# Remote HDL Prototyping and Testing Platform

**Felix Georg Braun**
**felix.braun@live.at**

**Philipp-Sebastian Vogt**
**philippvogt@gmx.at**

**Submitted for the 2019 Digilent Design Contest Europe**

**1.5.2019**

**Advisor: Dr. Nima TaheriNejad**

**TU Wien**
**Vienna, Austria**

Figure 1: Philipp-S. Vogt (left) and Felix G. Braun (right)

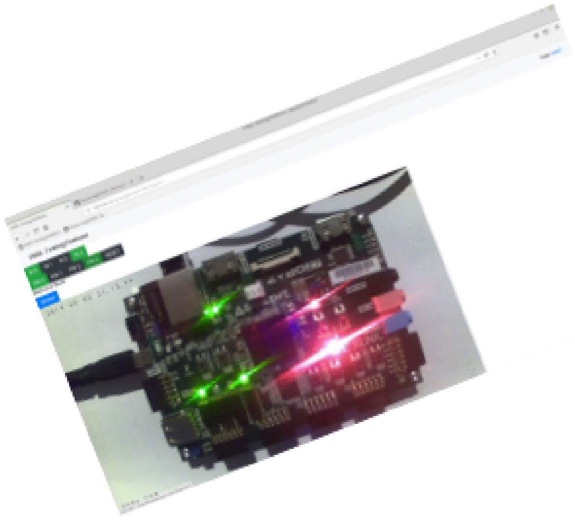
# Contents

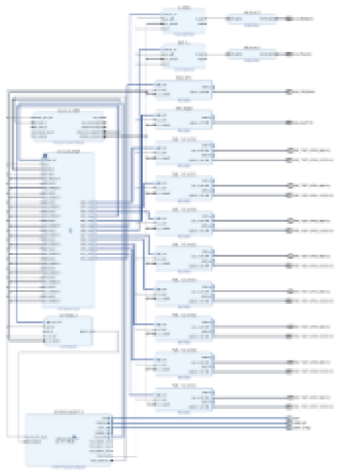
# VHDL Testing Platform

Easy to use Web Platform for VHDL!

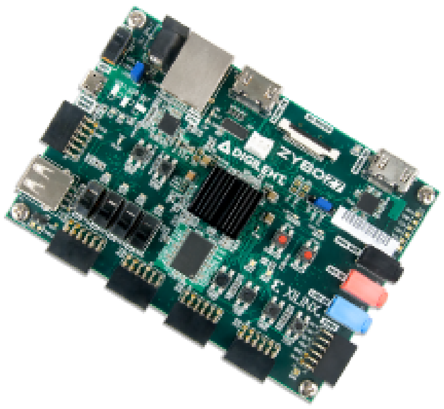
DIGILENT DESIGN CONTEST 2019

TU WIEN TECHNISCHE UNIVERSITÄT WIEN





**8x Partial Reconfiguration**



**Multi User Support**

Easy permission system, only one JSON to edit

**Cost and Time Efficient**

Up to 8 users on one board at the same time

**On Real Hardware**

Users can test their design on actual hardware, instead of simulating everything

https://projects.digilentinc.com/braun-vogt/vhdl-testing-platform

# 1 Introduction

Pursuing a career in FPGA design, while not being able to afford an FPGA development board is challenging. As a teacher too, it is more complicated to teach HDL and digital hardware design when not every student has an FPGA board. This project is there to help you! We have created an easy to use remote prototyping and testing platform at our university, which is particularly targeted toward education. So students can test their HDL design homework on real hardware at the comfort of their homes and using a simple web interface.

## 1.1 Abstract

The outcome of this project is an easy to use platform for students to prototype and test their HDL design on an actual hardware over the internet. A web interface and a webcam provide (digital and visual) feedback to the user regarding the inputs and outputs of their design running on real hardware.

## 1.2 Objectives

The main objective of this project is to enable a (large) group of HDL designers (in particular students) to prototype and test their design on a real hardware without needing to purchase or acquire a physical hardware.

