАШИНАМИ ТА АПАРАТАМИ

М. О. Поляков,Т.Ю. Ларіонова

ВСТУП ДО ЧАСТИНИ 2

Сучасний етап розвитку науки і техніки характеризується системною інтеграцією окремих апаратів, машин, установок на основі нових інформаційних технологій. Системи керування з контролерами – один з напрямів таких технологій. По мірі інтеграції України в світову спільноту, промислові контролери все ширше використовуються в проектних рішеннях розробників систем керування, у тому числі в енергетиці [1], експлуатуються на машинобудівних, металургійних, хімічних і інших підприємствах.

Для проектування і експлуатації контролерних систем керування складними електротехнічними установками і комплексами необхідні фахівці, які не лише знають об’єкт керування, але і елементну базу контролерних систем керування, мови і засоби програмування, методи формалізації проектних завдань. Відома література по програмуванню промислових контролерів або застаріла [2] або англомовна [3–5] і тому малодоступна фахівцям в Україні або не містить методів формалізації проектних рішень [6,7]. Таким чином розділи 4–6 можуть бути корисними спеціалістам з електромеханіки, які прагнуть навчитися проектувати інтелектуальні електромеханічні системи.

У четвертому розділі розглядається архітектура систем керування, різновиди, функціональна організація, апаратні засоби і властивості промислових контролерів.

П’ятий розділ містить опис мов програмування за стандартом МЕК 61131–3, в тому числі мови драбинних діаграм, методів формалізації завдань керування, включаючи моделі скінчених автоматів для опису поведінки системи керування, а також інструменти для роботи з віддаленою лабораторією GOLDI у режимі проектування систем керування за допомогою формалізму скінчених автоматів.

У шостому розділі описані методи програмної реалізації скінчених автоматів та інших типових завдань керування мовою драбин-

них діаграм, програмне забезпечення людино-машинного інтерфейсу, методи поведінкового синтезу та скриптингу завдань візуалізації. Наприкінці розділу наведені питання та вправи для самоперевірки, короткий опис лабораторних та практичних робіт за курсом.

Розглянуті в розділах 4–6 апаратні, програмні засоби систем промислової автоматизації та методи формалізації завдань керування перспективно використовувати при створенні систем керування фізичними моделями віддалених лабораторій для дистанційного інженерного навчання [18].

Матеріал розділів 4–6 базується на 16-ти річному досвіді викладання курсів із промислової автоматизації студентам Запорізького національного технічного університету [8–10] .

Modern stage of scientific and technologic advancement is characterized by system integration of separate devices, machines and facilities based on the new informational technologies. Control systems with controllers – one of the tendencies of such technologies. Along with Ukraine's integration into the global community, industrial controllers are being increasingly used in the design solutions of control systems, including energy sector [1]; industrial controllers are applied at the machine-building, metallurgical, chemical and other enterprises.

For the design and operation of controller control systems of complex electrotechnical machineries and complexes professionals who know not only the control object, but the basic elements of the controller control systems, languages and programming tools, methods of design problems formalization are required. Currently in use literature on programming of industrial controllers are outdated [2] or in English language [3–5] and therefore are not readily available for specialists in Ukraine or don't contain formalize methods of design solutions [6, 7]. Hence, chapters 4–6 would be useful for specialists in electromechanics, who want to learn how to design intelligent electromechanical systems.

In the fourth chapter the architecture of control systems, varieties, functional organization, hardware and properties of industrial controllers are examined.

The fifth chapter contains the description of the programming languages according to standard IEC 61131–3, including Ladder diagram

language, methods of control tasks formalizing, as well models of finite state machines, which specify the behavior of control system and also tools for the remote laboratory GOLDI in the mode of control system design using formalism of finite state machines.

In the sixth chapter software implementation methods of finite state machines and other common control tasks in Ladder diagram language, software of man-machine interface, methods of behavioral synthesis and scripting tasks visualization are described. At the end of the chapter questions and exercises for self-test, a brief description of laboratory and practical work according to the course are presented.

Considered in chapters 4–6 hardware, software of industrial automation systems and methods of formalization of control tasks are promising to be used at developing of control systems of physical models in remote laboratories for distance engineering education [18].

Material of chapters 4–6 is based on 16 years' experience in teaching Industrial Automation at Zaporizhzhya National Technical University [8–10] .

4 ВВЕДЕННЯ В КОНТРОЛЕРНІ СИСТЕМИ КЕРУВАННЯ

4.1 Різновиди і властивості контролерів

Контролер (Controller) – це пристрій, такий як програмований контролер або панель з реле, що контролює елементи устаткування або процесу.

Програмований контролер (Programmable controller) – напівпровідникова керуюча система, яка містить програмовану користувачем пам'ять для зберігання інструкцій без тих, що виконують спеціальні функції, такі як керування введенням/виводом, логічні, арифметичні, маніпуляцій з файлами даних, функції мережних комунікацій. Контролер містить центральний процесор, інтерфейс введення/виводу і пам'ять. Контролер спроектований як промислова керуюча система.

Іншими словами програмований контролер це обчислювальний пристрій, орієнтований на керування устаткуванням або процесом в умовах промислового середовища.

Щоб підкреслити, що контролер задовольняє вимогам промисловості по характеристиках надійності, діапазонів кліматичних, механічних, електромагнітних і інших чинників умов експлуатації, застосовують термін «промисловий контролер».

Абревіатура «PLC» (Programmable Logic Controller), введена компанією Allen Bradley (США), відображає одну з сфер застосування програмованих контролерів – логічне керування. Разом з тим сучасні програмовані контролери здатні вирішувати ширші завдання, наприклад, реалізувати системи автоматичного регулювання, штучного інтелекту та інші. А назва PLC зберігається, щоб уникнути плутанини термінів «програмовані контролери» і «персональні комп'ютери» – обидва англійською мовою «PC».

У еволюції контролера, як засобу програмного керування об'єктом можна виділити наступні етапи:

- жорстка логіка на елементній базі реле, дискретних транзисторів і логічних інтегральних схем;
- мікропрограмні автомати;
- мікропроцесорні і мікроконтролерні автомати;
- мультипроцесорні контролери.

Після винаходу мікропроцесора в 1971 році його сталі активно застосовувати в промислових контролерах. Компанія Allen Bradley (США) в 1977 році випустила перший контролер на основі мікропроцесора Intel 8080. Обчислювальний пристрій, зокрема контролер, на базі такого мікропроцесора є, як мінімум одноплатним. Він містить друковану плату, на якій встановлено декілька інтегральних мікросхем пам'яті програм і даних, зовнішніх портів, таймерів і ін. У міру розвитку мікроелектроніки ці вузли були інтегровані в один кристал і нові мікросхеми отримали назву однокристальних ЕОМ або мікроконтролерів. Широко відома родина мікроконтролерів Intel 8051, 80X96. Хоча мікроконтролери дозволяють створити однокристальний пристрій керування об'єктом, розробники розширюють його можливості шляхом підключення додаткових БІС. І, таким чином, процесор PLC, виконаний на базі мікроконтролера, як правило, є одноплатним. А мікроконтролерні системи керування, як альтернатива контролерним, характеризується вищою швидкодією, складністю програмування (як правило, мовою асемблеру) і проектування, а також на два порядки нижчими цінами на компоненти.

Мультипроцесорними називатимемо контролери, що містять більше одного процесора в локальному шасі контролера. Ці процесори виконують паралельні завдання по управлінню об'єктом і при цьому враховують дані, отримані один від одного.

У розподілених контролерах також є більше одного процесора, але вони розміщені в різних, географічно віддалених контролерах.

Контролери випускаються в наступних виконаннях, що характеризують можливість нарощування їх функцій:

- фіксовані – функціонально і конструктивно закінчені пристрої з фіксованими характеристиками (типом процесора, номенклатурою каналів введення/виводу і ін.), які випускаються серіями і проектант може вибрати готовий контролер, оптимальний за ціною і функціональними характеристиками;
- модульні контролери – пристрої, які комплектуються і збираються споживачем за лічені хвилини із стандартних модулів, виходячи з конкретних завдань керування;
- фіксовані контролери з можливістю підключення додаткових модулів введення/виводу.

На рис. 4.1 показаний модульний контролер SLC500 компанії Rockwell Automation (США) [11] .

Рисунок 4.1. Модульний контролер SLC500

Модулі випускаються сімействами і відрізняються конструкцією і системними параметрами, які приведені в табл. 4.1.

Якщо необхідна кількість модулів у системі перевищує число слотів в шасі, то системні магістралі декількох (як правило, до трьох) шасі об'єднують між собою коротким (десятки сантиметрів) кабелем. Така система називається локальним шасі, а вхідні в неї шасі – реками. Вона використовується в зосереджених системах керування.

Для побудови розподілених однопроцесорних систем керування до складу контролера включається один або більше модулів сканера. Сканер – це мережний модуль, який функціонально є процесором введення-виводу. Сканер забезпечує обмін інформацією центрального процесора з модулями введення-виводу, встановленими у віддалених шасі, з використанням промислової мережі. Мета застосування сканера – звільнити центральний процесор від рутинних завдань по передачі даних через мережу.

Структура розподіленого однопроцесорного контролера приведена на рис. 4.2.

Таблиця 4.1. Системні параметри модулів контролерів

Найменування модуля	**Системні параметри**
Модуль живлення	Напруга мережі, струм і напруга навантаження для модулів і живлення датчиків.
Шасі	Число слотів (посадочних місць) для установки модулів
Процесор	Швидкодія, розрядність даних, об'єм пам'яті користувача, системні канали, система інструкцій, струм споживання
Модуль введення, виводу	Число каналів, параметри каналу, спосіб конфігурації, струм споживання
Мережний модуль	Тип мережі, число каналів, струм споживання

Особливістю віддалених шасі є відсутність в них центрального процесора. Адаптер у віддаленому шасі – це теж мікропроцесорний

пристрій, але він не виконує програму користувача, а лише підтримує протокол мережі і керує читанням (записом) інформації із/в модулі введення/виводу і передачею її через мережу.

Однією з цілей застосування віддалених шасі є використання в одному контролері конструктивно різних модулів введення/виводу. Наприклад, одне з віддалених шасі повинне бути розміщене у вибухонебезпечній зоні об'єкту керування, побудоване на вибухобезпечних елементах. Решта шасі може бути реалізована в звичайному (не вибухобезпечному), як правило, дешевшого конструктивного виконання.

Більшість сучасних PLC є мережними. По мережі PLC підключається до станцій програмування, приладів операторського інтерфейсу, інших елементів системи промислової автоматизації (контролерів, датчиків, інформаційно-вимірювальних приладів, приводів і т.д.). При цьому мережеутворююче устаткування розміщується як в центральному процесорі, так і в спеціалізованих модулях встановлених в шасі. Якщо в контролері більше одного канала, то він може служити мостом з однієї мережі до іншої. До системних параметрів мереж можна віднести кількість вузлів в мережі, довжину сегментів мережі, швидкість передачі інформації по мережі, час доставки інформації, топологію мережі, середовище передачі. У промислових мережах, що застосовуються в даний час, кількість вузлів може досягати до декількох десятків, сумарна довжина сегментів мережі – до декількох кілометрів, швидкості передачі інформації – десятки і сотні кілобіт в секунду. Якщо контролери, що входять в систему керування, віддалені один від одного на значнішу відстань, то зв'язок між ними здійснюється за допомогою модемів з використанням телефонних ліній зв'язку, радіоканалів, каналів супутникового зв'язку і зв'язку по лініях електропередач. Існують мережі з негарантованим (наприклад Ethernet) і гарантованим (наприклад Controlnet) часом доставки повідомлень.

Застосування контролерів стало альтернативою прямому управлінню об'єктами за допомогою персонального комп'ютера. Прихильники прямого керування за допомогою ПК звертають увагу на можливість використання для керування пам'яті істотно великих розмірів, включаючи дискову, великі напрацювання по програмному забезпеченню, порівняльну дешевизну офісних комп'ютерів. Їх супротивники відзначають недостатню надійність, надмірну склад-

ність, як апаратного, так і програмного забезпечення ПК, невідповідність ПК вимогам систем реального часу.

Рисунок 4.2. Структура розподіленого однопроцесорного контролера

В результаті подолання відмічених недоліків з'явився деякий гібрид двох підходів до керування, який отримав назву софтвер-контролер. У цьому контролері функції процесора промислового контролера емулюються програмою, яка виконується на процесорі ПК в середовищі ОС реального часу, наприклад Windows CE. Цей ПК оснащений мережними картами на відповідні промислові мережі. По цих мережах відбувається обмін інформацією з віддаленими шасі, в яких розташовані модулі введення/виводу.

В деяких випадках персональний комп'ютер, що реалізовує софтвер-контролер випускається в спеціальному конструктиві без вентиляції, рухомих частин, з твердотільним Windows і можливістю безпосереднього підключення модулів введення/виводу. До таких приладів можна віднести контролер EZ1131 на базі комп'ютера «TECLA» (компанія Online Development Corp, США, сайт: www. oldi. com), зображений на рис. 4.3.

У обґрунтованих випадках керування відповідними об'єктами застосовується резервування каналів передачі інформації, пристроїв живлення, процесорів, модулів введення/виводу контролерів.

4.2 Апаратні засоби промислового контролера

Як вже наголошувалося вище, промисловий контролер – це обчислювальний пристрій із специфічними периферійними пристроями (модулями або блоками введення і виводу), який настроюється на рішення конкретної задачі користувача шляхом програмування. Програмно-апаратна реалізація контролера містить рішення типових завдань проектування систем керування, таких як зв'язок з об'єктом керування, контроль працездатності самого керуючого пристрою, аварійне відключення об'єкту керування, відновлення стану процесу керування після збоїв живлення контролера і ін.

Рисунок 4.3. Контролер EZ1131 на базі комп'ютера «Тесла»

Завдання зв'язку з об'єктом керування вирішуються вибором і конфігурацією модулів введення-виводу. Класифікація модулів введення-виводу приведена на рис. 4.4.

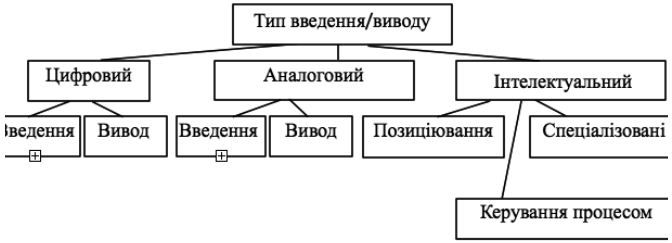


Рисунок 4.4. Типи модулів введення-виведення

Цифрові (дискретні) модулі – це модулі, в яких кожному вхідному (вихідному) каналу відповідає тег даних булевого типу в пам'яті процесора. У аналогових модулях кожному вхідному (вихідному) каналу відповідає тип даних цілочисельного типу. Інтелектуальні модулі забезпечують деяку внутрішню обробку вхідних значень або керування вихідними значеннями програмою користувача виконуваною в процесорі контролера.

За допомогою цифрових модулів введення до контролера підключають: селектори, датчики, обмежувачі, кінцеві вимикачі, кнопки, перемикачі, контакти пускачів двигунів, фотодатчики і т.п.

Цифрові модулі управляють включенням/відключенням аварійних пристроїв, реле керування, вентиляторів, ламп, сирен, клапанів, пускачів двигуна, соленоїдів і т.п.

Аналогові модулі введення призначені для вимірювання значень електричних струмів і напруги, а також для зв'язку з датчиками, що перетворюють значення фізичної величини (температури, швидкості, тиску, вологості і т.п.) в електричний сигнал. Ці модулі виконують функцію аналого-цифрового перетворення вхідних сигналів.

Аналогові модулі виводу керують аналоговими клапанами, приводами і т.п. Вони здійснюють цифро-аналогове перетворення значень даних контролера в електричний сигнал струму або напруги на виході модуля.

Інтелектуальні модулі керують такими пристроями як кодери, витратоміри, визначниками ваги, ASCII-пристрої, дисплеї, аналізатори параметрів електроенергії і т.п. Вони використовуються в специфічних додатках типу керування позиціонуванням, ПІД-регулювання, зв'язку із зовнішніми пристроями.

Структура основних інформаційних потоків між модулем і процесором приведена на рис. 4.5. Напрям передачі даних між модулем і процесором залежить від типу модуля: для вхідних – від модуля до процесора; для вихідних – від процесора до модуля.

Дані представляють інформацію, яка вводиться в процесор через вхідний модуль або виводиться з процесора через вихідний модуль. Конфігураційне слово – це набір полів або тегів, які задають режим роботи модуля. Слово стану – це набір полів або тегів, які відображають стан модуля.

Електричні схеми цифрових і аналогових вхідних і вихідних модулів містять вузол інтерфейсу з процесором контролера і схеми каналів.

Як видно з рис. 4.6, схема дискретного введення забезпечує: індикацію логічного рівня на вході, варисторний захист від перенапружень, омічне навантаження джерела вхідного сигналу змінного струму, завдання порогів напруги логічних рівнів «0» і «1», низькочастотну фільтрацію вхідного сигналу, оптоелектронну розв'язку основної частини контролера від джерела вхідного сигналу, формування фронтів сигналу (тригер Шмідта), періодичне (з періодом тактуючих імпульсів) збереження значення поточного логічного рівня в регістрі пам'яті.

Рисунок 4.5. Інформаційні потоки між модулем введення-виводу і процесором

Структурна схема типового дискретного входу наведена на рис. 4.6.

Деякі виробники забезпечують можливість підключення/виключення вхідних і вихідних модулів на працюючих контролерах RIUP (Remove In Up Power), контроль цілісності ланцюга датчика, відсутність короткого замикання, діагностику схеми самого каналу, супроводження міткою системного часу моменту зміни логічного рівня каналу і ін.

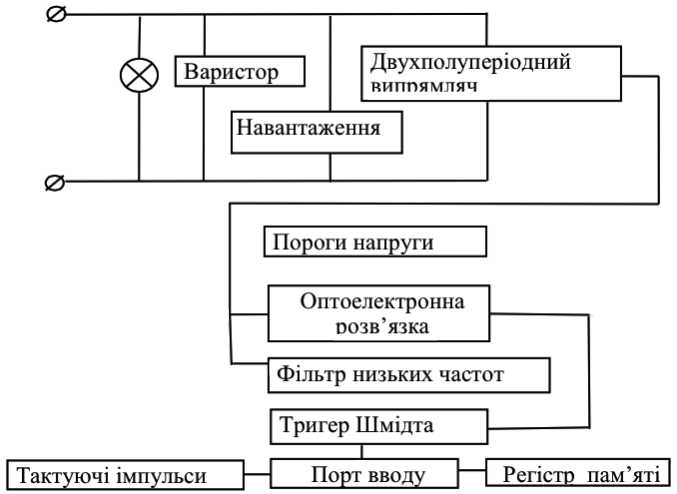


Рисунок 4.6. Структурна схема типового дискретного входу

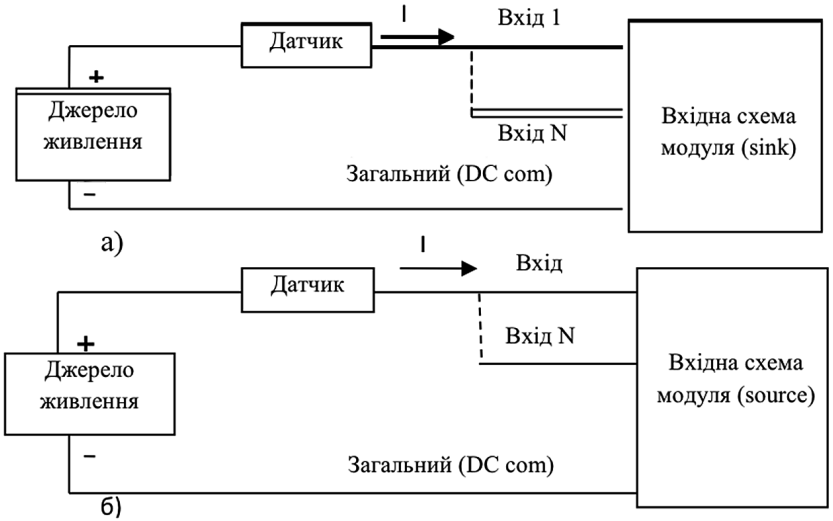


Рисунок 4.7. Схеми підключення датчиків до входів модулів:
а – sink типу; б – source типу

Для дискретних входів постійного струму істотним є тип (схема підключення до джерела живлення) датчика і наявність/відсутність міжканальної ізоляції. Залежно від типу датчика, схеми входів підрозділяються на одержуючи (sink) і такі, що видають (source) струм. Схеми підключення датчиків до входів модулів приведені на рис. 4.7.

Дискретні вихідні модулі повинні забезпечити видачу певної потужності в навантаження R. Комутація цієї потужності виконується транзистором, реле або симистором. Відповідні схеми виходів і їх підключення до навантаження приведені на рис. 4.8. Вони відрізняються структурою вихідного каскаду. Вихідний каскад схеми рис. 4.8 а побудований на польовому (FET) транзисторі VT із захисним діодом VD, запобіжником F і індикатором L.

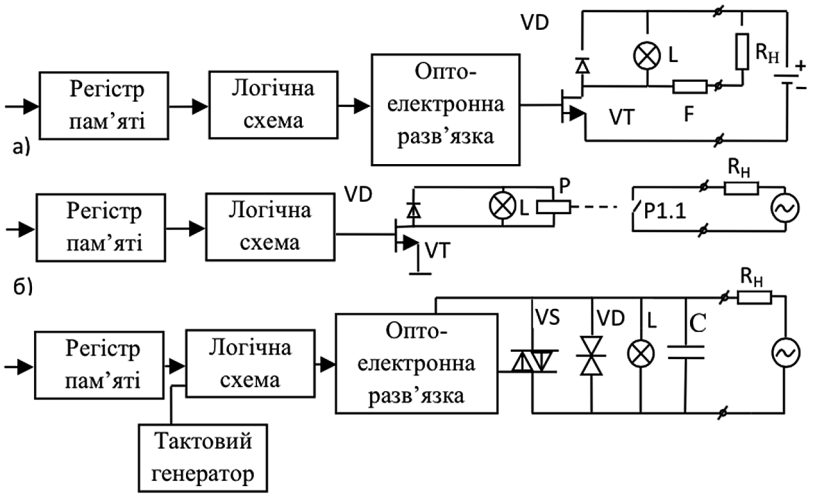


Рисунок. 4.8. Структурна схема дискретного виходу:
а – транзисторного; б – релейного; в – симисторного

Вихідний каскад схеми рис. 4.8 б також побудований на польовому транзисторі VT, який керує включенням реле P. Контакти P1.1 цього реле можуть підключатися до ланцюга змінного або постійного струму.

Вихідний каскад схеми рис. 4.8в побудований на симисторі VS. Він містить індикатор L і елементи захисту – варистор VD і конденсатор C.

Структурна схема каналів вхідного аналогового модуля приведена на рис. 4.9. Схема містить N (як правило, N = 4, 8, 16) підсилювачів каналів, що підключаються через мультиплексор до АЦП послідовно в часі. Мультиплексування каналів дозволяє скоротити число АЦП, але збільшує час оновлення інформації на виході АЦП.

Конфігурація каналу вихідного аналогового модуля включає завдання дозволу/заборони каналу, допустимого діапазону вхідних сигналів, формату представлення даних в процесорі, параметрів частотного фільтру каналу, значення даних каналу при обриві датчика та інші.

Як правило, конфігурування каналу проводиться шляхом занесення в схему каналу конфігураційного слова з процесора контролера. Рідше застосовується конфігурування каналу за допомогою перемикачів, встановлених на платі модуля. Наприклад, перемикач «струм – напруга».

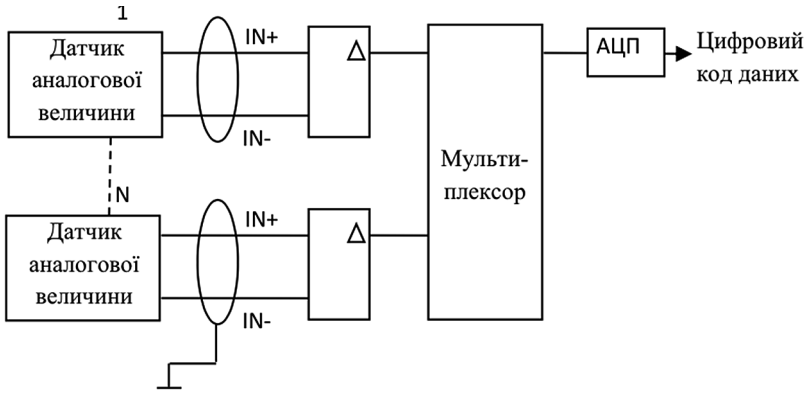


Рисунок 4.9. Структурна схема каналів вхідного аналогового модуля

Вибір значення «заборона» в полі «дозвіл/заборона каналу» дозволяє відключити канал і тим самим зменшити час оновлення даних на виході АЦП.

Діапазон допустимих значень вхідних сигналів повинен відповідати значенням реальних сигналів. Це дозволяє виконувати аналогово-цифрові перетворення з максимально можливою точністю. Для АЦП, що з'єднані з датчиками температури, конфігурується тип термопари або термоопору.

Найбільш поширені формати представлення даних в процесорі, описані нижче:

Інженерні одиниці – ціле число рівне кількості мілівольт (мікроампер) в перетворюваній напрузі (струмі). Наприклад, якщо перетворювана напруга рівна 2,53 вольта, то це відповідає 2530 інженерним одиницям.

Пропорційний рахунок – граничним значенням діапазону допустимих значень вхідних сигналів відповідають максимально і мінімально можливі коди на виході АЦП. Наприклад, допустимий діапазон вхідних значень [-10,+10] В, діапазон десяткових еквівалентів цифрових кодів на виході 14-ти розрядного АЦП [-32768,+32768], на вході АЦП поступив сигнал з напругою $U_{ВХ}$ = 1В. Тоді десятковий еквівалент коду цього сигналу визначимо по формулі:

$$код = 1\text{В} \cdot \frac{[32767-(-32768)]}{[10-(-10)]\text{В}} \approx 3268 \qquad (4.1)$$

Для частотного фільтру каналу конфігурується частота зрізу. Чим менше ця частота, тим ефективніше фільтрація високочастотних перешкод на вході каналу, але одночасно збільшується час реакції АЦП на стрибкоподібну зміну вхідного сигналу. На рис. 4.10 показана динаміка зміни коду на виході АЦП для двох варіантів фільтру. Крива 1 відповідає фільтру з більшою частотою зрізу, ніж у фільтру, відповідного кривій 2.

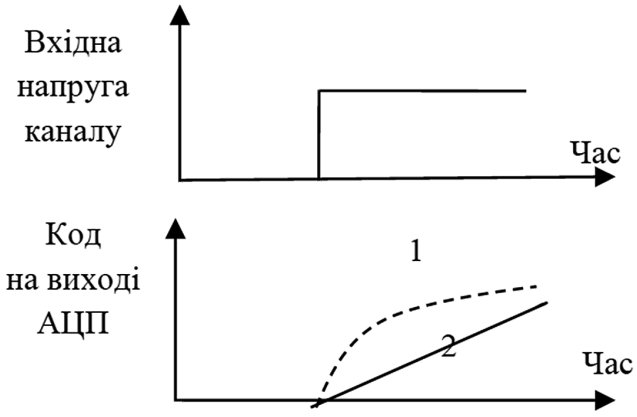


Рисунок 4.10. Реакція на виході АЦП

Щоб за допомогою регістру даних каналу відобразити факт обриву датчика від входу каналу, в цей регістр треба занести незвичайне значення, що не зустрічається при нормальному використанні. Наприклад, 0˚С, при вимірюванні температури металу в печі. Це значення вибирається при конфігуруванні модуля.

Слово стану вхідного модуля містить теги, підтверджуючі прийом модулем конфігураційного слова, а також теги несправностей модуля і його окремих каналів.

Вихідні аналогові модулі складаються з вузла інтерфейсу з шиною контролера, вузла керування і вузлів каналів. У вузол каналу входить регістр пам'яті, логічна схема і оптична розв'язка від основної схеми контролера, ЦАП, буферний підсилювач і перетворювач напруга – струм.

Вихідні модулі, також як і вхідні, підтримують різні формати представлення даних в процесорі і мають ряд додаткових можливостей. Наприклад, модулі родини Controllogix [12], дозволяють конфігурувати граничну швидкість зміни сигналу, виконують фіксацію (обмеження) вихідного сигналу, якщо цифровий код на вході модуля виходить за задані межі, виконують захоплення (завдання) значення вихідного сигналу для ініціалізації модуля при несправності або під час програмування, виявляють обрив проводу, що з'єднує вихід модуля з навантаженням, формують сигнали тривоги (біти стану) .

4.3 Функціональна організація контролера

В процесі рішення задачі користувача по керуванню об'єктом, в контролері використовується комплекс програмних і апаратних засобів, які доцільно розбити на функціональні рівні, як показано в табл. 4.2.

Таблиця 4.2. Рівні керування контролера

Рівень	Елементи	Інформаційні масиви	Місцеположення
Керування адаптацією	Продукційна система	База знань додатку, цільова функція	Пам`ять користувача

Продовження таблиці 4.2

Рівень	Елементи	Інформаційні масиви	Місцеположення
Керування станами	Керуючі автомати (КА)	Таблиці виходів і переходів КА	
Керування комплексними операціями	Вхідні, вихідні операційні автомати (ОА)	Файли (структури) даних, файли форсування, файли параметрів ОА	
Керування типовими операціями	Інструкція мови програмування	Структури керування	
Керування системними операціями	Операційна система контролера	Файл статусу (системних змінних) контролера	
		-	Системна пам`ять контролера
Керування апаратними засобами	Процесор, вхідні, вихідні, мережні модулі, блок живлення, операторський інтерфейс	-	Шасі, корпус контролера
Зв'язок з об'єктом керування	Датчики, виконавчі механізми		Об'єкт замовника
Об'єкт керування (ОК)	Агрегат, фізичний, технічний процес		

На нижньому рівні знаходиться об'єкт керування – агрегат, технічний процес, якого описується входами і виходами.

Входи підрозділяються на керовані і некеровані. Через некеровані входи в систему поступають збурювальні дії, наприклад, температура навколишнього середовища, параметри навантаження агрегату і т.д. Керовані входи ОК пов'язані з виконавчими механізмами. Виходи несуть інформацію про стан ОК, яка поступає в систему керування через датчики.

Якість взаємодії ОК з системою керування описується характеристиками керованості та спостережуваності.

Керованість (controllability) – показує, чи достатньо даного набору керованих входів для реалізації цілей керування, тобто досягнення необхідних станів ОК.

Спостережуваність (observability) характеризує достатність набору виходів і підключених до них датчиків для оцінки станів ОК в цілях керування.

На наступному рівні представлені апаратні засоби контролера: вхідні, вихідні модулі, процесор, мережний модуль, блок живлення, вузол операторського інтерфейсу. З погляду керування істотним є те, що ці засоби конфігуруються. Крім того, через апаратні засоби контролера можуть замикатися додаткові контури керування. Наприклад, на основі аналізу кольору і характеру освітлення (миготіння, безперервне) статусних світлодіодів на процесорі, оператор приймає рішення, яке доводиться до контролера через вузол операторського інтерфейсу, а інформація від датчика мережної напруги керує програмною процедурою обробки збоїв по живленню.

Через мережний модуль контролер обмінюється інформацією із станцією програмування, іншими контролерами, інтелектуальними приладами (наприклад, з приводами, аналізаторами якості електроенергії, засобами операторського інтерфейсу) і комп'ютерами автоматизованих робочих місць (АРМ) оператора. Джерела і споживачі цієї інформації можуть знаходитися на всіх вищих рівнях керування контролером.

Важливим рівнем керування є керування системними операціями контролера. Це керування здійснюється операційною системою (ОС) контролера, яка виконує наступні функції:

- завантаження, віддалене керування режимами програмних автоматів додатку користувача;
- збереження/відновлення стану додатку користувача при збої/включенні електроживлення контролера;
- організація програмного скану, включаючи обмін інформацією між апаратними засобами і пам'яттю користувача, керування багатозадачністю і перериваннями, обмін по мережах і діагностика працездатності апаратних засобів контролера.

Елементи ОС не доступні для користувача. Єдиною його можливістю є читання і запис системних змінних, що зберігаються в статусному файлі контролера.

Всі рівні, розташовані вище за рівень ОС, є програмно реалізованими автоматами, які зберігаються в пам'яті користувача.

Більшість інструкцій мови програмування мають досить складний алгоритм виконання, з внутрішніми станами, які відображаються через елементи структур керування. З цієї точки зору, їх можна розглядати як автомати типових операцій користувача і об'єднати в окремий рівень керування.

Автомати типових операцій використовуються при побудові операційних і керуючих автоматів.

На рівні керування операціями виділимо вхідні і вихідні ОА. Вхідні ОА виконують операції над вхідними змінними різних типів. Типові завдання, що вирішуються вхідними ОА, це:

- обчислення логічних функцій;
- обчислення предикатів;
- підрахунок подій;
- обчислення функцій з використанням арифметичних, логічних операцій, тригонометричних і інших функцій;
- обробка бази даних (пошук, фільтрація, сортування);
- розпаковування, перетворення кодів і величин, контроль даних, розпізнавання, статистична обробка даних, визначення пріоритетів.

Результати роботи ОА у вигляді умов переходу поступають на входи керуючого автомата (КА), і (або) у вигляді даних поступають на входи інших ОА.

Вихідні ОА формують значення вихідних змінних контролера. Розрізнятимемо вихідні керовані і некеровані ОА. Керовані ОА активізуються вихідним сигналом керуючого автомата, а некеровані –

виходами вхідних ОА. У окремому випадку, якщо потрібна постійна активність ОА, то вони взагалі не мають входів.

Типові вихідні ОА це: генератори і формувачі стимулюючих дій на ОК; квалифікатори кроків автомату, що управляє; регулятори параметрів ОК; синтезатори структур і масивів даних.

Рівень керування станами системи керування представлений в структурі керування керуючими автоматами. Кожен КА, як правило, керує станами окремого завдання, програми, процедури, тобто автомати можуть використовуватися паралельно. Тільки однозадачний додаток, в якому відсутні процедури обробки переривання, керуватися одним КА.

У кожен момент часу КА може знаходитися тільки в одному стані. При зміні вектора входів X, що поступає від ОА, КА переходить в новий стан. При цьому на виходах КА формується вектор виходів У, який управляє активністю вихідних ОА.

КА може бути побудований як автомат Мілі або як автомат Мура. Автомат Мура пов'язує окремий стан з вектором виходів, тобто зрештою з діями, які виконують вихідні ОА, активізовані КА в цьому стані. У автоматі Мілі стан задає спосіб інтерпретації вектора входів X. Кожному з дозволених в даному стані векторів X ставиться у відповідність свій вектор У і новий стан.

У обох випадках, стан – це інформація про минулі КА, необхідна для визначення його поведінки в майбутньому.

При проектуванні КА необхідно враховувати можливість утворення «паразитних» станів через зворотні зв'язки на нижчих рівнях керування. Наприклад, хай за допомогою контролера повинна бути реалізована функція керування включенням реле Р за допомогою контакту S1 або програмної ознаки X. На рис. 4.11 а приведений фрагмент програми на мові LD (див. розділ 5.1), що реалізовує відповідний КА.

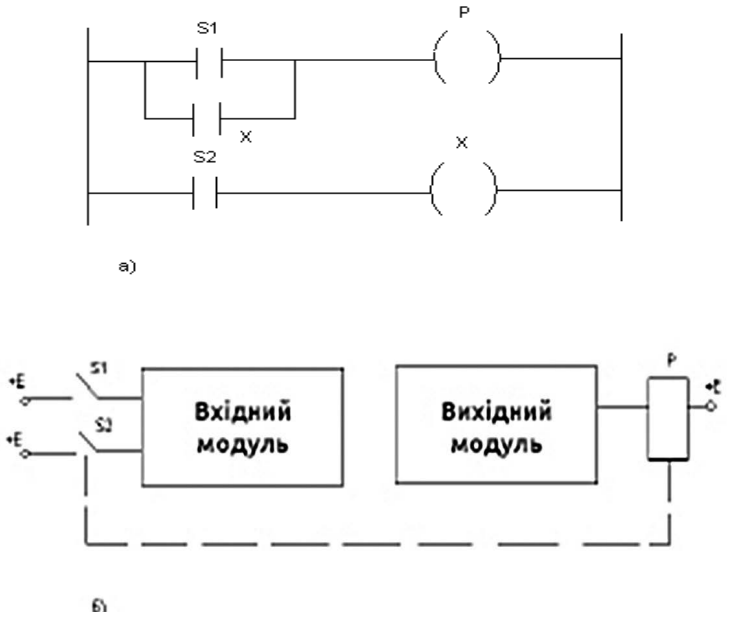


Рисунок 4.11. Приклад утворення «паразитних» станів: а) – програма; б) – схема підключення до контролера

У свою чергу, реле Р керує замиканням контакту S2 (рис. 4.11 б). Окремо ланцюги рис. 4.11 б є комбінаційною релейною схемою, а програма рис. 4.11 а реалізує логічні функції і стан виходу Р не володіє властивістю пам'яті. Разом з тим, через апаратний зворотній зв'язок, стан контакту S2 залежить від стану реле Р. А, відповідно до програми, стан реле Р залежить від стану контакту S2. Тому, при першому ж замиканні контакту S1, реле Р ввімкнеться і стане на самоблокування. Виникне новий стійкий стан контролера, а функція керування реле буде загублена.

Алгоритми обробки даних і керування станами можуть бути змінені під час експлуатації контролерної системи керування. Для цього застосовуються два способи: перепрограмування і адаптація.

