

Міністерство освіти і науки України
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ

до лабораторних робіт
з дисципліни

**“Валідація та верифікація цифрових систем
керування”**

для студентів
спеціальності 122

“Комп’ютерні науки та інформаційні технології”
всіх форм навчання

2016

Методичні вказівки до лабораторних робіт з дисципліни “Валідація та верифікація цифрових систем керування” для студентів спеціальності 122 “Комп’ютерні науки та інформаційні технології” всіх форм навчання / Г.В. Табунщик, Т.І. Каплієнко. – Запоріжжя: ЗНТУ, 2016. – 41 с.

Автори: Г. В. Табунщик, к.т.н., проф.
Т.І. Каплієнко, к.т.н., доцент

Рецензент: Р.К.Кудерметов, к.т.н., доцент

Відповідальний
за випуск: Г. В. Табунщик, к.т.н., проф.

Затверджено
на засіданні кафедри
програмних засобів

Протокол №
від “ ” _____ 2016 р.

ЗМІСТ

Вступ.....	4
Лабораторна робота №1 Основи верифікації вбудованих систем	6
Лабораторна робота №2 Модельно-орієнтоване тестування	15
Лабораторна робота №3 Робота з платою швидкого прототипування	18
Додаток А Код тест-кейсу test5.....	21

ВСТУП

Метою даного курсу є отримання теоретичних основ та практичних навичок з тестування цифрових систем керування. Дисципліна "Валідація та верифікація цифрових систем керування" спрямована на поглиблення знань та практичних навичок студентів з сучасної технології створення складних програмних продуктів на базі ідей і принципів об'єктно-орієнтованого методу.

На основі вивчення дисципліни " Валідація та верифікація цифрових систем керування " студент повинен знати:

- етапи життєвого циклу програмного забезпечення;
- метрики програмного забезпечення;
- методи керування якістю відносно до розробки проектуванню та експлуатації програмних продуктів;
- CASE-засоби для тестування програмного забезпечення.

Студент повинен вміти:

- оцінювати якість та ефективність програмного забезпечення при рішенні завдань розробки;
- перевіряти коректність;
- виконувати тестування;
- проводити вибір та вдосконалення програмного забезпечення.

Особливу увагу треба приділяти правильному оформленню звітів з практичних робіт. Звіт повинен містити: титульний аркуш (на ньому вказують назву міністерства, назву університету, назву кафедри, номер, вид і тему роботи, виконавця та особу, що приймає звіт, рік); мету, варіант і завдання роботи; лаконічний опис теоретичних відомостей; текст програми, що обов'язково містить коментарі; змістовний аналіз отриманих результатів та висновки.

Звіт виконують на білому папері формату А4 (210 x 297 мм). Текст розміщують тільки з однієї сторони листа. Поля сторінки з усіх боків – 20 мм. Аркуші скріплюють за допомогою канцелярських скріпок. Для набору тексту звіту використовують редактор MS Word: шрифт Times New Roman, 14 пунктів. Міжрядковий інтервал: полуторний – для тексту звіту, одинарний – для листингів програм і роздруківок даних.

Під час співбесіди студент повинний виявити знання про мету роботи, по теоретичному матеріалу, про методи виконання кожного етапу роботи, по змісту основних розділів розробленого звіту з демонстрацією результатів на конкретних прикладах. Студент повинний вміти правильно аналізувати отримані результати. Для

самоперевірки при підготовці до виконання і здачі роботи студент повинний відповісти на контрольні питання, приведені наприкінці опису відповідної роботи. Загальний залік студент одержує після виконання і здачі останньої роботи.

ЛАБОРАТОРНА РОБОТА №1

ОСНОВИ ВЕРИФІКАЦІЇ ВБУДОВАНИХ СИСТЕМ

1.1 Мета роботи

Аналіз і застосування засобів автоматизованого тестування вбудованих систем.

1.2 Основні теоретичні відомості

Традиційне тестування вбудованих систем, орієнтоване на розробку тестових наборів, що складаються з вхідних впливів і очікуваних реакцій, стає все більш ненадійним, тому що в системах реального часу проблематично відтворити ситуацію, що приводить саме до таких реакцій. Як в сфері наукових досліджень, так і розробці промислового інструментарію для верифікації розподілених систем, помітна тенденція до концентрації зусиль до просування формальних верифікації для паралельних і розподілених систем, які забезпечують гарантію того, що системи дотримуватимуться своїх ключові властивості для всіх своїх виконань.

Як об'єкт для розробки програмного забезпечення для роботи периферійними пристроями була обрана системна плата Digilent Basys2 Board, тому що вона є платформою для розробки і впровадження схем, яка дозволяє отримати досвід створення реальних цифрових схем, в тому числі і по роботі з периферійними пристроями.

Опис системної плати Digilent Basys2 Board

BASYS2 - відладочна плата компанії Digilent на базі програмованої логічної інтегральної схеми (ПЛІС) сімейства Spartan-3E XC3S250E ємністю 250 000 вентилів (рис. 1.1).



Рисунок 1.1 – Блок-схема плати Basys2

На плату встановлено ланцюг харчування, конфігураційна пам'ять, USB-контролер Full-speed, інтерфейси PS / 2 і VGA, 8 світлодіодів, 4 × 7-сегментних індикатора, 4 спеціальні кнопки, 8 перемикачів, що дозволяє розробнику відразу приступити проектування майбутнього устрою.

Для плати є велика кількість периферійних модулів (АЦП, ЦАП, управління двигунів, датчики), що підключаються до 6-вивідного роз'єму розширення. Для програмування плати можна використовувати фірмовий софт Xilinx ISE WebPack або софт від Digilent Adept.

Відмінні риси плати:

- ПЛІС: XC3S250E (Spartan-3E);
- кількість вентилів: 250 000;
- 18-розрядні помножувачі;
- конфігураційна пам'ять Flash ROM для зберігання конфігурації ПЛІС XCF02;
- посадочне місце для установки кварцу 25/50/100 МГц;
- 8 світлодіодів;
- 4 × 7-сегментних індикатора;
- 4 спеціальні кнопки;
- 8 перемикачів;
- порти: PS / 2 і VGA (8 розрядів);
- порти розширення: 4х 6-вивідних порту;

– захист від ESD і КЗ на всіх виводах I / O.