**Requirements:**

- Easy to access and use (especially for students).
- Multi-user support (partially re-configurable)
- Have a short waiting time even for a large group of users.
- Minimum work and financial burden on the user (provide only only the *.vhd file, hence no hardware or software license fees required)
- A good feedback to emulate real hardware usage as far as possible.

## 1.3 Project Summary

The project contains three major parts. A server program to provide the interface between the user and the hardware (which also generates the automatic TCL-script), an FPGA and a software running on Petalinux to compile and load submitted designs on the FPGA. We are going to integrate this project into our VELS system (an in-house developed online HDL homework platform, see section 2.2.1 on the following page for more information) to enable our students to not only verify the correctness of their HDL designs, but also prototype it on real hardware and see the results in action. The platform could be used for purposes other than the education ones too, e.g., commercial usages (FPGA prototype and test service over cloud).

Each part is functioning independently. So other than the current usage in the project one could implement the server part in a continuous integration pipeline for version controlled development of VHDL files. The backbone FPGA design could be used as a barebone design for other partial design projects. The software of the webserver could also be used standalone. For example, it would be possible (after some modifications), to upload *.bit files via web interface and test it over the web.

The whole project was tested on a Zybo Z7-20, but could be used on other Zynq platforms too. With small modifications it could be used on other Xillinx FPGAs as well.

### 1.4 Digilent Products Required

- Zybo Z7-20 Dev Board
- Various PMods (optional)

### 1.5 Tools Required

- Vivado 2017.4
- Petalinux 2017.4
- Vivado partial reconfiguration license
- Ubuntu 2016.04 LTS
- FreeCAD
- IP Cam (Keekoon KK001)
- Micro SD card (mininimum 16GB)

### 1.6 Design Status

All technical parts of the project are finished. We can receive project files (*.vhdl), automatically generate the *.bit files, flash them on the FPGA at run time, and provide both visual and input/output status feedback to the user. Due to unpredictable delivery bottlenecks, we could not finish the Plexiglas case-box. We hope to have it built till the presentations.

# 2 Background

## 2.1 Why This Project?

Access to real hardware for prototyping and testing HDL codes can always present a challenge in different contexts. For a large portion of students around the world or institutions in less privileged countries availability and access to any FPGA may present a barrier hard to surmount. High-end FPGAs with high performance are often not within the budget range of many academic groups or small companies in more privileged countries either.

At our university we have few lectures on VHDL and digital hardware design. The course requires the students to design digital hardware in VHDL as homework. In the introductory course, those files get processed in our online VELS platform (see section 2.2.1 for more details). In VELS, the homework only gets verified using simulations only. The student then receives an email if the design worked or not. We tried to improve this platform by sending the successfully simulated files to our project and thus implement the outcome on actual hardware. The students can therefore see the result of their design running on an actual hardware and interact with it live.

## 2.2 Reference Material

### 2.2.1 VELS

VELS[1] is an email-based simulation platform for VHDL design tasks (homeworks). It is used in at the TU Wien to teach digital hardware design using VHDL. The platform provides each student a custom generated assignment to solve. It simulates the file submitted by the student and if the test is successful, the student receives an email about the positive assessment of the task.

[1] `https://github.com/autosub-team/autosub`

# 3 Design

## 3.1 Features and Specifications

**Features:**

- Easy access and usage (via web interface)
- Multi-user support (via partial reconfiguration)
- Minimum waiting time (via partial reconfiguration)
- Efficient usage of hardware (via partial reconfiguration)
- Digital and visual feedback

**Constrain(s):**

- Only 8 users at the same time can use the hardware

## 3.2 Design Overview

The design consists of three smaller parts; a block design, some partial reconfigurable PBlocks and an input-output multiplexer. The main unit of the design is the block design. There an ARM processor is included, which is responsible for controlling all processes in the programmable logic, switching the multiplexer and programming the partial reconfiguration blocks. The PBlocks are the prototyping partitions which are used to flash the synthesised user files into them and thus implement and test them. The multiplexer is designed to connect the inputs/outputs of the partial blocks to certain user defined IOs. Table 1 shows the IOs of the FPGA, its direction, and a detailed description of its function.