Перепрограмування проводиться шляхом заміни виконуваного коду додатку за ініціативою програміста. Адаптація може виконуватися автоматично без участі людини. В ході адаптації накопичуються і оцінюються результати керування, формується прогноз зміни

контрольованих і неконтрольованих параметрів об'єкту керування. На підставі цих даних змінюються інформаційні масиви, що представляють структуру і параметри керуючих і операційних автоматів, системні змінні контролера.

Керування адаптацією можна виділити в окремий рівень керування, який використовує інструментарій систем штучного інтелекту. Ці інструменти вимагають значних ресурсів процесора, і тому рівень адаптації може бути реалізований в зовнішньому по відношенню до контролера додатку, наприклад, у вигляді моделі в середовищі додатку пакету програм MATLAB, який обмінюється даними з контролером через функції DDE.

Розглянемо, як взаємодіють апаратні засоби і операційна система контролера на рівні керування системними операціями. Важливими елементами цієї взаємодії є принципи розподілу пам'яті контролера, структура додатку користувача, режими роботи контролера, поняття програмного скану і форсування.

У пам'яті контролера можна виділити системну і призначену для користувача області. У системній області пам'яті поміщується операційна система контролера і ця частина пам'яті не доступна користувачеві. У деяких типах контролерів можна відновити операційну систему контролера, використовуючи спеціальне програмне забезпечення, що запускається на персональному комп'ютері станції програмування.

У призначеній для користувача пам'яті поміщаються дані і програми користувача. Дані характеризуються типом. Відповідно до стандарту на мови програмування контролерів [5] розрізняють елементарні і складені типи даних. До елементарних типів відносять цілочисельні (INTEGER), логічні (BOOL), дійсні (REAL), типи часу (TIME, DATA і ін.), рядкові (STRING). Елементарні типи служать основою для побудови складених типів. До складених типів відносяться перерахування, масиви, структури і ін. Ці типи визначаються також, як в інших мовах програмування, наприклад, у мові Паскаль. Специфічними є зумовлені керуючі структури для деяких інструкцій мови програмування, таких як таймери, лічильники, інструкції ПІД-регулювання, передачі повідомлень і ін.

Застосовується два типи організації даних в пам'яті. За першим способом однотипні дані об'єднуються у файли: бітових даних, цілочисельних даних, структур таймерів і т.д. В цьому випадку, для

звернення до даних треба вказати ім'я файлу, порядковий номер слова у файлі і номер біту. Наприклад, B3:2/5 означає п'ятий біт в другому слові третього файлу типу B (binary, логічний); T 4:1 – перша структура таймеру в четвертому файлі типу T (Timer, таймер) структур таймерів. За другим способом всім елементам даних присвоюються імена. Іменовані дані називаються тегами. Розміщення тегів в пам'яті виконує операційна система контролера. Звернення до тегів в програмі проводиться по їх іменах.

Організація програм в пам'яті користувача залежить від типу додатку: однозначне або багатозначне. У однозначному додатку виділяють основну програму (процедуру), процедури підпрограм, переривань за часом і по події, процедури обробок помилок. Основна програма (процедура) виконується постійно, тобто вона автоматично перезапускається після завершення свого виконання. Підпрограми запускаються з основної програми або з підпрограм (вкладення підпрограм). Процедури переривання за часом запускаються періодично, по закінченню заданого інтервалу часу. Конфігурація події, по якій запускається процедура переривання по події, включає завдання адреси каналів вхідних дискретних модулів і значення даних. На час виконання процедури переривання робота основної програми припиняється. Переривання можуть бути заборонені з основної або інших процедур.

В ході виконання програми в контролері можуть виникати різні «нештатні» ситуації (помилки), які приводять до неправильного керування об'єктом. Наприклад, виходи даних за діапазони допустимих значень, відмови модулів введення-виведення, каналів передачі даних і т.д. Для усунення наслідків деяких з цих помилок використовують процедуру обробки помилок. Ця процедура взаємодіє з елементами файлу системних змінних (статусного файлу) контролера, де накопичуються дані про типи помилок.

Один цикл виконання цього завдання називають програмними сканом. Його етапи приведені на рис. 4.12.

На першому етапі скану дані з регістрів пам'яті модулів введення по системній шині контролера записуються в призначену для користувача пам'ять процесора. На другому етапі власне виконується сканування програми керування і, відповідно до закладеної в неї логіки, формуються нові значення вихідних змінних, які записуються в призначену для користувача пам'ять процесора. І лише на третьо-

му етапі значення цих змінних по системній шині контролера записуються в регістри пам'яті модулів виходу. Внаслідок чого змінюються дані на входах об'єкта керування.

Рисунок 4.12. Етапи програмного скану

Таким чином, якщо вхідні дані контролера змінилися після виконання етапу 1, то ці зміни будуть враховані тільки в наступному програмному скані, а затримка між формуванням вихідних змінних і змінами дій на виходах модулів виходу залежить від часу виконання програми користувача і може досягати декількох секунд. Для критичних за часом завдань, наприклад, обробка аварійних ситуацій, є можливість оновлення окремих входів і виходів під час виконання сканування логіки. Ця можливість реалізована за допомогою спеціальних інструкцій мови програмування.

На четвертому етапі програмного скану виконується обмін повідомленнями по мережах, наприклад, із станцією програмування, іншими контролерами, засобами операторського інтерфейсу.

На п'ятому етапі програмного скану виконується контроль працездатності і діагностика несправностей апаратних засобів контролера. Це дозволяє оперативно виявити несправність.

Час виконання програмного скану фіксується в статусному файлі контролера і контролюється сторожовим таймером. Сторожовий таймер (Watchdog) – це апаратний лічильник часу виконання скану. Якщо цей час перевищує деяку задану межу, то встановлюються біти помилки в статусному файлі контролера.

До недоліків використання програмного скану можна віднести повторення в кожному скані етапів 1 і 3 оновлення входів і виходів,

незалежно від частоти зміни значень на входах (виходах) контролера. Наприклад, якщо до входу контролера приєднана кнопка, яку натискають кілька разів на добу, а в програмному скані стан цієї кнопки вводитиметься десятки разів кожну секунду доби. Тому в деяких контролерах обмін інформацією з модулями введення/виводу виконується тільки при визначених при конфігурації модуля змінах значень на входах (виходах) контролера. Ці зміни відбуваються асинхронно по відношенню виконуваним програмним процедурам. Тому, якщо проектант не хоче враховувати зміни вхідних даних процедури, що відбулися з моменту чергового її запуску, то він повинен передбачити на початку процедури буферизацію цих даних.

Типова структура багатозначного додатку приведена на рис. 4.13.

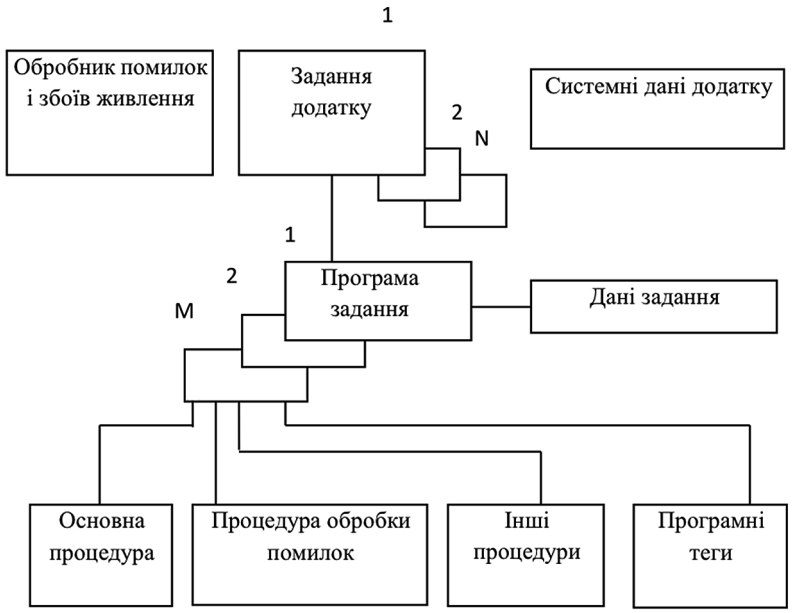


Рисунок 4.13. Структура багатозадачного додатку

Такий додаток містить завдання, обробник помилок і збоїв живлення і системні дані. До системних даних додатка відносять контролерні теги, дані введення/виводу, системні дані контролера. Завдання ділиться на програми. У додатку може бути тільки одне циклічне (що перезапускається після виконання) завдання і воно

має найменший пріоритет, тобто може бути перервано будь-яким періодичним завданням.

Програми складаються з процедур, програмних тегів і даних завдання. До даних завдання відносять конфігураційні дані, статус і сторожовий таймер завдання. Конфігураційні дані завдання містять інформацію про порядок запуску, пріоритет, час виконання і тип завдання (періодичне або циклічне). Програмні теги завдання, на відміну від контролерних тегів, не доступні іншим завданням додатку. Склад процедур завдання подібний до складу процедур однозадачного додатку.

Ряд функціональних можливостей контролера пов'язаний з режимами роботи його процесора. Характеристика типових режимів процесора наведена в табл. 4.3. Режим процесора задається положенням ключа на передній панелі процесора.

Таблиця 4.3. Характеристика типових режимів процесора

Найменування режиму	**Характеристика**
Program (програмування)	Зміна програми за допомогою пакету програмування. Програма користувача зупинена
Run (робота)	Виконується програма (додаток) користувача. Можливе переглядання результатів виконання програми в пакеті програмування. Неможливе редагування програми і зміна режиму роботи процесора з пакету програмування.
Remote run (віддалене керування)	Виконується програма (додаток) користувача. Можливе переглядання результатів виконання програми і зміна режиму роботи процесора з пакету програмування. Неможливе редагування програми

Серед режимів роботи процесора, що задаються при видаленому керуванні з пакету програмування, виділимо режим Test (тест), в якому програма користувача виконується як в режимі Run, тільки значення виходів, розраховані в програмі користувача, не передаються у вихідні модулі. Цей режим корисний при налагодженні програми, оскільки він дозволяє запобігти небезпечним переміщенням механізмів об'єкту керування через помилки в програмі.

Важливим елементом функціональних можливостей контролера є форсування (forsing) введення/виводу. Під форсуванням розуміється можливість змінювати значення вхідних і вихідних даних контролера. Якщо форсування дозволено, то ці значення беруться з таблиць форсування, які є елементом додатку користувача. Форсування використовується, наприклад, для примусового виключення, зупинки, переходу в безпечний стан агрегату або процесу користувача у разі аварії.

5 ПРОГРАМУВАННЯ КОНТРОЛЕРІВ ТА ФОРМАЛІЗАЦІЯ ПРОЕКТНИХ РІШЕНЬ

5.1 Стандартні мови програмування контролерів

Стандарт IEC 61131–3 визначає загальні вимоги до мов програмування і види мов для програмованих контролерів.

Створений спочатку як керівництво для розробників спеціалізованих засобів програмування контролерів, фірм-виробників контролерів, стандарт IEC 61131–3 породив нову, для контролерних систем, різновид програмних продуктів – універсальний засіб програмування на стандартних мовах за стандартом IEC 61131–3, наприклад пакет Codesys фірми 3s – Smart Software Solution (сайт http://www.3s-software.com) .

Мови програмування, визначені стандартом IEC 61131–3 приведені в табл. 5.1.

Мова LD – представляє логіку програми у формі, схожій на електричну схему з ланцюгами, що складаються з контактів і обмоток (котушок) реле. Така форма була особливо популярна на етапі переходу від релейних автоматів до програмованих контролерів. Програма на мові LD складається з рангів (rung). Кожен ранг – це програмний аналог логіки одного електричного ланцюга. Ранг містить умовні і виконавчі інструкції і еквівалентний операторові умовного виконання IF умова THEN дія, де умова представлена однією або декількома умовними інструкціями, логічні зв'язки між якими зада-

ні графічно гілками рангу, а дія представлена однією або декількома виконавчими інструкціями, послідовність виконання яких також задана графічно гілками рангу. Приклад рангу приведений на рис. 5.1.

Таблиця 5.1. Стандартні типи мов програмування контролерів

Позначення	**Найменування**	**Тип мови**	**Область найбільш ефективного застосування**
LD (RLL)	Ladder Diagram (драбинні діаграми) Relay Ladder Logic (релейна драбинна логіка)	графічна	Дискретна логіка.
SFC	Sequential Function Chart (послідовні функціональні схеми)	графічна	Побудова функціональної специфікації системи, організація керування як послідовності кроків або станів, через які проходить система.
ST	Structured Text (структурований текст)	текстова	Складні алгоритми, робота з рядковими даними, файлами, структурами даних
FBD	Function Block Diagram (функціональні блокові діаграми)	графічна	Процеси безперервного керування, алгоритми керування, які виконуються на постійній основі.
IL	Instruction List (список інструкцій)	текстова	Там, де потрібна мінімізація часу виконання і об’єму пам’яті програмного коду.

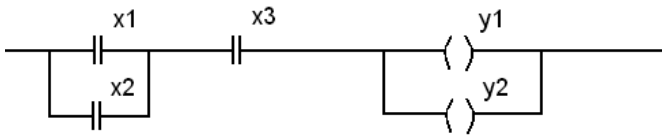


Рисунок 5.1. Приклад рангу програми на мові LD

Інтерпретуємо x1, x2, і x3 як контакти (датчики), а y1, y2 – як котушки (виконавчі механізми) підключені до контролера. Тоді оператор умовного виконання для рангу рис. 5.1 прийме вид IF (x1 OR x2) AND x3 THEN y1, y2. Тобто, якщо включені хоча б один з контактів x1 або x2 і контакт x3, то включити виконавчий механізм y1 і потім виконавчий механізм y2, інакше – вимкнути ці механізми.

Сучасна LD – це мова програмування високого рівня, яка містить близько півтори сотні інструкцій, включаючи такі складні інструкції як ПІД-регулювання, передача повідомлень по мережі іншим контролерам, операції під матрицями, синхронне переміщення осей. Детальніше мова LD і приклади програмування на ній розглянуті в наступних розділах.

Мова FBD також представляє логіку програми у формі, схожій на електричну схему, але, на відміну від LD, ця схема нагадує схему цифрового вузла з логічними елементами, суматорами, лічильниками і т.п. Діаграма FBD описує залежності вихідних змінних від вхідних змінних з використанням стандартних елементарних функціональних блоків і ліній з’єднання. Номенклатура функціональних блоків, приблизно така ж сама як інструкцій мови LD, тобто включає функціональні блоки, що виконують керування даними, булеві і арифметичні операції, операції порівняння, перетворення даних, математичні функції, блоки таймерів і лічильників та інші. Стандартний функціональний блок записується на діаграмі FBD у вигляді прямокутника, який має певне число входів і виходів. Входи стандартного функціонального блоку розташовуються на лівій стороні прямокутника і з’єднуються з вихідними змінними або виходами інших функціональних блоків. Виходи стандартного функціонального блоку розташовуються на правій стороні прямокутника і з’єднуються з вихідними змінними або вхідними інших функціональних блоків. Лінія з’єднання є направленою і передає свої дані від входу до виходу.

Діаграма FBD може містити декілька підсхем, що мають непересічні підмножини вхідних змінних. Такі підсхеми називають ланцюгами. Програми, відповідні ланцюгам виконуються послідовно зверху-вниз в порядку розташування в діаграмі FBD. Приклад ланцюга в діаграмі FBD приведений на рис. 5.2. Тут залежність виходу у від вхідних змінних in1, in2, in3 відповідає формулі

$$y = (in1 + in2)/2 - in3 \qquad (5.1)$$

Прообразом мови SFC є мова функціональних карт, яка описана стандартом IEC 848 Підготовка функціональних карт для систем, що управляють. А перша мова функціональних карт – Grafcet (Graphe de Commande Etape – Transition, фр.). Функціональна карта – покроковий перехід була розроблена у Франції в кінці 70-х років минулого століття.

Мова SFC представляє логіку програми у формі направленого графа. Вершини графа відповідають крокам, а дуги – переходам процесу керування. Процес виконання програми SFC можна представити як процес переходу від кроку до кроку. Коли крок стає активним, він ініціює виконання певних дій або може викликати дочірню SFC програму. Інтервал часу, в якому виконуються дії, може не співпадати з інтервалом активності кроку і задається квалификатором кроку. Переходи SFC програми описують умови припинення активності поточного кроку і передачі її наступному кроку. Дія кроків і логічні умови переходу програмуються на мовах LD, FBD, ST, IL.

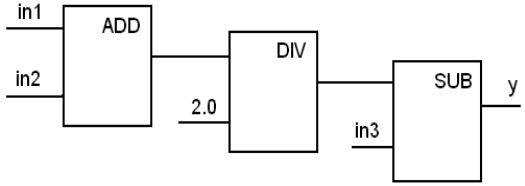


Рисунок 5.2. Приклад ланцюга в діаграмі FBD

Альтернативні гілки на графі SFC показуються одинарною лінією множинних з'єднань. Кожна гілка починається і закінчується власною умовою переходу. Перевірка альтернативних умов проводиться зліва направо. Якщо вірна умова знайдена, то інші альтернативи не розглядаються.

Паралельно виконувані гілки на графові SFC позначаються подвійними горизонтальними лініями. Ці гілки мають одну умову

на вході і одну умову на виході галузі. Умова переходу, що завершує паралельність перевіряється тільки тоді, коли активні останні кроки в кожній паралельній гілці.

На рис. 5.3 приведений приклад програми SFC. Крок, обведений подвійним квадратом, є початковим, кроки 1, 2 виконуються послідовно, кроки 3, 4 розташовані в альтернативних гілках, а фрагменти з кроками 6, 8 і 7, 9 виконуються паралельно.

Мову ST часто називають «паскалеподібною» за схожість синтаксису і операторів. Програма ST – це список операторів, що закінчуються крапкою з комою. Мова ST використовується для опису умов переходів і дій усередині кроків програми SFC. З програми ST легко викликати програми на мовах ST, IL, LD або FBD, а в деяких програмних засобах і «C – функції».

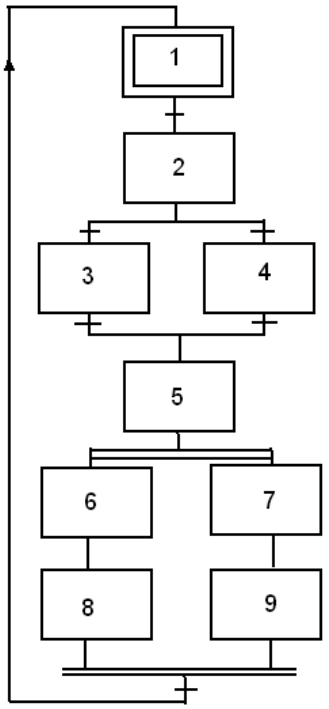


Рисунок 5.3. Приклад програми SFC

Основні компоненти мови ST: приведені в табл. 5.2.

Мова ST використовує традиційні операції (арифметичні, логічні, порівняння) і оператори (привласнення, переходу, розгалудження, циклів) .

Таблиця 5.2. Основні компоненти мови ST

Визначення	**Опис**	**Приклади**
Присвоєння (assignment)	Для присвоєння величин тегам	tag:= expression
Вираз (expression)	Вираз включає: - теги (tags) – найменування області пам'яті, в яких зберігаються дані; - безпосередні значення (immediate values) – числа, символи; - оператори (operators) – символи або мнемоніка для специфікації операцій; - функції (functions) – при виконанні вони повертають одне значення.	Value 1 4 tag1+tag2 tag1>= value 1 function (tag1)
Інструкції (instruction)	На відміну від функцій не можуть бути використані у виразах	Instruction (); Instruction (operand);

Конструкції (construct)	Задають вид алгоритму обробки даних, що розгалужується або циклічного	IF… THEN
		CASE
		FOR… DO
		WHILE… DO
		REPEAT… UNTIL
		EXIT.

Коментарії (comment)	Текст, який не впливає на виконання програми	// comment (* start of comment… end of comment*)

Приклад програми ST приведено на рис. 5.4.


```
PROGRAM ST_Example
VAR
in1, in2, in3, y: REAL;
END_VAR
BEGIN
y:= ((in1+in2) /2) -in3;
END.
```

Рисунок 5.4. Приклад програми ST

Мова IL – це «ассемблероподібна» мова. Асемблерні мови широко застосовуються при програмуванні мікроконтролерних систем [13], які можуть служити прототипами при проектуванні систем керування на промислових контролерах. Як вже наголошувалося, компактність коду і малий час виконання програм IL – ось мета застосування і вивчення мови IL.

Програма на IL є послідовністю інструкцій. Інструкція IL може містити від одного до чотирьох полів, тип і призначення яких приведені в табл. 5.3.

Таблиця 5.3. Структура полів інструкції IL

Найменування	**Тип**	**Призначення**
Метка	необов'язкове	Задає адресу переходу
Оператор	обов'язкове	Визначає дії, що виконує інструкція
Операнд	необов'язкове	Визначає дані, над якими виконуються дії. Якщо поле відсутнє, то операнд вибирається за замовчуванням
Коментарій	необов'язкове	Опис, роз'яснення дій виконуваних інструкцією

Стандартні оператори мови IL приведені в табл. 5.4.

Приклад програми IL, аналогічної по функціях програмі FBD рис. 5.2 та програмі ST рис5.4 приведений на рис. 5.5.

Таблиця 5.4. Стандартні оператори мови IL

Оператор	Опис
LD	Завантажити значення операнду в акумулятор
ST	Привласнити значення акумулятора операнду
S	Якщо акумулятор TRUE, встановити значення операнду TRUE
R	Якщо акумулятор TRUE, встановити значення операнду FALSE
AND	Порозрядне І
OR	Порозрядне АБО
XOR	Порозрядне виключаюче АБО
ADD	Складання
SUB	Віднімання
MUL	Множення
DIV	Ділення
MOD	Ділення по модулю
GT	>
GE	>=
EQ	=
NE	≠
LE	<=
LT	<
JMP	Перехід до мітки
CAL	Виклик підпрограми, функціонального блоку
RET	Вихід і повернення в програму, що викликала підпрограму

```
LD in1
ADD in2
DIV 2
SUB in3
ST y
```

Рисунок 5.5. Приклад програми IL

Окрім стандартних, існують додаткові оператори, які підтримуються не всіма апаратними платформами контролерів.

До деяких операторів можуть приєднуватися модифікатори, що змінюють сенс операції. До них відносяться наступні модифікатори: N – булеве заперечення операнду; «(« – відкладена операція; C – умовна операція. Наприклад, при виконанні інструкції AND з модифікатором N (ANDN x), операція AND виконується над інверсією операнду x. А послідовність інструкцій

```
AND (x1
OR x2
)
```

означає, що операція AND виконується над результатом операції OR з операндами x1 і x2.

Модифікатор C приєднують до операторів JMP, CAL, RET. За наявності модифікатора ці оператори приймають вид JMPC, CALC, RETC і відповідні інструкції виконуються тільки якщо регістр акумулятора (результату) має булеве значення TRUE (або відмінне від нуля не булеве значення). Модифікатор C може застосовуватися спільно з модифікатором N. Наприклад, інструкція з оператором JMPCN виконуватиметься, тільки якщо регістр акумулятора має булеве значення FALSE.

5.2 Структура програмного забезпечення систем промислової автоматизації

Програмне забезпечення (ПЗ) систем промислової автоматизації включає додаток користувача, засоби розробки і використання цього додатку в середовищі системи промислової автоматизації. У загальному вигляді, додаток користувача складається з декількох додатків виконуваних на контролерах, інтелектуальних інформаційно-вимірювальних приладах, приладах операторського інтерфейсу і на персональних комп'ютерах в середовищі програм людино-машинного інтерфейсу і інших програм, наприклад Excel, Access, MATLAB і ін. Програмне забезпечення для розробки і використання додатків користувача можна уявити собі у вигляді будинку (рис. 5.6) .

Операційна система контролера займає перший поверх будинку. Функції ОС контролера розглянуті вище.

Програмне забезпечення всіх подальших рівнів розміщується поза контролером – на ПК станції програмування або ПК АРМ оператора.

Поверх	Назва	Характеристика
6	Офісний	Пакети обміну інформацією з додатком керування через мережу Інтернет і мобільні мережі
5	Інформаційний	Пакети обміну інформацією між додатком керування і базами даних вищого рівня підприємства
4	Візуалізація	Пакети людино – машинного інтерфейсу, конфігуювання засобів операторського інтерфейсу
3	Програмування	Середовище програмування. Емулятор контролера
2	Комунікація	Комунікаційний сервер
1	ОС контролера	Операційна система реального часу

Рисунок 5.6. Структура програмного забезпечення для розробки і виконання додатків користувача

Завданням комунікаційного сервера є конфігурування і керування статусом прямих драйверів зв'язку при різних варіантах об'єднання контролерів в мережу, відображення активних вузлів в мережах підключених до ПК, конфігурування даних для DDE обміну даними між додатками, що входять до складу системи керування, і контроль працездатності мережі.

Середовище програмування забезпечує створення і документування додатку керування, його верифікацію, трансляцію з мови програмування в завантажуваний код, завантаження цього коду в контролер (Download), керування режимами роботи контролера, онлайнове спостереження за виконанням додатку в контролері, вивантаження коду, що виконується в контролері, в пакет програмування (Upload) і відображення його на стандартній мові програмування. Створення додатку передбачає вибір процесора, способу комунікацій з контролером, розміру шасі, конфігурування модулів введення/виходу (вибір і розміщення у слотах шасі, завдання параметрів каналів), завдання структури додатку (конфігурація завдань,

програм, процедур, даних), введення логіки користувача і коментарів на стандартній мові програмування і інші операції.

У тих випадках коли відсутній реальний контролер прогін і налагодження програми користувача можна виконувати за допомогою емулятора. Емулятор – це програмний продукт, який виконується на ПК станції програмування і емулює виконання додатку користувача для одного або декількох контролерів. Результати виконання програми користувача за допомогою емулятора контролера можна спостерігати за допомогою пакету програмування. Екран емулятора дозволяє управляти режимом емульованих контролерів, тобто запускати і зупиняти виконання програми, задавати ім'я файлу виконуваної програми. Емулятор не завжди забезпечує 100%-процентну відповідність реальному контролеру, особливо там, де це стосується тонкощів виконання програми в реальному часі. Проте, це дуже корисний інструмент розробника.

Пакет візуалізації забезпечує створення, документування і виконання додатку візуалізації на ПК або завантаження його в панель операторського інтерфейсу. Ці пакети спеціалізовані для кожного виду приладів, за допомогою яких виконується візуалізація.

Створення проекту візуалізації включає створення бази тегів, системи віртуальних екранів керування, підсистем тривожної сигналізації, реєстрації даних, діяльності і ряду інших підсистем.

Тег проекту візуалізації – це аналог змінної величини в звичайних мовах програмування. Вони підрозділяються на приладові і системні теги. Приладовий тег відповідає пойменованій області пам'яті в контролері, а системний генерується самою системою візуалізації. Типові приклади системних тегів – це теги секунд (System/second), хвилин (System/minute) системного часу. Значення тегу із заданою періодичністю заноситься в базу тегів під час виконання додатку візуалізації. Передача інформації у зворотному напрямі, тобто з бази тегів в контролер здійснюється як реакція на деяку подію в системі візуалізації. Наприклад, в певний час, при виконанні деякої умови, при виділенні курсором миші кнопки на екрані.

Кожний екран системи віртуальних екранів містить графічні об'єкти, візуальні параметри яких в реальному часі відображають параметри об'єктів керування, істотні для вибору дії оператору.

На п'ятому (інформаційному) рівні ПЗ керує обміном даними в реальному часі між компонентами додатку керування об'єктом (процесом) користувача і базами даних вищого рівня, наприклад, рівня підприємства.

На шостому (офісному) рівні ПЗ надає послуги видаленого доступу через мережу Інтернет до інформації додатку керування. Серед типових послуг відзначимо формування і пересилку підготовлених системою візуалізації листів по електронній пошті і SMS-повідомлень на мобільні телефони персоналу керування об'єктом користувача, надання у видаленому браузері доступу до віртуальних екранів керування додатку візуалізації. Грубо кажучи, знаходячись на курорті або в дальньому відрядженні, Ви можете зайти в Інтернет-кафе, набрати заповітну IP-адресу і пароль, відкрити екран керування додатку візуалізації, отримати необхідну інформацію і, наприклад, вимкнути виконавчий механізм в об'єкті керування.

5.3 Мова драбинних діаграм (LD): структура, елементи програм, інструкції

Мова драбинних діаграм (Ladder Diagram (LD)) – графічна мова високого рівня. Як вже зазначалося, програма на мові LD складається з рангів, які виконуються послідовно. Для зміни порядку виконання рангів, організації розгалужень і циклів у програмі використовуються інструкції керування потоком програми. Усі інструкції мови LD поділяються на умовні (вхідні), що перевіряють істинність деякої умови і виконавчі (вихідні), що виконують дії над даними. Інструкції мови поміщаються в послідовні і (або) паралельні гілки умовної і виконавчої частин рангу. По виконуваних функціях інструкції мови LD об'єднуються в групи: бітові, таймерів, лічильників, порівняння, математичні, логічні, керування потоком програми, переміщення, перетворення, спеціальних операцій, ASCII-інструкцій та інші. Перелік доступних для обраного процесору інструкцій наводиться в керівництвах по застосуванню контролера і в довідковому розділі середовища програмування. Кожна інструкція має графічне позначення, мнемокод, тип (виконавча, умовна), операнди і, в деяких випадках, керуючі структури. Результати виконання ряду інструкцій, зокрема математичних, можуть впливати на стан математичного регістра процесора.

Бітові інструкції використовується для контролю та керування станом бітів. Опис деяких бітових інструкцій наведені в табл. 5.5.

Таймери і лічильники контролюють операції, засновані на часі або кількості подій. Всі інструкції цієї категорії є вихідними і всі, крім інструкції RES, мають ряд параметрів, об'єднаних в структуру, асоційовану з тегом таймера (лічильника) .

Таблиця 5.5. Бітові інструкції

Мнемокод	**Назва**	**Тип**	**Опис**
XIC	eXamine If Close Перевірити на стан Включено	Умовна	Істина (дозволяє виходи), коли біт встановлений.
XIO	eXamine If Open Перевірити на стан Виключено	Умовна	Істина (дозволяє виходи), коли біт скинуто.
OTE	OuTput Energize Вихід включити	Виконавча	Коли інструкція дозволена, вона встановлює біт даних, а коли заборонена – скидає.
OTL	OuTput Latch Фіксація виходу	Виконавча	Коли інструкція дозволена, вона встановлює біт даних. Наступна заборона інструкції не змінює стан біта даних.

Найменування та опис інструкцій наведено в табл. 5.6, а параметри структур – в табл. 5.7.

Таблиця 5.6. Інструкції таймерів та лічильників

Мнемо-код	Назва	Опис
TON	Timer ON delay Таймер с затримкою на включення	Таймер що не зберігає, який накопичує час при дозволі виконання інструкції до тих пір, поки не буде заборонено виконання або коли накопичений час ACC не досягне уставки PRE. З моменту рівності ACC і PRE встановлюється біт виконання DN. Коли забороняється виконання інструкції TON накопичений час ACC і біт виконано DN обнуляється.
TOF	Timer OFF delay Таймер с затримкою на виключення	Таймер, який накопичує час затримки після завершення дозволу інструкції. Біт виконання DN встановлено від моменту дозволу інструкції до моменту рівності ACC і PRE.
RTO	Retentive Timer On Накопичу-вальний таймер	Таймер накопичує час при дозволі виконання інструкції. З моменту рівності ACC і уставки PRE встановлюється біт виконання DN. Коли забороняється виконання інструкції RTO, накопичене час ACC зберігається. Для скидання величини ACC використовується інструкція RES.
CTU	CounT Up Прямий рахунок	При кожному дозволі виконання інструкції CTU, накопичена величина ACC збільшується на одиницю. З моменту рівності ACC і уставки PRE встановлюється біт виконання DN. Для скидання величини ACC використовується інструкція RES.

Продовження таблиці 5.6

Мнемокод	Назва	Опис
CTD	CounT Down Зворотній рахунок	При кожному дозволі виконання інструкції CTD, накопичена величина ACC зменшується на одиницю. Біт виконання встановлюється коли ACC ≥PRE. Для скидання величини ACC використовується інструкція RES.
RES	RESet Скидання	В таймерах і лічильниках скидає накопичену величину ACC і керуючі біти стану, а в структурах керування – величину позиції поточного елемента POS і керуючі біти стану.

Таблиця 5.7. Параметри структур таймерів та лічильників

ACC	TON, TOF, RTO	Акумулятор (ACCumulator), величина накопичення часу
	CTU, CTD	Величина накопичення подій

Позначення	Інструкція	Опис
PRE	TON, TOF, RTO	Уставка (PREset) часу
	CTU, CTD	Уставка (PREset) подій
DN	TON, TOF, RTO	Біт виконання (DONe)
EN		Біт дозволу (ENable)
TT		Біт рахунку (TimerTimed)
CU	CTU	Біт дозволу прямого рахунку (Count Up)
CD	CTD	Біт дозволу зворотнього рахунку (Count Down)

Продовження таблиці 5.7

Позначення	Інструкція	Опис
OV	CTU, CTD	Біт переповнення (Over flow), встановлюється в момент переходу від максимальної позитивної величини в акумуляторі до негативній величині
UN		Біт втрати значущості (Under flow), встановлюється в момент переходу від максимальної негативної величини в акумуляторі до позитивної величині

Інструкції порівняння – це категорія вхідних інструкцій призначених для порівняння величин. Вони включають наступні інструкції: EQUal (EQU) – перевіряє на рівність дві величини; Not Equal (NEQ) – перевіряє на нерівність дві величини; Greater then oR eQual (GRQ) – перевіряє, коли величина А більше величини В або дорівнює їй; GReaTer then (GRT) – перевіряє, коли величина А більше величини В; Less then or EQual (LEQ) – перевіряє, коли величина А менше величини В або дорівнює їй; LESs then (LES) – перевіряє, коли величина А менше величини В; LIMit test (LIM) – перевіряє, коли величина Test перебуває всередині діапазону від Low Limit до High Limit.

Математичні інструкції – це категорія вихідних інструкцій, які виробляють арифметичні операції додавання (ADD), віднімання (SUB), ділення (DIV), множення (MUL), добування кореня (SQR), обчислення логарифма (LOG), синуса (SIN) та інші. Вони виконуються щоразу, коли інструкція дозволена. Результати виконання інструкції обчислення заносяться в операнд Dest (Destination, Призначення), а також змінюють арифметичні прапори стану, такі як С – перенос (Carry), V – переповнення (oVerflow), Z – нуль (Zero) і S – знак (Sign) .

Логічні вихідні інструкції, такі як bitwise AND (AND), OR (OR), eXclusive OR (XOR) виконують логічні операції з бітами. Результат виконання інструкцій заноситься в операнд Dest, а також змінюють арифметичні прапори стану.

Інструкції керування потоком програми застосовуються для зміни логіки виконання програми, тобто послідовності виконання рангів. Типові представники цієї категорії інструкцій: JuMP (JMP), LaBeL (LBL), Jump to SubRoutine (JSR), SuBRoutine (SBR), Return (RET) .

Якщо інструкція JMP не виконується, то далі програма виконує черговий ранг. При виконанні інструкції JMP відбувається перехід до рангу, заданого інструкцією LBL із таким самим операндом, якій має інструкція JMP. Розрізняють перехід вперед і перехід назад по програмі. При переході вперед, ранги програми між інструкціями JMP та LBL пропускаються (не виконуються). Перехід назад по програмі, виконується тоді, коли номер рангу, в якому розміщена інструкція LBL, менше номера рангу, в якому розміщена інструкція JMP. При переході назад в програмі утворюється цикл і збільшується час програмного скану.

При виконанні інструкцій JSR здійснюється перехід до виконання підпрограми, ім’я якої є операндом інструкції JSR. Підпрограма обов’язково повинна починатися інструкцією SBR та завершуватися інструкцією RET. Після завершення підпрограми продовжується виконання програми, що викликала підпрограму.

До розширених інструкцій віднесемо інструкції секвенсорів, зсуву бітових ланцюжків і PID-регулювання.

Інструкції секвенсорів служать для виконання послідовних і повторюваних операцій з елементами масивів даних. При кожному спрацьовуванні інструкції SeQuencer Output (SQO) за адресою операнда призначення пересилається чергове слово з масиву даних. Біти цього слова можуть містити вказівки щодо включення/вимикання виконавчих механізмів системи керування.

Інструкції зсуву бітових ланцюжків, наприклад, Bit Shift Left (BSL) при кожному спрацьовуванні інструкції виробляють зсув на один розряд значень елементів бітових ланцюжків, які задаються в інструкції початковим адресою і довжиною.

Пропорційно-інтегрально-диференційна (ПІД) інструкція – це вихідна інструкція за допомогою, якої контролюються фізичні властивості, такі як температура, тиск, рівень рідини або швидкості за допомогою петлі зворотного зв’язку процесу. Під час програмування ви конфігуруєте параметри блоку керування, змінної процесу і керуючої змінної. Блок керування це є файл, в якому зберігаються

дані, необхідні для роботи інструкції. Змінна процесу – це адреса елемента, який зберігає значення вихідного параметру процесу. Керуюча змінна – це адреса елементу, який зберігає результат ПІД-регулювання, тобто розраховане значення впливу на процес.

5.4 Формалізація завдань керування

Одним з істотних етапів проектування, у тому числі проектування контролерних систем керування, є формалізація завдань керування. Різноманіття застосовуваних при цьому засобів формалізації описів викликано описаними нижче причинами.

У результаті теоретичного опрацювання завдання, існує аналітичний або графічний опис задачі або її фрагментів у вигляді булевих виразів, предикатів, автоматних таблиць, графів автоматів, алгоритмів, мереж Петрі, тощо.

На момент проектування існує документація до функціонального аналогу системи керування на інший елементній базі. Наприклад, у вигляді електричної схеми цифрового логічного вузла, релейно-контактного вузла, програми на іншій, ніж використовується в контролері, мові програмування.