Приклади варіантів тестування для плати Digilent Basys2 Board наведено нижче.

Завдання 1

Розробити тест-кейс для плати Digilent Basys2 Board і модифікувати проект test1 відповідно до завдання: змінити номер функціонального перемикача і кількість світлодіодів, що святяться, відповідно до номеру варіанту .

Тест-кейс test1

Унікальний ідентифікатор варіанти тестування – test1.

Короткий опис варіанта тестування – самий правий перемикач sw[0] буде вмикати / вимикати самий правий світлодіод ld[0].

Порядок виконання - ввімкнути плату, ввімкнути правий перемикач.

Вимоги - тест завантажений на плату, плата підключена до комп'ютера.

Критерій завершеності - при включеному правому перемикачі світиться правий світлодіод, при вимкненому - не світиться.

Категорія тесту - тестування системних компонент плати.

Автор - Іванов І.І.

Автоматизований - так.

Код тесту:

```
module test1(
    input [0:0] sw,
    output [0:0] ld);

    assign ld[0] = sw[0];
endmodule
```

Завдання 2

Розробити тест-кейс для плати Digilent Basys2 Board і модифікувати проект test2 відповідно до завдання: змінити вхідні і вихідні параметри відповідно до таблиці.

Таблиця 1.1 Таблиця вхідних і вихідних параметрів

№ варіанту	Вхідні параметри (перемикачі)								Вихідні параметри (світлодіоди)							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	0	1	0	1	0	1	0	0	1	0	1	0	1	0	1
3	1	1	0	0	1	1	0	0	0	0	1	1	0	0	1	1
4	1	1	1	0	0	0	1	1	0	0	0	0	1	0	0	0
5	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1
6	0	0	0	0	1	0	1	0	1	0	1	0	0	0	0	1
7	1	0	0	0	0	0	0	1	0	1	0	0	1	1	0	1
8	0	0	0	1	1	0	0	0	1	0	0	0	0	0	0	1

Тест-кейс **test2**

Унікальний ідентифікатор варіанти тестування – test2.

Короткий опис варіанта тестування - значення важелів sw[0] и sw[1] (0=FALSE або 1=TRUE) подаються в якості значень вхідних параметрів модуля basic_boolean a і b, а світлодіоди ld [0] ... ld [4] будуть показувати значення результатів роботи модуля basic_boolean not_a, not_b, a_and_b, a_or_b, a_nand_b (0 = FALSE = викл або 1 = TRUE = вкл).

Порядок виконання - включити плату, поперемінно включати і вимикати в різних комбінаціях два правих перемикача.

Вимоги - тест завантажений на плату, плата підключена до комп'ютера.

Критерій завершеності - при включеному перемикачі sw [0] і вимкненому sw [1] світиться перший, третій і четвертий світлодіод; при вимкненому перемикачі sw [0] і включеному sw [1] світиться нульовий, третій і четвертий світлодіод; при вимкненому перемикачі sw [0] і вимкненому sw [1] світиться нульовий, перший і четвертий світлодіод; при включеному перемикачі sw [0] і включеному sw [1] світиться другий і третій світлодіод.

Категорія тесту - тестування системних компонент плати, логічні операції.

Автор - Іванов І.І.

Автоматизований - так.

Код тесту:

```
module basic_boolean(input a, input b,
```

```

output not_a, output not_b, output a_and_b,
output a_or_b, output a_nand_b
);

assign not_a = ~a; // NOT
assign not_b = ~b; // NOT
assign a_and_b = a & b; // AND
assign a_or_b = a | b; // OR
assign a_nand_b = ~(a & b); // NAND
endmodule

module basic_boolean_basys2(
    input [0:1] sw,
    output [0:4] ld);

basic_boolean impl(.a(sw[0]), .b(sw[1]),
    .not_a(ld[0]), .not_b(ld[1]),
    .a_and_b(ld[2]), .a_or_b(ld[3]),
    .a_nand_b(ld[4]));
endmodule

```

Завдання 3

Розробити тест-кейс для плати Digilent Basys2 Board і модифікувати проект test3 відповідно до завдання: змінити першу виведену цифру на номер варіанту, другу - на число, що дорівнює номеру варіанту плюс 1.

Тест-кейс test3

Унікальний ідентифікатор варіанти тестування – test3.

Короткий опис варіанта тестування - при включеному стані самого правого перемикача sw [0] буде світитися другий і четвертий цифровий дисплей з цифрою «7», при вимкненому - буде світитися перший і третій цифровий дисплей з цифрою «5».

Порядок виконання - включити плату, включити правий перемикач.

Вимоги - тест завантажений на плату, плата підключена до комп'ютера.

Критерій завершеності - при включеному стані самого правого перемикача sw [0] світиться другий і четвертий цифровий дисплей з цифрою «7», при вимкненому - перший і третій цифровий дисплей з цифрою «5».

Категорія тесту - тестування системних компонент плати.

Автор - Іванов І.І.

Автоматизований - так.

Код тесту:

```

module seven_driver(
    input value,
    output [0:3] an,
    output [0:6] segments
);
    // value=true - 7
    // value=false - 5
        assign an[0] = value ? "1" : "0";
        assign an[1] = value ? "0" : "1";
        assign an[2] = value ? "1" : "0";
        assign an[3] = value ? "0" : "1";
    assign segments[0] = value ? 0 : 0;
    assign segments[1] = value ? 0 : 1;
    assign segments[2] = value ? 0 : 0;
    assign segments[3] = value ? 1 : 0;
    assign segments[4] = value ? 1 : 1;
    assign segments[5] = value ? 1 : 0;
    assign segments[6] = value ? 1 : 0;
endmodule
module seven_driver_basys2(
    input [0:0] sw,
    output [0:3] an,
    output [0:6] seg
);
    seven_driver driver57(.value(sw[0]), .an(an),
    .segments(seg));
endmodule

--      0
--      ---
--  5 |    | 1
--      ---  <- 6
--  4 |    | 2
--      ---
--      3

```

Завдання 4

Розробити тест-кейс для плати Digilent Basys2 Board і модифікувати проект test4 відповідно до завдання: змінити номер миготливого світлодіода відповідно до варіанта, і частоту миготіння на 1 / варіант сек.