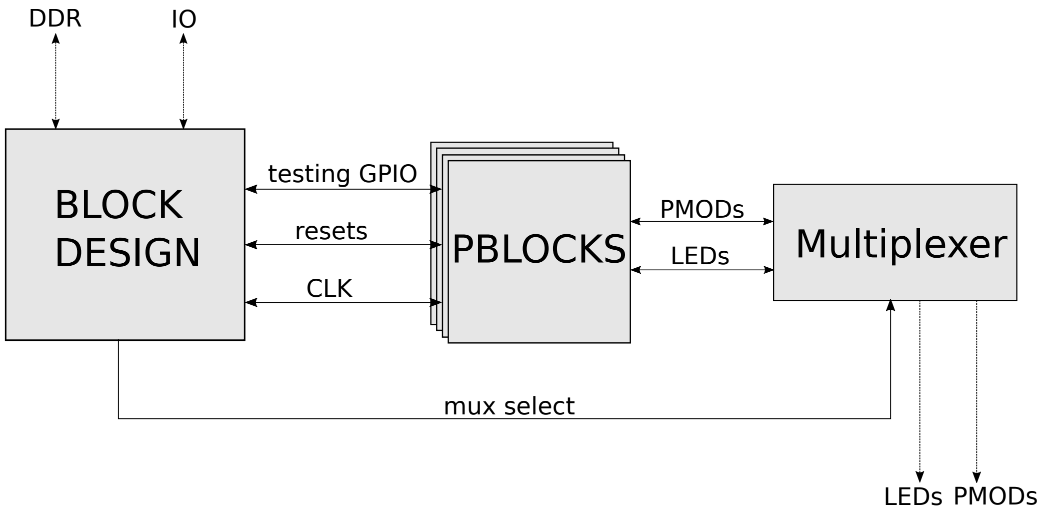


Figure 2: Top block diagram of the system.

The block design is shown in Figure 3. It consists of five main parts. The ARM processor, the AXI_HWICAP, the PAR_TEST_GPIO*, the dynamic clocks CLK_***, and the GPIO controller ( PAR_RESET and MUX_GPIO). All these parts are connected via AXI interface.

Table 1: The inputs and outputs of the FPGA development board and their respective function.

| IO LIST | | | |
|---|---|---|---|
| Pin Group Name | Pin Name | Pin Direction | Description |
| DDR | | IO | Pins for the interface between the arm and the DDR ram. |
| PMODs | | IO | Bidirectional PMOD interface for the PBlocks |
| | PMODB | IO | Interface to PMOD header B on the zybo-z7 board |
| | PMODC | IO | Connection to PMOD header C on the zybo-z7 board |
| | PMODD | IO | Interface to the PMODD connector |
| | PMODE | IO | Contact to the PMODE header |
| LEDS | | O | Conglomerate of all ports, which are connected to leds. |
| 0 | RGB_LED1 | O | RGB LED 5 on the zybo board |
| 0 | RGB_LED2 | O | RGB LED 6 on the zybo board |
| 0 | LEDS | O | Connection to the user leds on the board |
| IO | | IO | Basic IO from the Processor (reserved for future use) |

## 3.3 Detailed Design Description

### 3.3.1 Top design

The top design can be seen as simple connection between different units. The main role of top.vhd is connecting the “Blockdesign” to the Pblocks and the multiplexer. It implements a crossover connection to connect them