Існує функціональний аналог системи керування, отриманий в результаті моделювання об’єкта і контролера в універсальних пакетах моделювання, наприклад у пакетах Simulink, Stateflow.

Проектування системи керування ведеться із застосуванням одного з методів архітектурного проектування, наприклад COMET (Concurrent Object Modeling and Architectural Design Method) [14] і документується уніфікованою мовою моделювання UML (The Unified Modeling Language) [15] .

Отже, програміст повинен мати уявлення про ці види описів системи керування та способах їх «конвертації» в програму мовою сприйманою контролером.

Термін «алгоритм» походить від імені середньовічного узбецького математика Аль-Хорезмі і застосовується для завдання порядку виконання деяких операцій для вирішення всіх завдань будь-якого типу. Зокрема, порядку виконання керуючим пристроєм операцій в ході функціонування системи «керуючий пристрій – об’єкт керування».

Для формального опису алгоритмів використовуються мови логічних схем алгоритмів, формули переходів, матричні та графічні схеми алгоритмів та їх модифікації. Логічною схемою алгоритму (ЛСА) називають вирази, складені зі слідуючих один за одним операторів та логічних умов, а також розставлених певним чином нумерованих стрілок розгалуження. Наприклад, нехай задана ЛСА:

$A_0\ z_1 \uparrow^1 A_1 \downarrow^1 z_2 \uparrow^2 A_2\ A_3\ z_3 \uparrow^2 A_4 \downarrow^2 A_5\ A_k$

Ця ЛСА має оператори початку A_0 і закінчення A_k, п’ять операторів ($A_1 – A_5$) та три логічних умови ($z_1 – z_3$). Оператори алгоритму виконуються по ЛСА зліва направо з урахуванням значень логічних умов. Так, з початкового стану A_0 перевіряється умова z_1. Якщо z_1 виконується, то наступним виконується оператор, що стоїть в ЛСА праворуч від z_1, тобто A_1. Інакше виконується перехід по шляху: верхня стрілка ($\uparrow^1$) – нижня стрілка з тим же номером ($\downarrow^1$). Далі перевіряється умова z_1, що стоїть праворуч від нижньої стрілки, тобто z_2. Робота алгоритму закінчується, як правило, по досягненню A_k, або тоді, коли на деякому етапі не виявляється такого члена схеми, який мав би виконуватися.

5.5 Моделі скінчених автоматів для опису поведінки системи керування

Теорія автоматів є фундаментом сучасної теоретичної та практичної інформатики [16]. Моделі автоматів широко використовуються при проектуванні як апаратного, так і програмного забезпечення систем логічного керування. На відміну від функціональних перетворювачів, на основі яких побудовані цифрові комбінаційні вузли, автомати мають властивість пам’яті. Тобто їхня реакція залежить не тільки від входу в даний момент часу, але і від значень входів у попередні моменти часу. Свою передісторію автомат зберігає в станах. Стан – це характеристика, яка однозначно визначає реакцію автомата на наступні вхідні події. На практиці широко використовуються автомати з кінцевим числом станів. Їх називають кінцевими автоматами (FSM, Finite State Machine) .

Важливе місце в теорії займають два основних автомата Мілі та Мура. Нехай кінцевий автомат заданий шісткою об’єктів $A = <S, So, X, Y,, \delta, \lambda >$, де S – безліч станів, So – початковий стан, X – безліч

вхідних сигналів, Y – безліч вихідних сигналів; δ: S×X→S – функція переходів; λ – функція виходів. Для автомата Мілі функція λ: S×X→Y, а для автомата Мура λ: S →Y. Функції переходів і виходів зручно задавати в табличній формі, а для графічного опису автомата використовувати спрямований граф, вершини якого відповідають станам, а дуги – переходам з одного стану в інший. По автомату Мура можна побудувати еквівалентний йому автомат Мілі і навпаки. У еквівалентного автомата може бути інша кількість і семантика станів. Часто кажуть також, що у автомата Мілі дії прив'язані до переходів, а в автомата Мура – до станів. Автомати можуть бути реалізовані програмно і апаратно. Програмна реалізація автомата документується у вигляді блок-схеми алгоритму (БСА), апаратна – у вигляді принципової схеми пристрою. Оскільки контролерні системи часто проектують для заміни існуючих апаратних і програмних автоматів, то представляють інтерес методи конвертації цих документів в графи автоматів і навпаки.

Процедура побудови графа автомата згідно з графічною схемою алгоритму (ГСА) включає розмітку ГСА для визначення набору станів автомата і знаходження шляхів на ГСА для визначення переходів автомата. Побудова автоматів Мілі та Мура відрізняється правилами розмітки ГСА. Для автомата Мілі правила розмітки наступні:

1. Символом S_0 відзначається вхід вершини, наступної за початковою, а також вхід кінцевої вершини;
2. Причому різним вершинам привласнюють різні символи;
3. Якщо вхід вершини вказується, то тільки одним символом.

Для автомата Мура правила розмітки ГСА полягають в наступному:

1. Символом S_0 відзначається початкова та кінцева вершини ГСА;
2. Кожна операторна вершина відзначається єдиним і відмінним від інших символом S_i;
3. Кожному символу, отриманому в результаті розмітки, ставиться у відповідність стан автомата, причому символу S_0 відповідає початковий стан.

Нехай по ГСА від позначки S_i до позначки S_j відповідає переходу автомата зі стану S_i в стан S_j. Для автомата Мілі цей шлях позначають:

$$S_i x(S_i, S_j) y(S_i, S_j) S_j, \tag{5.2}$$

де x (S_i, S_j) – кон'юнкція логічних умов x_k, відповідних умовним вершинам ГСА на шляху від S_i до S_j; у (S_i, S_j) – вміст операторної вершини, на ГСА між відмітками S_i і S_j.

Якщо шлях проходить через вихід «так» («1», «істина») умовної вершини ГСА, то умову беруть без інверсії, а якщо через вихід «ні» («0», «брехня») – то умова включається в кон'юнкцію з інверсією. Шляхи, в яких деяка умова включається як у прямій, так і в інверсній формі, виключаються з опису переходів автомата. Шляху, що не містить умовних вершин, відповідає безумовний перехід в графі автомата, а шляху, що не містить операторної вершини – порожній оператор.

Для автомата Мура шлях від розмітки Si до розмітки Sj позначають:

$$S_j x(S_i, S_j) S_j \tag{5.3}$$

а вихідний сигнал уі в кожному стані відповідає вмісту операторної вершини ГСА, зазначеної як S_i.

5.6 Інструменти для роботи з віддаленою лабораторією

Гібридна лабораторія GOLDI (Grid of Online Lab Devices Ilmenau) розроблена департаментом інтегрованих комунікаційних систем (Department of Integrated Communication Systems) в університеті Ільменау (Ilmenau University of Technology). Вона надає набір інструментів, що підтримують всі етапи проектування для складних завдань управління (наприклад, в області техніки керування, робототехніки, телемеханіки). Мета системи GOLDI – показати сучасні способи і проблеми дистанційного контролю та дистанційного спостереження реальних процесів, з комплексним та інтерактивним використанням сучасних інтернет і інтранет технологій, таких як HTML5, JavaScript, і т.д. Вона пропонує різні функції, такі як візуалізація та анімація, яка дозволяє спостерігати і перевірити всі властивості конструкції. Зв'язок з формальними методами проектування, моделювання і прототипування використовується для створення основи для розвитку надійної конструкції системи. Структура лабораторії приведена на рис. 5.7.

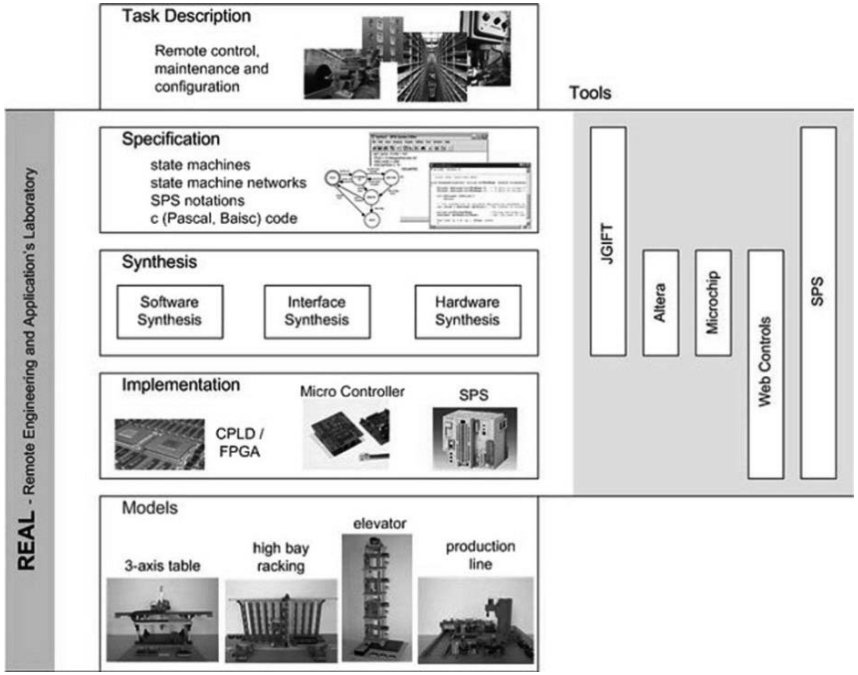


Рисунок 5.7. Структура лабораторії GOLDI

Для використання лабораторії GOLDI необхідно мати підключений до мережі Internet персональний комп'ютер із встановленими на ньому програмами браузера і Java – машини. Для входу в лабораторію слід використати посилання http://www.tu-ilmenau.de/GOLDI, або інше посилання, яке вкаже лектор. Після реєстрації можливо обрати зручну мову спілкування. Далі на домашній сторінці проглянути презентацію лабораторії, а на сторінці «Фізична система» ознайомитись з описом моделей та переліком їх давачів та виконавчих механізмів. На сторінці «Стартовий експеримент» обрати блок керування «Кінцевий автомат» та одну з запропонованих фізичних систем. Якщо експеримент не доступний, обрати віртуальний режим роботи та почати експеримент.

У вікні експерименту в блоці керування можливо завантажити свій файл або демонстраційний приклад, проглянути опис входів та виходів моделі. У блоці керування потоком виконується старт/зупинка процесу виконання програми керування.

У основному вікні представлена анімація роботи віртуальної моделі об'єкту керування та є можливість коригування виразів файлу з програмою керування, які визначають поведінку виходів та машин моделі. Приклад екрану для проектування системи керування конвеєром наведено на рис. 5.8.

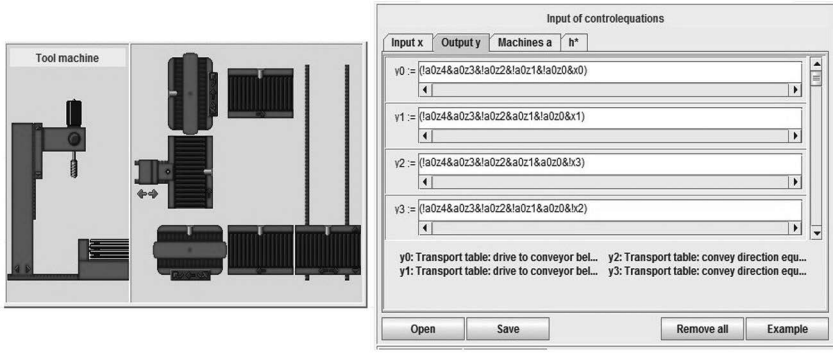


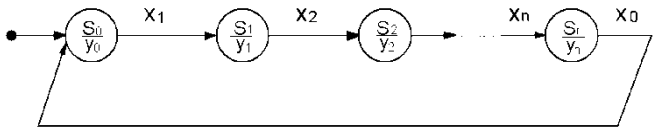
Рисунок 5.8. Приклад екрану

Приклад проекту можна зберегти на своєму комп'ютері і переглянути за допомогою стандартної програми Блокнот. У опису FSM-моделі використані наступні позначення: x – вхідна змінна, y – вихідна змінна, z – стан, a – автомат; # – логічна операція OR (АБО), & – логічна операція AND (І),! – логічна операція NOT (НІ) .

6 ТИПОВІ ЗАВДАННЯ КЕРУВАННЯ

6.1 Реалізація моделей скінчених автоматів мовою дробинних діаграм

Розглянемо особливості FSM-моделювання систем керування мовою LD. Особливістю автомата з фіксованою зміною станів є наявність в ньому тільки одного шляху з початкової вершини до кінцевої. Граф такого автомата наведено на рис. 6.1.



S_i - стан ($i = \overline{0,n}$); y_i – вихід у стан S_i; x_i - умова переходу з стану S_{i-1} в стан S_i.

Рисунок 6.1. Граф автомата з фіксованою зміною станів

Перехід X_i із стану S_{i-1} в S_i може здійснитися за подією і (або) за часом. У програмі автомата, як правило, можна виділити ранги установки початкового стану автомата, зміни станів з приходом нової вхідної події та формування вихідних сигналів в кожному стані розглянуті далі..

На рис. 6.2. представлена програма, що реалізує автомат зі зміною станів за подією за допомогою інструкцій OTU, OTL, XIC і XIO. На програмних аналогах лічильників – інструкціях CTU, CTD також можна побудувати програмну реалізацію автомата, наприклад як показано на рис. 6.3 для графа автомата, зображеного на рис. 6.1.

Будемо вважати, що стан цього автомата кодується значенням акумулятора лічильника CounT1. На рис. 6.3 не показаний ранг установки автомата в початковий стан при першому прогоні програми. Умовна частина першого рангу програми рис. 6.3 містить n – гілок (по числу станів автомата і лічильника на інструкції CTU). Кожна гілка визначає умови збільшення стану лічильника CounT1, а саме поточний стан лічильника і відповідний вхідний сигнал. Ранг з інструкцією RES скидає акумулятор лічильника в нуль, тобто переводить автомат в нульовий стан. Наступні ранги з інструкціями EQU і OTE формують виходи автомата в кожному стані.

На відміну від автоматів з фіксованою зміною станів, автомати з довільною зміною мають більш ніж один шлях від початкової вершини до кінцевої, і, отже, в загальному вигляді, вершина графа такого автомата може бути інцидентна більш ніж одній вхідній та (або) вихідній дузі. Крім того, істотним стає тип автомата – Милі або Мура.

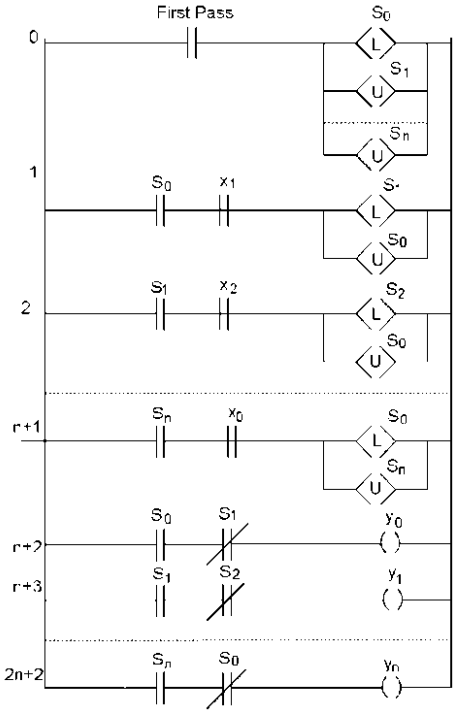


Рисунок 6.2. Програма автомата – аналога цифрового вузла із запам'ятовуванням станів в RS-тригерах

Процес програмування автомата з довільною зміною станів містить ті ж етапи, що для автомата з фіксованою зміною станів: визначення елементів пам'яті для зберігання станів; програмування переходів з одного стану в інший, включаючи перехід в початковий стан при ініціації автомата; програмування дій (виходів) у кожному стані.

Таким чином, для програмування, що містить N станів і M переходів буде потрібно $(N + M + 1)$ рангів. Кількість рангів можна зменшити, якщо об'єднати в одному ранзі переходи (дуги) інцидентні одному стану (вершині) автомата і діям в цьому стані. Два підходи до такого об'єднання: «для всіх вхідних» і «для всіх вихідних» з S_i – й вершин дуг автоматів Мілі та Мура наведено на рис. 6.4–6.7.

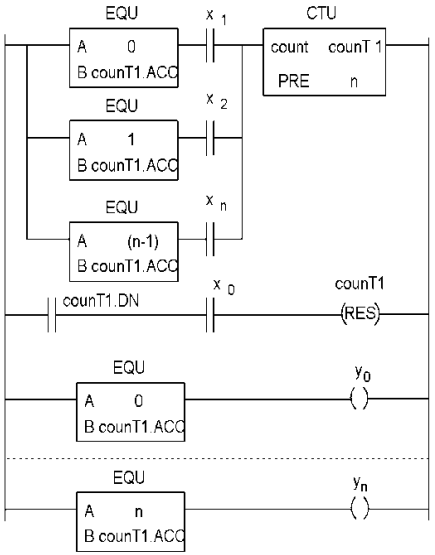


Рисунок 6.3. Програма автомата із запам'ятовуванням станів в акумуляторі інструкції лічильника

Програма, що реалізує автомат Мура за принципом «всі вихідні», має N рангів (по числу станів автомата). Кожен ранг (рис. 6.4б) описує вихідні переходи одного S_i – го стану (рис. 6.4а) і значення виходу y_i автомата в цьому стані.

Як видно з рис. 6.4–6.7, програми автомата Мілі вимагають запам'ятовування не лише нового стану, але і значення виходу в новому стані, а, отже, скидання попереднього стану і попереднього виходу. Наприклад, якщо автомат Мілі на рис. 6.6 знаходиться в стані S_i, то активними можуть бути один з виходів y_{01}, y_{02}, y_{0m} і їх активність необхідно скидати при будь-якому переході зі стану Si: в S_1 по X_{i1}; в S_2 по X_{i2}; … В S_n по X_{in}. Якщо автомат Мілі на рис. 6.7 знаходиться в стані Si, то встановлюється активність одного з виходів y_{i1}, … y_{in} і для кожного з цих виходів потрібно скинути активність попереднього стану.. Наприклад, якщо в стан S_i автомат перейшов по входу X_{i2} зі стану S_2, то встановлюється активність виходу y_{i2}, а скидаються активності виходів y_{21}, … y_{2m}.

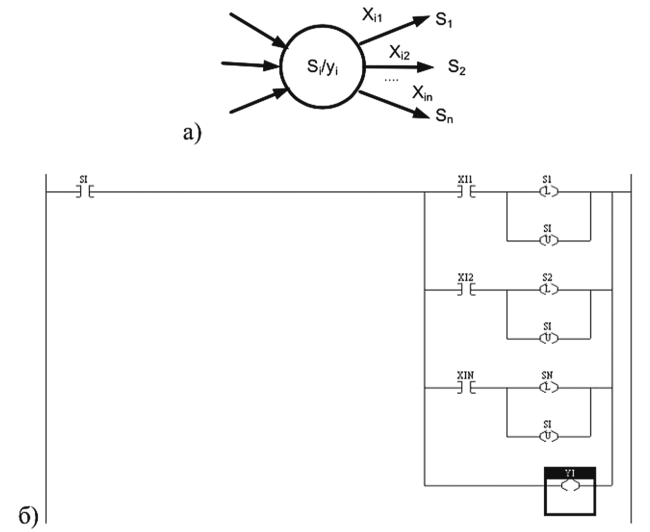


Рисунок 6.4. Програмування автомата Мура за принципом «всі вихідні»: а – фрагмент графа з вихідними дугами, інцидентними стану S_i; б – фрагмент програми

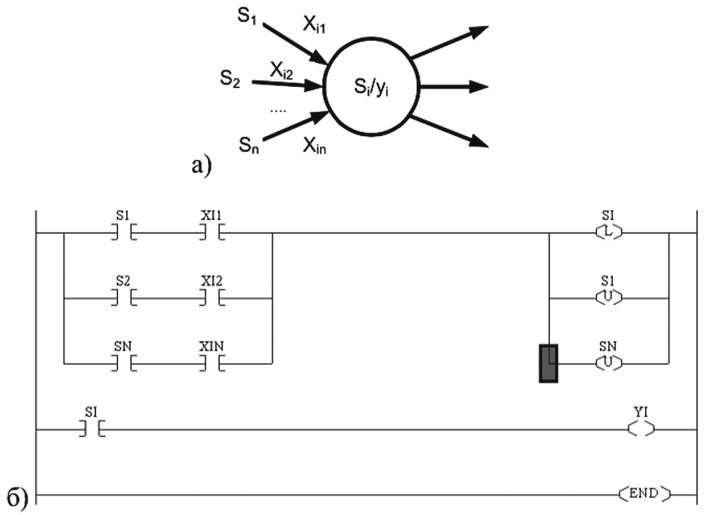


Рисунок 6.5. Програмування автомата Мура за принципом «всі вхідні»: а – фрагмент графа (вхідні дуги інцидентні стану S_i); б – фрагмент програми

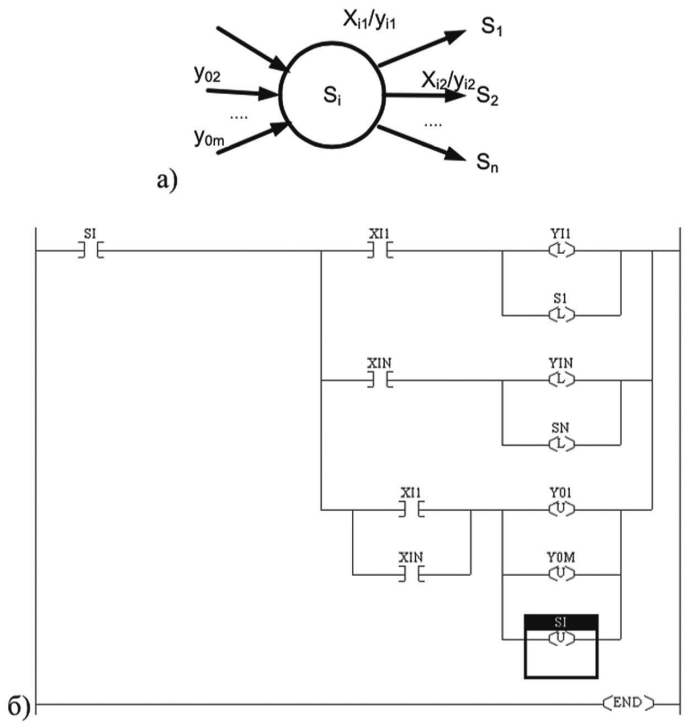


Рисунок 6.6. Програмування автомата Мілі за принципом «всі вихідні»: а – фрагмент графа з вихідними дугами інцидентними стану S_i; б – фрагмент програми

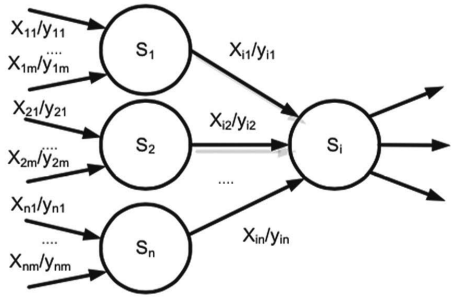


Рисунок 6.7. Програмування автомата Мілі за принципом «всі вхідні»: а – фрагмент графа з вхідними дугами, інцидентними станом S_i; б – фрагмент програми

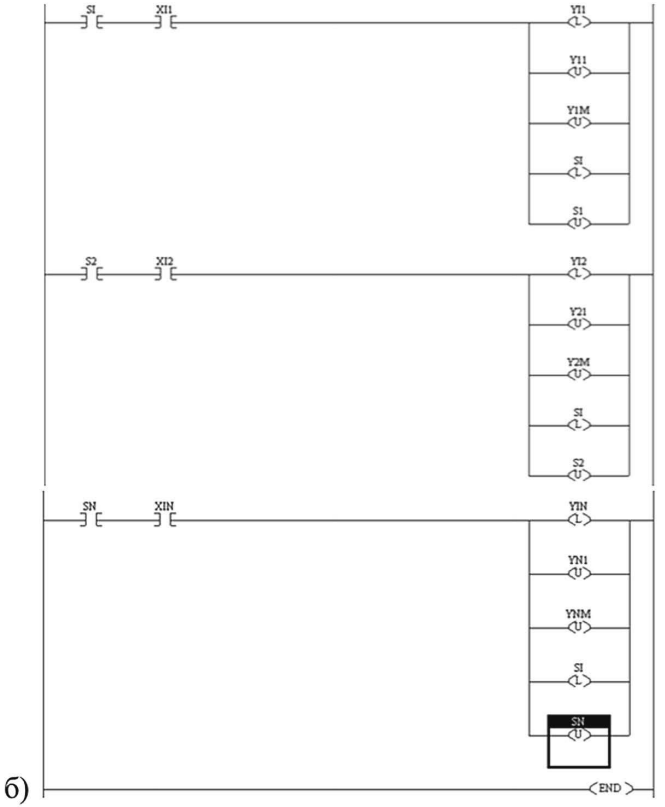


Рисунок 6.7 (продовження)

Як приклад, на рис. 6.8 наведено граф і програма автомата Мура сконструйована за принципом «всі вихідні»

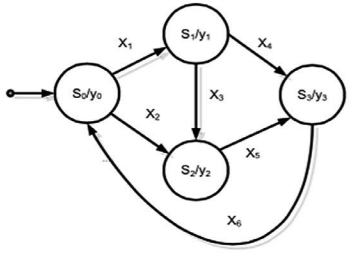
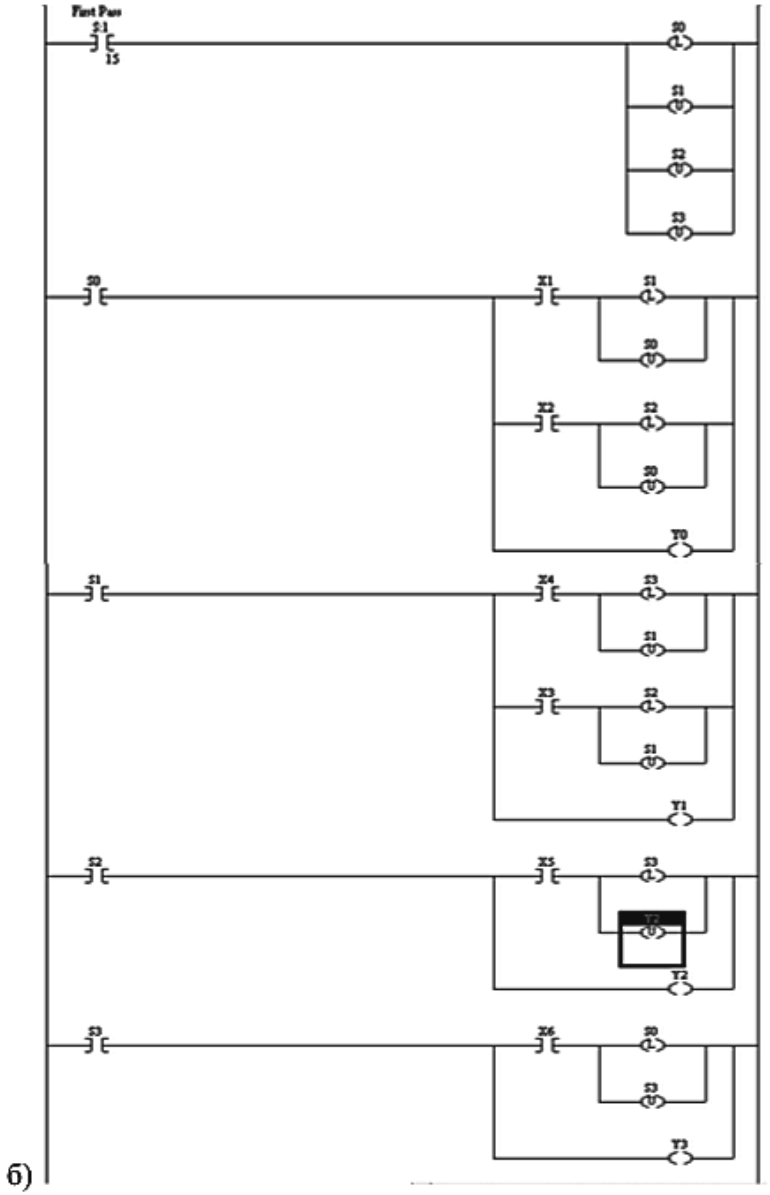


Рисунок 6.8. Приклад програмування автомата Мура за принципом «всі вихідні»: а – граф автомата; б – програма



б)

Рисунок 6.8 (продовження)

Розглянуті вище конструкції автоматів не є єдиними. Тут можуть бути використані й інші рішення, розглянуті при побудові автоматів з фіксованою зміною станів та в джерелах [14, 16,17] .

6.2 Типові завдання керування

До типових завдань контролю віднесемо програмування логічних функцій, програмування генераторів впливів.

В результаті програмування логічних функцій складається один або кілька рангів програми з інструкціями XIC, XIO, OTE, операндами яких є теги бітів вхідних змінних і функцій. Якщо істинність логічної функції дозволяє виконання інших вихідних інструкцій, то ці інструкції заносяться в ранг додатково або замість інструкції OTE.

Нехай вихідна формула логічної функції задана в диз'юнктивній нормальній формі (ДНФ), де кожен терм k_i (або кон'юнктив) являє собою кон'юнкцію взятих із запереченням або без двійкових змінних функції. Тоді терми $k_1,...,k_n$ представляються в ранзі програми паралельними вхідними гілками, а функція – вихідною гілкою. Ранг, який представляє терм, містить ланцюг послідовно з'єднаних інструкцій XIC і (або) XIO, операндами яких є теги двійкових змінних функції. У вихідний гілки розміщується інструкція OTE, операндом якої є тег функції. Допускається в гілці терма поміщати тільки одну інструкцію XIC (XIO), операндом якої є тег проміжної змінної, що представляє істинність терма. Обчислення цієї проміжної змінної проводиться в окремому ранзі. Приклад програмування логічної функції:

$$y = \overline{\left(x_1 \wedge \bar{x}_2\right) \vee \left(x_3 \wedge x_4\right)}, \tag{6.1}$$

з використанням проміжних змінних y_1 і y_2 наведено на рис. 6.9.

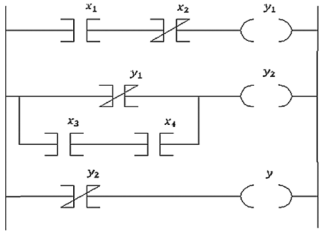


Рисунок 6.9. Програмування логічної функції в кон'юнктивній нормальній формі (КНФ) з використанням проміжних змінних

Нехай логічна функція задана в КНФ, де кожен терм являє собою диз'юнкцію, взятих з запереченнями або без, двійкових змінних функції. Тоді, терми $D_1 \ldots D_n$ представляються в ранзі програми послідовними вхідними гілками, а функція – вихідною гілкою. Розгалудження – частина рангу, яка, яка представляє терм, в свою чергу містить паралельні гілки з інструкціями XIC і (або) XIO, операндами яких є теги довічних змінних функції. У вихідний гілки розміщується інструкція OTE. Приклад програмування логічної функції наведено на рис. 6.10.

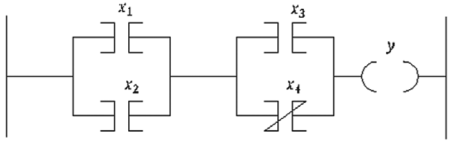


Рисунок 6.10. Програмування логічної функції представленої в КНФ

Під генераторами стимулюючих впливів (ГСВ) будемо розуміти вихідні ОА, рівень сигналу на виході яких змінюється протягом активності ОА. Виходи ГСВ з'єднані з входами об'єкту керування. Формовані ними сигнали змінюють стан об'єкта керування, що фіксується шляхом аналізу виходів ОУ. Крім того, виходи ГСВ можуть підключатися до входів вхідних ОА, моделюючи зміну збурюючих впливів або вихідних сигналів об'єкта керування. На рис. 6.11 зображена програма генератора нескінченної послідовності імпульсів прямокутної форми з прогальністю два.

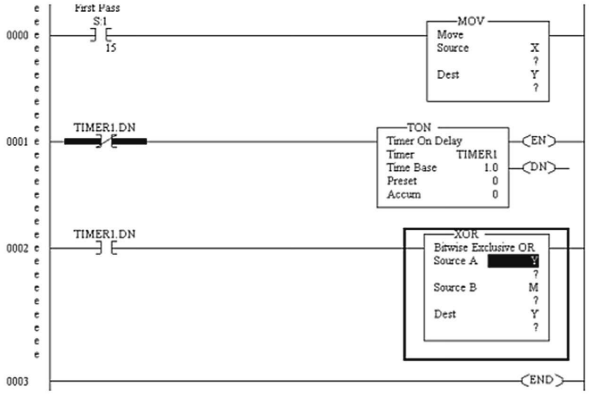


Рисунок 6.11. Програма генератора послідовності імпульсів з прогальністю два

У ранзі 0, при першому прогоні програми, на виходах генератора встановлюються початкові значення розрядів коду X. Ранг 1 формує часовий інтервал тривалістю TIMEBASE * PRE, протягом якого на виходах Y зберігаються поточні значення. Назвемо цей інтервал фаза 1. У ранзі 2 ці значення змінюються на протилежні для тих розрядів Y_i, для яких значення розряду – маски M_i дорівнює одиниці. У наступному скані програми в ранзі 1 починається відлік фази 2. Після завершення фази 2, знову змінюються на протилежні значення розрядів виходів Y.

На рис. 6.12 зображена програма генератора сигналів пилкоподібної (лінійно зростаючої) форми. У цій програмі значення акумулятора інструкції лічильника COUNT1.ACC змінюється від 0 до COUNT1.PRE, збільшуючись на одиницю по закінченні інтервалу часу TIME BASE * PRE заданого операндами інструкції таймера TIMER1. Потім акумулятор лічильника обнуляється, і цикл формування пилкоподібної напруги повторюється.

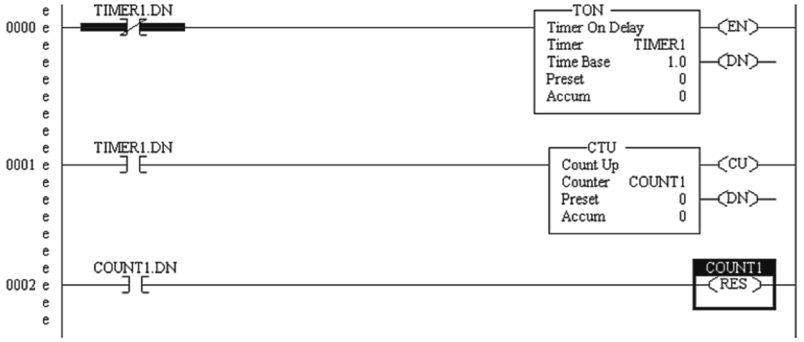


Рисунок 6.12. Програма генератора сигналів пилкоподібної (лінійно зростаючої) форми

На рис. 6.13 зображена програма генератора сигналів трикутної форми з максимальним (N1) і мінімальним (N2) значенням вихідного сигналу.

Рис. 6.14 ілюструє спосіб побудови генератора тригонометричної функції. Нехай потрібно сформувати кусково-лінійну апроксимацію синусоїдального сигналу

$$Y = A \times \sin(wt) \qquad (6.2)$$

Вихідними даними для завдання операндів інструкцій у програмі генератора є період T функції і кількість лінійних ділянок апроксимованої функції за її період. Звідки

$$w = 2\pi / t, \tag{6.3}$$

а тривалість лінійної ділянки

$$\Delta t = T / n \tag{6.4}$$

Тривалість Δt задається в програмі як добуток значень операндів TIME BASE та PRE інструкції таймера TON.

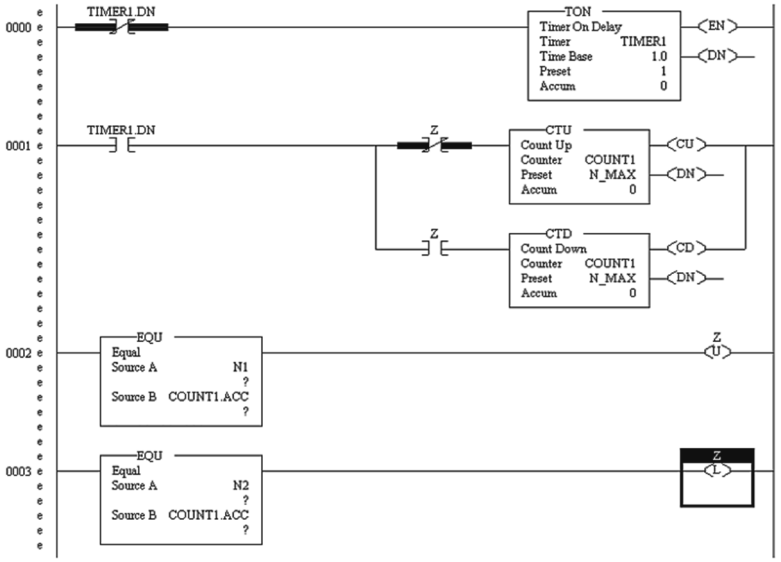


Рисунок 6.13. Програма генератора сигналів трикутної форми

Якщо довільна функція задана таблично, то для побудови генератора значень функції зручно використовувати інструкцію секвенсера SQO, як показано на рис. 6.15. В цій програмі таймер TIMER1 задає тривалість інтервалу лінійної ділянки апроксимованої функції. По закінченні цього інтервалу на вихід надходить нове значення функції, взяте з чергового рядка таблиці TABL.