Тест-кейс test4

Унікальний ідентифікатор варіанти тестування - test4.

Короткий опис варіанта тестування - необхідно вмикати / вимикати самий правий світлодіод ld [0] з частотою раз в секунду.

Порядок виконання - включити плату.

Вимоги - тест завантажений на плату, плата підключена до комп'ютера.

Критерій завершеності - при включенні плати самий правий світлодіод ld [0] загоряється з частотою раз в секунду.

Категорія тесту - тестування системних компонент плати.

Автор - Іванов І.І.

Автоматизований - так.

Код тесту:

```
module clock_divider(
    input clk,
    output one_second
);

reg [25:0] counter;

// count up on each positive clock
always @(posedge clk)
    counter <= counter + 1;

// one second signal would be emitted when
// 26th bit of the counter would become "1"
assign one_second = counter[25];
endmodule

module clock_divider_top(
    input clk,
    output [0:0] ld
```

);

```
clock_divider count(.clk(clk),
.one_second(ld[0]));
endmodule
```

Завдання 5

Розробити тест-кейс для плати Digilent Basys2 Board (на ваш розсуд) і модифікувати проєкт test5 відповідно ним.

Тест-кейс test5

Унікальний ідентифікатор варіанти тестування - test5.

Короткий опис варіанта тестування - кожен перемикач запалює відповідний йому за номером світлодіод, натискання кнопки вимикає відповідний за номером індикатор, сьомий перемикач запускає циклічну зміну цифр на індикаторах.

Порядок виконання - включити плату, включити по черзі різні перемикачі.

Вимоги - тест завантажений на плату, плата підключена до комп'ютера.

Критерій завершеності - при включеному перемикачі світиться відповідний світлодіод, при вимкненому - не світиться; при включеному сьомому перемикачі запускається циклічна зміна цифр на індикаторах; при натисканні кнопки вимикає відповідний за номером індикатор.

Категорія тесту - тестування системних компонент плати.

Автор - Іванов І.І.

Автоматизований - так.

Код тесту (Додаток А)

1.3 Завдання на лабораторну роботу

1.3.1 Ознайомитися з теоретичними відомостями щодо засобів автоматизованого тестування для вбудованих систем.

1.3.2 Вивчити мову Verilog.

1.3.3 Розробити варіанти тестування для Digilent Basys2 Board.

1.3.4 Виконати аналіз отриманих результатів.

1.3.5 Оформити звіт та відповісти на контрольні питання.

1.4 Зміст звіту

1.4.1 Тема та мета роботи.

1.4.2 Розроблені варіанти тестування для Digilent Basys2 Board.

1.4.3 Аналіз отриманих результатів тестування.

1.4.5. Висновки, що містять відповіді на контрольні запитання, а також відображують результати виконання роботи та їх критичний аналіз.

ЛАБОРАТОРНА РОБОТА №2

МОДЕЛЬНО-ОРІЄНТОВАНЕ ТЕСТУВАННЯ

2.1 Мета роботи

Вивчити основи побудови FSM моделей та реалізувати модель «ліфт».

2.2 Основні теоретичні відомості

Тестування на основі моделі (англ. Model-based testing) - це тестування програмного забезпечення, в якому варіанти тестування частково або повністю виходять з моделі, яка описує деякі аспекти (частіше функціональні) тестованої системи (англ. System under test). Моделі можуть відображати бажане поведінку системи або використовуватися для створення тестових стратегій або середовища тестування.

Тестове покриття - покриття всіх можливих дій користувача в контексті даної програми

Повне тестове покриття - достатня інформація, щоб проходження всіх згенерованих тестів означало покриття всіх можливих дій користувача

Етапи тестування, заснованого на моделях:

- Побудова моделі (модель кінцевих автоматів, модель подій системи і т.д.);

- Генерування очікуваних вхідних даних;

- Генерування очікуваних вихідних значень;

- Порівняння з фактичними результатами;

- Прийняття рішень.

Структура моделі:

- 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 в наслідку, 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.

Тестування переходів (transition testing)

Варіант тестування розробляється для кожного переходу окремо

Мета - перевірити чи вплине вихід системи, що знаходиться в стані s1, на вхід a і призведе до стану s2.

2.3 Завдання на лабораторну роботу

2.3.1 Ознайомитися з теоретичними відомостями щодо модельно-орієнтованого тестування.

2.3.2 Для моделі ліфт розробити FSM модель, яка дозволить:

Варіант 1.

При циклічному переміщенні ліфта, при натисканні кнопки вгору 1 поверху, приїхати на 1 поверх, зупинитися і відкрити двері.

Варіант 2.

При циклічному переміщенні ліфта, при натисканні кнопки вгору 2 поверху, приїхати на 1 поверх, зупинитися і відкрити двері.

Варіант 3.

При циклічному переміщенні ліфта, при натисканні кнопки вниз 3 поверху, приїхати на 3 поверх, зупинитися і відкрити двері.

Варіант 4.

При циклічному переміщенні ліфта, при натисканні кнопки вниз 4 поверху, приїхати на 4 поверх, зупинитися і відкрити двері.

2.3.3 Виконати аналіз отриманих результатів.

2.3.4 Оформити звіт та відповісти на контрольні питання.

2.4 Зміст звіту

2.4.1 Тема та мета роботи.

2.4.2 Розроблена FSM модель.

2.4.3 Аналіз отриманих результатів.

2.4.5. Висновки, що містять відповіді на контрольні запитання, а також відображують результати виконання роботи та їх критичний аналіз.

2.5 Контрольні запитання

2.5.1 Які позначки використовуються для логічних операцій в виразах?