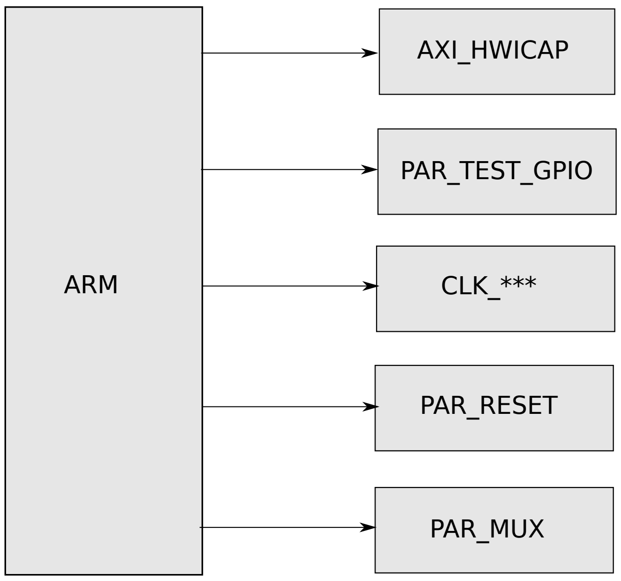


Figure 3: Overview of “Blockdesign” sub-units.

properly. These crossovers are used at the PAR_TEST_GPIO ports. Additionally the top design instantiates the IOBUFs for the outgoing PMOD ports, an entity which shows all incoming and outgoing ports. These can be easily split in 3 parts. The PMOD ports, the LEDs port and the DDR ports. As explained in Section 3.2 on page 6 they are connected to the on-board resources, for instance the physical LEDs on the board.

### 3.3.2 Blockdesign

As mentioned in the Section 3.2 on page 6 the block design consists of five major parts. These parts are the ARM processor core, the HWICAP IP core, the two dynamic clocks, all AXI testing GPIOs and finally the AXI (IPs PAR_RESET and MUX_GPIO). More in detail, the ARM processor core, is used for controlling and validating the operations of the system. If we look at the software on this processor, there is a Petalinux instance installed, where a custom program is running. Further details are inserted at Section 5, Petalinux. At the hardware side there are some AXI slaves. The PAR_TESTING_GPIOs can be controlled with the ARM processorand are used to test the implemented user design. Therefore, these are directly connected to the Pblocks. Additionally, there are two different AXI GPIOs connected, namely the MUX_GPIO which is responsible for switching the lanes at the multiplexer and the PAR_RESET GPIO. The PAR_RESET GPIO lanes are divided between all Pblocks, which means that every partial reconfiguration unit gets only one pin. This pin should be used to reset the implemented design. This allows the user to have a defined starting point at the beginning of his testing. Even though there are 16 pins, only 8 are in use (the others are reserved for future use). Another part consists of the dynamic clocks, CLK_***. These are used to provide a different clock compared to the system_clock to the partial reconfiguration blocks, since each design may need a different clock. Finally, there is also a AXI_HWICAP port, which is used to program the partial reconfiguration units via the ARM processor. All additional blocks are for the AXI bus. The ps7_0_axi_periph is just a simple interconnect to connect the M_AXI_GP0 of the zynq processor to the axi slave devices. The rst_ps7_0_100M IP block is a reset manager which distributes the FCLK_RESET0_N to all AXI devices. The util_ds_buf_1 is responsible for connecting the clock signals to a specific clock area in the FPGA.