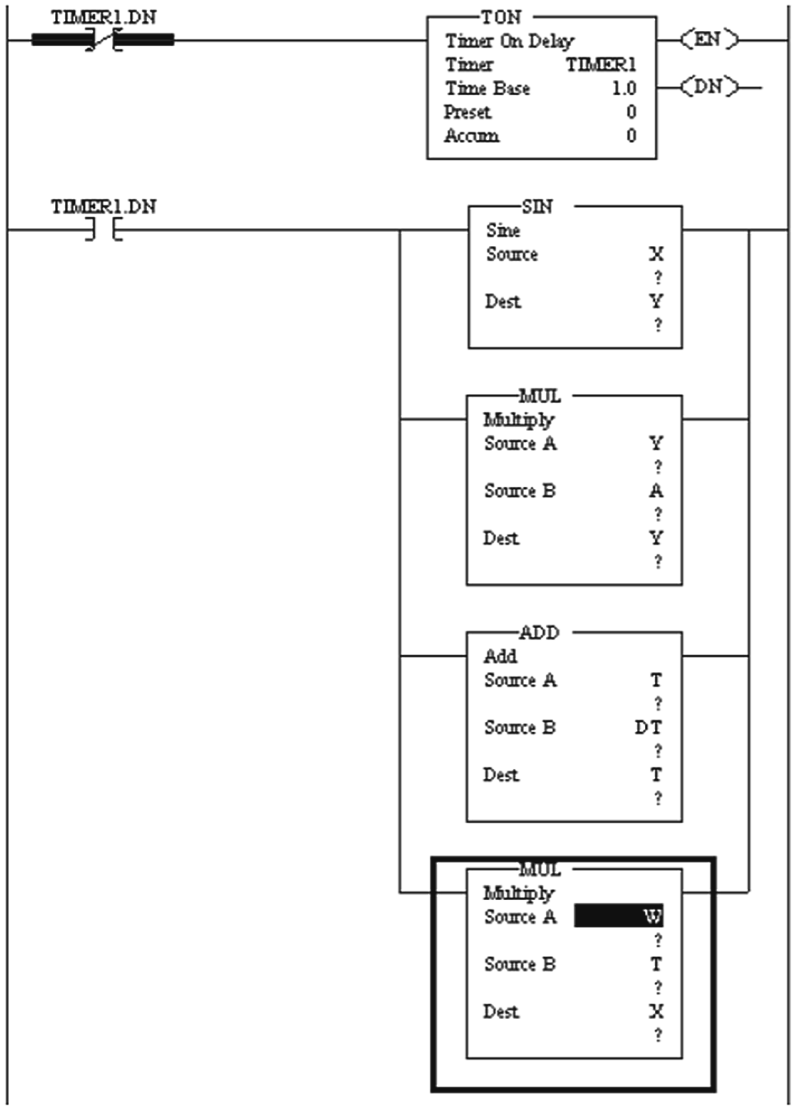


Рисунок 6.14. Генератор синусоїдальної функції

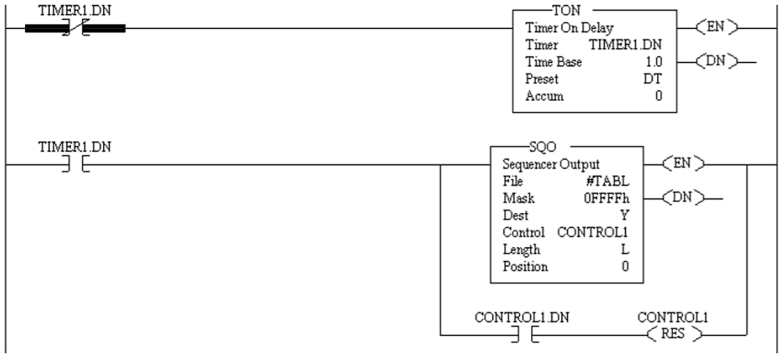


Рисунок 6.15. Табличні обчислення довільної функції

Після «прочитання» всієї таблиці проводиться скидання структури керування інструкції SQO і починається новий період формування значень функції.

6.3 Програмне забезпечення людино-машинного інтерфейсу

Характерними рисами сучасних контролерних систем керування є багаторівнева структура, розподіленість, участь людини у процесі керування, програмна реалізація алгоритмів керування. Такі системи існують у галузях виробництва, передачі та розподілу електричної енергії, видобування, транспортуванні та розподілі нафти та газу, керування на транспорті, комунальному господарстві тощо. На верхньому рівні систем керування, як правило, вирішують задачі людино-машинного інтерфейсу (MMI–$5an Machine Interface, HMI – Human Machine Interface), збору даних та супервізорного керування (SCADA – Supervisory Control And Data Acquisition) .

Основні структурні компоненти SCADA-систем – це віддалений термінал (RTU – Remote Terminal Unit), диспетчерський пункт керування (головний термінал) (MTU – Master Terminal Unit, MS – Master Station) та комунікаційна система (канали зв'язку) (CS – Communication system). Основним елементом програмного забезпечення MTU є додаток користувача, який створюється и виконується у середовищі пакетів програм візуалізації. Для оцінки та вибору SCADA-системи важливо знати її характеристики, які розподіляють по чотирьох категоріях: базові властивості, можливості середовища розробки, можливості конфігурації системи, характеристик супроводження/експлуатації системи. До базових властивостей належить відкритість, підтримка контролерів, реєстрація, архівація, робота з трендами, аларми, можливості HMI, WEB – технології, підтримка звітів, операційна система, масштабованість, резервування, захист доступу, вартість та інші.

SCADA-система вважається відкритою якщо для неї описані формати даних та інтерфейс для приєднання незалежно розроблених компонентів, таких як власні програмні модулі, драйвери введення-виведення, компоненти, розроблені за технологіями DDE (Dynamical Data Exchange), OLE (Object Linking and Embedding), OPC (OLE for Process Control), COM/DCOM (Component Object Model/Distributed COM), є можливості обміну даними з базами даних реального часу різних рівнів та ASCII – експорту (імпорту) конфігураційних параметрів проектів користувача.

Реєстрація даних – це процес у якому відбувається відбір у даних за номенклатурою та часом надходження, їх структурування та організація їх зберігання на комп'ютері з метою постобробки. Архівація – це спосіб зменшення об'єму пам'яті для зберігання даних.

Аларми (від англійського «alarm» – тривога) – це повідомлення підсистеми тривожної сигналізації. Засоби конфігурації алармів відрізняються можливостями присвоєння пріоритетів, групування, онлайнової фільтрації, маскування, структурою повідомлень оператору та іншими.

Тренди (від англійського «trend» – тенденція, направлення) – це засоби візуалізації залежностей даних, як правило від часу, рідше від іншої змінної (так звана XY-залежність). Тренди підрозділяються на тренди реального часу та історичні. До характеристик цих засобів належить кількість кривих у тренді, можливість оперативної зміну масштабу вікна, формування тренду з використанням ActiveX-об'єкту та інші.

Можливості HMI SCADA-системи характеризуються наявністю об'єктно-орієнтованого графічного редактору, бібліотек стандартних графічних символів, технікою «drug-and-drop», засобів навігації

по окремим екранам, типами анімації графічних об'єктів, мультимедійними засобами та інше.

WEB-технології, що використовуються у SCADA-системах дозволяють виконувати моніторинг даних та керування об'єктом з віддаленого комп'ютеру, який приєднаний до системи керування через комп'ютерну мережу.

Підтримка звітів, у першу чергу, – це можливість друкування та архівації звітів, які, в загальному вигляді, містять первинні дані, що отримані від об'єкту керування, результати їх обробки, зведення подій, алармів та діяльності у системі керування. Масштабованість припускає можливість існування декількох реалізацій однієї SCADA-системи, які відрізняються кількістю тегів у базі даних реального часу або кількістю вузлів у системі керування.

Резервування – це спосіб забезпечення працездатності системи керування в умовах ймовірних відмов її елементів. Що до SCADA-систем – це вузли та їх елементи (наприклад процесор, джерело живлення, накопичувач на магнітних дисках тощо), канали обміну даними. Захист доступу – це система заходів, що обмежують несанкціонований доступ до даних і засобів конфігурування додатку користувача. Ці заходи включають паролі та рівні доступу до даних та деяких видів діяльності. Серед інших властивостей можна відзначити наявність опцій статистичного керування, підтримки «рецептів», планування заданій, засобів нечіткої логіки та автоматичної обробки подій.

Розробка додатку користувача виконується у пакеті MMI за допомогою так званого менеджеру проекту, із якого викликаються проблемно-орієнтовані редактори та вже створені частини проекту. До типових проблемно-орієнтованих редакторів належать редактори каналів зв'язку з контролерами, вузлів у мережах, редактори тегів, подій, прав користувача, властивостей підсистем реєстрації даних та діяльності тощо. Результатом проектування є графічні, логічні та тегові моделі задачі керування. Графічна модель – це система графічних екранів та засобів навігації між ними. Графічний екран містить графічні об'єкти які мають поведінку. Вони змінюють свою форму, положення на екрані, колір та інші атрибути у разі зміни значення керуючого тега. Логічна модель пов'язує певні умови у системі з певними діями. Тегові моделі описують для кожного тега, який використовується в процесі керування об'єктом, його тип, джерело, діапазон допустимих значень та інші.

6.4 Поведінковий синтез та скриптинг завдань візуалізації

Поведінковий синтез завдань керування виконується за допомогою редактору подій. Кожна подія містить умовну та виконавчу частини, а їх сукупність задає скінчений автомат, який реалізує частину поведінки системі керування, що не була реалізована у середовищі промислових контролерів. Умовна частина події – це вираз, елементами якого теги вхідних даних системи. Виконавча частина містить множину команд пакету.

Команди пакету – це підпрограми, що виконуються як елемент реакції на певні події, що виникають при виконанні додатку користувача або складова частина макросу чи програми на мові програмування. У вигляді команд реалізовані дії, що пропонуються пунктами функціонального меню вікна програми SCADA-пакету.

Для розширення функціональних можливостей SCADA-пакету використовуються макроси та скриптинг. Макроси – це списки команд, які автоматично виконуються у порядку їх розташування у макросі. Команда «виконати макрос <ім'я макросу>» дозволяє включати до складу певного макросу інші макроси.

Скриптинг складається з об'єктної моделі додатку та мови програмування. Найбільш поширені мови – це Microsoft Visual Basic для додатків (VBA) та C. Приклади використання скриптингу: розробка сценаріїв (підпрограм VBA або C, що виконують умовні переходи); прив'язка даних додатку візуалізації до інших додатків, таких як Microsoft Access або Microsoft SQL Server; керування пакетом візуалізації з підпрограми VBA шляхом видачі команд пакету візуалізації.

ЛІТЕРАТУРА ДО ЧАСТИНИ 2

1. Рассальский А. Н. Система мониторинга и управления силовых трансформаторов.– Електротехніка і Електромеханіка. 2005, № 2, с.54–58.
2. Мишель Ж., Лоржо К., Эспьо Б., пер. с франц. А. П. Сизова – М.: Машиностроение, 1986 – 176 с., ил.
3. Parr E. A. Programmable Controllers. An engineer's guite. Third edition. Oxford: Newnes, 2003, 429 p.
4. Lewis R. W. Programming industrial control systems using IEC1131–3/Revised ed. The Institution of Electrical Engineers. London, United Kindom, 1998, 344p.
5. John K.– H., Tiegelkamp, M. IEC 61131–3: Programming Industrial Automation Systems. Concepts and Programming Languages, Requirements for Programming Systems, Aids to Decision-Making Tools.– Шпрингер. 2001 VI, 376 pp., 139 figs.
6. Петров И. В. Программируемые контроллеры. Стандартные языки и инструменты/Под. Ред. Проф. В. П. Дьяконова.– М.: СОЛОН-Пресс, 2003, – 256с., ил.
7. Олсон Г., Пиани Д. Цифровые системы автоматизации и управления.– СПБ.: Невский Диалект, 2001. – 557с.: ил.
8. Рассальский А. Н., Поляков М. А. Опыт обучения промышленной информатике в учебно-научной лаборатории АСУ ТП Запорожского технического университета // «Новые информационные технологии в электротехническом образовании»: Сб. Трудов пятой международной научно-методической конф. Астрахань: Изд-во ЦНТЭП, 2000, с.199–201.
9. Поляков М. А., Рассальский А. Н.. Перспективы применения средств промышленной автоматизации для целей обучения // «Новые информационные технологии в региональной инфраструктуре и образовании»: Сб. Трудов четвертой международной научно-методической конф. Астрахань: Изд-во ЦНТЭП, 2001, с.123-124.
10. Рассальский А. Н., Поляков М. А. Опыт обучения контроллерным системам управления в учебно-научной лаборатории АСУ ТП Запорожского технического университета. // Вісник національного технічного університету «ХПІ», 2001, № 16, с.141.
11. Rockwell automation. User manuals/ Режим доступу: www/ URL: http://literature.rockwellautomation.com/idc/groups/public/documents/webassets/browse_category.hcst (EN) .
12. Модули аналового ввода-вывода ControlLogix. Компания Allen Bradley. Публикация 1756–6.5.9. – Ноябрь 1998.
13. Сташин В. В. и др. Проектирование цифровых устройств на однокристальных микроконтроллерах.– М.: Энергоатомиздат, 1990, 224 с.
14. Гома Х. UML проектирование систем реального времени, распределеных и параллельных приложений.– М.: ДМК, 2002–704 с.
15. Буч Г., Рамбо Д., Джекобсон А.,– Язык UML. Руководство пользователя. Пер. с англ.– М.: ДМК Пресс, 2001. – 432 с ил. (Серия «Для программистов») .
16. Карпов Ю. Г. Теория автоматов.– СПб.: Питер, 2002. – 224 с.
17. Hugh Jack Automating Manufacturing Systems with PLCs (version 5.0, may 4, 2007 (EN) .
18. Поляков, М. А. Теоретико-множественная модель функциональной структуры удаленной лаборатории для обучения проектированию систем управления / М. А. Поляков, Т. Ю. Ларионова // Системные технологии. Региональный межвузовский сборник научных работ. – Выпуск 3 (98). – Днепропетровск, 2015. – С. 48-56.

ЧАСТИНА 3. ЯКІСТЬ ІНФОРМАЦІЙНИХ СИСТЕМ

Г.В. Табунщик,Т.І. Каплієнко

ВСТУП ДО ЧАСТИНИ 3

На всіх етапах життєвого циклу складних технічних об'єктів питання верифікації є з найважливіших.

Сучасні інформаційні системи (ІС) це складні програмно-апаратні комплекси, що вимагають всебічну верифікацію, але існуючи підходи не орієнтуються на одночасну верифікацію як програмного так і апаратного забезпечення складної вбудованої системи.

Для автоматизації процесу верифікації інформаційних систем необхідно мати формалізовану модель, тому розділ 7 містить запропоновану авторами формалізовану модель верифікації інформаційних систем.

Ефективним інструментом верифікації інформаційних систем є тестування, що дозволяє провести всебічну верифікацію всіх частин системи де були внесені зміни. Розділ 8 містить методи тестування програмного та апаратного забезпечення вбудованих систем, зокрема авторський модифікований метод регресійного тестування, що дозволяє автоматизувати процес формування взаємозалежних компонент при повторному тестуванні інформаційної системи.

Дев'ятий розділ присвячений прикладам використання віддаленої лабораторії GOLDi при вивченні методів тестування вбудованих систем.

For all stages of lifecycle of compound technical objects the questions of verification are highly important.

Concurrent informational systems are compound systems consistes from software and hardware but existed approaches don't concider complex verification of hardware and software of the embedded systems.

For complex automatization of the informational system its important to have formalized model for objects included in the system and description of the processes of data transmition, so the chapter 7 consists developed by the authors the model of verification.

One of the most effective instruments of verification is testing, which allow to provide verification of all modificated parts of the developing system. Chapter 8 depicts software and hardware testing techniques, including modificated method for regression testing, which allow to form automatevly list of connected compotents of the system within the repeated testing of the informational system.

Chapter 9 devoted for examples of online lab GOLDi developed by Integrated Communication Systems Group at the Ilmenau University of Technology usage for the tasks of teaching of the embedded systems testing teqniques.

7 ОСНОВИ ВЕРИФІКАЦІЇ ВБУДОВАНИХ СИСТЕМ

Верифікація системи в найзагальнішому сенсі – це перевірка відповідності між вимогами до системи і властивостями працюючої системи [13].

7.1 Методи верифікації вбудованих систем

Розрізняють такі методи верифікації як експертиза, статичний аналіз, формальні методи, динамічні методи, синтетичні методи (рис. 9.1) [13]. Зазвичай серед експертиз виділяють такі: організаційні експертизи (management review), технічні експертизи (technical review), наскрізний контроль (walkthrough), інспекції (inspection) та аудити (audit).

З середини 1990-х активно розвиваються методи оцінки архітектури програмного забезпечення (ПЗ) на основі сценаріїв (scenario based software architecture evaluation), які зазвичай не співвідносяться з «традиційними» експертизами. Від інших методів верифікації експертизу відрізняє можливість виконувати її, використовуючи тільки самі артефакти життєвого циклу, а не їх моделі (як у формальних методах) або результати роботи (як в динамічних). Експертиза застосовується до будь-яких характеристик ПЗ і до будь-яких артефактів життєвого циклу на всіх етапах життєвого циклу, хоча для різних цілей можуть використовуватися різні її види. Вона дозволяє виявляти широкий спектр помилок, причому робити це на етапі проектування відповідного артефакту, тим самим мінімізуючи час існування дефекту і його наслідки для якості похідних артефактів.

У той же час експертиза не може бути автоматизована і вимагає активної участі людей.

Рисунок 9.1. Методи верифікації

Ефективність експертизи істотно залежить від досвіду та мотивації її учасників, організації процесу, а також від забезпечення коректної взаємодії між різними учасниками. Це накладає додаткові обмеження на розподіл ресурсів у проекті і може призводити до конфліктів між розробниками, якщо керівництво проекту звертає мало уваги на комунікативні аспекти проведення експертиз.

Кожний вид експертизи має свої властивості:

- технічна експертиза (*technical review*) – це систематичний аналіз артефактів проекту кваліфікованими фахівцями для оцінки їх внутрішньої узгодженості, точності, повноти, відповідності стандартам і прийнятим в організації процесів, а також відповідності один одному і загальним завданням проекту;
- наскрізний контроль (*walkthrough*) – метод експертизи, в рамках якого один з членів команди перевірки надає її учасникам послідовно всі характеристики артефакту, що перевіряється, і вони аналізують цей артефакт, ставлячи питання, вносячи зауваження, відзначаючи можливі помилки, порушення стандартів і інші дефекти;

- інспекція (*software inspection*) – це послідовне вивчення характеристик артефакту, зазвичай слідує деякому плану, з метою виявлення в ньому помилок і дефектів;
- аудит (*audit*) – аналіз артефактів і процесів життєвого циклу, що виконується людьми, які не входять до команду проекту, для оцінки відповідності цих артефактів і процесів завданням проекту, укладеним контрактом, загальним стандартам, один одному і ін.

Статичний аналіз використовується для перевірки формалізованих правил коректної побудови артефактів ІС і пошуку дефектів за визначеними шаблонами.

Такий аналіз добре автоматизується і може бути практично повністю покладений на програмний інструментарій, хоча іноді необхідно вручну визначити, наприклад, прийняті в проекті стандарти кодування. Однак він застосовується лише до коду або до певних форматів уявлення проектних артефактів ІС, і здатний виявляти тільки обмежену кількість типів помилок. Проблемою цих методів є або велика надмірність, при використанні методів строго аналізу, або вірогідність пропустити помилки, при використані методів що ще генерують повідомлення про помилку. Інструменти автоматичної верифікації на основі статичного аналізу застосовуються досить широко, оскільки не вимагають спеціальної підготовки і досить зручні у використанні. Більшість технік статичної перевірки коректності програм, які довели свою ефективність на практиці, рано чи пізно стають частиною компіляторів або навіть перетворюються в семантичні правила мов програмування.

На практиці, формальні методи верифікації набагато частіше застосовуються до апаратного забезпечення ніж до програмного. Це обумовлено більш високою вартістю помилок, більшою однорідністю його структури, більш широким багаторазовим використанням проектної інформації, а також більшою звичністю строгих обмежень і точних описів для інженерів.

Логіко-алгебраїчні моделі (*property-based models*), вони ж – логічні або алгебраїчні обчислення. При моделюванні ПЗ ІС модель такого типу описує деякий набір властивостей системи, що змінюється з часом, але не дає точного уявлення про те, за рахунок чого змінюються ці властивості.

Здійснимі моделі (або операційні, *executable models*) характери-

зуються тим, що їх можна визначеним способом виконати, щоб простежити зміну властивостей модельованого ПЗ. Кожна модель, що виконується, є, по суті, програмою для деякої досить строго визначеної віртуальної машини.

Всі види здійсненних моделей можна вважати розширенням і узагальненням моделей на базі кінцевих автоматів.

Моделі проміжного типу мають риси як логіко-алгебраїчних, так і здійсненних моделей. Частина наведених вище прикладів може по ряду властивостей бути віднесена до обох цих класів.

Методи та інструменти перевірки моделей. Перевірка моделей (*model checking*) використовується для перевірки виконання набору властивостей, записаних у вигляді тверджень визначеного логіко-алгебраіческого обчислення за здійсненною моделлю що моделює певні проектні рішення. Найчастіше для перевірки опису властивостей використовується деяка тимчасова логіка або – числення, а в якості моделі, властивості якої перевіряються, виступає кінцевий автомат, стани якого відповідають наборам значень елементарних формул у властивостях, що перевіряється, зазвичай він називається моделлю Кріпке. Перевірку моделі виконує спеціалізований інструмент, який або підтверджує, що модель дійсно володіє заданими властивостями, або видає сценарій її роботи, в кінці якого ці властивості порушуються, або не може прийти до певного вердикту, оскільки аналіз моделі вимагає занадто великих ресурсів. Властивості, що перевіряються, зазвичай поділяють на властивості безпеки (*safety properties*), які перевіряють умову, що щось небажане при будь-якому варіанті роботи системи ніколи не трапиться, і властивості живучості (*liveness properties*), які, навпаки, перевіряють умову, що щось бажане рано чи пізно відбудеться. Іноді додатково виділяють властивості стабільності (або збереження, *persistence properties*) – при будь-якому сценарії роботи системи задане твердження в деякий момент стає істинним і з тих пір залишається виконаним, і властивості відповідності (*fairness properties*) – деяке твердження при будь-якому сценарії роботи буде виконано в нескінченній множині моментів часу. Ті чи інші властивості відповідності, часто є вихідними припущеннями, при виконанні яких потрібно перевірити властивості безпеки або живучості.

Методи та інструменти перевірки узгодженості. При перевірці узгодженості аналізується відповідність між двома здійснимими моделями, одна з яких моделює артефакт, що перевіряється, зазвичай проект або реальну роботу системи (її компонента), а друга – перевіряються властивості. Перевіряються властивості в цьому випадку – це вимоги до поведінки системи або її компонента, представлені у вигляді узагальненого автомата (системи переходів, мережі Петрі та ін.), Всі сценарії роботи якого оголошуються правильними. В цьому випадку зазвичай перевіряється умова, що всі можливі сценарії поведінки реалізації можливі також і в специфікації. Іноді встановлюється їх еквівалентність, тобто додатково перевіряється, що всі сценарії поведінки специфікації є і у реалізації. Більшість методів та інструментів перевірки узгодженості використовують для цього тестування, і тому відносяться до синтетичних методів верифікації – до тестування на основі моделей.

Динамічні методи верифікації, в рамках яких аналіз і оцінка властивостей програмної системи виконуються за результатами її реальної роботи або роботи деяких її моделей і прототипів. Прикладами такого роду методів є звичайне тестування або імітаційне тестування, моніторинг. Для застосування динамічних методів необхідно мати працюючу систему, або хоча б деякі її компоненти, чи їхні прототипи, тому зазвичай вони не використовуються на перших стадіях розроблення. З допомогою цих методів можна контролювати характеристики роботи ІС в її реальному оточенні, коли інші підходи використати неможливо. Методи динамічної верифікації дозволяють виявляти тільки помилки, що проявляються при її роботі. Застосування динамічних методів верифікації зазвичай вимагає додаткової кваліфікації для створення тестів, розробки тестової системи, що дозволяє їх виконувати, або системи моніторингу, яка дозволяє контролювати певні характеристики поведінки ІС. Але системи тестування або моніторингу можуть бути зроблені один раз і використовуватися багаторазово для широких класів ІС, лише самі тести необхідно готувати заново для кожної тестуємої ІС. У той же час підготовка тестів на ранніх етапах створення ІС дозволяє виявити безліч дефектів в описі вимог і проектних документах – фактично, розробники тестів змушені в ході своєї діяльності виконувати експертизу артефактів, які є основою для тестів. Створення тестових наборів, які дозволять отримати адекватну оцінку якості складної системи, є досить трудомістким завданням. Оскільки тестування це досить трудомісткий процес, тому зазвичай використовуються

не надто надійні, але досить дешеві техніки, такі як (нестроге) вірогідне тестування, при якому тестові дані генеруються випадковим чином, або ж тестування на основі найпростіших сценаріїв використання, що перевіряють лише найбільш прості ситуації.

Відокремились динамічні методи, що використовують елементи формальних, – до них відносяться тестування на основі моделей (*model-based testing, model driven testing*) і моніторинг формальних властивостей (*runtime verification, passive testing*). Ряд інструментів побудови тестів істотно використовує як формалізацію деяких властивостей ПЗ, так і статичний аналіз коду. Загальна ідея таких методів міститься в наступному – поєднувати переваги основних підходів до верифікації для балансування їх недоліків.

Тестування на основі моделей (*model based testing*) використовує для побудови тестів формальні моделі вимог до ІС. Як критерії повноти тестування, так інші критерії будуються на основі інформації, що міститься в цих моделях. Отримані в результаті тести зазвичай слабко пов'язані зі специфічними особливостями коду тестованої системи, але містять репрезентативну вибірку ситуацій з точки зору вихідної моделі.

В даний час методи тестування на основі моделей використовують такі типи моделей і технік як методи перевірки узгодженості автоматів і систем переходів. Такі методи відносяться до одного з трьох типів, залежно від моделей, що використовуються.

Методи побудови тестів на основі кінцевих автоматів найбільш глибоко розроблені, відомі їхні точні обмеження і гарантії повноти виконуваних перевірок. Методи, що використовують розширені автомати, зводять їх до звичайних, але застосовують більш детальні критерії покриття, засновані на використанні даних в розширених автоматах.

Системи переходів. Такі методи частіше використовуються при тестуванні розподілених систем, оскільки моделювання таких систем за допомогою кінцевих автоматів є дуже трудомістким. Більшість цих методів не визначають практично застосовних критеріїв повноти і не дають кінцевих тестових наборів для реальних систем, тому інструменти, що їх використовують, спираються на певні евристики для забезпечення завершеності набору тестів.

Методи побудови тестів на основі формального аналізу властивостей ПЗ використовують формальний аналіз для класифікації тестових ситуацій і націленої генерації тестів.

В рамках методів на основі перевірки моделей тестові ситуації вибираються як представники класів еквівалентності, що задаються критерієм покриття. Відповідна ситуації тестова послідовність будується як контр приклад при перевірці моделі (model checking) на виконання властивості, що є запереченням умови досягнення цієї ситуації.

Методи на основі дедуктивного аналізу. Обирають тестові ситуації, що відповідають особливим випадкам в дедуктивному аналізі властивостей тестованої системи.

7.2 Формалізована модель верифікації інформаційних систем

Для автоматизації комбінованих методів верифікації ІС авторами запропонована формалізована модель верифікації [19], що дозволяє визначити основні концепти системи.

Модель домену в запропонованій моделі визначає концептуальну основу та семантику змісту інформаційної системи. Модель домену – це набір

$$DM = (C, D, RC, RD),$$

де $C = CC \cup CA$ – набір понять (композиційних та атомарних), D – набір екземплярів концепції домену, RC – набір відносин між концепціями, RD – набір відносин між екземплярами концепції.

Елемент ІС *ed* – це будь-який ресурс, доступний по мережі і ідентифікований унікальним URL. ed визначається функцією:

$$rd: CA \rightarrow Ed | \forall c \in CA: ed = rd(c), ed \in Ed,$$

де CA – набір атомарних концепцій, Ed – набір ресурсів програмного комплексу (ПК) ІС, тобто сукупність інтегрованих програмно-апаратних та технічних засобів, а також інформації, призначеної для публікації в мережі Інтернет, та яка відображається в певній текстовій, графічній або звуковій формах.

Модель вимог користувача має спеціальну архітектуру, яка була складена з вимог до типів збереженої інформації та необхідних методів її обробки. Розділимо дані користувача на дві частини – профіль користувача і неявні вимоги користувача.

Явні вимоги (профіль) користувача *u* – набір *UP (u) = (A, V, rp)*

$$rp: A \to V \mid \forall a \in A: rp\ (a) \in Va,$$

де *A* – набір атрибутів, визначених в якості словника характеристик користувача, $V = \bigcup_{a \in A} V_a$ – набір значень атрибута і *Va* – діапазон атрибуту а.

Неявні вимоги користувача в даній термінології позначають модель, що містить неявні характеристики користувача. У той час як явні характеристики встановлюються самим користувачем, неявні характеристики визначаються адаптивною системою. Система збирає різні значення, які стосуються конкретного об'єкта домена.

Неявні вимоги користувача *u* – це набір *UM (u) = (D, A, V, rm)*

$$rm: D \times A \to V \mid \forall a \in D \times A: rm\ (a) \in Va,$$

де *D* – набір екземплярів концепції домена;

A – набір атрибутів, визначених в якості словника неявних користувальницьких характеристик;

$V = \bigcup_{a \in A} V_a$ – набір значень атрибута;

Va – діапазон атрибуту а.

Сукупність явних *UP (u)* та неявних *UM (u)* вимог користувача, виражених у вигляді множини елементів прототипу інтерфейсу, являють собою початкову версію ІС, яка використовується для демонстрації концепцій, закладених в системі, перевірки варіантів вимог, а також пошуку проблем:

$$interprototype = (A, D, V, rp, rm). \qquad (7.1)$$

Адаптивна функція *fa* є перетворенням між елементами ІС та вимог користувача, вираженими у вигляді прототипу. Елемент ІС, наприклад може вважатися частиною коду, який в свою чергу є частиною підсистеми:

$$fa: ea \to e,$$

де *fa AF*,

ea – елементарна одиниця interprototype ПК ІС, *ea {rp (a), rm (a)};*

e= {ed} – підмножина ресурсів ПК ІС, $e \subset E_d$, *ed* – елемент ІС, *Ed* – набір web-ресурсів.

Ефективність функціонування ІС може бути оцінена такими найбільш важливими критеріями як відповідність вимогам користувача та достовірність інформації, тому в формальному вигляді постановка задачі оцінки ефективності може бути представлена наступним чином:

$$\begin{cases} interprototype \to E_d \\ DB \subset (DB_1 \cap DB_2 \cap ... \cap DB_N), \end{cases} \qquad (7.2)$$

де interprototype – об'єднана множина явних та неявних вимог користувача до функціональності ПК ІС;

E_d = *{ed}* – існуюча функціональність ПК ІС, виражена множиною елементів ІС;

$DB_1, DB_2, ..., DB_N$ – сховища даних ІС у разі використання поширення даних або федеративного підходу до інтеграції даних (дані підходи частіше використовуються у зв'язку з можливістю отримувати найбільш актуальну інформацію, без затримок в оновленні, як у випадку з консолідацією даних). Кожна *DB* містить в собі множину D.

Розроблену модель будемо використовувати для наступних завдань:

- враховуючи те, що вимоги користувача найнаочніше можуть бути представлені у вигляді прототипу користувача інтерфейсу, виникає необхідність у засобах порівняння розроблених функціональних елементів ІС та елементів прототипу:

$$interprototype \equiv E_d.$$

- для обліку невизначеності розроблюваної ІС та висунутих до неї вимог необхідні методи визначення ризиків розроблення ІС, а також методів спрямованих на їх зниження з урахуванням специфіки розробляється ІС та моделі її розроблення;

- для контролю відповідності даних, що зберігаються в кількох БД, необхідний організований процес тестування інтегрованих БД, що дозволяє забезпечити достовірність та актуальність даних при мінімальних тимчасових витратах.

Будемо розглядати $f = \bigcup_{d \in D} \{e_d, p\}$, $e_d \in E_d$, $f \in F$, p – варіант тестування для f, набір з одного або декількох елементів ІС, як «функціональну одиницю системи» – елементарну структурну складову ІС, що реалізує закінчений функціональний блок, для перевірки якого може бути розроблений один або більше автоматизованих або автоматичних перевірочних тестів:

$$F = (E_d, P, r)\mid_{\forall\ e_d},\ e_d \in Ed, \qquad (7.2)$$
$$\text{rf: ed} \to \text{p}.$$

Функціональну одиницю (ФО) розробки необхідно вибирати залежно від виду ІС. Наприклад, якщо розробляється ІС вимагає зв'язку з БД або є об'єднанням різних апаратних платформ, то ФЕ це функція обміну інформацією між вказаними компонентами.

На відміну від функціональної точки використання при оцінюванні поняття ФО дає можливість використовувати більш узагальнений функціональний блок, характерний для конкретного типу ІС, що дозволяє спростити процедуру планування та тестування цих елементів.

Враховуючи зазначені переваги використання ФО і відсутність її в існуючих моделях, модель верифікації MV отримає вигляд:

$$MV = (interprototype, Ed, DB, rmv)$$

$$rmv: interprototype \rightarrow Ed \times DB \mid \forall\ ea, ea \in \{rp\ (a), rm\ (a)\}, ea \rightarrow e,$$

$$e = \{ed\},$$

$$e_d \in E_d,\ E_d \in W_{res},\ DB \in W_{res}$$

8 МЕТОДИ ТЕСТУВАННЯ ВБУДОВАНИХ СИСТЕМ

Вбудовані системи це складні системи, що можуть об'єднувати різноманітні архітектури. Крім того більшість вбудованих систем – це системи реального часу тому коректність функціонування залежить не тільки від логічної достовірності але й від часу коли отриманий результат.

Для більшості вбудованих систем проектування апаратних платформ та програмного забезпечення це відокремлені процеси. Але для діагностування систем існує істотний зв'язок між програмно-апаратним забезпеченням вбудованих систем. Відмови системи можуть виникнути у зв'язку з дефектами як програмного так и апаратного забезпечення. Але більшість підходів не розглядає тестування на системному рівні і рівні реалізації як єдине ціле. Тому цей розділ присвячений як методам тестування апаратного так і програмного забезпечення.

8.1 Загальні поняття та визначення

Тест (test) являє собою набір операцій, призначених для отримання одного або більшої кількості очікуваних результатів в певній програмній системі. Якщо отримані всі очікувані результати, вважається, що тест пройдено (тобто тест виконано успішно). Якщо фактичний результат відрізняється від очікуваного, вважається, що тест не пройдено (тобто тест завершився невдало) .

Перше, що слід відзначити в наведеному визначенні, так це те, що кожен тест складається з двох компонентів: сукупність виконуваних тестувальником дій та послідовність подій, що повинні відбутися в результаті цих дій.

Тестові дії в сукупності утворюють методику тестування. Послідовність подій, що відбуваються в результаті цих дій, називаються очікуваними результатами. Щоб тест був ефективним, повинні бути чітко й однозначно визначені як методика, так і очікувані результати.

По-друге, якщо методика тестування та очікувані результати визначені правильно, тест повинен давати результат, за яким можна зробити однозначний висновок щодо успіху або невдачі випробування. При введенні в програму двох чисел з метою отримання їх суми тест вважається пройденим, якщо на виході програми буде отриманий коректний результат, інакше тест розглядається як не пройдений.

Варіант тестування (тестовий сценарій) – це опис початкових умов, вхідних даних, дій користувача і очікуваного результату (рис. 8.1). Гарною практикою вважається використовувати прецеденти в якості варіанта тестування.

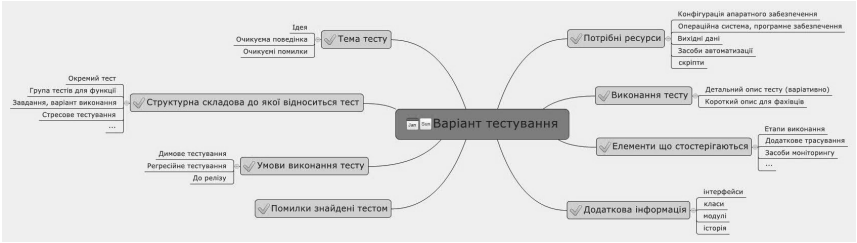


Рисунок 8.1. Варіант тестування

З варіантом тестування повинні бути пов'язані такі ознаки:

- він повинен володіти високою імовірністю виявлення дефекту;
- він повинен бути відтворюваним;
- він повинен володіти чітко визначеними очікуваними результатами і критеріями успішного або невдалого виконання тесту;

- він не повинен бути надмірним.

Мета тестування полягає в знаходженні дефектів.

Дефект – це певна невідповідність продукції вимогам, встановленим нормативно-технічною документацією;

Брак – це дефектна одиниця продукції, тобто продукція, що має хоча б один дефект.

Необхідно розрізняти дефекти апаратного забезпечення та помилки програмного забезпечення.

Дефект апаратного забезпечення [11] це різниця між реалізованим обладнанням та його проектом. Це може бути дефекти процесу виробництва, дефекти матеріалу, вікові дефекти або дефекти пакування.

Помилкою називається неправильний вихідний сигнал дефектної системи. Помилка це «ефект», що призводиться визначеним «дефектом». Помилки викликають відмови системи, тобто відхилення від відповідної поведінки. Якщо відмова може призвести до аварійної ситуації, вона являє собою ризик.

Під відмовою (несправністю) будемо вважати уявлення про «дефект» на рівні абстракції, будемо називати його несправністю. Несправності, це фізичні або логічні дефекти в конструкції або реалізації пристрою.