2.5.2 Які елементи виступають індикаторами параметрів фізичної моделі?

2.5.3 Де в віртуальній лабораторії вказується вираз кінцевого автомата?

2.5.4 В яких видах тестування можливо використовувати FSM?

ЛАБОРАТОРНА РОБОТА №3 РОБОТА З ПЛАТОЮ ШВИДКОГО ПРОТОТИПУВАННЯ

3.1 Мета роботи

Розробити варіанти тестування для плати швидкого протипування на мові VHDL в Quartus II.

3.2 Основні теоретичні відомості

Під терміном «проект» в Quartus II розуміється набір файлів, пов'язаних з проектуванням пристроєм, і набір відповідних бібліотек. Автоматично при створенні нового проекту створюються файли з розширенням .qpf (quartus project file) і .qsf (quartus settings file). Файли можуть бути двох типів - логічні і допоміжні. Логічні файли описують поведінку або структуру окремих модулів приладу. До них відносяться файли з текстовим описом на мовах опису апаратури (HDL файли, Hardware Description Language), файли з графічним представленням схем, файли з поданням окремих модулів у вигляді кінцевих автоматів. Допоміжні файли містять додаткову інформацію про проєктований пристрій: файли установок і призначень (.qsf), символні файли (.bsf), файли звітів (.rpt).

3.3 Завдання на лабораторну роботу

3.3.1 Робота з перемикачами

Для свого варіанту реалізувати включення LED відповідним перемикачем

```
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;

```

3.3.2. Робота з 7 сегментним дисплеєм

Реалізуйте вивід на дисплей номер вашого варіанту по включенню 0 перемикача

Приклад роботи з кнопками і дисплеєм

```

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;

```

3.3.3 Виконати аналіз отриманих результатів.

3.3.4 Оформити звіт та відповісти на контрольні питання.

3.4 Зміст звіту

3.4.1 Тема та мета роботи.

3.4.2 Результати виконання свого варіанту завдання.

3.4.3. Аналіз отриманих результатів.

3.4.5. Висновки, що містять відповіді на контрольні запитання, а також відображують результати виконання роботи та їх критичний аналіз.

Додаток А

Код тест-кейсу test5

CompSel.vhd

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CompSel is
    Port (regEppAdrIn : in std_logic_vector(7 downto
0);
        CS80_9F : out std_logic;
        CS0_7 : out std_logic;
        CS8_F : out std_logic;
        CS10 : out std_logic);
end CompSel;

architecture Behavioral of CompSel is

begin

    CS0_7 <= '1' when regEppAdrIn(7 downto 3) = "00000"
else '0';
    CS8_F <= '1' when regEppAdrIn(7 downto 3) = "00001"
else '0';
    CS10 <= '1' when regEppAdrIn = "00010000" else '0';
    CS80_9F <= '1' when regEppAdrIn(7 downto 5) = "100"
else '0';

end Behavioral;

```

PinDrivers.vhd

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PinDrivers is
    Port ( -- ck : in std_logic;
        -- sys clk: 50Mhz
        busIn : in std_logic_vector(7 downto 0);

```

22

```

busOut : out std_logic_vector(7 downto 0);
Wr : in std_logic;
      -- write cycle
adrIn : in std_logic_vector(7 downto 0);--
Epp address
CompSel : in std_logic;
      -- component select
--      HandShakeReqOut : out std_logic;
--      unused; steady '0'
--      DoneOut : out std_logic;
--      unused; steady '0'
      btn : in std_logic_vector(3 downto 0);
      sw : in std_logic_vector(7 downto 0);
      PIO : inout std_logic_vector(87 downto 72));
      -- JA, JB, JC, JD,

PS2 and VGA pins
--      PS2C: inout std_logic;
--      PS2D: inout std_logic);
end PinDrivers;

architecture Behavioral of PinDrivers is

signal PIOint : std_logic_vector(87 downto 70):=
      x"FFFF"&"11";

begin

-- Combinatorial busOut
      busOut <= "00000000"      when CompSel = '0' else
-- prepared for OR-ing
--
-- when not selected
      btn(3 downto 0) &  -- BTNs (read only)
--      "00" & PS2C & PS2D
--      "0000"
--      when adrIn(3 downto 1) =
"000" else
--      sw(7 downto 0)      when adrIn(3
downto 1) = "001" else
--      PIO(79 downto 72) when adrIn(3
downto 1) = "011" else
--      JB4...1 & JA4...1
--      PIO(87 downto 80) when adrIn(3
downto 1) = "100" else
--      JD4...1 & JC4...1

```

```

                                "00000000";           -- prepared
for OR-ing

-- Output Latches
--   outReg: process(ck, Wr) is
--       outReg: process(Wr) is
--           begin
--               if ck'event and ck = '1' then
--                   if wr'event and wr = '0' and           --
write cycle
                                CompSel = '1' then -- component
selected
--                               if adrIn(3 downto 1) = "000" then
--                                   PIOint(71 downto 70) <= busIn(1
downto 0); -- Sync and PS2
--                               -- PS2C, PS2D
--                                   if adrIn(3 downto 1) = "011" then
--                                       PIOint(79 downto 72) <= busIn;
--JB&JA
--                                   elsif adrIn(3 downto 1) = "100" then
--                                       PIOint(87 downto 80) <= busIn;
--JD&JC
--                                   else
--                                       null;
--                                   end if;
--                               end if;
--       end if;
--   end process;

-- Combinatorial Output buffers
--   outBuf: process(PIOint) is
--       begin
--           for i in 72 to 87 loop
--               if PIOint(i) = '0' then
--                   PIO(i) <= 'L';
--               else
--                   PIO(i) <= 'Z';
--               end if;
--           end loop;
--       end process;

--   outBufPS2D: process(PIOint) is
--       begin
--           if PIOint(70) = '0' then
--               PS2D <= 'L'; -- low

```

```

--                                     else
--                                     PS2D <= 'Z';  -- weak high with
pullup
--                                     end if;
--      end process;
--
--      outBufPS2C: process(PIOint) is
--      begin
--          if PIOint(71) = '0' then
--              PS2C <= 'L';  -- low
--          else
--              PS2C <= 'Z';  -- weak high with
pullup
--          end if;
--      end process;