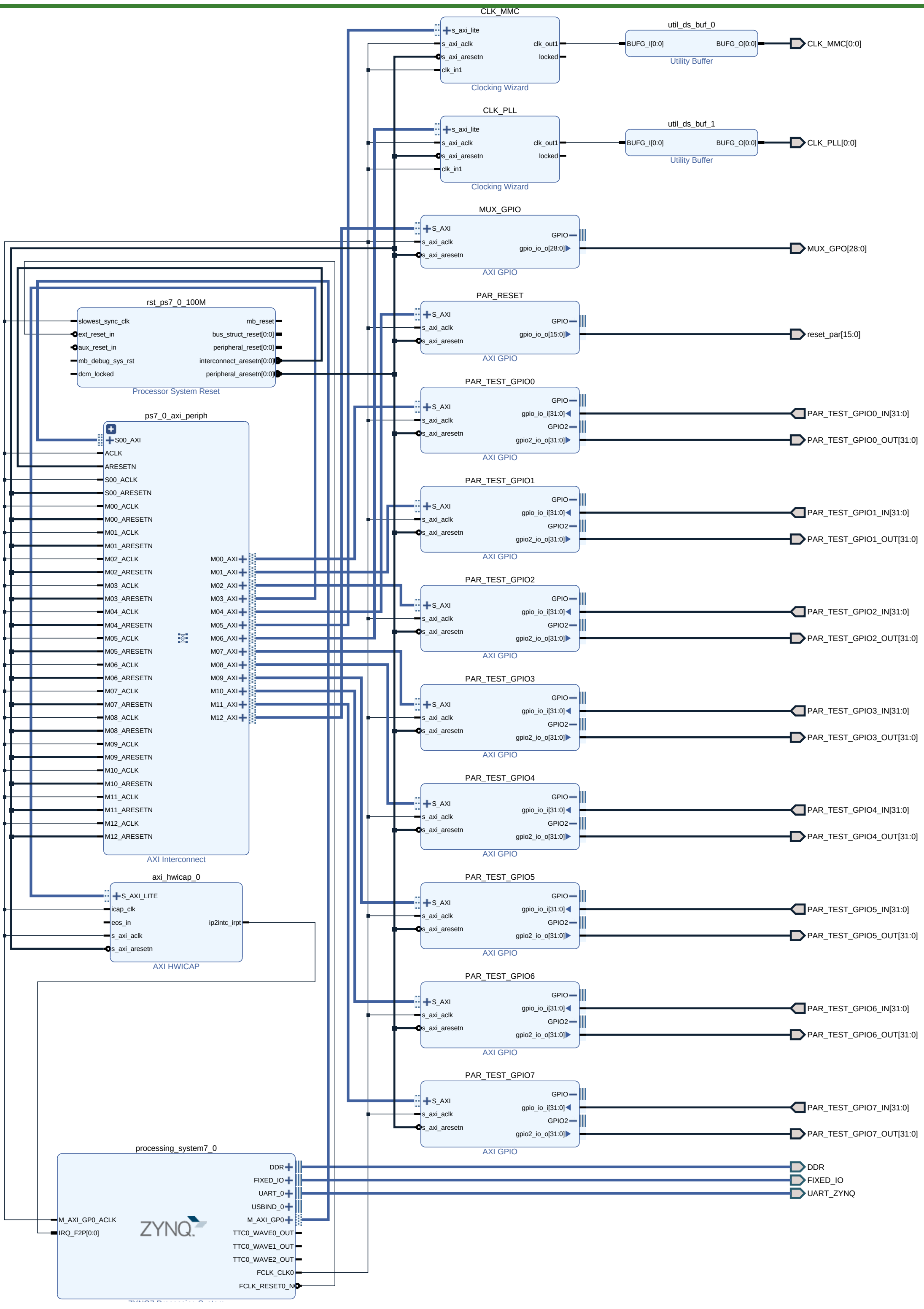


Figure 4: block_design.vhd

### 3.3.3 Mux

The mux is a standard multiplexer, as shown in Figure 5. It takes all LED and PMOD ports of the partial reconfiguration blocks and routes the used outputs of the blocks to the desired outputs on the board. This allows a safe connection to the outputs, as there are no double connections possible (due to a design rule error). Moreover, the processor can easily control this multiplexer via the MUX_GPIO controlling lines. At the actual design there are 8 input lanes defined but these can easily be extended if more parts are required. Each PMOD port consists of 3 lanes. One enable lane, one output lane and one input lane. This is necessary to convert this vectors to IO ports in the top design.

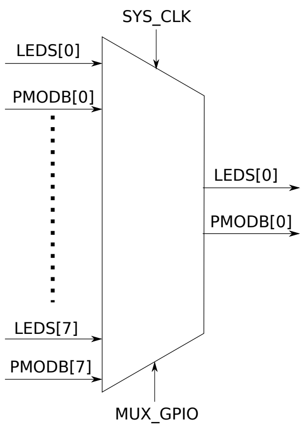


Figure 5: Mux block and its connections.