8.2 Моделі відмов апаратного забезпечення

З розвитком технологій складність та потужність апаратних засобів збільшується. Модель повинна допомогти визначити цільову функцію для тестування та аналізу відмов. Крім того, ефективність моделі відносно фактичних відмов визначаються експериментально. В більшості випадків несправності цифрових систем можна розділити на три групи [11]: помилки проектної документації, помилки виробництва та експлуатаційні відмови. Помилки проектування виникають завдяки людському фактору або помилкам програмного забезпечення САПР (симулятори, генератори та генераторів шарів) і відбуваються в процесі проектування. Ці відмови не пов’язані безпосередньо з процесом тестування. Відмови, що обумовлені недосконалістю процесів виробництва, призводить до дефектів самого обладнання (пропущені при металізації контактні вікна або ділянки оксиду, паразитичні транзистори, пробій оксиду (в МОП-структурах) і т.п.). Фізичні відмови також називаються дефект-орієнтованими відмовами (*defect-oriented faults*). Експлуатаційні або логічні відмови відбуваються через відхилення в умовах експлуатації вбудованих систем. Прикладом таких відхилень виступають електромагнитні перешкоди, помилки оператора, вібрації.

Несправність є моделлю, яка є ефектом фізичного дефекту на логічному або функціональному рівні. Зазначимо, що кілька різних дефектів можуть представлятися однією і тією ж несправністю (має місце відношення «багато до одного»). З іншого боку, одному фізичному дефекту іноді може відповідати кілька несправностей (відношення «один до багатьох») .

Апаратні несправності класифікуються як константні відмови, наприклад, несправності типу замикання, дефекти обриву ланцюга, збоїв пам’яті та інші.

Крім того, їх можна класифікувати як структурні та функціональні. Ті несправності, що визначаються на структурній моделі системи, називаються структурними несправностями. Їх ефект, як правило, зводиться до зміни з’єднань компонент. Функціональні несправності визначаються на функціональної моделі системи. Наприклад, ефект функціональної несправності може проявлятися у зміні функції, реалізованої компоненти системи або оператором мови опису апаратури.

Типовими несправностями сполучень компонентів системи є обрив (open) і замикання (short). Обрив відповідає порушенню з’єднання компонентів схеми. Причиною порушення з’єднання може бути нестача або відсутність провідного матеріалу, наприклад, в металевому провіднику. З іншого боку, відсутність з’єднання може виникнути внаслідок наявності зайвих частинок діелектрика, наприклад, між провідними шарами. Замикання утворюється в результаті з’єднання ліній схеми, які в справній системі ізольовані один від одного. Воно може бути викликано наявністю зайвих провідних частинок між провідниками, пробоєм оксиду в МОП-структурах, який утворює з’єднання з деяким невеликим, але необов’язково нульовим опором і т.п.

Константні відмови (Stuck-at fault). Для визначення константних несправностей використовується структурна модель у вигляді

логічної схеми. Вважається, що одиночна константна несправність (single stack-at fault) діє тільки на з'єднання між вентилями (при цьому логічні елементи функціонують правильно). Кожна лінія схеми може мати два типи цих несправностей: константа 0 і константа 1 (sa-0, sa-1). Отже, константна несправність фіксує на даній лінії постійне значення сигналу 0 або 1 (sa-0, sa-1), незалежно від значення сигналу, що подається на неї. Часто такі несправності моделюють замикання лінії схеми на землю (sa-0) або джерело живлення (sa-1).

Ця модель може використовуватися для генерації тестів незалежно від її адекватності реальних фізичних дефектів. Відзначимо, що для деяких технологій адекватність цієї моделі досить висока, для інших нижче. Але в цілому, дана модель надзвичайно корисна в силу своєї виняткової простоти і задовільної адекватності, і тому використовується в якості базової для багатьох методів моделювання несправностей і генерації тестів.

Несправності типу замикання мають місце в тому випадку, коли відбувається з'єднання двох або більше ліній схеми і утворюється «дротова логіка» (wired logic) в місці виниклої електричного зв'язку. Кратні замикання (з'єднання більше двох ліній) виникають звичайно на зовнішніх входах ІС. В певний час кількість дефектів, провідних до замикань, збільшується внаслідок зменшення розмірів схем і збільшення щільності вентилів у кристалі. Очевидно, що кількість простих замикань (між двома лініями) в схемі, що має m ліній одно. Однак, звичайно, не всі лінії схеми можуть замкнутися між собою. Тому реальна кількість можливих замикань істотно менше і залежить від фізичного сусідства провідників.

Поведінка логічної схеми при замиканні залежить від технології виготовлення цієї схеми.

Слід зазначити, що дефекти замикання можуть викликати функціональні зміни в логічній схемі, які не можна уявити традиційними моделями – несправностями.

Деякі фізичні дефекти не можуть бути представлені константними несправностями. Основна причина полягає в тому, що МОП комбінаційні схеми не завжди залишаються комбінаційними при деяких фізичних дефектах. Найбільш поширеними є такі види відмов у МОН технології: 1) обрив і замикання транзисторів; 2) обриви між стоком, витоком і затвором; 3) короткі замикання: витік – стік, затвор – стік, затвор – витік. Дефекти третьої групи зазвичай обумовлені пробоєм оксиду.

У сучасних цифрових системах можливі ситуації, коли схема структурно і логічно коректна, але час поширення сигналів по деяким її шляхах перевищує допустиме для правильного функціонування значення. У таких випадках говорять про наявність несправності типу «затримка» (поширення сигналів). Такі несправності не можуть бути виявлені на низькій частоті роботи схеми. Метою тестування несправностей «затримка» є визначення правильного функціонування схеми на високих тактових робочих частотах. При цьому виявляється, чи містить схема шляху поширення сигналів, які є занадто повільними або швидкими при зміні вхідних наборів. Для цих цілей використовуються дві основні моделі: затримка вентиля та затримка шляху.

Перша модель передбачає, що затримка обумовлена несправним логічним елементом. Слід зазначити, що час перемикання елемента, може залежати від напрямку зміни сигналу – його підйому або спаду. Це є недоліком даної моделі, оскільки не дозволяє у затримці одного елемента врахувати затримки інших елементів шляху. Очевидно, тут також повністю ігноруються затримки з'єднань між елементами.

Друга модель бере до уваги загальну затримку поширення сигналу від зовнішнього входу до зовнішнього виходу схеми. Хоча дана модель вимагає розгляду занадто багатьох можливих шляхів у схемі, вона більш реалістична, особливо для сучасних технологій, де затримки поширення сигналів мають місце насамперед за рахунок ліній з'єднань. Як правило, тестування затримок проводиться шляхом подачі на схему пари вхідних наборів на бажаної швидкості та спостереженні для кожного зміненого виходу швидкості його перемикання.

Часові несправності. При даних несправностях відбувається тимчасово поява неправильних сигналів в схемі. Вони зустрічаються в різних цифрових елементах, але найчастіше в мікросхемах пам'яті і мікропроцесорів. Серед цих несправностей розрізняють «короткочасні» (transient) «відмови» intermittent. Короткочасні несправності відбуваються, коли сигнали змінюють своє значення внаслідок, наприклад, шумів. Такі несправності важко виявити та виправити. Тут важливо мінімізувати шуми і підвищити перешкодозахищеність

схеми. Дані несправності можуть бути викликані, наприклад, флуктуаціями напруги або космічним випромінюванням.

Аварії є однією з причин відмов при експлуатації комп'ютерних систем. Серед них можна виділити несправності залежні від коду, які зустрічаються в мікросхемах пам'яті і мікропроцесорах.

Несправності рівня кристалу. В даний час нові технології дозволяють проектувати складні цифрові системи (ЦС) на одному кристалі (System-on-Chip – SOC). При цьому проектування ЦС виконується за допомогою досить складних мов високого рівня опису апаратури (HDL), таких як VHDL, Verilog і SpecC. Тому актуальною є проблема верифікації та тестування складних ЦС, описаних за допомогою цих мов.

На логічному рівні моделювання, де ЦС представляється у вигляді логічної схеми, основною моделлю фізичних дефектів є константні несправності, які еквівалентні постійним сигналам 0 або 1 на лініях схеми. На відміну від логічного рівня моделей несправностей, де зазвичай можна встановити відповідність між фізичним дефектом провідників у кремнії і з'єднаннями в логічній схемі, на поведінковому рівні, як правило, важко встановити відповідність між описом ЦС на HDL і структурним описом. Один оператор HDL може відповідати сотням логічних вентилів, з'єднаних між собою. Тому необхідно розглядати функціональні моделі несправностей безпосередньо на мовних конструкціях HDL. При цьому якість (або адекватність) функціональних моделей несправностей, як правило, перевіряється за допомогою логічного моделювання ЦС, яке визначає повноту тесту щодо одиночних константних несправностей схеми, реалізує ЦС. Тому даний підхід орієнтований швидше на досягнення високої повноти тесту для константних несправностей, а не виявлення помилок в мовних конструкціях HDL. Більш того, при цьому ефективність тестової послідовності не може бути визначена безпосередньо на функціональному рівні. Тому в даний час для верифікації та тестування ЦС, описаних на HDL, застосовуються методи, запозичені з тестування програмного забезпечення.

8.3 Функціональне тестування апаратного забезпечення

Функціональне тестування з'явилося найпершим. Постійне збільшення складності виробів роблять процес підготовки функціонального тесту нескінченно довгим. Діагностування несправностей, виявлених у процесі функціонального тестування, може бути досить складним, що вимагає залучення кваліфікованих фахівців. Тому перед тестом системи в цілому часто здійснюється тестування на рівні окремих плат. Тестування плат може бути здійснено і на функціональному рівні, але даний поділ робить діагностування несправностей і підготовку тестів більш гнучкими. Швидко зростаюча складність інтегральних мікросхем викликає схожі проблеми із функціональним тестуванням на рівні плат, так само як і в системному тестуванні – довгий час підготовки тестів, неточне тестове покриття, слабка діагностика.

Наступний широко поширений метод тестування – це внутрішнє-схемне тестування (In-Circuit Test, ICT). Цей метод дозволяє знаходити дефекти і помилки монтажу шляхом забезпечення прямого електричного доступу до компонентів на платі через адаптер. Внутрісхемне тестування ідеально підходило для DIP-компонентів і технології штирьового монтажу. Але у зв'язку з появою багатошарових друкованих плат і більш складних корпусів мікросхемтестовий доступ став сильно обмежений. Технологія внутрисхемного тестування не може розвиватися також швидко, як мініатюризація розмірів компонентів і виробів.

Електронна індустрія передбачала ці проблеми заздалегідь, тому був розроблений метод периферійного сканування, закріплений стандартом IEEE 1149.1, який описує порт тестового доступу (TAP – Test Access Port) і архітектуру периферійного сканування [8, 9]. Метою створення даного стандарту було подолання недоліків інших методів тестування.

Дивлячись на еволюцію тестових методів можна зробити наступні спостереження:

(1) розроблення тестопригодності виробів (Design-For-Test, DFT) стає все більш і більш необхідним доповненням функціонального тестування, дозволяючи зробити контроль більш повним і інформативним, (2) для того щоб виробляти і тестувати сучасні передові розробки, тестопригодність необхідна.

Спочатку, тестування було похідною процесу налагодження нової розробки і пошуку дефектів монтажу. Через зростання складності схем пристроїв керованість цими процесами могла бути підвищена тільки при роздільному їх проведенні. Виявлення і виправлення виробничих дефектів на стадії налагодження дослідних зразків стало необхідністю.

З ростом складності продукції багато виробників почали застосовувати багатоступеневу стратегію тестування, метою застосування якої є якомога більш раннє виявлення і виправлення помилок виробничого процесу.

Перша версія стандарту Boundary-Scan [8], з'явилася на початку 1990 року, і отримала ім'я, яке зберіглося і сьогодні – IEEE 1149.1. Стандарт цієї технології також називається Test Access Port and Boundary-Scan Architecture (порт тестового доступу та архітектура граничного сканування). Проект був розроблений міжнародною групою експертів, яка носила назву JTAG (Joint Test Action Group – об'єднана група розробки методів тестування).

Сама архітектура цифрового стандарт Boundary-Scan не відрізняється особливою складністю, на відміну від своїх можливостей. Відповідно до стандарту IEEE 1149.1, так звана Boundary-Scan-IC, повинна бути оснащена чотирма обов'язковими елементами:

- TAP-портом, який складають чотири обов'язкових сигналів, і п'ятий за рішення розробників безпосередньо самої плати (TCK – контакт синхронізації роботи механізму Boundary-Scan; TMS – контакт вибору тестового режиму; TDI – контакт введення тестових даних; TDO – контакт виведення тестових даних (знаходиться в постійно в третьому стані, окрім режиму зсуву); RST – контакт асинхронного скидання стану TAP-контролера (може зовсім не бути присутньою)). TAP-контролер виступає одним з важливих елементів управління всією роботою технології Boundary-Scan;
- IR (Instruktion Register від анг. регістр команд) – перша група регістрів, в якій обов'язковим за стандартом повинен бути присутнім хоча б один Регістр Команд (РК);
- DR (Data Registers від анг.– Регістри Даних) – друга група регістрів, відповідно до стандарту зобов'язана в себе містити як мінімум два регістри: Регістр Обходу (РО, також його іноді називають Шунт-Регістр), Регістр Boundary-Scan.

Такий мінімальний комплект елементів вимагає стандарт IEEE 1149.1. Інші регістри, які можуть доповнити групу, як IR, так DR на розсуд розробників плат, також допускаються створеним стандартом.

Для того, щоб отримати хороше тестове покриття, немає необхідності в тому, щоб всі компоненти на платі мали JTAG-інтерфейс. Наприклад, багато блоків, складається з нескановних компонентів (кластера), можуть тестуватися, незважаючи на відсутність прямого доступу для периферійного сканування. У дійсності, існують практичні приклади, коли здійснюється контроль і детальне тестування абсолютно всієї плати (включаючи пам'ять) за допомогою одного або двох компонентів, що підтримують периферійне сканування.

На рис. 8.2 зображена архітектура Boundary-Scan. На TAP-контролер подаються 2 (3) сигналу, за допомогою яких контролер встановлює відповідний режим роботи схеми. Сам TAP-контролер є автомат з кінцевою кількістю вершин.

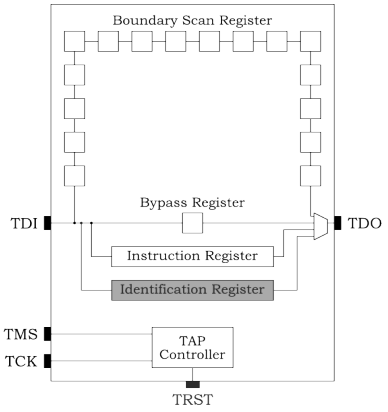


Рисунок 8.2. Архітектура JTAG

Сьогодні на світовому ринку в цій області лідирують чотири представники США і Європи, які поставляють програмно-апаратні комплекси (ASSET InterTech Inc. і CORELIS Inc – США; GOEPEL Electronic – Німеччина; JTAG Technologies – Нідерланди). Такі розробки називаються BS-тестери.

Використання JTAG і технології граничного сканування в мікросхемі, на платі або в пристрої додає вартість і збільшує час розро-

блення проекту. Але, все ж ці витрати легко окупаються при проведенні тестування, яке забезпечується на кожній стадії циклу життя виробу. Крім безпосередньо граничного тестування, проектувальники використовують технологію JTAG для того, щоб виробляти самотестування (BIST) (у тих компонентах, де воно реалізовано) і завантажувати внутрішні значення в регістри пристрою або програмувати мікросхеми ПЗУ. Тести, які були розроблені та використані на етапі проектування, можуть бути передані виробництву, для того щоб забезпечити додаткове зниження вартості і часу на перевірку виробів при вихідному контролі.

8.4 Тестування програмного забезпечення вбудованих систем

Тестування програмного забезпечення (*software testing*) – це планова, впорядкована діяльність, процес аналізу або експлуатації програмного забезпечення з метою виявлення дефектів.

Всі види тестування програмного забезпечення вбудованих систем можливо класифікувати: за рівнем елементів, що перевіряються, за характеристикам якості, за джерелами даних, по ролі команди.

За масштабом перевіряємих елементів розрізняють тестування модулів, комплексні випробування, системне тестування, приймальне тестування.

За характеристиками якості можливо тестування функціональності, тестування надійності, тестування переносимості, тестування зручності використання.

За джерелами даних виділяють тестування чорного ящика, сірого ящика та білого ящика.

Функціональні тести базуються на функціях і особливостях, а також взаємодії з іншими системами, і розглядають зовнішню поведінку системи. Далі перераховані одні з найпоширеніших видів функціональних тестів: функціональне тестування (Functional testing); тестування безпеки (Security and Access Control Testing); тестування взаємодії (Interoperability Testing).

Нефункціональне тестування описує тести, необхідні для визначення характеристик програмного забезпечення, які можуть бути виміряні різними величинами. В цілому, це тестування того, «як» система працює. Це всі види тестування продуктивності такі як тестування навантаження (Performance and Load Testing); стресове тестування (Stress Testing); тестування стабільності або надійності (Stability / Reliability Testing); об'ємне тестування (Volume Testing); тестування інсталяції (Installation testing); тестування зручності користування (Usability Testing); тестування на відмову і відновлення (Failover and Recovery Testing); конфігураційне тестування (Configuration Testing).

Після проведення необхідних змін, таких як виправлення бага/дефекту, програмне забезпечення повинне бути протестовано для підтвердження того факту, що проблема була дійсно вирішена. Нижче перераховані види тестування, що необхідно проводити після установки програмного забезпечення, для підтвердження працездатності програми або правильності здійсненого виправлення дефекту:

1. Димове тестування (Smoke Testing);
2. Регресійне тестування (Regression Testing);
3. Тестування збірки (Build Verification Test);
4. Санітарне тестування або перевірка (Sanity Testing).

Тестування на різних рівнях проводиться протягом усього життєвого циклу розробки і супроводу програмного забезпечення. Рівень тестування визначає те, над чим виробляються тести: над окремим модулем, групою модулів або системою, в цілому. Проведення тестування на всіх рівнях системи – це запорука успішного завершення проекту.

8.5 Метод регресійного тестування інформаційних систем

Розглянемо метод регресійного тестування ІС, зокрема ІС з web-орієнтованим інтерфейсом (BOIC) [16, 13]. Даний клас ІС будуються за принципом еволюційного прототипування, тобто послідовно створюється макет системи, який буде з кожною ітерацією ближче до реального продукту. Такий підхід має ту перевагу, що на кожному кроці розробники манипулюють працюючою системою, яка має часткову функціональність, яку можна тестувати і покращувати з кожною ітерацією.

Для тестування будемо виділяти ФО, що є атомарним об'єктом, пов'язаним з вимогами користувача (прототипом ІС) і функціональністю системи за допомогою адаптивних правил, залежно від логіки додатка. Таким чином, якщо прийняти за *F* – безліч виділених в ІС ФО, а за *Fk* – безліч реалізованих і пройшли тестування в k-ої версії ФО, то прогрес розробки *IC* в першому наближенні можна оцінити за допомогою критерію:

$$M_{progress1} = \frac{|F_k|}{|F|} \times 100\%. \quad (8.1)$$

Більш точну інформацію про стан розроблення ІС дасть критерій, який враховує трудомісткість інтеграції ФО і відсоток завершеності розробки ФО і показує прогрес виконання ІС за поточним кале

$$M_{progress2} = \frac{\sum_{i=1}^{N}(tp_i \times p_i + tp_i^{(int)} \times p_i^{(\text{int})})}{100 \times \sum_{i=1}^{N}(tp_i + tp_i^{(int)})}, \quad (8.2)$$

де *tp* – планова трудомісткість ФО; $tp^{(int)}$ – Планові трудовитрати на інтеграцію ФО у ІС; *N* – загальна кількість ФО, $N = |F| - |F_{cansel}|$, | F | – загальна кількість виділених в ІС ФО; $|F_{cansel}|$ – Кількість ФО, знятих з розробки; p_i – відсоток завершеності розробки ФЕ; $p_i^{(int)}$ – відсоток завершеності інтеграції ФО.

Величини, визначають за результатами тестування, виходячи з кількості пройдених тестів для даної ФО:

$$\frac{n\_test\_passed_i}{n\_test\_total_i} \times 100\%,$$

де $n\_test\_passed_i$ – кількість тестів, для яких виконаний критерій успішності, по і-тої ФО, $n\_test\_total_i$ – загальна кількість тестів з і-тої ФО.

Відхилення фактичних трудовитрат на розробку ІС від календарного графіка дозволяє визначити критерій:

$$M_{dev} = \sum_{i=1}^{N}\left[\left(\frac{tf_i \times p_i + tf_i^{(int)} \times p_i^{(\text{int})}}{100}\right) - (tp_i + tp_i^{(int)})\right], \quad (8.3)$$

де tf – поточні фактичні трудовитрати на ФО; $tf^{(int)}$ – поточні фактичні трудовитрати на інтеграцію ФО.

Всі запропоновані критерії оцінки прогресу прагнуть до 100% (відповідає повністю розробленої ІС, що задовольняє всім вимогам), а значення критерію відхилення календарного графіка дозволяє визначити неправильну оцінку трудомісткості ФО (від'ємне значення свідчить про завищену оцінку трудомісткості, позитивне визначає поточне перевищення календарного плану).

Дані критерії є заходами зусиль і вимірюються відносної шкалою по відношенню до початкових виділеним завданням і їх тривалості. Розраховані значення (8.1–8.3) дають на проміжних етапах розробки ІС менеджеру розробки кількісну оцінку виконання проекту та можливість визначити відповідність процесу розробки календарним планом. Використання критеріїв окремо може спричинити за собою невірні висновки, в той час як їх комплекс дозволяє точно визначити поточний стан розроблення ІС.

Таким чином, запропоновані критерії є об'єктивними обчислюваними динамічними критеріями, що служать заходом визначення прогресу розробки ІС і дозволяють на будь ітерації визначити відповідність фактичного стану процесу розроблення до плану. Отримані значення дозволяють визначити чи виконані на даний момент всі заплановані завдання, немає перевищення термінів виконання та незавершених завдань. Отримані значення критеріїв дозволяють переоцінити трудомісткість окремих модулів і ІС в цілому, щоб потім скорегувати процес розробки з метою зменшити відхилення від запланованої дати передання до експлуатації, таким чином, знизивши ризик порушень календарного планування.

Для ІС з веб-інтерфейсом (ВОІС) рівень критичності web-ресурсів різного рівня вкладеності значно відрізняється і тому необхідно розглядати їх окремо.

Рівень вкладеності – це параметр, який відповідає за стан вебсторінки в загальній структурі сайту. Рівень вкладеності визначається по мінімальній кількості кілків (переходів), які потрібно зробити, щоб досягти цієї сторінки з головної сторінки сайту. Головна сторінка сайту завжди має рівень вкладеності 1. Всі інші сторінки, на які можна потрапити з неї за один клік – рівень вкладеності 2. Сторінки з рівнем вкладеності 4 з точки зору оптимізації є неприпустимими при розробці структури сайтів, так як пошукові роботи проводять їх індексацію не дуже часто.

Таким чином, структура ПК ІС може бути представлена наступним чином:

$$WOIS = <structure0, S_p, structure, S>,$$

де structure0=$< Ed2_p, EdN3_p, EdN^p_{®\infty} >$ – первинна структура прототипу ВОІС, яка не містить інформації про рівень вкладеності web-ресурсів;

Ed2p – web-ресурси другого рівня вкладеності прототипу інтерфейсу ПК ВОІС;

Ed3p – web-ресурси третього рівня вкладеності прототипу інтерфейсу ПК ВОІС;

$Ed2_p, EdN3_p, EdN^p_{®\infty}$ – Web-ресурси глибше третього рівня вкладеності прототипу інтерфейсу ПК ВІС;

Sp – матриця зв'язків між web-ресурсами в прототипі інтерфейсу ПК ВОІС;

structure =$< Ed2, EdN3, EdN_{\to\infty} >$ – структура ПК ВОІС;

Ed2 – web-ресурси другого рівня вкладеності ПК ВОІС;

Ed3 – web-ресурси третього рівня вкладеності ПК ВОІС;

$EdN_{\to\infty}$ – Web-ресурси глибше третього рівня вкладеності ПК ВОІС;

S – матриця зв'язків між web-ресурсами в ПК ВОІС.

Відповідно, |*F*| – кількість виділених в ІС ФО – прийме вигляд:

где *N2p* – кількість елементів прототипу другого рівня вкладеності; *N3p* – кількість елементів прототипу третього рівня вкладеності; $N^p{}_{®\infty}$ - кількість елементів прототипу глибше третього рівня вкладеності;

а F_k – кількість реалізованих і пройшли тестування в k-ої версії ФО – прийме вигляд:

$$|F_k'| = N2 + N3 + N_{\to\infty},$$

де *N2* – кількість web-ресурсів версії ВОІС другого рівня вкладеності; *N3* – кількість web-ресурсів версії ВОІС третього рівня вкладеності; $N_{\to\infty}$ – кількість web-ресурсів версії ВОІС глибше третього рівня вкладеності.

Аналогічно критеріями для оцінки відповідності прототипу запропоновано тестування наступних критеріїв стану:

1.– критерій відповідності кількості web-ресурсів другого рівня вкладеності версії ВОІС прототипу:

$$M_{N_2} = \frac{N2}{N2_p} \times 100\%, \quad lim(M_{N_2}) \to 100\% \tag{8.4}$$

де *N2* = |*Ed2*| – кількість web-ресурсів версії ВОІС другого рівня вкладеності;

N2p = |*Ed2p*| – кількість елементів прототипу другого рівня вкладеності;

2.– критерій відповідності кількості web-ресурсів третього рівня вкладеності версії ВОІС прототипу:

$$M_{N3} = \frac{N3}{N3_p} \times 100\%, \; lim(M_{N_3}) \to 100\% \tag{8.5}$$

де *N3* – кількість web-ресурсів версії ВОІС третього рівня вкладеності;

N3p – кількість елементів прототипу третього рівня вкладеності;

3.– критерій кількості web-ресурсів в прототипі інтерфейсу глибше третього рівня вкладеності:

$$M_{N_{\to\infty}} = \frac{1}{N2_p + N3_p} \times \sum_{i=4}^{q} N_i \times 100\%, \quad lim(M_{N_{\to\infty}}) \to 0 \tag{8.6}$$

де *q* – максимальний рівень вкладеності версії ВОІС;

N2p і *N3p* – відповідно кількість елементів прототипу другого і третього рівня вкладеності;

для кращої індексації ВОІС $lim(M_{N_{\to\infty}}) \to 0$, тобто кількість web-ресурсів більш ніж третього рівня вкладеності має бути як можливо меншим порівняно з кількістю *N3* і *N2*.

4. *MS* – критерій відповідності структури версії ПК ВОІС і прототипу інтерфейсу (через симетричності відносно головної діагоналі розглядається тільки нижня трикутна матриця):

$$M_s = 2 \times \frac{\sum_{i=1}^{N} \sum_{j=1}^{i-1} a[i][j]}{N^2 - N}, lim(M_s) \to 1, \tag{8.7}$$

де *N* – кількість web-ресурсів ВОІС і прототипу (у разі кількість web-ресурсів не збігається, додаються відсутні рядок і стовпець і заповнюються нулями);

$$a[i][j] = \begin{cases} 1, если\ S_p[i][j] = S[i][j] \\ 0, если\ S_p[i][j] \neq S[i][j] \end{cases},$$

S – матриця зв'язків між web-ресурсами ВОІС;

Sp – матриця зв'язків між елементами прототипу.

У заголовках рядків і стовпців матриці Sp знаходяться назви всіх елементів прототипу ВОІС:

$$S_p[i][j] = \begin{cases} 0, е,е\ звязок\ між\ і\text{-}тойи\ \ j\text{-}т\text{-}той\ стор. відсутній; \\ 1, е,е\ \ існує\ перехід\ зі\text{-}т\text{-}той\ на\ую\ стор \\ 2, е,е\ існує\ \ перехід\ сj\text{-}т\text{-}той\ на\тую\ стор \\ 3, е,е\ існує\ двосторонній\ перехід. \end{cases}$$

Матриця *S* будується аналогічно.

5. Співвідношення трудомісткості розроблених та затверджених web-ресурсів версії ВОІС у загальної запланованої трудомісткості ВОІС розраховується аналогічно (8.6):

$$M_{progress} = \sum_{i=1}^{N} \left[\left(\frac{tp_i \times pct_i}{100} + tS_p[i] \right) - (tf_i + tS[i]) \right], lim(M_{progress}) \to 0,$$

де $i = 1..N$, $N = |F|$ – загальна кількість запланованих web-ресурсів;

tp_i – планові трудовитрати на ФО;

pct_i – відсоток завершеності і-ого web-ресурсу;

$tS_p[i]$ – планові трудовитрати на реалізацію зв'язків і-ого web-ресурсу;

tf_i – поточні фактичні трудовитрати на web-ресурс;

$tS[i]$ – поточні фактичні трудовитрати на реалізацію зв'язків і-ого web-ресурсу.

Дані критерії перевіряються на кожному етапі ЖЦ ВОІС за допомогою автоматизованого тестування посилань ВОІС. Це дозволяє перевіряти відповідність кількості web-ресурсів усіх рівнів необхідному замовником кількістю.

8.6 Метод регресійного тестування web-орієнтованих систем

В основу методу покладено прототип інтерфейсу ІС (7.1), на підставі якого обчислюються критерії оцінки якості (8.1–8.7), що зв'язують логічну модель ІС, що розробляється, з артефактами на різних етапах створення ВОІС.

Розглянемо обчислювальну схему методу:

Етап 1. Будується матриця атрибутів вимог – до атрибутів функціональних вимог *V* додається атрибут «Реалізувати на сторінці або для ФО», в якому вказується ідентифікатор web-ресурсу, в якому планується реалізувати дану функціональність.

Етап 2. З поля «Реалізувати на сторінці або для ФО» витягуються ідентифікатори web-ресурсів, прибираються повторювані і формується масив імен web-ресурсів майбутньої ІС structure0:

$$V_a^{e_d^j} == "Реалізувати\ \ для\ ФО" \Rightarrow structure0,$$

де $V_a^{e_d^j}$ – набір атрибутів web-ресурсу; *j = 1..N, N* – кількість web-ресурсів;

structure0- первинна структура ІС, яка не містить інформації про рівень вкладеності web-ресурсів.

Етап 3. Будується модифікована структура прототипу Sp, тобто структура взаємодії web-ресурсів, матриця можливих переходів між ними Sp [N+1] [N+1] .

Етап 4. В результаті розробки на k-ой ітерації отримують модифіковану структуру ПК ІС structure і матрицю можливих переходів між реалізованими web-ресурсами S [N + 1] [N + 1] .

Етап 5. Після етапу розробки, на кожній ітерації на етапі тестування розраховуються модифіковані критерії для оцінки стану процесу розробки ПК ІС, записані в формулах (8.1–8.6), тобто порівнюються модифікована і первинна структура прототипу і структури, отримані в результаті аналізу розробленої на поточній ітерації версії системи:

$$\begin{cases} S\ [N+1][N+1] == Sp\ [N+1][N+1] \\ structure == structure0 \end{cases}$$

Етап 6. Обираються методи тестування для змінених ФО ІС. Проводиться тестування і знайденим помилок залежно від результатів тестування присвоюються такі статуси:

$$status(bug_i) \in \{"open","fixed","verified\ fixed","verified\ closed"\},$$

де «open» – відкритий і вимагає тестування; «fixed» – виправлений; «verified fixed» – перевірений виправлений; «verified closed» – закритий виправлений.

Етап 7. Вибираються методи тестування для пов'язаних ФО ІС за рахунок аналізу структури зв'язків між ресурсами ІС.

Формально процес вибору методів тестування виглядає наступним чином:

1. Для дефектів (помилок) зі статусом « $fixed$ », необхідно виконати тестування web-ресурсу, з яким пов'язано функціональне вимога, до якого відноситься виправлений дефект:

$$if\ (fixed(bug_i)\ \&\ bug_i \in V^{e^g_d}) \Rightarrow test(e^g_d),$$

де bug_i – будь-який з незакритих дефектів, i=1..Nb, Nb – кількість дефектів;

$fixed(bug_i)$ – привласнення дефекту статусу « $fixed$ »;

$V^{e^g_d}$ – функціональна вимога, яка ставиться відповіно до ресурсу e^g_d, g = 1..N, N – кількість web-ресурсів;

$test(e^g_d)$ – запуск безлічі тестів для web-ресурсу e^g_d.

2. Визначаються ресурси, пов'язані зі зміненим, для тестування:

$$if\ (S_p[g][k] > 0) \Rightarrow test(e^k_d),$$

де k = 0..N – номер web-ресурсу.

3. Якщо тестований web-ресурс і все, пов'язані з ним, пройшли тестування, поточного дефекту присвоюється статус «перевірений виправлений» і потім «закритий виправлений»:

$$if\ (test(e^g_d) == passed\ \&\ test(e^k_d) == passed)$$

$$\Rightarrow verified\ fixed(bug_i)\ \&\ fixed\ closed(bug_i),$$

де $verified\ fixed(bug_i)$ – присвоєння bug_i статусу «перевірений виправлений»;

$fixed\ closed(bug_i)$ – присвоєння bug_i статусу «закритий виправлений».

Етап 8. Прийняття рішення про завершення розробки.

Етапи 1–7 виконуються на кожній ітерації до тих пір, поки результати тестування не зійдуться з прописаними в плані тестування критеріями закінчення тестування. В успішних розробках критерії $M_{N2,3} = 100\%$, $\lim M_S = 100\%$ і $\lim M_{N\infty} \to 0$ (граничний рівень може встановлюватися замовником) .

Необхідно, щоб $\lim M_{N2,3} \to 100\%$, $\lim M_S \to 100\%$ и $\lim M_{N\to\infty} \to 0\%$, тому при інших значеннях необхідно визначити через що відбулося відхилення від норми і виправити невідповідність прототипу (етап 5) або неактуальне прототип (етап 2). У разі зміни або додавання вимог до системи – повернення на 1 етап.

Запропонований метод виходить за рамки стандартних цілей контролювати якість і дозволяє забезпечити якість на різних етапах ЖЦ. Це можливо за рахунок планомірного контролю та відстеження відповідності розроблюваної ІС вимогам замовника до функціональності і структурі ІС. Створені на кожному етапі артефакти тестуються на відповідність прототипу і, таким чином, даний метод надає кошти оцінки якості на різних етапах створення ІС за рахунок аналізу відповідності логічної моделі розроблюваної системи, прийнятої замовником.

9 ВИКОРИСТАННЯ ВІДДАЛЕНОЇ ЛАБОРАТОРІЇ GOLDI ДЛЯ ВИВЧЕННЯ ЗАСОБІВ ТЕСТУВАННЯ ВБУДОВАНИХ СИСТЕМ

9.1 Аналіз можливостей лабораторії GOLDi для цілей навчання

При сучасному навчанні неможливо не використовувати інформаційні комунікаційні технології (ІКТ).

Кожна з п'яти фаз, що присутні при викладанні матеріалу – вступ до матеріалу, мотиваційна фаза, фаза викладання матеріалу, фаза фіксації та діагностична фаза, містить свої ІКТ.

Однією з найважливіших фаз є фаза фіксації матеріалу, коли учень за допомогою лабораторного практикуму закріплює отриманий матеріал (рис. 9.1).

Рисунок 9.1. Піраміда ефективності методів навчання

Але в сучасних умовах дуже складно забезпечити постійний доступ до обладнання, особливо коли йде мова про короткосрочні курси для представників підприємств. На підтримку приходять віддалені лабораторії [1–7].

Для грунтовного використання існуючих віддалених лабораторій необхідно провести всебічний аналіз результатів навчання, що планується забезпечити та потужностей віддалених лабораторій. Ефективним інструментом для такого аналізу є метод стратегічного планування SWOT (Strengths, Weaknesses, Opportunities, Threats). Шаблон для використання віддалених лабораторій наведений у таблиці 9.1 [4].

Для вивчення дисципліни Якість інформаційних систем були проаналізовані цілі і завдання (рис. 9.2). Було проведено аналіз режимів функціонування лабораторії GOLDi [3–6] та проведено SWAT (таблиця 9.2).

Таблиця 9.1. Загальний шаблон SWOT аналізу для адаптації існуючих віддалених лабораторій

	Можливості Які можливості віддаленої лабораторії? Яка галузь використанні віддаленої лабораторії?	**Погрози** Які обмеження на вхідні та вихідні данні при використанні віддаленої лабораторії? Які передумови використання віддаленої лабораторії?
Сильні стороні Яка мета курсу? Які завдання навчання? Які методи повинен вивчити студент? Які ІКТ можуть бути використані в курсі? Для яких завдань можуть бути використання ІКТ?	Які можливості віддаленої лабораторії дозволять вирішити завдання навчання?	Які можливості використання матеріалу курсу, що дозволять запобігти передумовам віддаленої лабораторії?
Слабкі сторони Які існують передумови для вивчення дисципліни? Чи є в курсі матеріал для якого не можливо використовувати ІКТ?	Як потужності віддаленої лабораторії можуть бути використані для запобігання передумов курсу?	Як можливо мінімізувати внутрішні слабості для запобігання зовнішнім погрозам?

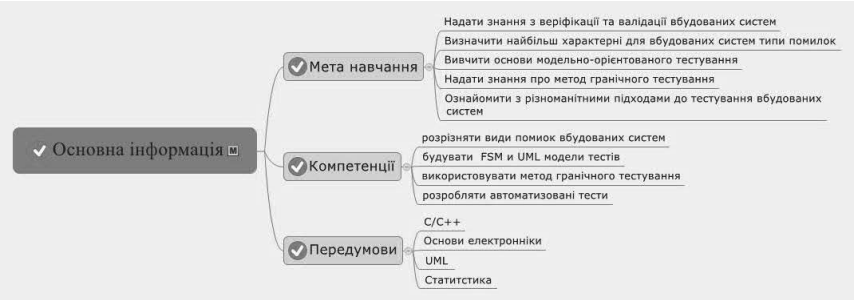


Рисунок 9.3. Основна інформація з дисципліни «Якість інформаційних систем»

Таблиця 9.2. Шаблон SWOT для адаптації лабораторії GOLDi для дисципліни «Якість інформаційних систем»

	Можливості	**Погрози**
Сильні стороні	Використання наступних режимів лабораторій “Stand alone mode” для навчанню генерації тестів з використанням FMS. “Remote control Mode” для виконання розроблених тестів “Rapid Prototype mode” для функціонального тестування	Необхідно розробити практикум для генерації тестів, що можуть використовуватись для плати швидкого прототипування
Слабкі сторони	Використання віртуального режиму “Virtual control Mode” перед використанням обладнання	Необхідно розробити додаткові тести з FMS, VHDL and C для вбудованих систем у системі LMS Moodle.