```

```
end Behavioral;
```

EppCtrl.vhd

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity EppCtrl is
    Port (

-- Epp-like bus signals
        clk      : in std_logic;          -- system clock
(50MHz)
        EppAstb: in std_logic;            -- Address strobe
        EppDstb: in std_logic;            -- Data strobe
        EppWr   : in std_logic;            -- Port write
    signal
        EppRst  : in std_logic;            -- Port reset
    signal
        EppDB   : inout std_logic_vector(7 downto 0);  -
-- port data bus
        EppWait: out std_logic;            -- Port wait signal
-- User signals
        busEppOut: out std_logic_vector(7 downto 0); --
    Data Output bus

```

```

        busEppIn: in std_logic_vector(7 downto 0);    --
Data Input bus
        ctlEppDwrOut: out std_logic;                -- Data
Write pulse
        ctlEppRdCycleOut: inout std_logic;          -- Indicates
a READ Epp cycle
        regEppAdrOut: inout std_logic_vector(7 downto 0)
:= "00000000";
                                                    -- Epp Address
Register content
        HandShakeReqIn: in std_logic;                -- User
Handshake Request
        ctlEppStartOut: out std_logic;              -- Automatic
process Start
        ctlEppDoneIn: in std_logic                  -- Automatic
process Done
        );
end EppCtrl;

```

architecture Behavioral of EppCtrl is

```

-----
-----
-- Constant and Signal Declarations
-----
-----

```

```

-- The following constants define state codes for the
EPP port interface
-- state machine.
-- The states are such a way assigned that each
transition
-- changes a single state register bit (Grey code -
like)
        constant stEppReady      : std_logic_vector(2 downto
0) := "000";
        constant stEppStb       : std_logic_vector(2 downto
0) := "010";
        constant stEppRegTransf : std_logic_vector(2 downto
0) := "110";
        constant stEppSetProc    : std_logic_vector(2 downto
0) := "011";
        constant stEppLaunchProc: std_logic_vector(2 downto
0) := "111";

```

26

```
constant stEppWaitProc : std_logic_vector(2 downto
0) := "101";
constant stEppDone      : std_logic_vector(2 downto
0) := "100";

-- Epp state register and next state signal for the Epp
FSM
    signal stEppCur: std_logic_vector(2 downto 0) :=
stEppReady;
    signal stEppNext: std_logic_vector(2 downto 0);

-- The attribute lines below prevent the ISE compiler
to extract and
-- optimize the state machines.
-- WebPack 5.1 doesn't need them (the default value is
NO)
-- WebPack 6.2 has the default value YES, so without
these lines would
-- "optimize" the state machines.
-- Although the overall circuit would be optimized, the
particular goal
-- of "glitch free output signals" may not be reached.
-- That is the reason of implementing the state machine
as described in
-- the constant declarations above.

attribute fsm_extract : string;
attribute fsm_extract of stEppCur: signal is "no";
attribute fsm_extract of stEppNext: signal is "no";

attribute fsm_encoding : string;
attribute fsm_encoding of stEppCur: signal is "user";
attribute fsm_encoding of stEppNext: signal is "user";

attribute signal_encoding : string;
attribute signal_encoding of stEppCur: signal is
"user";
attribute signal_encoding of stEppNext: signal is
"user";

-- Signals used by Epp state machine
    signal busEppInternal: std_logic_vector(7 downto
0);
--    signal ctlEppDir      : std_logic;
    signal ctlEppAwr      : std_logic;
```

```

-----
-----
-- Module Implementation
-----
-----

```

```
begin
```

```

-----
-----
-- Map basic status and control signals
-----
-----

```

```

-- Epp signals
-- Port signals

```

```
-- Synchronized Epp inputs:
```

```
process(clk)
```

```
begin
```

```
    if clk'event and clk='1' then
```

```
        if stEppCur = stEppReady then
```

```
            ctlEppRdCycleOut <= '0';
```

```
        elsif stEppCur = stEppStb then
```

```
            ctlEppRdCycleOut <= EppWr;
```

```
            -- not equivalent to EppWr due to default
```

```
state
```

```
    end if;
```

```
end if;
```

```
end process;
```

```
    busEppOut <= EppDB;    -- name meaning change!!!
```

```
    EppDB <=    busEppInternal when (ctlEppRdCycleOut
= '1') else "ZZZZZZZZ";
```

```
    busEppInternal <= regEppAdrOut when EppAstb = '0'
else busEppIn;
```

```
-- Epp State machine related signals
```

```
EppWait <= '1' when stEppCur = stEppDone else '0';
```

```
ctlEppAwr <= '1' when stEppCur = stEppRegTransf and
```

28

```

                                EppAstb = '0' and
                                EppWr = '0' else
                                '0';
    ctlEppDwrOut <= '1' when (stEppCur = stEppRegTransf
or
                                stEppCur = stEppSetProc)
                                and EppDstb = '0'
                                and EppWr = '0' else
                                '0';
    ctlEppStartOut <= '1' when stEppCur =
stEppLaunchProc else
                                '0';

```

```

-----
-- EPP Interface Control State Machine
-----

```

```

    process (clk)
    begin
        if clk = '1' and clk'Event then
            if EppRst = '0' then
                stEppCur <= stEppReady;
            else
                stEppCur <= stEppNext;
            end if;
        end if;
    end process;

    process (stEppCur)
    begin
        case stEppCur is
            -- Idle state waiting for the beginning of
an EPP cycle
            when stEppReady =>
                if EppAstb = '0' or EppDstb = '0' then
                    -- Epp cycle recognized
                    stEppNext <= stEppStb;
                else
                    -- Remain in ready state
                    stEppNext <= stEppReady;
                end if;

                when stEppStb =>

```

```

        if EppDstb = '0' and HandShakeReqIn =
'1' then
            stEppNext <= stEppSetProc;
        else
            stEppNext <= stEppRegTransf;
        end if;

-- Data or Address register transfer

when stEppRegTransf =>
    stEppNext <= stEppDone;

-- Automatic Process

when stEppSetProc =>
    stEppNext <= stEppLaunchProc;

when stEppLaunchProc =>
    if ctlEppDoneIn = '0' then
        stEppNext <= stEppLaunchProc;
    else
        stEppNext <= stEppWaitProc;
    end if;

when stEppWaitProc =>
    if ctlEppDoneIn = '1' then
        stEppNext <= stEppWaitProc;
    else
        stEppNext <= stEppDone;
    end if;

when stEppDone =>
    if EppAstb = '0' or EppDstb = '0' then
        stEppNext <= stEppDone;
    else
        stEppNext <= stEppReady;
    end if;

-- Some unknown state
when others =>
    stEppNext <= stEppReady;

end case;
end process;

```

```

-- EPP Address register

process (clk, ctlEppAwr)
begin
    if clk = '1' and clk'Event then
        if ctlEppAwr = '1' then
            regEppAdrOut <= EppDB;
        end if;
    end if;
end process;

end Behavioral;

```

SimpleSsegLedDemo.vhd

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity SimpleSsegLedDemo is

    Port (ck: in std_logic;
          btn: in std_logic_vector(3 downto 0);
          sw: in std_logic_vector(7 downto 0);
          led: out std_logic_vector(7 downto 0);
          seg: out std_logic_vector(6 downto 0);
          dp: out std_logic;
          an: out std_logic_vector(3 downto 0);
          PS2C: in std_logic; -- PS2 serial clock
          PS2D: in std_logic -- PS2 serial data
          );

end SimpleSsegLedDemo;

architecture Behavioral of SimpleSsegLedDemo is

    signal cntDiv: std_logic_vector(28 downto 0); --
    general clock div/cnt
    alias cntDisp: std_logic_vector(3 downto 0) is
    cntDiv(28 downto 25);
    alias clkDisp: std_logic is cntDiv(16); -- divided
    clock
    -- 50MHz/2^17=381Hz
    alias ck1: std_logic is cntDiv(9); -- divided clock

```

```

-- 50MHz/2^10=48828Hz

signal s_buf:std_logic_vector (9 downto 0); --
buffer for PS2 receiver
signal key_code:std_logic_vector (7 downto 0):=
x"00"; -- scan code from keyboard
signal par:std_logic; -- parity bit for PS2 receiver
signal wdg: integer range 0 to 50000000; -- watch dog
counter (1 sec at 50MHz)
signal digit:std_logic_vector (3 downto 0); --
curent displayed digit
signal PS2Cold:std_logic; -- stored PS2C for edge d

begin

    led <= sw;

    ckDivider: process(ck)
    begin
        if ck'event and ck='1' then
            cntDiv <= cntDiv + '1';
        end if;
    end process;

-- selecting the content of the seven segment display
process(sw(6))
begin
    case SW(6) is
        when '0' => -- sw6 LOW
            digit <= cntDisp; -- simple counter on all
digits
            an <= btn; -- pressed buton turns
digit OFF
            dp <= '0'; -- al decimal points ON
            when others => -- sw6 HIGH
                -- displaying the keyboard scan code on seven
segment display last two digits
                dp <= '1'; -- al decimal points OFF
                case clkDisp is
                    when '0' =>
                        digit <= key_code(3 downto 0); -- digit
0 (LSD)
                        an <= "1110";
                        when others =>

```

1

```

        par<='0';
    else
        -- incorrect byte
        s_buf<="1111111111";
        par<='0';
        key_code<="00000000";
-
- no valid byte available
        end if;
    else
        -- not yet 11 bits
        s_buf<=PS2D&s_buf(9 downto 1);
-- shift bits to right, adding the new received one
        par<=par xor PS2D;
        -- parity check. Includes the received parity
        bit.
        end if;
    end if;
end if;
end process;

--HEX-to-seven-segment decoder
--  HEX:   in   STD_LOGIC_VECTOR (3 downto 0);
--  LED:   out  STD_LOGIC_VECTOR (6 downto 0);
--
-- segment encoinputg
--      0
--      ---
--      5 |   | 1
--      ---   <- 6
--      4 |   | 2
--      ---
--      3

    with digit SElect
    seg<= "1111001" when "0001", --1
          "0100100" when "0010", --2
          "0110000" when "0011", --3
          "0011001" when "0100", --4
          "0010010" when "0101", --5
          "0000010" when "0110", --6
          "1111000" when "0111", --7
          "0000000" when "1000", --8
          "0010000" when "1001", --9

```

```

"0001000" when "1010",  --A
"0000011" when "1011",  --b
"1000110" when "1100",  --C
"0100001" when "1101",  --d
"0000110" when "1110",  --E
"0001110" when "1111",  --F
"1000000" when others;  --0

```

```
end Behavioral;
```

ckMux.vhd

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

```

```

library UNISIM;
use UNISIM.VComponents.all;

```

```

entity ckMux is
    Port ( ck0 : in  STD_LOGIC;
          ck1 : in  STD_LOGIC;
          sel : in  STD_LOGIC;
          ckOut : out STD_LOGIC);
end ckMux;

```

```
architecture Behavioral of ckMux is
```

```
begin
```

```

    BUFGMUX_inst : BUFGMUX
    port map (
        O => ckOut,  -- Clock MUX output
        I0 => ck0,   -- Clock0 input
        I1 => ck1,   -- Clock1 input
        S => sel     -- Clock select input
    );

```

```
end Behavioral;
```

DispCtrl.vhd

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

```

```

use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity DispCtrl is
  Port (ck: in std_logic;  -- 50MHz
  --      Hcnt: in std_logic_vector(9 downto 0);  -
  - horizontal counter
  --      Vcnt: in std_logic_vector(9 downto 0);  -
  - verical counter
      HS: out std_logic;  -
  - horizontal synchro signal
      VS: out std_logic;  -
  - verical synchro signal

      outRed  : out std_logic_vector(2 downto 0); --
final color
      outGreen: out std_logic_vector(2 downto 0);
-- outputs
      outBlue : out std_logic_vector(2 downto 1)
      );
end DispCtrl;