Так для викладання в рамках міжнародного проекту ICo-op для кожної фази викладання були використанні наступні ІКТ – вступ до матеріалу – презентація лабораторії GOLDi, мотиваційна фаза – демо режим лабораторії GOLDi, фаза викладання матеріалу – презентаційні матеріали, додаткові теоретичні матеріали були завантажені в університетську систему LMS Moodle, фаза фіксації – режими віддаленої лабораторії GOLDi для вивчення FSM для використання при генерації тестів та основи функціонального тестування, для діагностичної фази – система керування контентом Moodle

9.2 Використання лабораторії GOLDi для модельно-орієнтованого тестування

Тестування на основі моделі (англ. Model-based testing) – це тестування програмного забезпечення, в якому варіанти тестування частково або цілком виходять з моделі, яка описує деякі аспекти (частіше функціональні) тестованої системи (англ. System under test). Моделі можуть відображати бажану поведінку системи або використовуватися для створення тестових стратегій або середовища тестування. Тестування на основі моделі виконується в наступні етапи:

1. Побудова моделі (Модель кінчений автоматів, модель подій системи, і т.д.).
2. Генерування очікуваних вхідних даних.
3. Генерування очікуваних вихідних значень.
4. Порівняння з фактичними результатами.
5. Прийняття рішень

При побудові моделі, вона повинна відповідати наступним вимогам:

- простота – модель повинна бути настільки простою, щоб витрати на її побудову могли б окупитися. Хороша проста модель не вимагає великих вкладень на навчання персоналу. Модель повинна бути інтуїтивно зрозумілою кожному співробітнику;
- детальність – модель повинна бути настільки детальною, щоб з її допомогою можна було б описати всі стани і параметри програми для проведення повноцінного тестування;

- тестованість – модель повинна бути побудована таким чином, щоб при генерації тестів можна було б отримати тестові набори, придатні для ручного тестування (а згодом, і автоматизованого);
- автоматизованість – модель повинна підтримувати потенційну автоматизацію тестування. Тобто переходи між станами повинні бути настільки «низькорівневими», щоб їх можна було передавати на вхід інструменту автоматизації тестування.

Для побудови варіантів тестування заснованих на моделях ефективними інструментами виступають: діаграми станів UML, кінцеві автомати, автомати Мілі, автомати Мура, контекстно-незалежні граматики, марківські мережі, таблиці рішень.

При використанні кінцевих автоматів для тестування умови можуть бути обрані за наступними критеріями:

1. Які значення може приймати параметр? Перевірити граничні значення, неприпустимі значення, нормальні значення
2. Які сукупності значень мають особливий сенс? Якщо параметри залежать один від одного – простежити залежності
3. Які комбінації параметрів приведуть до коректній роботі програми / методу / модуля?
4. Які комбінації параметрів викличуть помилку, неправильний код повернення або аномальна поведінка?

На рисунку 9.4 відображені можливі ситуації при формуванні моделей з використанням FSM.

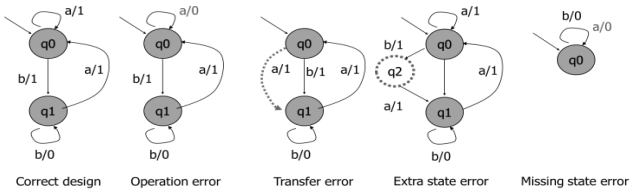


Рисунок 9.4. FSM моделі

Найбільш ефективним є використання FSM моделей для таких видів тестування як підтверджуюче тестування (conformance testing), тестування переходів (transition testing) та тестування на основі критерію текстового покриття (coverage testing).

Для навчання моделюванню з використанням FSM віддалена гібридна лабораторія GOLDi має віртуальний режим (рис. 9.5) та віддалений експеримент з використанням веб-клієнту (рис. 9.6) [5].

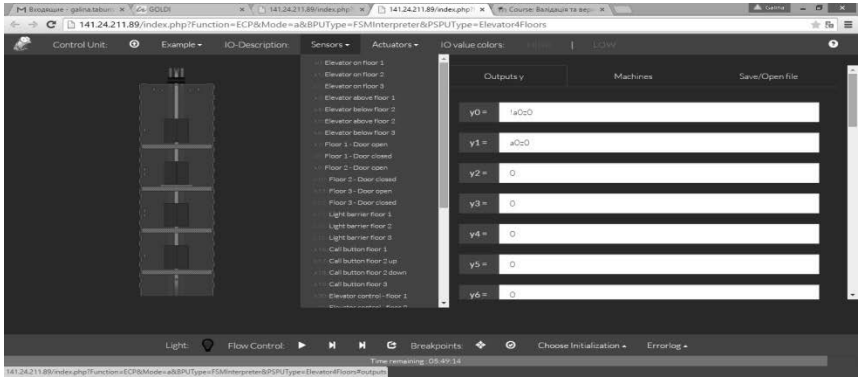


Рисунок 9.5. Віртуальний режим роботи ліфта

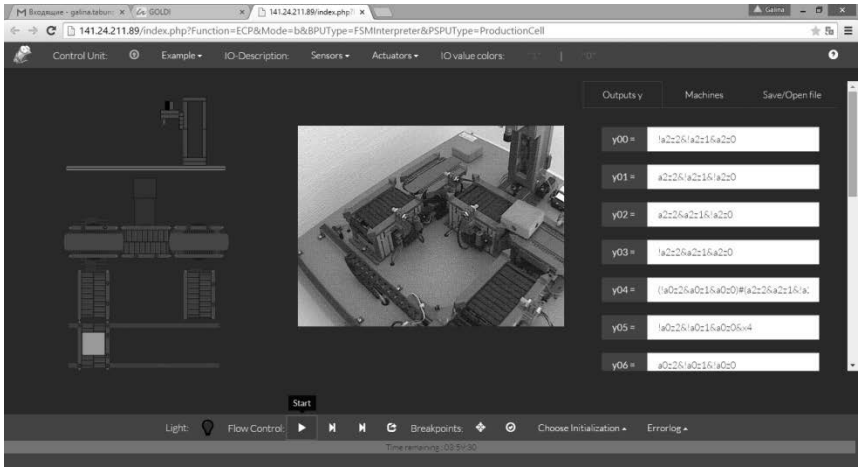


Рисунок 9.6. Віддалений експеримент для конвеєрної лінії

Завдання для студентів можуть бути наступними:

1. Побудуйте модель роботи 3-вісного порталу відповідно до графіка на рис. 9.7.

2. Розширити тестові послідовності і використовувати їх для конструювання покриття всіх переходів для моделі 3-вісний портал.
3. Розробити найпростіший критерій тестового покриття для 3-вісного порталу.

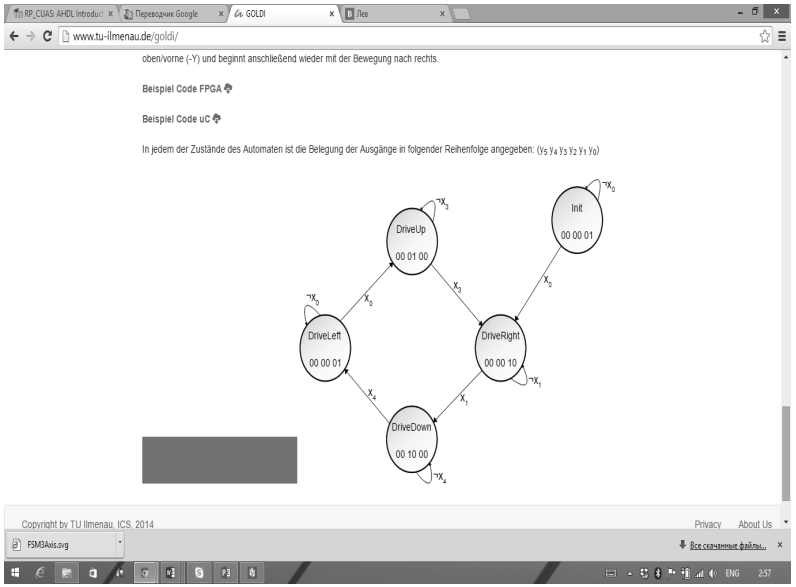


Рисунок 9.7. Модель роботи 3-вісного порталу

4. Розширити тестові послідовності і використовувати їх для покриття всіх переходів для моделі ліфт.
5. Розробити найпростіший критерій тестового покриття для моделі ліфт.
6. Розширити тестові послідовності і використовувати їх для покриття всіх переходів для моделі конвеєрна лінія.
7. Розробити найпростіший критерій повного тестового покриття для конвеєрної лінії.

9.3 Використання плати швидкого прототипування для навчання функціональному тестуванню вбудованих систем

Серцем віддаленої гібридної лабораторії GOLDi є плата швидкого прототипування. Робота з цією платою можлива як з використанням автономного програматора (рис. 9.9) так і з використанням віртуального та віддаленого режимів лабораторії (рис. 9.10–9.11) [4, 5].

Рисунок 9.9. Плата швидкого прототипування

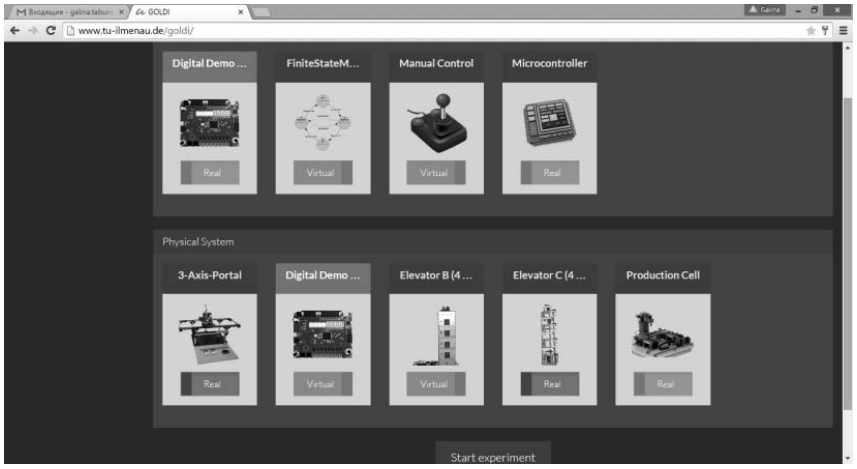


Рисунок 9.10. Режими роботи віддаленої лабораторії

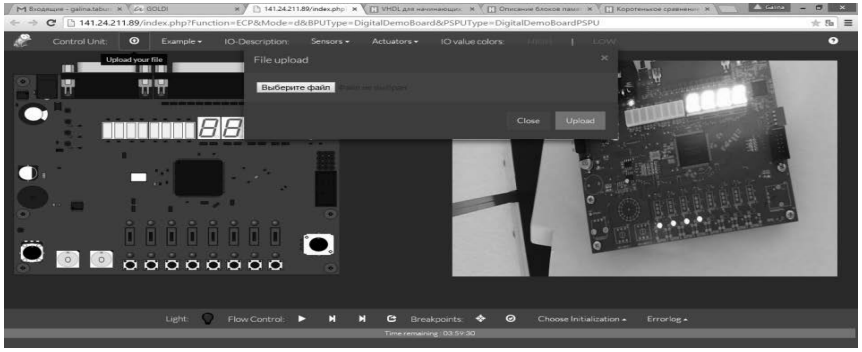


Рисунок 9.11. Робота з платою швидкого прототипування

Для роботи з платою вихідними даними є .pof файл, отриманий при компіляції проекту в Quartus II [5].

Для виконання функціонального тестування можливо використовувати програми розроблені з використанням VHDL у середовищі Quartus II Web-edition від Altera (рис. 9.12).

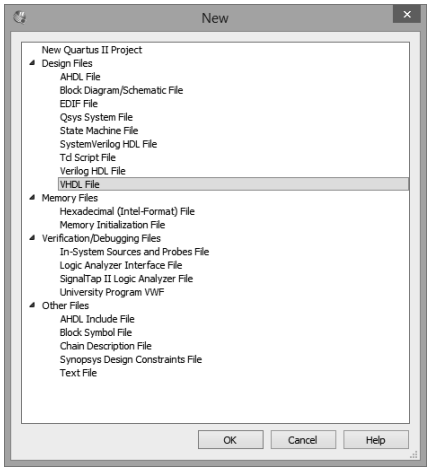


Рисунок 9.12. Створення VHDL проекту

Створення тесту складається з наступних кроків:

Крок 1. Введення проекту на мові vhdl.

Крок 2. Компіляція проекту (рис. 9.13) .

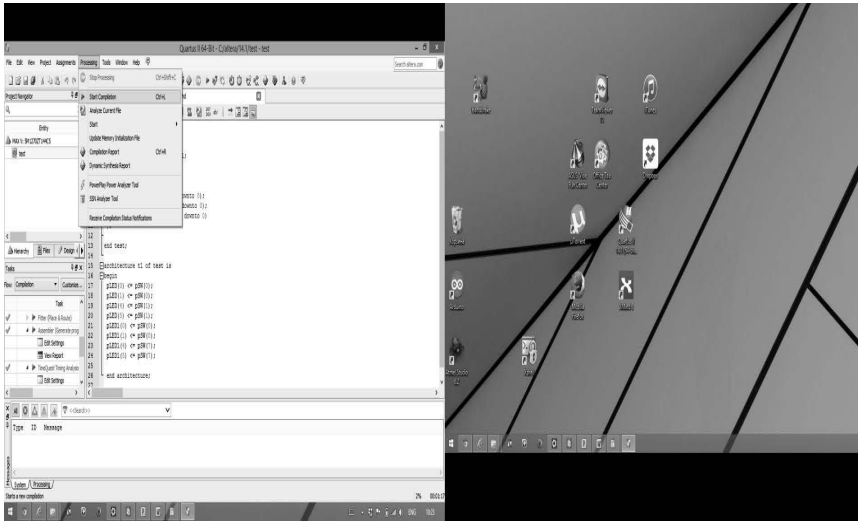


Рисунок 9.13. Компіляція проекту

Крок 3. Зв'язування контактів з використанням Pin Planner (рис. 9.14)

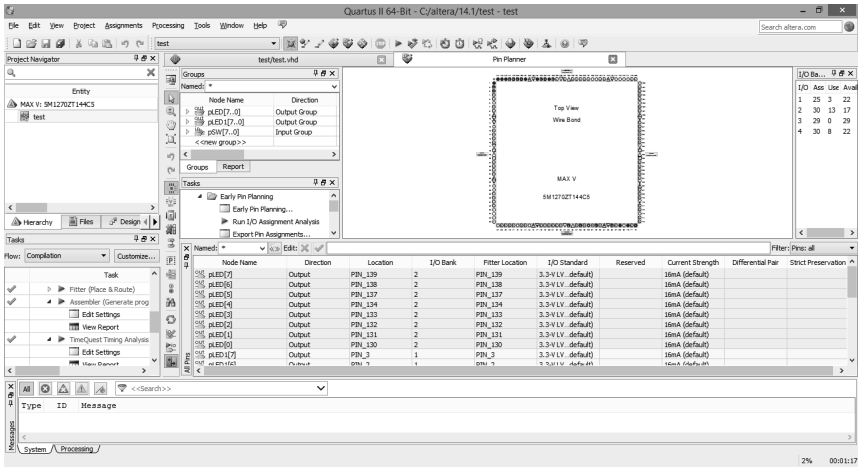


Рисунок 9.14. Pin Planner для Quartus II

На рис. 9.15 міститься відповідність виходів та контактів для MAX® V – 5M1270Z CPLD від Альтера.

Name	Input/Output	Pin
Hex0[2]	Output	PIN_85
Hex0[1]	Output	PIN_86
Hex0[0]	Output	PIN_87
Hex1[7]	Output	PIN_96
Hex1[6]	Output	PIN_102
Hex1[5]	Output	PIN_103
Hex1[4]	Output	PIN_104
Hex1[3]	Output	PIN_105
Hex1[2]	Output	PIN_97
Hex1[1]	Output	PIN_98
Hex1[0]	Output	PIN_101
Hex2[7]	Output	PIN_108
Hex2[6]	Output	PIN_112
Hex2[5]	Output	PIN_113
Hex2[4]	Output	PIN_114
Hex2[3]	Output	PIN_117
Hex2[2]	Output	PIN_109
Hex2[1]	Output	PIN_110
Hex2[0]	Output	PIN_111
Hex3[7]	Output	PIN_118
Hex3[6]	Output	PIN_122
Hex3[5]	Output	PIN_123
Hex3[4]	Output	PIN_124
Hex3[3]	Output	PIN_125
Hex3[2]	Output	PIN_119
Hex3[1]	Output	PIN_120
Hex3[0]	Output	PIN_121
HexCode0[3]	Input	PIN_52
HexCode0[2]	Input	PIN_50
HexCode0[1]	Input	PIN_51
HexCode0[0]	Input	PIN_49
HexCode1[3]	Input	PIN_48
HexCode1[2]	Input	PIN_44
HexCode1[1]	Input	PIN_45
HexCode1[0]	Input	PIN_43
Incremental[1]	Input	PIN_80
Incremental[0]	Input	PIN_81
Switch[7]	Input	PIN_55
Switch[6]	Input	PIN_58
Switch[5]	Input	PIN_60
Switch[4]	Input	PIN_62
Switch[3]	Input	PIN_66
Switch[2]	Input	PIN_68
Switch[1]	Input	PIN_70
Switch[0]	Input	PIN_72

Name	Input/Output	Pin
Bargraph0[7]	Output	PIN_139
Bargraph0[6]	Output	PIN_138
Bargraph0[5]	Output	PIN_137
Bargraph0[4]	Output	PIN_134
Bargraph0[3]	Output	PIN_133
Bargraph0[2]	Output	PIN_132
Bargraph0[1]	Output	PIN_131
Bargraph0[0]	Output	PIN_130
Bargraph1[7]	Output	PIN_3
Bargraph1[6]	Output	PIN_2
Bargraph1[5]	Output	PIN_1
Bargraph1[4]	Output	PIN_144
Bargraph1[3]	Output	PIN_143
Bargraph1[2]	Output	PIN_142
Bargraph1[1]	Output	PIN_141
Bargraph1[0]	Output	PIN_140
Button[7]	Input	PIN_53
Button[6]	Input	PIN_57
Button[5]	Input	PIN_59
Button[4]	Input	PIN_61
Button[3]	Input	PIN_63
Button[2]	Input	PIN_67
Button[1]	Input	PIN_69
Button[0]	Input	PIN_71
Buzzer	Output	PIN_4
Clock	Input	PIN_18
Frequency	Input	PIN_20
FTDI_nCTS	Output	PIN_5
FTDI_nRTS	Input	PIN_6
FTDI_RXD	Output	PIN_7
FTDI_TXD	Input	PIN_8
GPIO[11]	Output	PIN_127
GPIO[10]	Output	PIN_73
GPIO[9]	Output	PIN_74
GPIO[8]	Output	PIN_75
GPIO[7]	Output	PIN_76
GPIO[6]	Output	PIN_77
GPIO[5]	Output	PIN_79
GPIO[4]	Output	PIN_94
GPIO[3]	Output	PIN_95
GPIO[2]	Output	PIN_106
GPIO[1]	Output	PIN_107
GPIO[0]	Output	PIN_129
Hex0[7]	Output	PIN_84
Hex0[6]	Output	PIN_88
Hex0[5]	Output	PIN_89
Hex0[4]	Output	PIN_91
Hex0[3]	Output	PIN_93

Рис 9.15. Відповідність контактів для MAX® V – 5M1270Z CPLD

Крок 4. Повторна компіляція програми та отримання.pof файлу для подальшого програмування плати швидкого прототипування.

Командою, що відповідає за розробку та тестування програмного забезпечення для вбудованих систем (рис. 9.16) були розроблені навчальні та методичні матеріали для використання в міжнародному проекті ICo-op [16, 18, 20].

Рисунок 9.16. Команда з тестування вбудованих систем

Розглянемо приклади завдань для розробки варіантів тестування для функціонального тестування.

Завдання 1

Розробити тест-кейс і модифікувати проект test1 у відповідності до завдання: змінити номер функціонального перемикача і кількість світлодіодів, що світяться відповідно до номера варіанту.

Тест-кейс test1

Унікальний ідентифікатор варіанта тестування – test1.

Короткий опис варіанта тестування – крайній справа перемикач sw [0] буде вмикати/вимикати крайній справа світлодіод ld [0].

Порядок виконання – ввімкнути плату, ввімкнути правий перемикач.

Вимоги – тест завантажений на плату, плата підключена до комп'ютера.

Критерій завершеності – при ввімкненому правому перемикачі світиться правий світлодіод, при вимкненому – не світиться.

Категорія тесту – тестування системних компонентів плати.
Автор – Іванов І. І.
Автоматизований – так.

Приклад програми

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.all;
entity test is
port (
pSW: in std_logic_vector (7 downto 0);
pLED: out std_logic_vector (7 downto 0);
pLED1: out std_logic_vector (7 downto 0)
);
end test;
architecture t1 of test is
begin
pLED (0) <= pSW (0);
pLED (1) <= pSW (0);
pLED (4) <= pSW (1);
pLED (5) <= pSW (1);
pLED1 (0) <= pSW (0);
pLED1 (1) <= pSW (0);
pLED1 (4) <= pSW (7);
pLED1 (5) <= pSW (7);
end architecture;
```

Завдання 2

Розробити тест-кейс і модифікувати проект test2 у відповідності до завдання: змінити номер функціонального перемикача і кількість світлодіодів, що світяться відповідно до номера варіанту.

Тест-кейс test2

Унікальний ідентифікатор варіанта тестування – test2.
Короткий опис варіанта тестування – вмикання крайнього справа перемикача sw [0] буде виводити на дісплей номер варіанту.
Порядок виконання – ввімкнути плату, ввмікнути правий перемикач.
Вимоги – тест завантажений на плату, плата підключена до комп'ютера.

Критерій завершеності – при ввімкненому правому перемикачі на 7 сегментному дисплеї світеться варіант користувача, при вимкненому – не світиться.
Категорія тесту – тестування системних компонентів плати.
Автор – Іванов І. І.
Автоматизований – так.

Приклад програми роботи з кнопками та дісплеєм

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity btn_test is
port (
pbtn: in STD_LOGIC_VECTOR (7 downto 0);
pHex0: out  STD_LOGIC_VECTOR (7 downto 0);
pHex1: out  STD_LOGIC_VECTOR (7 downto 0);
pHex2: out  STD_LOGIC_VECTOR (7 downto 0);
pHex3: out  STD_LOGIC_VECTOR (7 downto 0)
);
end btn_test;
architecture numbers of btn_test is
begin
if (pbtn = «000000001») then
pHex0<=»11110000»;
phex1 <=»00000000»;
phex2 <=»00000000»;
phex3 <=»00000000»;
elsif (pbtn=»00000010») then
pHex1<=»11110000»;
phex0 <=»00000000»;
phex2 <=»00000000»;
phex3 <=»00000000»;
end if;
end numbers;
```

ЛІТЕРАТУРА ДО ЧАСТИНИ 3

1. Arras, P. E-learning concept for the properties of materials remote study/ Arras, P.; Tabunshchyk, G.; Kozik, T.// IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing

Systems (IDAACS), 2013 Volume: 02, 2013, P3: 742–747. DOI: 10.1109/IDAACS.2013.6663024

2. Arras, P. E-learning environment for the remote study in material properties courses/ Arras, P., Kolot, Y., Galyna, T., Kozik, T // International Journal of Computing, 12 (3), 2014, 233–238.
3. Arras, P. Architectural Characteristics and Educational Possibilities of the Remote Laboratory in Materials Properties/ P. Arras, G. Tabunshchyk, Ye. Kolot, B. Tanghe// REV2014 Conference26–28 February 2014, Porto, Portugal, PP. 94–97
4. Arras, P. Iterative Pattern for the Embedding of Remote Laboratories in the Educational Process / P. Arras, K. Henke, G. Tabunshchyk, D. V. Merode // 12th International Conference on Remote Engineering and Virtual Instrumentation (REV 2015) 25–28 February 2015, Bangkok, Thailand, PP. 52–55 (DOI 978–1–4799–7838–0/15)
5. Henke, K. Using Interactive Hybrid Online Labs for Rapid Prototyping of Digital Systems/ G. Tabushchyk, H-D Wuttke, St. Ostendorff, T. Vietzke //REV2014 Conference26–28 February 2014, Porto, Portugal, PP.48–59
6. Henke, K. Using Interactive Hybrid Online Labs for Rapid Prototyping of Digital Systems/G. Tabushchyk, H-D Wuttke, St. Ostendorff, T. Vietzke // iJOE, Volume 10, Issue 5, 2014,– PP. 57–62
7. Kozik, T. Techniques and tools for virtual and remote experiments/ Kozik T., Arras P., Tabunshchyk G. // Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій: зб. тез доп. VII міжнар. наук.– практ. конф. (Запоріжжя, 17–19 вересня 2014 р.) .– Запоріжжя: ЗНТУ, 2014.– С. 112–113.
8. «JTAG Boundary – Тестування сканування в MAX V пристроїв» http://www.altera.com/literature/hb/max-v/mv51008.pdf
9. JTAG Boundary – Тестування сканування для Cyclone IV пристроїв http://www.altera.com/literature/hb/cyclone-iv/cyiv-51010.pdf
10. Modelling, diagnostics and testing of digital systems [Available electronically] / INTUIT. Access mode: http://www.intuit.ru/studies/courses/3440/682/info [RU]

11. Yu. Skobtsov. Logical modelling and testing of digital systems / Yu. Skobtsov,. V. Skobtsov.– Donetsk: IPMM NASU, DonNTU, 2005.–436c [RU]
12. Брагина, Т. И. Информационная технология риск-ориентированного оценивания функциональности web-ориентированных систем /Т.И. Брагина, Г. В. Табунщик // Системи обробки інформації. Вип.2 (118) .– 2014.– С. 245–252.
13. Брагина, Т. И. Риск-ориентированный метод тестирования интегрированных баз данных /Т.И. Брагина, Г. В. Табунщик // «Электротехнические и компьютерные системы».– Одесса, 2014.– № 13 (89). – С. 223–230.
14. Табунщик Г. В. Інженерія якості програмного забезпечення: навч. посіб. / Табунщик Г. В., Кудерметов Р. К., Брагіна Т. І.; Запоріз. нац. техн. ун-т.– Запоріжжя: Дике поле, 2013.– 173 с.
15. Табунщик, Г. В. Перспективи впровадження віртуальної інженерії у навчальний процес ЗНТУ/ Г. В. Табунщик, А. В. Пархоменко //Тиждень науки – 2014: зб. тез доп. щоріч. наук.– практ. конф. викладачів, науковців, молодих учених, аспірантів, студентів ЗНТУ (Запоріжжя, 20–25 квіт. 2014 р.). – Запоріжжя: ЗНТУ, 2014.
16. Притула А. В. Практично-орієнтовані методи викладання в галузі вбудованих систем/ Притула А. В., Пархоменко А. В., Табунщик Г.В // Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій: зб. тез доп. VII міжнар. наук.– практ. конф. (Запоріжжя, 17–19 вересня 2014 р.) .– Запоріжжя: ЗНТУ, 2014.– С. 216–217.
17. Шитикова Е. В. Оптимизация процесса исследовательских испытаний сложных технических систем/ Шитикова Е. В., Табунщик Г. В. // Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій: зб. тез доп. VII міжнар. наук.– практ. конф. (Запоріжжя, 17–19 вересня 2014 р.) .– Запоріжжя: ЗНТУ, 2014.– С. 252–253.
18. Щербак Н. В. Обучающий эксперимент для интерактивной образовательной среды /Щербак Н.В., Табунщик Г. В. //Сучасні проблеми і досягнення в галузі радіотехніки, телекомунікацій та інформаційних технологій: зб. тез доп. VII міжнар. наук.– практ.

конф. (Запоріжжя, 17–19 вересня 2014 р.) .– Запоріжжя: ЗНТУ, 2014.– С.254–255.

19. Брагина, Т. И. Модель верификации web-ориентированных систем [Текст] / Т. И. Брагина, Г. В. Табунщик // ІНТЕРНЕТ-ОСВІТА-НАУКА-2014: тези доп. 19-ої міжнар. наук.– практ. конф., ІОН–2014, жовтень 14–17, 2014.– Вінниця: ВНТУ, 2014.– С. 23–25.
20. Табунщик Г. В. Нові технології в підготовці фахівців з вбудованих систем / Г. В. Табунщик, А. В. Пархоменко // ІНТЕРНЕТ-ОСВІТА-НАУКА-2014: тези доп. 19-ої міжнар. наук.– практ. конф., ІОН–2014, жовтень 14–17, 2014.– Вінниця: ВНТУ, 2014.– С.248–250.

CONTENTS

АВТОРЫ/AUTHORS

ПАРХОМЕНКО Анжеліка Володимирівна, кандидат технічних наук, доцент кафедри програмних засобів Запорізького національного технічного університету. Випускниця Запорізького машинобудівного інституту ім. В. Я. Чубаря. Захистила кандидатську дисертацію на тему «Розробка комплексних моделей елементної бази мікроелектронної апаратури для систем автоматизації проектування». Автор більш ніж 100 наукових публікацій та навчально-методичних робіт, у тому числі 1 навчального посібника з грифом Міністерства освіти та науки України, 2 авторських свідоцтв. Основні напрямки наукових досліджень: вбудовані системи управління рухомими об'єктами; технології та системи віртуальної та віддаленої інженерії; CAD/CAM/CAE-системи.

Anzhelika PARKHOMENKO, Ph.D., Associate professor of Software Tools Department of Zaporizhzhya National Technical University. A graduate of Zaporizhzhya Machine-Building Institute named V. Ya. Chubarya. Defended the PhD thesis on topic "Development of complex models of microelectronic equipment components for CAD". Author of over 100 scientific publications and educational works, including one textbook approved by Ministry of Education and Science of Ukraine, 2 Certificates for invention. Main research fields: Embedded Systems for moving objects control; technologies and systems of virtual and remote engineering; CAD/CAM/CAE-system.

ГЛАДКОВА Ольга Миколаївна, аспірантка кафедри програмних засобів Запорізького національного технічного університету. Закінчила магістратуру за спеціальністю «Програмне забезпечення систем». Автор 18 наукових публікацій, 1 авторського свідоцтва, 3 навчально-методичних робіт. Основні напрямки наукових досліджень: CAD/CAM/CAE системи; вбудовані системи управління рухомими об'єктами; Інтернет Речей; технології та системи віртуальної та віддаленої інженерії.

Olga GLADKOVA, a postgraduate student of Software Tools Department of Zaporizhzhya National Technical University. She has a master's degree in the specialty "Software systems". The author of 18 scientific publications, 1 Certificate for invention, three educational works. Main research fields: CAD/CAM/CAE systems; Embedded Systems for moving objects control; Internet of Things; technologies and systems of virtual and remote engineering;

ПОЛЯКОВ Михайло Олексійович, доцент, к.т. н., доцент кафедри електричних та електронних апаратів ЗНТУ. Після закінчення в 1974 р. ЗМІ (ЗНТУ) працював розробником систем управління в НДІ "Марс" в Росії. Науково-педагогічний стаж – близько 40 років в університетах Києва, Ульяновська, Запоріжжя, Дніпропетровська, автор понад 100 наукових праць. Сфера наукових інтересів: проектування систем управління з контролерами, моніторинг і прогнозування технічного стану трансформаторів.

Mykhailo POLIAKOV, associate professor, PhD., assistant professor at department of Electrical and electronic devices at ZNTU. After graduating in 1974 ZNTU worked as a developer of control systems at the Research Institute "Mars" in Russia. Scientific-teaching experience – 40 years in universities of Kiev, Ulyanovsk, Zaporozhe, Dnepropetrovsk, author of more than 100 scientific works. Research interests: design of control systems with controllers, monitoring and forecasting of technical states of transformers.

ЛАРІОНОВА Тетяна Юріївна, асистент кафедри «Електричні та електронні апарати», аспірант Запорізького національного технічного університету (ЗНТУ). Закінчила магістратуру ЗНТУ за спеціальністю «Електричні машини та апарати» в 2011-му році. Має 7 наукових праць. Коло наукових інтересів: проектування інтелектуальних систем керування з контролерами, розробка перетворювачів постійного струму, енергоефективність систем енергоживлення.

Tetiana LARIONOVA, teaching assistant at department of Electrical and electronic devices, postgraduate student at the Zaporozhye National Technical University (ZNTU). Graduated from ZNTU with a master degree in "Electrical machines and apparatuses" in 2011, has 7 scientific papers. Scientific interests: design of intelligent control systems with controllers, development of DC-DC converters, power efficiency of electricity supply systems.

ТАБУНЩИК Галина Володимирівна, к.т.н., доцент, професор Запорізького національного технічного університету. Закінчила Запорізький державний технічний унверситет за спеціальністю програмне забезпечення автоматизованих систем, захистила дисертацію на звання кандидата технічних наук за спецальністю 05.13.03 – системи і процеси керування. Автор понад 100 наукових праць. Наукові інтереси – інженерія програмного забезпечення, верифікація інформаційних систем, програмування вбудованих систем, керування ризиками.

Galyna TABUNSHCHYK, PhD, Prof of Software Tool Department of Zaporizhzhya National Technical Univerity. Graduated from Zaporizhzhya National Technical University with speciality Software Engineering, in 2004 finished PhD work in control systems and process. Have more than 100 scientific works. Scientific interests – Software Engineering, System Verification, Embeded Systems, Risk Management.

КАПЛІЄНКО Тетяна Ігорівна, старший викладач кафедри програмних засобів Запорізького національного технічного університету. Закінчила магістратуру за спеціальністю «Програмне забезпечення автоматизованих систем», аспірантуру за спеціальністю «Інформаційні технології». Автор 33 наукових публікацій, 2 авторських свідоцтв і одного патенту. Основні напрямки наукових досліджень: аналіз та верифікація якості програмного забезпечення; керування програмними проектами; керування ризиками.

Tetiana KAPLIIENKO, a senior lecturer of Software Tools Department at Zaporizhzhya National Technical University. She has a master's degree in "Automatic systems software" and has finished the postgraduate course in "Information technology". She is the author of 33 scientific publications, 2 Certificates for invention and 1 patent. Main research fields: the analysis and verification for the software quality; software design managements; risks management.

Appendix 1. Using Interactive Hybrid Online Labs for Rapid Prototyping of Digital Systems

K. Henke[1],G. Tabunshchyk[2],H.-D. Wuttke[3],T. Vietzke[4], St. Ostendorff[5]

iJOE Volume 10, Issue 5, 2014
(http://dx.doi.org/10.3991/ijoe.v10i5.3994)

Abstract

The Ilmenau Interactive Hybrid Online Lab offers several fields of application. In this article an enhancement of the existing Online Lab will be described to provide additional functionalities for a Web-based rapid prototyping of digital systems as well as the Web-based verification of such systems. For this new operation mode of the online lab a special rapid-prototyping board for digital systems was developed to fulfill the design tasks of digital systems. All the components of the rapid prototyping board will be described in detail. The implemented online lab infrastructure allows to interconnect online labs and to exchange remote lab experiments among different universities worldwide.

Index Terms

control engineering education, laboratories, Web-based education, virtual and remote labs, Web-based design tools, distance learning, rapid prototyping.

1 Ilmenau University of Technology, Ilmenau, Germany
2 Zaporizhzhya National Technical University, Zaporizhzhya, Ukraine
3 Ilmenau University of Technology, Ilmenau, Germany
4 Ilmenau University of Technology, Ilmenau, Germany
5 Ilmenau University of Technology, Ilmenau, Germany

INTRODUCTION

In our contribution we would like to present an enhancement of the Ilmenau Interactive Hybrid[6] Online Lab to provide additional functionalities for a Web-based rapid prototyping of digital systems as well as the Web-based verification of such systems. Facilities of Hybrid Online Labs provide permanent online access for students and supervisors, and give possibilities to check different parts of the designs most easily. This gives possibilities to realize correct designs, to organize self-study process of the student more efficiently, to control student's work and to broaden the ways of communication in research work with companies. This solution is intended for the use in teaching materials dealing with the design of digital control systems and embedded systems – from the basics up to complex design tasks as well as within the newly established Tempus project "ICo-op – Industrial Cooperation and Creative Engineering Education based on Remote Engineering and Virtual Instrumentation", founded by the European Commission of the program "Tempus", Grant No 530278TEMPUS-1-2012-1-DE-TEMPUS-JPHES [1].

The main topic of this Tempus project is to empower university-enterprise partnerships in Armenia, Georgia, and Ukraine by modernizing engineering education based on remote engineering and virtual instrumentation enhanced with transversal knowledge and competences at universities. It also offers new possibilities for bidirectional cooperation between universities and enterprises in education and research. The remote teaching of enterprise's staff or usage of universities remote labs for research purposes is one example for such cooperation. In order to achieve this, Ilmenau Interactive Hybrid Online Labs will be put in place at several project partners to develop common learning modules and to interchange these modules between all the partners based on EU best practices, partners' industry expertise, and knowledge of business demand of target countries.