```

architecture Behavioral of DispCtrl is

```

-- constants for Synchro module
  constant PAL:integer:=640;          --Pixels/Active
Line (pixels)
  constant LAF:integer:=480;          --Lines/Active
Frame (lines)
  constant PLD: integer:=800; --Pixel/Line Divider
  constant LFD: integer:=521; --Line/Frame Divider
  constant HPW:integer:=96;          --Horizontal
synchro Pulse Width (pixels)
  constant HFP:integer:=16;          --Horizontal
synchro Front Porch (pixels)
--  constant HBP:integer:=48;          --Horizontal
synchro Back Porch (pixels)
  constant VPW:integer:=2;          --Verical synchro
Pulse Width (lines)
  constant VFP:integer:=10;          --Verical synchro
Front Porch (lines)
--  constant VBP:integer:=29;          --Verical synchro
Back Porch (lines)

```

```

-- signals for VGA Demo
  signal Hcnt: std_logic_vector(9 downto 0);      --
horizontal counter
  signal Vcnt: std_logic_vector(9 downto 0);      --
verical counter
  signal intHcnt: integer range 0 to 800-1; --PLD-1 -
horizontal counter
  signal intVcnt: integer range 0 to 521-1; -- LFD-1 -
verical counter

  signal ck25MHz: std_logic;          -- ck 25MHz

begin

-- divide 50MHz clock to 25MHz
div2: process(ck)
begin
  if ck'event and ck = '1' then
    ck25MHz <= not ck25MHz;
  end if;
end process;

syncro: process (ck25MHz)
begin

if ck25MHz'event and ck25MHz='1' then
  if intHcnt=PLD-1 then
    intHcnt<=0;
    if intVcnt=LFD-1 then intVcnt<=0;
    else intVcnt<=intVcnt+1;
    end if;
  else intHcnt<=intHcnt+1;
  end if;

  -- Generates HS - active low
  if intHcnt=PAL-1+HFP then
    HS<='0';
  elsif intHcnt=PAL-1+HFP+HPW then
    HS<='1';
  end if;

  -- Generates VS - active low
  if intVcnt=LAF-1+VFP then
    VS<='0';
  elsif intVcnt=LAF-1+VFP+VPW then

```

```

        VS<='1';
    end if;
end if;
end process;

-- mapping internal integers to std_logic_vector ports
Hcnt <= conv_std_logic_vector(intHcnt,10);
Vcnt <= conv_std_logic_vector(intVcnt,10);

mixer: process(ck25MHz,intHcnt, intVcnt)
begin
    if intHcnt < PAL and intVcnt < LAF then      -- in
the active screen

        if Vcnt(7 downto 6) = "00" then
            outRed <= Vcnt(5 downto 3);
            outGreen <= "000";
            outBlue <= "00";
        elsif Vcnt(7 downto 6) = "01" then
            outRed <= "000";
            outGreen <= Vcnt(5 downto 3);
            outBlue <= "00";
        elsif Vcnt(7 downto 6) = "10" then
            outRed <= "000";
            outGreen <= "000";
            outBlue <= Vcnt(5 downto 4);
        else
            outRed(2 downto 1) <= Vcnt(5 downto 4);
            outGreen(2 downto 1) <= Vcnt(5 downto 4);
            outBlue <= Vcnt(5 downto 4);
        end if;

    else
        outRed <= (others => '0');
        outGreen <= (others => '0');
        outBlue <= (others => '0');
    end if;
end process;

end Behavioral;

```

Basys2_100_250General.ucf

```

# This file is a general .ucf for Basys2 rev C board
# To use it in a project:

```

```

# - remove or comment the lines corresponding to unused
pins
# - rename the used signals according to the project

# clock pin for Basys2 Board
NET "mclk" LOC = "B8"; # Bank = 0, Signal name = MCLK
NET "uclk" LOC = "M6"; # Bank = 2, Signal name = UCLK
NET "mclk" CLOCK_DEDICATED_ROUTE = FALSE;
NET "uclk" CLOCK_DEDICATED_ROUTE = FALSE;

# Pin assignment for EppCtl
# Connected to Basys2 onBoard USB controller
NET "EppAstb" LOC = "F2"; # Bank = 3
NET "EppDstb" LOC = "F1"; # Bank = 3
NET "EppWR"      LOC = "C2"; # Bank = 3

NET "EppWait" LOC = "D2"; # Bank = 3

NET "EppDB<0>" LOC = "N2"; # Bank = 2
NET "EppDB<1>" LOC = "M2"; # Bank = 2
NET "EppDB<2>" LOC = "M1"; # Bank = 3
NET "EppDB<3>" LOC = "L1"; # Bank = 3
NET "EppDB<4>" LOC = "L2"; # Bank = 3
NET "EppDB<5>" LOC = "H2"; # Bank = 3
NET "EppDB<6>" LOC = "H1"; # Bank = 3
NET "EppDB<7>" LOC = "H3"; # Bank = 3

# Pin assignment for DispCtl
# Connected to Basys2 onBoard 7seg display
NET "seg<0>" LOC = "L14"; # Bank = 1, Signal name = CA
NET "seg<1>" LOC = "H12"; # Bank = 1, Signal name = CB
NET "seg<2>" LOC = "N14"; # Bank = 1, Signal name = CC
NET "seg<3>" LOC = "N11"; # Bank = 2, Signal name = CD
NET "seg<4>" LOC = "P12"; # Bank = 2, Signal name = CE
NET "seg<5>" LOC = "L13"; # Bank = 1, Signal name = CF
NET "seg<6>" LOC = "M12"; # Bank = 1, Signal name = CG
NET "dp" LOC = "N13"; # Bank = 1, Signal name = DP

NET "an<3>" LOC = "K14"; # Bank = 1, Signal name = AN3
NET "an<2>" LOC = "M13"; # Bank = 1, Signal name = AN2
NET "an<1>" LOC = "J12"; # Bank = 1, Signal name = AN1
NET "an<0>" LOC = "F12"; # Bank = 1, Signal name = AN0

```

```
# Pin assignment for LEDs
```

```
NET "Led<7>" LOC = "G1" ; # Bank = 3, Signal name = LD7
NET "Led<6>" LOC = "P4" ; # Bank = 2, Signal name = LD6
NET "Led<5>" LOC = "N4" ; # Bank = 2, Signal name =
LD5
NET "Led<4>" LOC = "N5" ; # Bank = 2, Signal name =
LD4
NET "Led<3>" LOC = "P6" ; # Bank = 2, Signal name = LD3
NET "Led<2>" LOC = "P7" ; # Bank = 3, Signal name = LD2
NET "Led<1>" LOC = "M11" ; # Bank = 2, Signal name =
LD1
NET "Led<0>" LOC = "M5" ; # Bank = 2, Signal name =
LD0
```

```
# Pin assignment for SWs
```

```
NET "sw<7>" LOC = "N3"; # Bank = 2, Signal name = SW7
NET "sw<6>" LOC = "E2"; # Bank = 3, Signal name = SW6
NET "sw<5>" LOC = "F3"; # Bank = 3, Signal name = SW5
NET "sw<4>" LOC = "G3"; # Bank = 3, Signal name = SW4
NET "sw<3>" LOC = "B4"; # Bank = 3, Signal name = SW3
NET "sw<2>" LOC = "K3"; # Bank = 3, Signal name = SW2
NET "sw<1>" LOC = "L3"; # Bank = 3, Signal name = SW1
NET "sw<0>" LOC = "P11"; # Bank = 2, Signal name = SW0
```

```
NET "btn<3>" LOC = "A7"; # Bank = 1, Signal name =
BTN3
NET "btn<2>" LOC = "M4"; # Bank = 0, Signal name =
BTN2
NET "btn<1>" LOC = "C11"; # Bank = 2, Signal name =
BTN1
NET "btn<0>" LOC = "G12"; # Bank = 0, Signal name =
BTN0
```

```
# Loop back/demo signals
```

```
# Pin assignment for PS2
```

```
NET "PS2C" LOC = "B1" | DRIVE = 2 | PULLUP ; #
Bank = 3, Signal name = PS2C
NET "PS2D" LOC = "C3" | DRIVE = 2 | PULLUP ; #
Bank = 3, Signal name = PS2D
```

```
# Pin assignment for VGA
```

```
NET "HSYNC" LOC = "J14" | DRIVE = 2 | PULLUP ; #
Bank = 1, Signal name = HSYNC
NET "VSYNC" LOC = "K13" | DRIVE = 2 | PULLUP ; #
Bank = 1, Signal name = VSYNC
```

```

NET "OutRed<2>" LOC = "F13" | DRIVE = 2 | PULLUP ; #
Bank = 1, Signal name = RED2
NET "OutRed<1>" LOC = "D13" | DRIVE = 2 | PULLUP ; #
Bank = 1, Signal name = RED1
NET "OutRed<0>" LOC = "C14" | DRIVE = 2 | PULLUP ; #
Bank = 1, Signal name = RED0
NET "OutGreen<2>" LOC = "G14" | DRIVE = 2 | PULLUP ;
# Bank = 1, Signal name = GRN2
NET "OutGreen<1>" LOC = "G13" | DRIVE = 2 | PULLUP ;
# Bank = 1, Signal name = GRN1
NET "OutGreen<0>" LOC = "F14" | DRIVE = 2 | PULLUP ;
# Bank = 1, Signal name = GRN0
NET "OutBlue<2>" LOC = "J13" | DRIVE = 2 | PULLUP ;
# Bank = 1, Signal name = BLU2
NET "OutBlue<1>" LOC = "H13" | DRIVE = 2 | PULLUP ;
# Bank = 1, Signal name = BLU1

# Loop Back only tested signals
NET "PIO<72>" LOC = "B2" | DRIVE = 2 | PULLUP ; #
Bank = 1, Signal name = JA1
NET "PIO<73>" LOC = "A3" | DRIVE = 2 | PULLUP ; #
Bank = 1, Signal name = JA2
NET "PIO<74>" LOC = "J3" | DRIVE = 2 | PULLUP ; #
Bank = 1, Signal name = JA3
NET "PIO<75>" LOC = "B5" | DRIVE = 2 | PULLUP ; #
Bank = 1, Signal name = JA4

NET "PIO<76>" LOC = "C6" | DRIVE = 2 | PULLUP ; #
Bank = 1, Signal name = JB1
NET "PIO<77>" LOC = "B6" | DRIVE = 2 | PULLUP ; #
Bank = 1, Signal name = JB2
NET "PIO<78>" LOC = "C5" | DRIVE = 2 | PULLUP ; #
Bank = 1, Signal name = JB3
NET "PIO<79>" LOC = "B7" | DRIVE = 2 | PULLUP ; #
Bank = 1, Signal name = JB4

NET "PIO<80>" LOC = "A9" | DRIVE = 2 | PULLUP ; #
Bank = 1, Signal name = JC1
NET "PIO<81>" LOC = "B9" | DRIVE = 2 | PULLUP ; #
Bank = 1, Signal name = JC2
NET "PIO<82>" LOC = "A10" | DRIVE = 2 | PULLUP ; #
Bank = 1, Signal name = JC3
NET "PIO<83>" LOC = "C9" | DRIVE = 2 | PULLUP ; #
Bank = 1, Signal name = JC4

```

```
NET "PIO<84>" LOC = "C12" | DRIVE = 2 | PULLUP ; #  
Bank = 1, Signal name = JD1  
NET "PIO<85>" LOC = "A13" | DRIVE = 2 | PULLUP ; #  
Bank = 2, Signal name = JD2  
NET "PIO<86>" LOC = "C13" | DRIVE = 2 | PULLUP ; #  
Bank = 1, Signal name = JD3  
NET "PIO<87>" LOC = "D12" | DRIVE = 2 | PULLUP ; #  
Bank = 2, Signal name = JD4
```