Within the Tempus project a bilateral agreement between Ilmenau University of Technology and Zaporizhzhya National Technical University (ZNTU) was signed as well. The main purpose of this cooperation

[6] Hybrid online labs provide both remote experiments on real electromechanical models (physical systems) in the remote lab as well as simulation models of these physical systems in virtual labs [2].

is to engage in joint scientific work in the fields of Automation Systems, Computer and Software Engineering and Remote Engineering, using the Ilmenau Interactive Hybrid Online Lab for this purpose. The Institute of Computer Science and Radio Electronics of ZNTU is in close contact with the leading specialized enterprises of the Ukrainian region. But unfortunately it is tended to reduce the number of diplomas ordered by enterprises during last years. One of the factors is difficulties in communication between student, university supervisor and company supervisor. Further is considering how Hybrid Online Lab can solve this problem as well providing an effective remote tool for different level designs.

RAPID PROTOTYPING OF DIGITAL SYSTEMS

With the described enhancement of the functionality we want to provide exciting and challenging Web-based lab experiments in the field of digital system design. Course material starts with the basics of Boolean algebra, combinational logic and simple sequential circuits. This is followed by various minimization techniques for logical expressions, dynamic effects in combinational and sequential circuits and the design of digital control systems based on Finite State Machines (FSM). Finally we offer different methods and tool concepts to create, implement and validate digital systems to solve complex design tasks.

The goal is to introduce methods and technologies for a rapid prototyping of digital (embedded) control systems, especially the hardware oriented part of these systems. In connection with the mentioned Tempus project design engineers working in industry may also want to consider the offered learning scenarios of the project to handle modern CAE tools, logic simulation and logic synthesis using hardware description languages (e.g. VHDL), design hierarchy, and current generation of field programmable gate array (FPGA) technology – if they have not had previous experience with these rapidly evolving technologies. With modern logic synthesis tools and large FPGAs, more advanced designs are needed to present challenging laboratory projects.

ARCHITECTURE

The concept and the architecture as well as different fields of application of the Ilmenau Interactive Hybrid Online Lab were presented in various publications during the last years in detail, e.g. in [3, 4, 5]. A special rapid prototyping board for digital systems was developed for the new operation mode of the Ilmenau Interactive Hybrid Online Lab, presented in this paper to fulfill the mentioned design tasks. All the components of the rapid prototyping board will be described in detail in the following sections.

The Ilmenau Interactive Hybrid Online Lab

Figure 1 gives an overview about the Ilmenau Interactive Hybrid Online Lab. The infrastructure is based on a universal grid concept which guaranties a reliable, flexible as well as robust usage of this online lab. A more detailed description of this grid concept as well as the main components is presented in [5, 6].

The server side infrastructure (remote lab) consists of three parts:

an internal serial remote lab bus to interconnect all parts of the remote lab, realized as CAN bus, a bus protection unit (BPU) to interface the control units to the remote lab bus and to protect the bus from blockage, misuse and damage as well as a physical system protection unit (PSPU), which protects the physical systems (the electro-mechanical models in the remote lab) against deliberate damage or accidentally wrong control commands and which offers different access and control mechanisms.

The interconnection between the Web-control units and the selected physical systems during a remote lab work session (experiment) as well as the webcam handling is done by the lab server as part of the remote lab infrastructure.

During a running experiment, the client application will interact directly with the online lab grid infrastructure. Based on this infrastructure we offer several operation modes, described in detail in [3, 5]. In the following we would like to present an enhancement of the existing operation modes to provide additional functionalities for a Web-based rapid prototyping of digital systems as well as the Web-based verification of such systems.

The Rapid Prototyping Board

To fulfil all the mentioned design tasks for the design of digital systems, we have developed a special rapid prototyping board, shown in Figures 2 and 8. Over the last years, a number of interesting and challenging rapid prototyping boards were developed for educational purposes, the UP 3 from Altera [7] or development boards from Xilinx [8]. But most of them support very specific design tasks – not the whole spectrum, needed from the beginning (e.g. students in the first semester) to exciting and challenging design tasks in high-level courses.

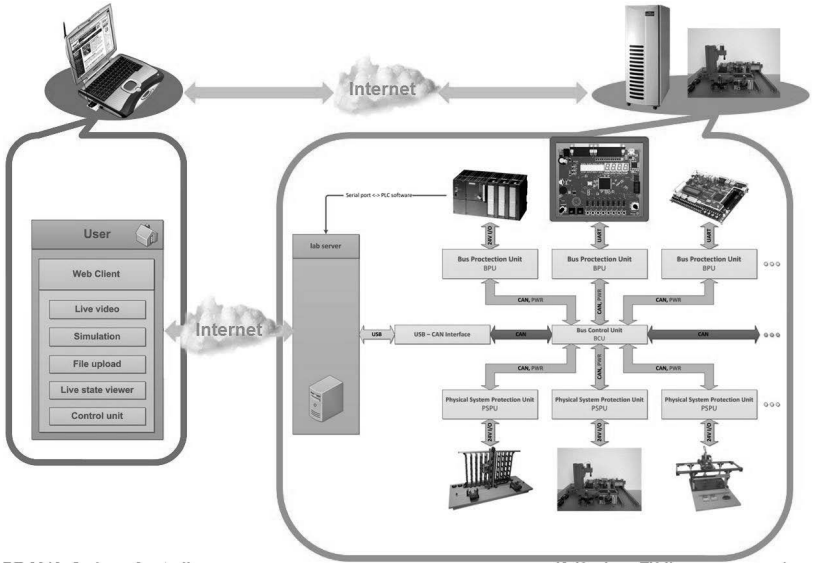


Figure 1. Overview of the Ilmenau Interactive Hybrid Online Lab infrastructure

Our rapid prototyping board is based on the MAX® V 5M1270Z CPLD from Altera [9]. With its mix of low price, low power, and new features, the MAX V CPLD family delivers the market's best value. Featuring a unique, non-volatile architecture and one of the industry's largest density CPLDs, MAX V devices provide robust new features at up to 50 percent lower total power compared to competitive CPLDs. The MAX V architecture integrates previously external functions, such as flash, RAM, oscillators, and phase-locked loops [9].

MAX V CPLDs are supported by the free Quartus® II Web Edition development software. With Quartus II software, students will get productivity enhancements resulting in faster simulation, faster board bring-up, and faster timing closure [10].

Finally, students can use the free Quartus II simulator tool QSim (with an integrated waveform editor). QSim is a graphical user interface (GUI) that is used to run simulations and launch the Waveform Editor. Also, QSim is used to set whether the simulation should be a functional or timing simulation. The Vector Waveform Editor is used to draw the test input signals for the simulation and select which signal should be shown in the simulation results [11]. In High-level courses students can use the more powerful simulation tool ModelSim® to validate the design. ModelSim supports behavioral and gate-level simulations, including VHDL testbenches [12].

Besides the MAX V CPLD the rapid prototyping board consists of the following components (see Figure 2):

- Input buttons:
- 8 push buttons:
- PB_7 .. PB_4 (active low)
- PB_3 .. PB_0 (active high)
- 2 rotary hexadecimal encoder
- 8 slide switches
- LED outputs:
- 4 7-segment displays (active low)
- 1 LED bar display with 8 LED (active low)

Other components:

- 10.000 MHz crystal oscillator
- Frequency synthesizer (200 Hz .. 10 kHz)
- Piezoelectric oscillator (for “sounds”)
- Incremental encoder
- UART (over USB)
- 25-pin SUB-D connector (to connect additional hardware boards, e.g. for a VGA adapter)

To program the MAX V CPLD on this stand-alone version of the rapid prototyping board, an additional FTDI chip [13] was placed on the board. The student has to connect the prototyping board via USB to his PC or laptop to upload the synthesized CPLD design to the FTDI chip at

the prototyping board. This chip will program the CPLD automatically via JTAG interface (see Figure 3).

The Online Lab Rapid Prototyping Mode

For a Web-based usage of the rapid prototyping board, an additional “interconnection” FPGA is placed on the bottom side of the PCB to realize the communication with the remote lab infrastructure (see Figure 4). All inputs of the board (buttons, synthesizer, incremental encoder and oscillator) have to be removed from the PCB and are replaced by a direct connection to the outputs of the “interconnection” FPGA, which will set all input signals according to the user’s input via the user interface at the student’s client PC. The generated outputs of the prototyping board can be directly read by the “interconnection” FPGA without removing any LED. The FPGA is interconnected to the BPU within the remote lab infrastructure.

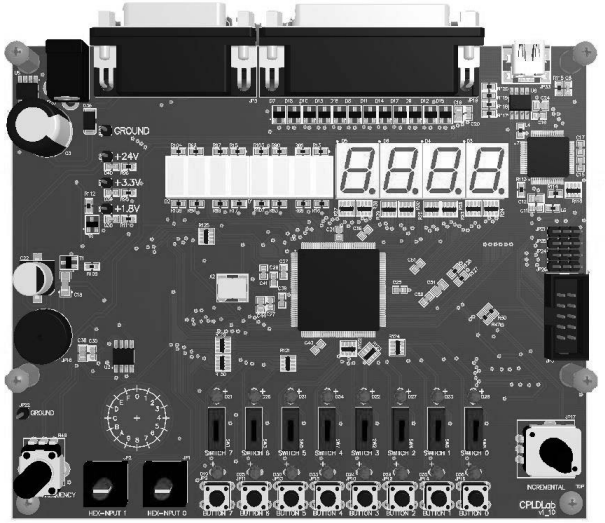


Figure 2. Rapid prototyping board (schematic view)

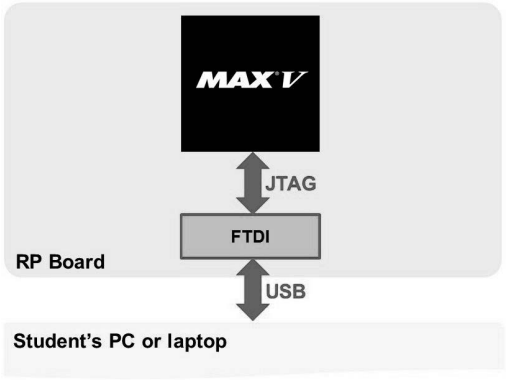


Figure 3. Programming of the stand-alone rapid prototyping board

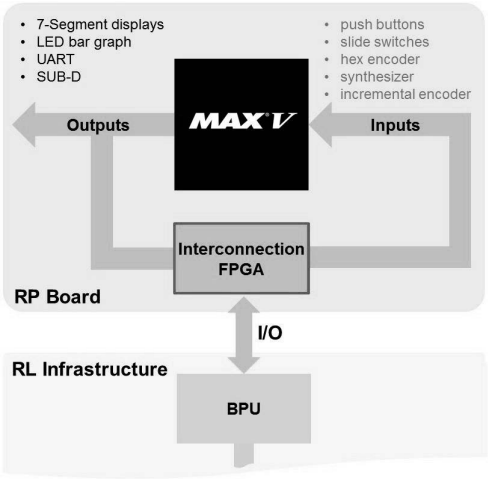


Figure 4. Interfacing the rapid prototyping board to the remote lab

To program the MAX V CPLD on the prototyping board in the remote lab via the Internet, the existing remote lab infrastructure can be used, as shown in Figure 5. The student has to upload his synthesized design (FPGA programming file) via the RIA (Rich Internet Application – a consistent enhancement of previous Java applets – see next section) on his client PC at home to the remote lab. Then the remote lab server will forward the data to the corresponding BPU (compare Figure 1). The BPU in turn will program the connected CPLD automatically via JTAG.

The Web-based User Interface

The increasing capacity of wireless communication and the growing number of mobile devices (e.g. smartphones and tablets) on the one hand as well as modern Internet technologies like JavaScript, HTML5 and Web Sockets on the other hand provides new possibilities and challenges in the area of mobile learning. Therefore a realization as HTML5 RIA was chosen for the Web-interface. Figure 6 gives an impression of this Web-interface.

By using the client‘s Web-interface, the student is able to upload the synthesized CPLD code of the design to program the CPLD on the rapid prototyping board (automatically in the remote lab – see section III.C) and to handle the whole lab procedure. This Web-interface allows the student to manipulate all the inputs of the rapid prototyping board virtually (slide switches, hex coding switches, pushbuttons and incremental encoder).

For the look-and-feel of the RIA, we use a visual model of the prototyping board (on the upper left side). All the inputs are realized as HTML5 control elements and can be activated via a mouse interactively. Changes are immediately sent to the rapid prototyping board in the remote lab and the corresponding results are displayed again inside the visual model. Furthermore a webcam will be used to observe the rapid prototyping board (on the upper right side) to watch the results of the user’s actions directly as reaction in the remote lab.

FIELDS OF APPLICATION

In this section possible fields of application of the developed rapid prototyping platform will be discussed.

Following the design flow for digital systems, students have to use common design tools to implement their control tasks. As mentioned in Section III.B they have to use Altera’s development system (Quartus II, simulation tools).

After synthesizing the bit file as shown in Figure 7, students can use the prototyping board as stand-alone system (see Section IV.A) or Web-based rapid prototyping system (see Section IV.B and IV.C).

Stand-alone Rapid Prototyping System

The simplest way to use the designed hardware platform is as a stand-alone solution for the rapid prototyping of digital systems (without Internet connectivity), shown in Figure 8. This application is not the focus of this article and therefore not discussed in detail.

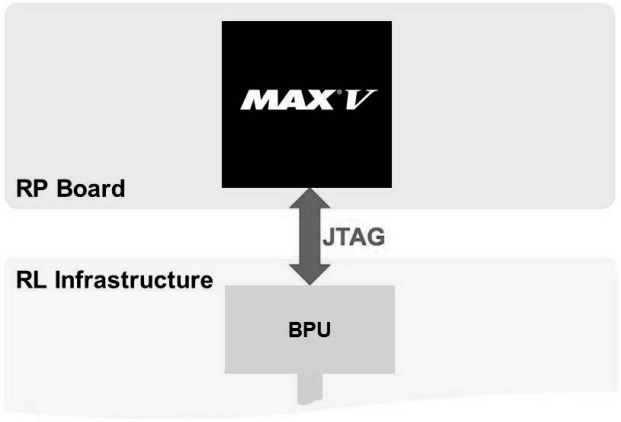


Figure 5. Web-based programming of the rapid prototyping board via the remote lab

Figure 6. Web-interface of the rapid prototyping board (RIA)

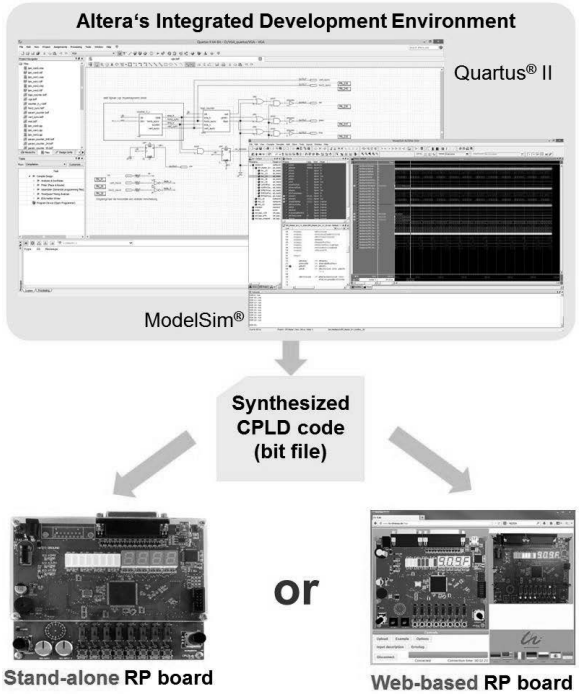


Figure 7. Using Altera's IDE and the rapid prototyping board for the design of digital control tasks

Web-based Prototyping of Digital systems

By using the mentioned Altera's development system, students are able to specify their design via:

- Text based design methods,

Here the student can enter his design by means of logical equations, truth tables or hardware description languages (AHDL, VHDL or Verilog), as shown in Figure 9.

- Graphically based design methods,

The student can use block or schematic diagrams to input his design, as shown in Figure 10.

- Integrated FSM editors.

In case of sequential design tasks he can directly enter the derived automaton graph (or graphs of parallel automata) with the built-in FSM editor, as shown in Figure 11.

The editor itself generates VHDL code for the further design steps.

Furthermore, students can specify, describe, implement and verify different digital systems using “self-made” IP core libraries (e.g. digital control systems, serial communication modules, robot sensors control, models of RISC processors, etc.).

After specification of the given task, students have to simulate their design. They can choose between different simulation tools within Altera’s development system, e.g. ModelSim for challenging designs in high-level courses.

Once the design is completed and error free, the student can upload the synthesized design to the remote lab, program the CPLD on the prototyping board (as described in Section III.C) and can test his solution using the rapid prototyping platform.

Web-based Validation of Digital Systems

Another use case of the rapid prototyping board is to identify the function of a given design (black box) or to find malfunctions of a given well-known design.

The CPLD will be pre-programmed by the teacher and the student has no ability to reprogram the device.

By manipulating the inputs of the unknown black box (e.g. to enter all the input sets of a truth table or special input sequences), the student attempts to analyze the response (the real output signals) of the board to find out the function of the given design. The student has to enter his test vectors based on truth tables or input sequences one by one using the provided controls and observe the results.

It is planned to support the upload of a whole truth table or input sequence as a text file via the Web-interface of the prototyping board. In this case the student will be able to record the responses corresponding to the specified inputs creating a waveform file. This file can then be used for further investigation of the current design.

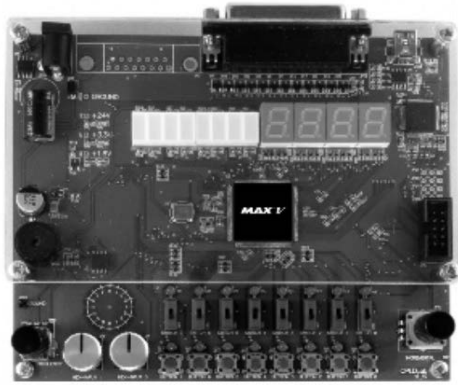


Figure 8. Rapid prototyping board

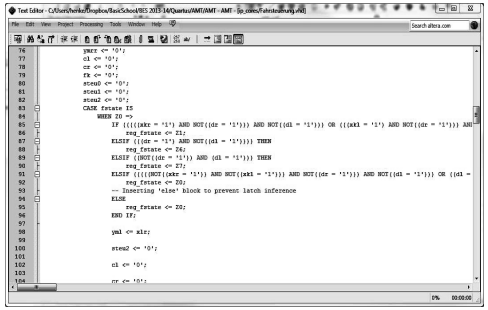


Figure 9. VHDL text-based specification

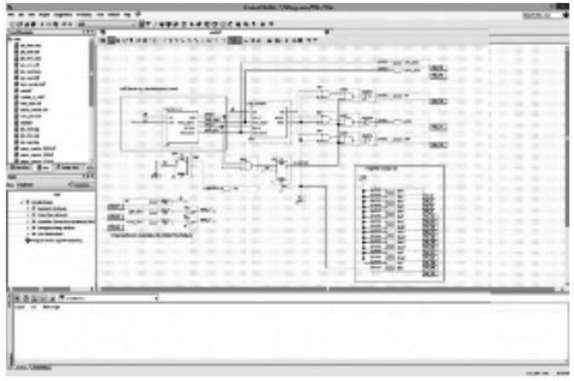


Figure 10. Block diagram-based specification

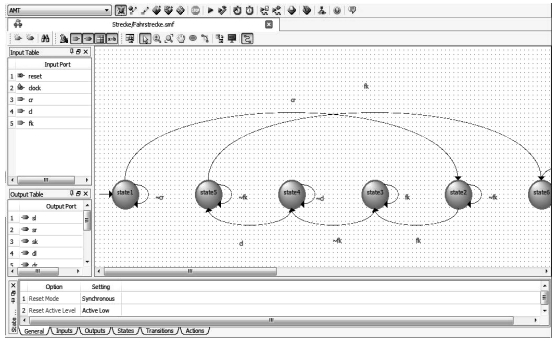


Figure 11. FSM-based specification

CONCLUSION

A universal hardware platform has been discussed, which can be used for rapid prototyping of digital systems. It is used for teaching purposes, from the basics to complex control design tasks including the ability for both local and web-based remote access.

Within several new European projects in the area of "Remote Engineering" e.g. ICo-op [1], eScience [14] and DesIRE [15] it is increasingly necessary to allow and organize a shared use of equipment. Therefor the main focus is a Web-wide usage of design tools and remote labs for the design of digital systems.

Using both, online tool support and laboratories, has the potential of removing the obstacles of cost, timeinefficient use of facilities, inadequate technical support and limited access to design and laboratory resources. This would also benefit students and researchers with special needs and students/researchers working from home, so they do not have to travel to their companies' facilities to perform their work. Even students/researchers working at their university's facilities can use remote specialized equipment at another university without travelling.

Students will learn more about the possibilities and limits of remote control and observation via Internet on practical examples.

The student – besides knowledge consolidation by executing design and practical experiments using these new Internet technologies – is forced to estimate the technologies critically.

ACKNOWLEDGMENT

The authors would like to acknowledge the work of Alexander Härtel and Tobias Fäth in the development of the Web-based programmer as well as the Web-based user interface (RIA).

REFERENCES

[1] ICo-op project Website: http://www.ico-op.eu.

[2] K. Henke, St. Ostendorff, H.-D. Wuttke, Th. Vietzke, Ch. Lutze, "Fields of Applications for Hybrid Online Labs", International Journal of Online Engineering (iJOE), Vol 9 (2013) Special Issue REV2013; pp. 20–30 , Vienna, May 2013.

[3] K. Henke, St. Ostendorff, St. Vogel, H.-D. Wuttke, "A Grid Concept for Reliable, Flexible and Robust Remote Engineering Laboratories", International Journal of Online Engineering (iJOE), Vol 8 (2012), pp. 42–49, Vienna, December 2012.

[4] K. Henke, St. Ostendorff, H.-D. Wuttke, Th. Vietzke, Ch. Lutze, "Fields of Applications for Hybrid Online Labs", Remote Engineering & Virtual Instrumentation, REV2013, Sydney, Australia, 06–08 February 2013.

[5] K. Henke, St. Ostendorff, H.-D. Wuttke, "A Flexible and Scalable Infrastructure for Remote Laboratories Robustness in Remote Engineering Laboratories", The Impact of Virtual, Remote and Real Logistics Labs ImViReLL2012 in: CCIS 282 pp. 13–24, Springer Verlag, DOI: 10.1007/978-3-642-28816-6_2, Bremen, Berlin, Heidelberg, February 2012. http://dx.doi.org/10.1007/9783-642-28816-6_2

[6] K. Henke, St. Ostendorff, St. Vogel, H.-D. Wuttke, "A Grid Concept for Reliable, Flexible and Robust Remote Engineering Laboratories", International Conference on Remote Engineering and Virtual Instrumentation, REV2012, Bilbao, Spain, July 04–06, 2012.

[7] Altera Corporation, http://www.altera.com. [8] Xilinx, Inc., http://www.xilinx.com.

[9] MAX V CPLD, http://www.altera.com/devices/cpld/max-v/mxvindex.jsp

[10] Quartus II Web Edition Software, http://www.altera.com/products/software/quartus-ii/subscription-edition/qts-se-index.html.

[11] Quartus II Simulator Tools for Education, http://www.altera.com/education/univ/software/qsim/unvqsim.html.

[12] ModelSim-Altera Software, http://www.altera.com/products/software/quartus-ii/modelsim/qts-modelsim-index.html

[13] http://www.ftdichip.com/Products/ICs/FT2232H.htm

[14] eScience Website, http://www.esience.org

[15] DesIRE Website, http://tempus-desire.thomasmore.be

AUTHORS

Karsten Henke is with the Ilmenau University of Technology, Faculty of Computer Science and Automation, Integrated Communication Systems Group, 98684 Ilmenau, Germany, POB 10 05 65 (e-mail: karsten.henke@tu-ilmenau.de).

Galina Tabunshchyk is with the Zaporizhzhya National Technical University, Software Tools Department, 69063 Zaporizhzhya, Ukraine, Zhukovskogo 64, (e-mail: galina.tabunshchik@gmail.com).

Heinz-Dietrich Wuttke is with the Ilmenau University of Technology, Faculty of Computer Science and Automation, Integrated Communication Systems Group, 98684 Ilmenau, Germany, POB 10 05 65 (e-mail: dieter.wuttke@tu-ilmenau.de).

Tobias Vietzke is with the Ilmenau University of Technology, Master Student in Computer Engineering at the Faculty of Computer Science and Automation, 98684 Ilmenau, Germany, POB 10 05 65. He obtained his BSc. in Computer Engineering in 2011 (e-mail: tobias.vietzke@tu-ilmenau.de).

Steffen Ostendorff is with the Ilmenau University of Technology, Faculty of Computer Science and Automation, Integrated Communication Systems Group, 98684 Ilmenau, Germany, POB 10 05 65 (e-mail: steffen.ostendorff@tu-ilmenau.de).

This work was supported in part by the European Commission within the program “Tempus”, “ICo-op – Industrial Cooperation and Creative Engineering Education based on Remote Engineering and Virtual Instrumentation”, Grant No 530278-TEMPUS-1-2012-1-DE-TEMPUSJPHES. Submitted 01 July 2014. Published as resubmitted by the authors 13 September 2014.

Appendix 2. Fields of Applications for Hybrid Online Labs

K. Henke,St. Ostendorff,H.-D. Wuttke,T. Vietzke and Ch. Lutze (Ilmenau University of Technology,Ilmenau,Germany)

iJOE – Volume 9, Special Issue 3, 2013
(http://dx.doi.org/10.3991/ijoe.v9iS3.2542)

Abstract

Based on a grid concept of an interactive hybrid online laboratory we will describe different fields of applications in different learning scenarios. The infrastructure is based on a universal grid concept which guaranties a reliable, flexible as well as robust usage of this online lab. By using the online lab, students are able to design control algorithms with different specification techniques to control electromechanical models in the online lab. Additionally, the reconfigurable rapid prototyping platform of the REAL system can be used to test all the taught topics of a given lectures in the field of digital system design. Finally, a special demonstration platform (a ball in a labyrinth on a balance plate) can be used to give the students a better feeling about the possibilities and limitations of remote control and observation via Internet and to evaluate these technologies critically. The implemented online lab infrastructure is based on the iLab architecture of the MIT, which allows to interconnect online labs and to exchange remote lab experiments among different universities worldwide.

Index Terms

control engineering education, laboratories, Web-based education, virtual and remote labs, Web-based design tools, distance learning.

INTRODUCTION

Our Integrated Communication Systems Group at the Ilmenau University of Technology has many years of experience in integrated hardand software systems and over 10 years of experience in dealing with Internetsupported teaching in the field of digital system design ([1] and [2]).

We have developed a new teaching concept, called “Living Pictures” [3] that we use in several phases of the learning process. Living Pictures are highly interactive Java applets that can be used for demonstrations as well as for experimental purposes, and also serve as tools in certain steps of the design process of digital systems. To complete the learning outcomes by own experiences, the students have to pass hands on examination in a lab. A task during this examination is to design an algorithm for a control system that controls one of various physical systems, for instance an electromechanical model of an elevator or a production cell.

For all students, hands-on experiences are important to deepen their knowledge about topics they learned during lectures (see Figure 1). At our university we offer an online laboratory, which gives the students the possibility to work on real physical systems without the need to stand in line at a lab or the need to take care of opening hours.

With our hybrid[7] online lab we want to offer the students a working environment that is as close as possible to a real world laboratory. Under real laboratory conditions disturbances can appear and lead to failures of the control algorithm that cannot be detected under virtual lab conditions.

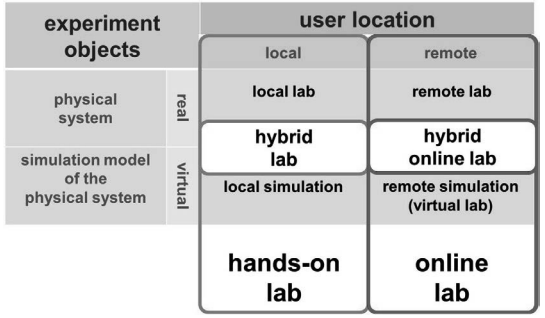


Figure 1. Classification of lab experiments

Therefore, it is important to include such real disruptive factors for a closer relation to practical conditions. Furthermore, the online lab should offer the students stimulus with regards to the design of safety critical control systems. By the conception of our lab (described in the next section), the virtual worlds were embedded within a real laboratory.

[7] Hybrid online labs provide both remote experiments on real electromechanical models (physical systems) in the remote lab as well as simulation models of these physical systems in virtual labs.

In principle, we would like to concentrate on giving students the chance to check their prepared control algorithms in real environmental conditions, which they can interactively influence themselves, and correct or modify the received results

- via Web-based simulation,
- via Web-based remote control of the existing physical system (that means before the tutorial course).

Currently, there are no common standards for the architecture of remote labs. This is the reason why universities develop their own remote lab solutions – suitable for their specific requirements, for example [4], [5], [6] and [7]. This handicaps a networking of different online labs among each other and also a usage by different institutions. Missing uniform interfaces prevent the programming of universal plug-ins and other software modules.

ARCHITECTURE OF THE REAL SYSTEM

In the following, we will describe a hybrid interactive online lab, supporting all the design steps for complex control tasks to control various electromechanical models the REAL system. Goal of the REAL (Remote and Applications Laboratory) system is to show new ways and chances of remote controlling and remote observation of real processes (e.g., in the fields of control engineering, robotics, tele-control engineering), dealing with the integrated and interactive usage of modern Internet and intranet technologies, like HTML5, jQuery, JavaScript etc. It was developed at the Department of Integrated Communication Systems at the Ilmenau University of Technology [8] – see Figure 2.

The implemented REAL infrastructure is based on the iLab Shared Architecture of the MIT (see Figure 3) which meanwhile is established as a standardized implementation for online laboratories and will be implemented in more and more locations all over the world. Furthermore, it allows to interconnect online labs and to exchange remote lab experiments among different universities worldwide.

As mentioned in many papers (e.g., [9], [10] and [11]), interactive online labs can open opportunities which allow an experimental approach for a wider audience and are also independent of opening hours of the laboratory rooms and their staff.

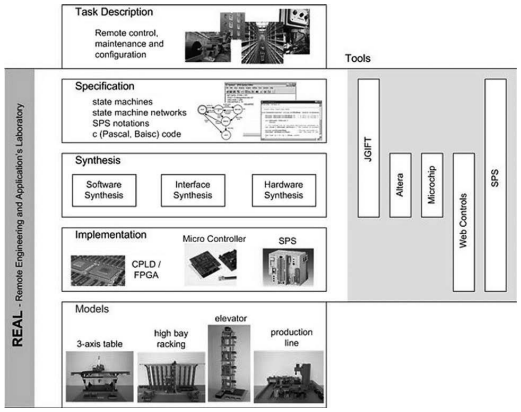


Figure 2. Overview of the REAL system

Interactive labs will be used, when:

- an experiment requires real-time processing or
- the user wants to observe the whole physical system (the electro-mechanical models) during the experiment or
- some parameters (e.g., input variables to a control system) need to be changed interactively during the experiment.

They offer various features like visualization and animation, which allows to observe and to test all the properties of the design. In connection with formal design techniques, simulation and prototyping are used to establish a foundation for the development of a reliable system design. To check the functionality of the whole design, some special simulation and validation features are included as integral part of the REAL system. This offers various possibilities for the execution of simulations, such as:

- usage of simulation models of the physical system for visual prototyping,
- step by step and parallel execution of these prototypes,
- visualization of the simulation process with the tools also used for specification,
- features for test pattern generation and
- code generation for hardware and software synthesis.

REAL offers a Web-based environment supporting the above mentioned features to generate and execute a design by using simulation models. An example will be given in Section III.A.

At any time the students have the chance to adjust their algorithms in case of faults. Therefore, they are able to achieve a fault free solution (a validated control algorithm) step by step. For more details see the previous publications [2], [12] and [13].

Our online lab is used for teaching practical lessons as well as giving hands-on experiences for the development of embedded electronics. This is not done using on-site lessons, but remote via the Internet. This has the advantage that courses can be offered internationally world-wide and gives students from different countries, speaking different languages, the same access and equal possibilities in the lab.

Figure 3. iLab Shared Architecture of the MIT (4)

Additionally, even for local students, the lab offers extended opening hours (twenty-four-seven) when compared to a regular lab. Besides the advantages for students, this also reduces the costs for academic teaching and improves the quality by offering more practical training possibilities.

One implementation challenge is to protect the physical systems in the lab against wrong control algorithms of students without defining too many design constraints. Students should be free in their decisions and develop own creative solutions. They can implement their own design strategies, therefore a reference design and a method to check the students' design against this reference is needed [14] to protect the physical system in the lab (see Figure 4).

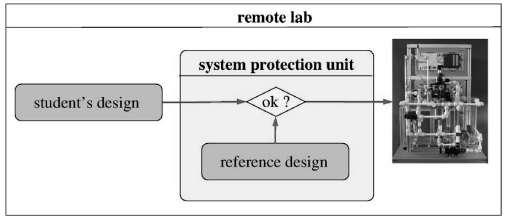


Figure 4. Observation of the student's design by the physical system protection unit

The reference design should be independent of the used control unit and the development tools. This is done by the protection unit of the physical system.

Figure 5 illustrates the grid architecture of the REAL system. The server side infrastructure (remote lab) consists of three parts:

- an internal serial remote lab bus to interconnect all parts of the remote lab,
- a bus protection unit to interface the control units to the remote lab bus and to protect the bus from blockage, misuse and damage as well as
- a physical system protection unit, which protects the physical systems (the electro-mechanical models in the remote lab) against deliberate damage or accidentally wrong control commands and which offers different access and control mechanisms.

For a more detailed description of this grid concept as well as of the main components see papers [15, 16].

The interconnection between the Web-control units and the selected physical systems during a remote lab work session (experiment) as well as the webcam handling is done by the lab server as part of the iLab architecture (see Figure 3).

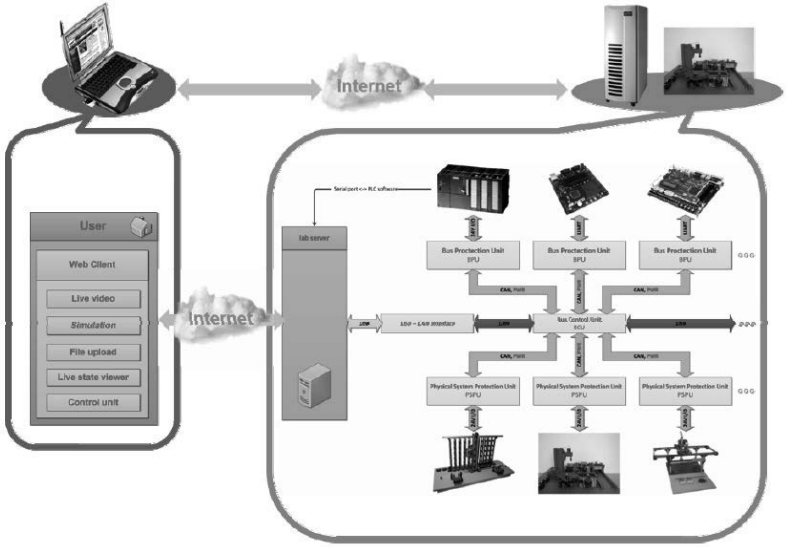


Figure 5. Grid architecture of the REAL system (server side) with Web-client

The iLab Service Broker is mainly responsible for the user management and time scheduling. All experiments are made available by this service and integrated into the iLab Cloud.

During a running experiment, the client application will interact directly with the online lab infrastructure as depicted in Figure 5 without any more access to the Service Broker. Details about the iLab architecture are beyond the scope of this paper. Please refer to [17] for further information.

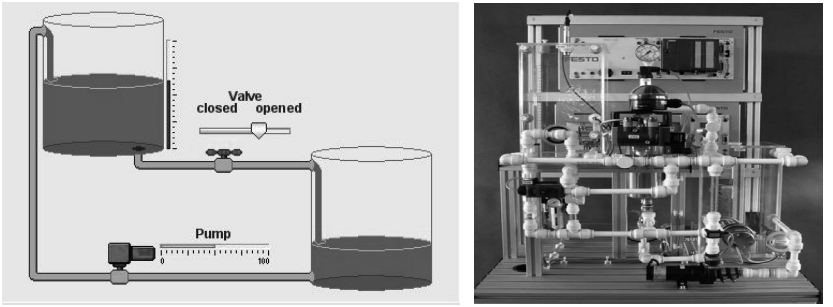


Figure 6. Simulation model and physical system of a water level control

FIELDS OF APPLICATIONS

Based on this flexible online lab structure we offer different operation modes to test the developed control task. Simulation and visual prototyping help to find functional errors. Before starting practical work on real systems, simulations and animations in “virtual worlds” are often used to verify the developed solutions. The behaviour of the physical system that should be controlled, as well as its environment, will be emulated as a simulation model. The student can influence this “virtual world” and analyze the caused reaction of his control algorithm. Figure 6 shows an example of such a simulation model.

These steps have to be executed until no more errors are detected. But there is an essential disadvantage in this method. Real disruptive factors (e.g., failure of single components, mechanical problems or process variations) cannot be recognized by the underlying virtual environmental model.

Generally, only a simulation of predetermined malfunctions is possible. After some time, all these effects are well known in the student’s

community. Unconsidered sources of errors lead to undetected failures of the control because the corresponding environmental situation was not simulated before [13]. That is why a fault free design algorithm finally should be tested on real physical systems (e.g., the water level control, shown in Figure 6) in the online laboratory as well. In the following we will describe possible operation modes of the REAL system based on the schematic view in Figure 5:

- Stand-alone Mode (visual prototyping)
- Remote Control Mode (via Web-client)
- Remote Control Mode (via control unit)
- Virtual Control Mode (visual prototyping)
- Virtual Control Mode (test mode)
- Local Control Mode (via control unit)
- Local Control Mode (manually)
- Rapid Prototyping Mode
- Visitor Mode

The complete schematic view (client and server side) of the online lab architecture is shown in Figure 7.

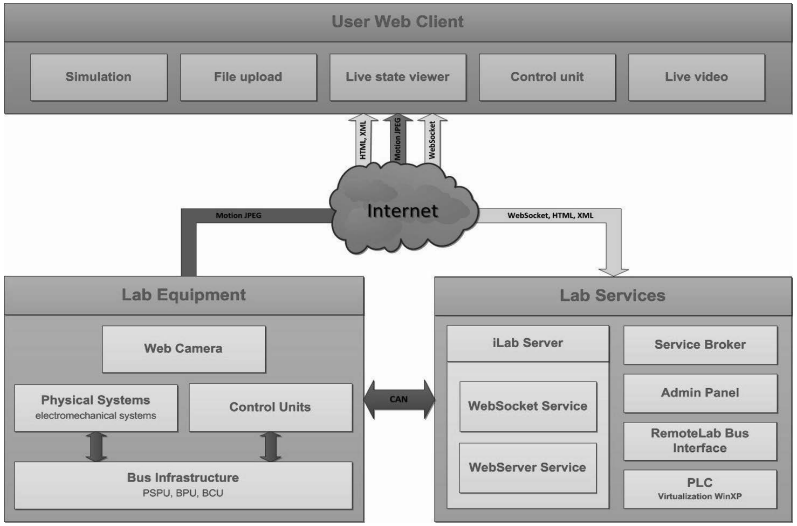


Figure 7. Schematic view of the remote lab components

Via the Web-client running on the student's PC at home the physical system (e.g., the electromechanical model of the water level control) can be controlled by using different control units (e.g., microcontroller, FPGA, PLC).

For the Web-clients modern Internet technologies like HTML5, jQuery and JavaScript can be used. These technologies make rich internet applications (RIAs) possible, which run in nearly any browser and on many terminal devices without needing any plugins.

These RIAs carry out many functions like animation, simulation, interpretation and control and therefore release the server from these tasks. The interaction with the user is speedup and can react faster and more direct.

By using this Web-interface, the student is able to:

- handle the experiment (e.g., start, stop, reset),
- change environmental variables if necessary and
- watch the experiment by manipulating environmental variables inside an I/O monitor or by observing the control of the physical system directly via a webcam.

Via an optional local control panel the student is able to manipulate the lab environment (e.g., the water level in the tank) or the actuators (e.g., the pump) when working on-site. Heart of this architecture is the physical system protection unit, which is described in detail in [15, 16].

Stand-alone Mode – Visual Prototyping

In case of using finite state machines (FSM) for specification, based upon an automaton graph, a student can use the JGIFT design environment [20] of the REAL system.

Figure 8 gives an impression of the verification and simulation features of the Web-based environment of the REAL system. The simulation model will be driven directly through the I/O signals by the control algorithm running on the embedded interpreter within the applet.

Assuming the student achieved a validated design, he gets the required next state and the output equations. By accessing the Web-browser interface of the REAL system, he is able to enter his algorithm (the received equations), handle the laboratory experiment (e.g., start, stop, reset) and change environmental variables if necessary. The control al-

gorithm is executed by an interpreter running inside the student's client PC (e.g., implemented as a HTML 5 RIA). No Internet connectivity to the laboratory and the physical systems in lab is necessary for this mode (see Figure 9).

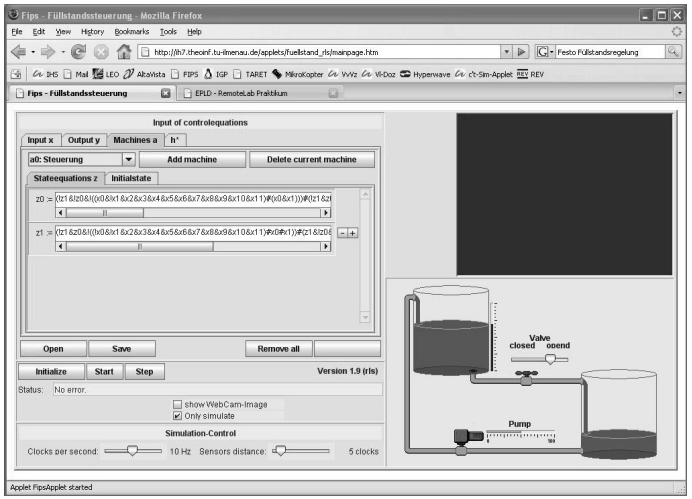


Figure 8. Offline regulation of the water level control (without Internet)

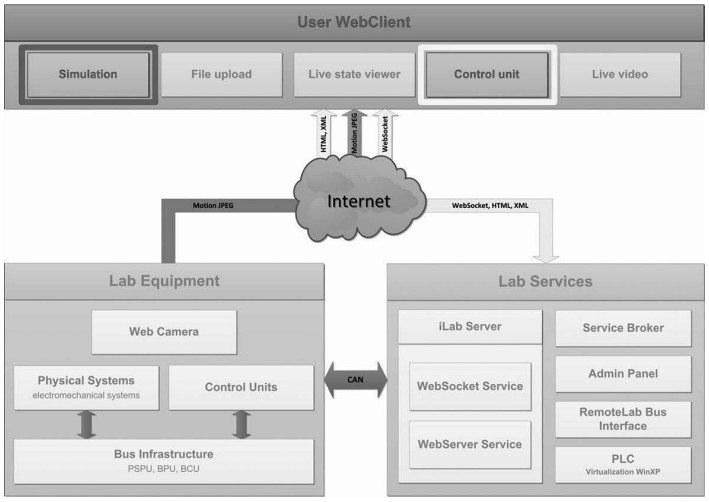


Figure 9. Stand-alone mode for visual prototyping

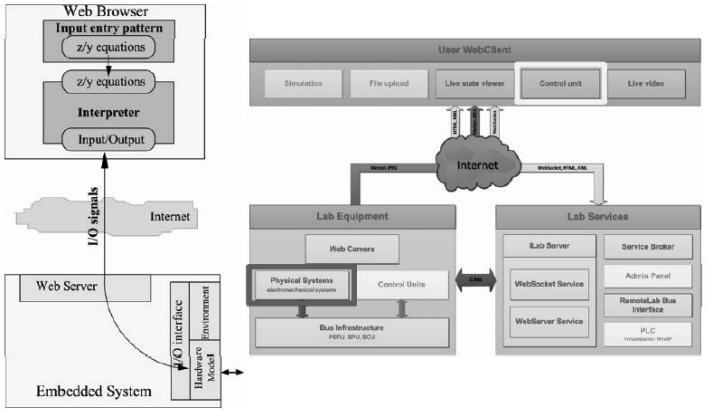


Figure 10. Remote control mode via a Web-client

Remote Control Mode – via Web-client

This operation mode is also for the FSM based specification based on equations. In this case, the physical system will be controlled via the Internet "from a distance" through the interpreter running inside the student's client PC. No additional control units in the remote lab are necessary (see Figure 10). In this case, only the input and output signals of the physical system will be transferred via Internet.

An example of such a Web-client is shown in Figure 11.

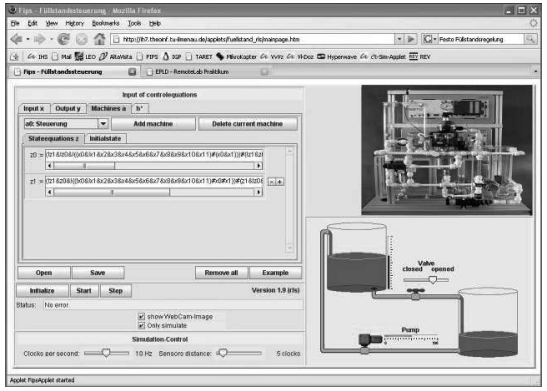


Figure 11. Online regulation of the water level control (with Internet)

Remote Control Mode – via Control Unit

This mode can be used to realize software or hardware oriented control tasks via connected microcontrollers or FPGAs as control units (see Figure 13).

Students can implement their control algorithm directly into a microcontroller for a software-oriented implementation. Therefore, they use common (non-commercial) development tools, for example MPLAB IDE and/or C18 C-compiler by Microchip [21], or AVR Studio by Atmel [22], to develop Assemblerand/or C-coded software projects. After compilation, the generated software control algorithm is transferred via REAL Web-interface to the remote lab, where the hex code is programmed into the microcontroller (see Figure 12).

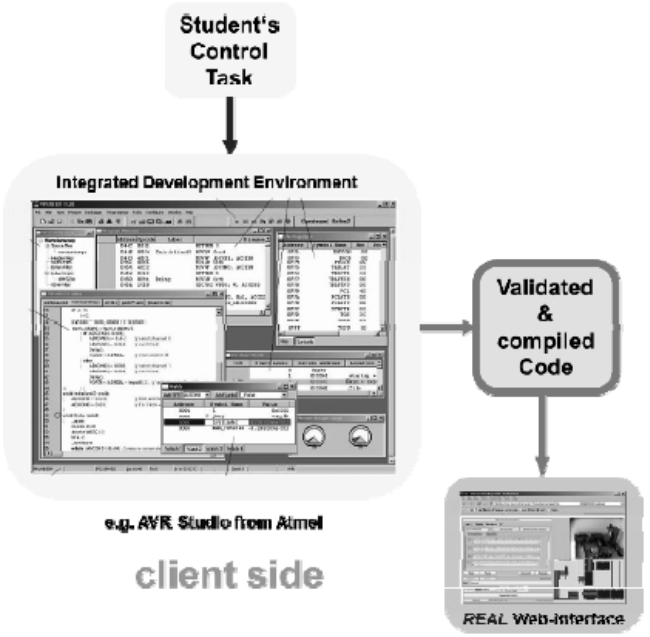


Figure 12. Software-oriented design of the control task

Now, the student can begin with his experiment, to check if his algorithm fulfills the requirements of the given control task.

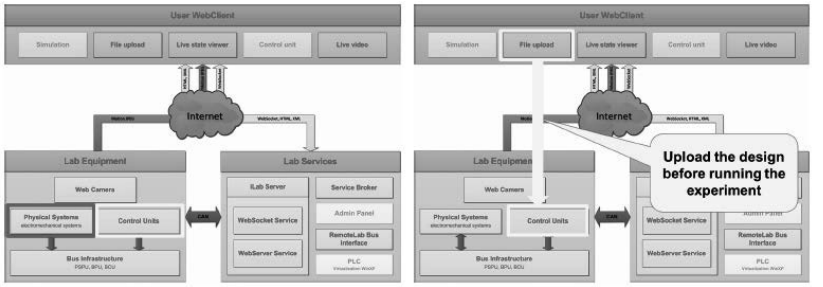


Figure 13. Remote control mode via a control unit

If a student prefers an exclusive hardware-oriented design using an FPGA and applying a hardware description language like VHDL as specification technique, he can prepare his design with common development tools, for example ISE, Quartus II, Diamond or others. The generated bit file is uploaded via the REAL Web-interface to the remote lab, where the FPGA will be programmed (see Figure 14). After programming the connected FPGA, the FPGA board operates as control unit for the designed control algorithm, and the student can start his experiment.

The student has the possibility to debug his design, by defining breakpoints depending on input/output signals from the physical system protection unit and stop the electromechanical model at certain sensor readings to validate the correct actor settings. It is also possible to do single step processing, by pausing the execution on every sensor/actor change.

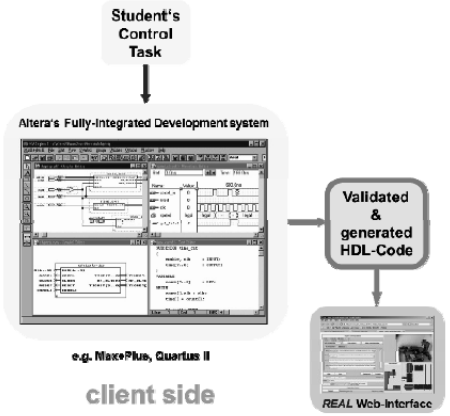


Figure 14. Hardware-oriented design of the control task

Virtual Control Mode – Visual Prototyping

This mode is comparable to the operation mode A (for visual prototyping). In this case, the simulation model is not connected to the interpreter, running inside the client, but via the Internet to the real control units which are running inside the remote lab. In this case, the student can test his prepared software or hardware oriented design on the corresponding control unit (microcontroller or FPGA) without the need for a real physical system – before he will use operation mode C (remote control mode – via control unit) to test his design task on the physical system in the remote lab (see Figure 15).

Furthermore it is possible to have many students working on control algorithms for the same electromechanical model without disturbing each other. They can test and validate their design before connecting it to real hardware. This can eliminate the need to install multiple instances of the same electromechanical model in one lab and therefore reduce the costs of this lab.

Besides this, it is also possible to emulate electromechanical models that are not available in the lab.

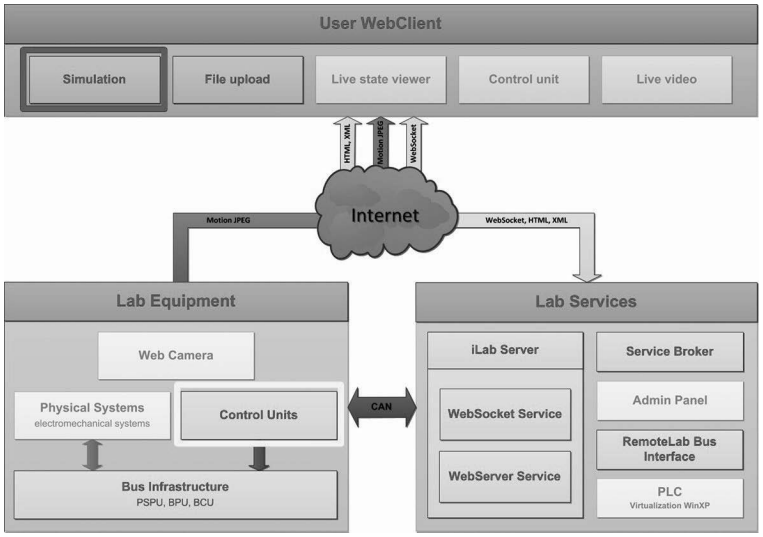


Figure 15. Virtual control mode for visual prototyping

Virtual Control Mode – Test Mode

This operation mode (see Figure 16) is mainly for debugging, testing and maintenance of the physical system protection unit.

By using this operation mode, it is possible to check the implemented reference design (as seen in Figure 4) in the physical system protection unit without the need to use a control unit or a physical system. This will be done by transmitting input and output pattern from the Web-client to the protection unit and by analysing its response.

Because this mode is not preferential to execute lab experiments, it is not explained in detail.

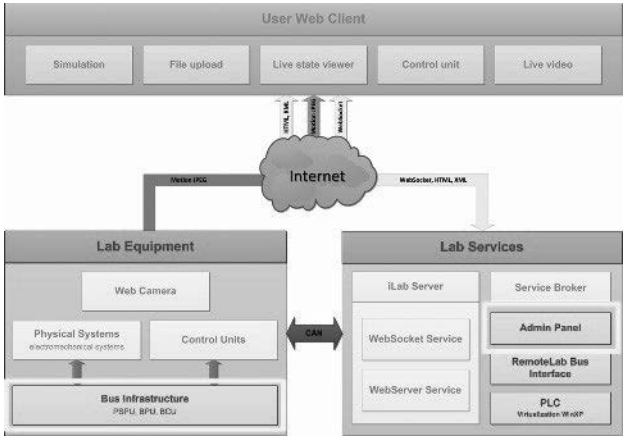


Figure 16. Virtual control mode to test the protection unit

Local Control Mode – via a Control Unit

Besides the possibility to work in the lab remotely, the following two operation modes explain the usage of the REAL system for on-site experiments or demonstrations.

During an on-site lab experiment students can observe the whole hardware setup as well as all the physical system and environmental variables directly in the lab room. In this case they can check their control task (running on a connected control unit). They have access to the whole lab setup via a connected control panel. Figure 17 illustrates this operation mode.

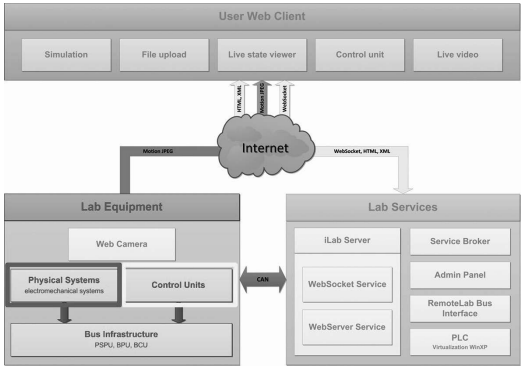


Figure 17. Local control mode via a control unit

In addition to its use for student experiments, this mode can also be used to demonstrate the remote lab on guided tours during open house presentations to inspire new students to study engineering courses.

Local Control Mode – manually

This operation mode (see Figure 18) can be used for demonstrations and maintenance of the connected electromechanical models.

By activating actuators (e.g., a water pump) manually without any control algorithm the sensor signals can be observed via the connected control panel.

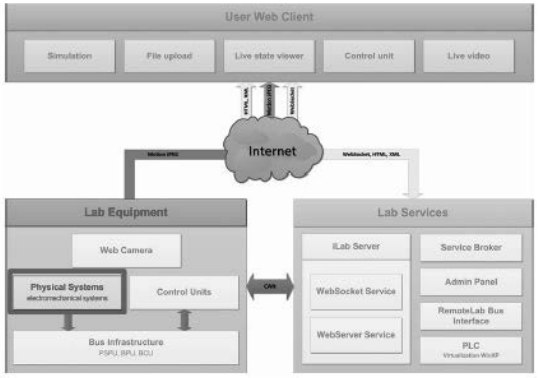


Figure 18. Manual local control

This mode will also be used for hands-on usage of the demonstration system “labyrinth on ball balance plate” (see Figure 19) or can be used for testing and debugging the electromechanical parts of the physical systems for service and inspection.

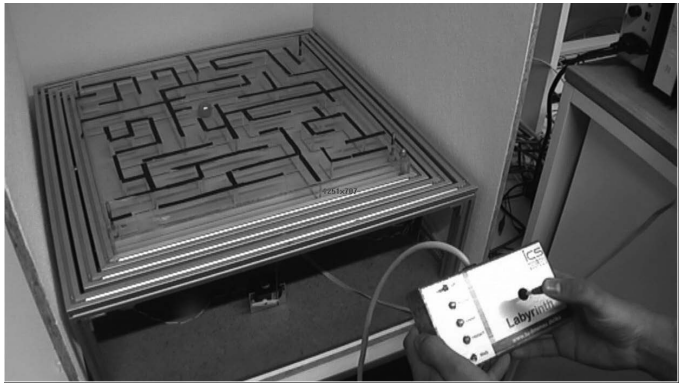


Figure 19. Labyrinth model

We would like to provide this game-like experiment especially for promotional means for open house presentations at the university to attract new students in engineering disciplines.

Rapid Prototyping Mode

For this special operation mode a rapid-prototyping board for digital systems was developed, which is directly connected to the serial remote lab bus. In this case, no physical system protection unit is needed.

By using the REAL Web-interface, the student is able to

- upload his/her design,
- program the FPGA and
- handle the lab procedure.

The applet allows the student to manipulate all the inputs of the rapid prototyping board (slide switches, hex coding switches, and pushbuttons). He can observe the outputs of the board (7-segment displays, row of LEDs) virtually inside the Java applet. Figure 21 gives an impression of the applet’s Web-interface including a “photo” (image) of the rapid prototyping board.

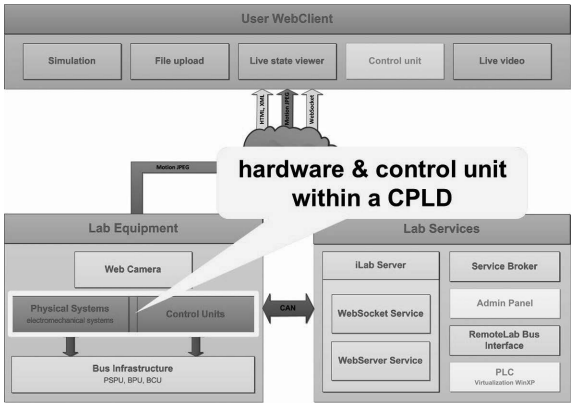


Figure 20: Rapid prototyping mode

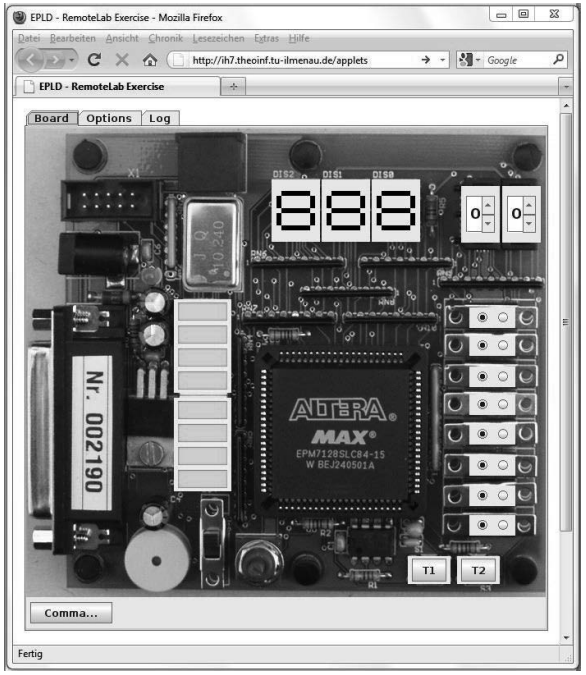


Figure 21. Web-interface of the Rapid Prototyping board

The manipulation of the input and output signals of the board are done virtually in the following way: For the look-and-feel of the applet, we use a "photo" of the board as background. All the inputs are realized as Java control elements and can be manipulated via the mouse interactively.

Changes are immediately sent to the rapid prototyping board and the corresponding results are displayed inside the applet. There are two general options to display the output results within the Web-interface:

- Representation by Java components

The graphical outputs are represented directly inside the "photo" using read-only Java components. Updates of the display only require information about the current state of the corresponding output signal. This enables students without fast Internet access to use the applet.

- Webcam based feedback

Another option is the use of a webcam to monitor the rapid prototyping board inside the lab room. The webcam image is replacing the background "photo" of the applet. This allows the user to watch the results of his/her actions directly as if present in the lab. The overlaying Java display components are not used in this option and therefore invisible.

Both options can be configured using custom user settings in the corresponding tab of the applet. All actions are documented in a separate Log Tab.

By using this Web-based rapid prototyping board the following application fields are possible:

Web-based Rapid Prototyping of Digital System

By using common design tools (e.g. MaxPlus+, Quartus II from Altera [23]), the student is able to specify his design via

- Text based design methods,

Here the student can enter his design by means of logical equations, truth tables or hardware description languages (like VHDL or Verilog).

- Graphically based design methods,

The student can use block or schematic diagrams to input his design.

- Integrated FSM editors.

In case of sequential design tasks he can directly enter the derived automaton graphs with the built in FSM editor.

The editor itself generates VHDL code for the further design steps.

Finally, students will specify, describe, implement and verify different digital systems using “selfmade” IP core libraries, e.g.

- Digital control systems
- Serial communication modules,
- Robot sensors,
- Model of a RISC processor.

Once the design is completed and error free, the student can test his solution using the described platform.

Web-based Verification of Digital Systems

Another use case of the rapid prototyping board is to identify the function of a given design (black box) or to find malfunctions of a well-known design. The CPLD will be programmed by the teacher, but the student has no ability to reprogram the device.

By manipulating the inputs of the unknown black box in order to find the malfunction (e.g. to enter all the input sets of a truth table or special input sequences), the student attempts to analyze the response (the real output signals of the board) to find out the function of the given design. The student has to enter his test vectors based on truth tables or input sequences one by one using the provided controls and observe the results.

It is planned to support the upload of a whole truth table or input sequence as a text file to the Webinterface of the prototyping board. In this case the student will be able to record the responses corresponding to the specified inputs creating a waveform file. This file can then be used for further investigation of the current design.

Visitor Mode

This mode of operation is intended for visiting the lab and passively taking part in experiments. It supplies a live video feed as well as a display of sensor and actor values from the electromechanical system of the selected experiment (see Figure 22).

Using the visitor mode does not require any login or booked time slot, but gives any interested person the possibility to view the lab and any

running experiment. This mode is also useful for tele-teaching, where a teacher wants to present an experiment to a student via Internet.

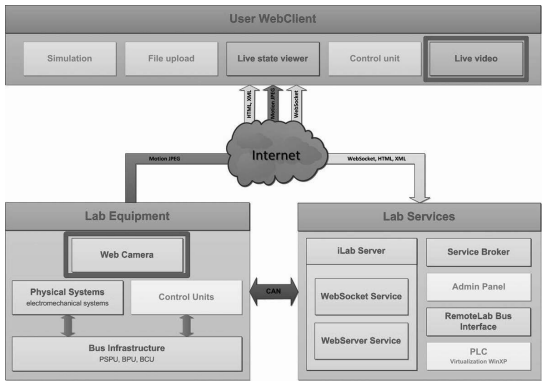


Figure 22. Visitor mode

CONCLUSION AND FORESIGHT

We have discussed different operation modes of the REAL system for various fields of applications – based on a new flexible grid-based online lab structure. In addition to simulation-based experiments, on-side and off-side experiments with real physical systems can be offered for students as well.

Besides the features already mentioned in this article, even more functionality can be added using the new concept of having a Web-based protection unit that checks the user input against a reference model. This protection unit can be connected to a learning management system like “moodle” to forward any experimental results of the user. For an effective usage of the REAL system within learning management systems, the reference design and a method to check the student’s design against this reference design step by step will be traced by the LMS. Figure 23 shows this idea.

The increasing capacity of wireless communication and the growing number of mobile devices (e.g. smartphones and tablets) on the one hand as well as modern Internet technologies like JavaScript, HTML5 and Web Sockets on the other hand provide new possibilities and challenges in the area of mobile learning (see Figure 24). In [24] a first Android client application for the iLab Shared Architecture is described.

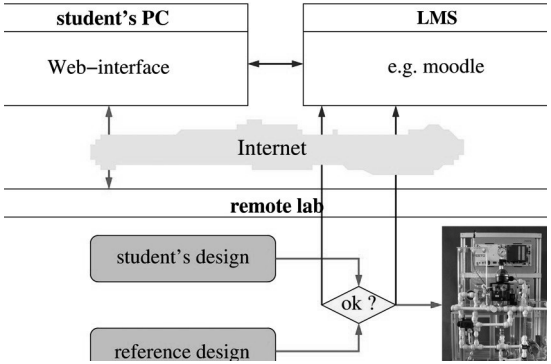


Figure 23. Observation of the student's design under LMS control

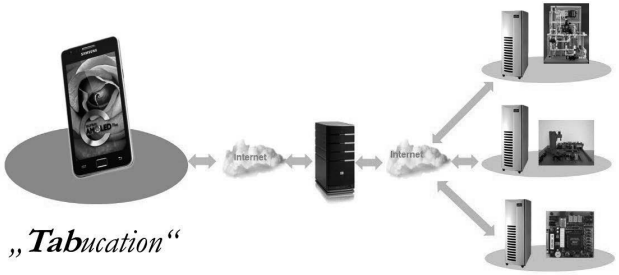


Figure 24. Mobile learning by using modern wireless technologies

Our Integrated Communication Systems Group at the Ilmenau University of Technology is involved in different national and international e-Learning projects (e.g., [25]) in which it is increasingly necessary to allow and organize a shared use of equipment. That is why, the main focus of the REAL system is

- a Web-wide usage of different design tools and control units to control different physical systems in the lab room,
- a robust, fault-protected access to any connected physical system,
- an LMS-coupling for all control units and physical systems used in the remote lab as well as
- a worldwide interchange of online experiments with other universities by interconnecting the iLab Service Broker to an "iLab cloud" (see Figure 25), as proposed by the iLab Europe consortium, where we are involved, as well [26], [27].

All these requirements can be fulfilled using the concept and the infrastructure presented in this paper.

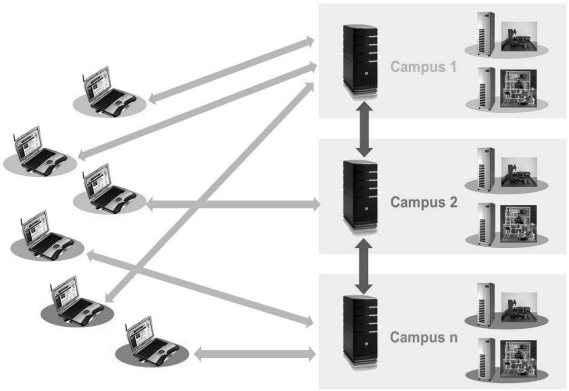


Figure 25. Interconnection of various iLab Broker to an "iLab cloud"

Future work will be concentrated on analyzing the student's behavior during the experiments and developing an adaptive feedback and assessment system to support the students in a better way.

ACKNOWLEDGMENT

The authors would like to thank Silvia Krug, Stephan Simon and Stefan Vogel for their work within the REAL framework.

Parts of the REAL project are supported by the society "Friends of the Faculty of Computer Science and Automation". This work is also supported by the project "ICo-op – Industrial Cooperation and Creative Engineering Education based on Remote Engineering and Virtual Instrumentation" by the European Commission within the program "Tempus", Grant No 530278TEMPUS-1-2012-1-DE-TEMPUS-JPHES [26].

REFERENCES

[1] S. Sire, F. Geoffroy and D. Gillet: "A Virtual Assistant for Sending Hints and Perturbations to Students based on an Electronic Laboratory Journal (eJournal)", Proceedings of the ITHET'03, Marrakech, Morocco, July 7-9, 2003.

[2] K. Henke and H.-D. Wuttke: “Web-based educational tool access”, IASTED International Conference Computers and Advanced Technology in Education – CATE 2003, Rhodes, Greece, June 30 July 2, 2003.

[3] H.-D. Wuttke and K. Henke: “Living Pictures – tool-oriented learning modules and laboratory for teaching digital via Internet”, Proceedings of the ICEE-2002 International Conference on Engineering Education UMIST, Manchester, Great Britain, August 18-22, 2002.

[4] The iLab Project, from: https://wikis.mit.edu/confluence/ display/ ILAB2/Home

[5] WebLab Deusto, from: https://www.weblab.deusto.es/web. [6] VISIR, from: http://openlabs.bth.se/index.php.

[7] LabShare Sahara, from: http://www.labshare.edu.au/project/ index.php.

[8] REAL: “Remote Engineering and Applications Laboratory”, http://lanai.theoinf.tu-ilmenau.de/applets/index.htm.

[9] Y. Torroja, et al.: “A Modular Environment for Learning Digital Control Applications”, Microelectronics Education, Marcombo, S.A, 2002.

[10] T. A. Fjeldly, J. O. Strandman, R. Berntzen and M. S. Shur: “Advanced Solutions for Laboratory Experiments over the Internet”, Engineering Education and Research-2001, Begell House Publishing.

[11] H.-D. Wuttke, K. Henke and N. Ludwig: “remote labs versus Virtual Labs for Teaching Digital System Design”, Proceedings of the Int. Conf. On Computer Systems and Technologies CompSysTech’05, Varna, 2005.

[12] K. Henke, H.-D. Wuttke and T. Braune: “Virtual and remote labs in the Educational Process”, International Conference on Remote Engineering and Virtual Instrumentation, REV2007, Porto, Portugal, June 25-27, 2007.

[13] K. Henke, H.-D. Wuttke and S. Hellbach: “Laboratory via Internet – new ways in education and research”, Int. Journal of Computers and Applications, vol. 25, ACTA press, 2002.

[14] K. Henke, H.-D. Wuttke and T. Braune: “Rapid Prototyping Modules for Remote Engineering Applications”, International Conference on Remote Engineering and Virtual Instrumentation, REV2008, Düsseldorf, Germany, June 23-25, 2008.

[15] K. Henke, St. Ostendorff and H.-D. Wuttke: “A Flexible and Scalable Infrastructure for Remote Laboratories Robustness in Remote Engineering Laboratories”, The Impact of Virtual, Remote and Real Logistics Labs ImViReLL2012, Bremen, February 28 – March 02, 2012.

[16] K. Henke, St. Ostendorff, H.-D. Wuttke and St. Vogel: “A Grid Concept for Reliable, Flexible and Robust Remote Engineering Laboratories”, International Conference on Remote Engineering and Virtual Instrumentation, REV2012, Bilbao, Spain, July 4-6, 2012.

[17] MIT iLab Service Broker, Project homepage, from: http://ilab.mit.edu/iLabServiceBroker.

[18] H.-D. Wuttke, R. Ubar, K. Henke and A. Jutman: “Assessment of Student’s Design Results in E-Learning-Scenarios”, 8th Conference on Information Technology Based Higher Education and Training (ITHET2007), Kumamoto City, Japan, July 10-13, 2007.

[19] K. Henke: “Reusable Assessment Objects for Learning Management Systems”, Computers and Advanced Technology in Education”(CATE 2007), Beijing, China, October 8–10, 2007.

[20] JGIFT: “Java-based Graphical Interactive FSM Tools”, http://wcms1.rz.tu-ilmenau.de/fakia/index.php?id=780, TU Ilmenau.

[21] Microchip Corporation, http://www.microchip.com. [22] Atmel Corporation, http://www.atmel.com.

[23] Altera Corporation, http://www.altera.com.

[24] B. Deaky, D.G. Zutin and P.H. Bailey: “The First Android Client Application for the iLab Shared Architecture”, International Journal of Online Engineering (iJOE), Vol 8, No 1 (2012).

[25] TRE – International Summer School in Technologies for Remote Engineering, www.ingenieria.deusto.es/summerschool2012.

[26] ICo-op project Website: http://www.ICo-op.eu.

[27] The iLab Europe Network Service Broker, from: http://ilabeurope.net/iLabServiceBroker/.

AUTHORS

Karsten Henke is with the Ilmenau University of Technology, Faculty of Computer Science and Automation, Integrated Communication Systems Group, 98684 Ilmenau, Germany, POB 10 05 65 (e-mail: karsten.henke@tu-ilmenau.de).

Steffen Ostendorff is with the Ilmenau University of Technology, Faculty of Computer Science and Automation, Integrated Communication Systems Group, 98684 Ilmenau, Germany, POB 10 05 65 (e-mail: steffen.ostendorff@tu-ilmenau.de).

Heinz-Dietrich Wuttke is with the Ilmenau University of Technology, Faculty of Computer Science and Automation, Integrated Communication Systems Group, 98684 Ilmenau, Germany, POB 10 05 65 (e-mail: dieter.wuttke@tu-ilmenau.de).

Tobias Vietzke is with the Ilmenau University of Technology, Master Student in Computer Engineering at the Faculty of Computer Science and Automation, 98684 Ilmenau, Germany, POB 10 05 65. He obtained his BSc. in Computer Engineering in 2012 (e-mail: tobias.vietzke@tu-ilmenau.de).

Christian Lutze is with the Ilmenau University of Technology, Master Student in Computer Engineering at the Faculty of Computer Science and Automation, 98684 Ilmenau, Germany, POB 10 05 65. He obtained his BSc. in Computer Engineering in 2012 (e-mail: christian.lutze@tu-ilmenau.de).

This work was supported in part by the European Commission within the program “Tempus”, Grant No 530278-TEMPUS-1-2012-1-DETEMPUS-JPHES and by the “Friends of the Faculty of Computer Science and Automation” of the TU Ilmenau. It is an extended and modified version of a paper presented at the International Conference on Remote Engineering & Virtual Instrumentation (REV2012), held at University of Deusto, Bilbao, Spain, July 4-6, 2012. Received 01 March 2013. Published as resubmitted by the authors 20 March 2013.

Наукове видання

ПАРХОМЕНКО Анжеліка Володимирівна,
ГЛАДКОВА Ольга Миколаївна,
ПОЛЯКОВ Михайло Олексійович,
ЛАРІОНОВА Тетяна Юріївна,
ТАБУНЩИК Галина Володимирівна,
КАПЛІЄНКО Тетяна Ігорівна

ВІДДАЛЕНИЙ ТА ВІРТУАЛЬНИЙ ІНСТРУМЕНТАРІЙ В ІНЖИНІРИНГУ

Монографія

Комп’ютерний набір	*Г. В. Табунщик*
Дизайн обкладинки	*Д. М. Головань*
Технічний редактор	*Л. А. Рябоконь*
Коректор	*Н. В. Чечеко*
Художник	*А. П. Кондаков*

Формат 60x84/16.
Папір офсетний. Гарнітура *Times*. Друк офсетний.
Підписано до друку 30.09.2015. Ум. друк. арк. 14,53. Наклад 300 прим.

Видавництво «Дике Поле»
Україна, 69063, м. Запоріжжя, вул. Чекістів, 31-А.
Тел.: (061) 213–75–95; 213–75–05.
Свідоцтво суб’єкта видавничої справи ЗЗ № 004 від 23.08.2001 р.

В-42 **Віддалений** та віртуальний інструментарій в інжинірингу: монографія /за заг. ред. Карстена Хенке. – Запоріжжя: Дике Поле, 2015. – 250 с.
ISBN 978–966–2752–74–8

Книга містить основні концепції використання віртуальних, керованих дистанційно пристроїв, а також розподілених віддалених лабораторій. Розглянуті можливості їх використання при проектуванні вбудованих систем, для розробки систем керування складними електротехнічними установками і комплексами та для оцінювання якості вбудованих систем. Видання може бути корисним для фахівців в галузі проектування вбудованих систем, спеціалістів з електромеханіки, студентів та аспірантів.

УДК 004.41

Видання здійснено за підтримки міжнародного проекту ICo-op «Промислове співробітництво та креативна інженерна освіта на основі дистанційного інженерного та віртуального інструментарію» (530278-TEMPUS-1-2012-1-DE-TEMPUS-JPHES) за програмою TEMPUS Європейської комісії.
Зміст даного матеріалу відображає думку авторів та Європейська комісія не несе відповідальності за використання інформації, що міститься в монографії.