## 4 Server and TCL Vivado

### 4.1 Overview

To enable automated implementation for flashing the partial reconfiguration blocks, a standard C program is provided. This C program is capable of polling VHDL files from a specific folder and automatically create TCL scripts for Vivado. These scripts are used to automatically synthesise, create partial bitstreams for each design and implement the given designs into the partial reconfiguration blocks. More in detail, the program polls a specific folder for VHDL files. If there is one file, it expects a specific entity in these file which is parsed and checked for its port usage. Having this infomation, it is possible to synthesise the VHDL files and create a JSON file which includes all port information. This is necessary for the program on ARM which performs user management. After this step the Vivado report is parsed, and the design usage is analysed. More precisely, the LUT, DSP, and SLICE count is stored and the design is added to the smallest possible Pblock. This information is also written to the JSON file. Afterwards, the only remaining step is the creation of a bitstream generation TCL. This file will be automatically opened by Vivado and the bitstream generation will be started. The output partial bitstreams will be stored in a specific folder, where the Petalinux program (see Petalinux Section 5 on the following page) will be able to fetch it. After this process, the C program will look for new files to synthesise and the procedure starts again.

#### 4.1.1 Scheduler

As written in the above, the program will always look for new files and automatically processes them. At this point a small scheduler is implemented which analyses the free partial reconfiguration areas. This means that small designs will also be implemented in larger parts, if there are no free small partitions. After all partitions are filled or after a specific time, the scheduler will start from the beginning with a new "fileset". Additionally, this scheduler looks for remaining unfetched bitfiles, to prevent unintentionally overwriting them.

#### 4.1.2 Parser

There are three different parsers implemented. One for VHDL entity identification, one for TCL scripts and one for JSON. The VHDL parser searches for specific words and automatically filters and saves the necessary information, for instance the port usage. The TCL parser is used to modify predefined TCL scripts and adapt them for specific userfiles. The JSON parser is designed to write the collected information into a JSON file and edit the file, if necessary (It has to be modified if a user is added to a fileset).

# 5 Petalinux

## 5.1 Features and Specifications

**Feature:**

- Muli User support via JSON handoff
- Webserver on the FPGA board itself
- Easy to use web interface

## 5.2 Overview

The software part consists of the Petalinux operating system running on the ARM processor, the busybox webserver and a C program as CGI script running on the webserver. The program is executed everytime a user visits the website. It checks the username and flashes the correct *.bit file. The user is then allowed to use all predefined pins displayed on the website.

## 5.3 Detailed Description

### 5.3.1 Petalinux Configuration

In this project, Petalinux 2017.4 was used because of some known issues for partial reconfiguration (See section 6.1.2 on page 14).
Under petalinux-config -c kernel, the kernel was switched to the latest digilent linux kernel[2] for easier UIO[3] handling.

Under petalinux-config -c rootfs

- busybox
- digilent-apps[4]
- gpio(the webinterface)

were enabled.

[2]`https://github.com/Digilent/linux-digilent`
[3]Universal I/O
[4]`https://github.com/Digilent/digilent-apps`

### 5.3.2 Program

The program consists of a JSON parser, a partial flasher, libgio and the web handling section. It uses the JSON file to decide which *.bit file should be flashed and allows the user to toggle all enabled pins. It also features a live-stream of the webcam.

**JSON Parser** The JSON parser reads the necessary information out of the JSON file. It creates a struct for easier data handling. The created struct is shown below. json-c[5] is used for easier parsing.

```
typedef struct json_s {
        char users[9][256];
        char designs[9][256];
        int pblocks[9];
        char peripherals[9][7][256];
        int peripheral_count[9];
        int pins[9][32];
        int pin_count[9];
        int length;
} json_t;
```

Listing 1: JSON struct.

**Partial Flash** With this part, the *.bit file is parsed into a buffer and flashed into the /dev/xdevcfg device, after the partial_bitstream flag is set. Afterwards, it moves the file into a folder, or deletes it.

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

#define PATH "/home/root/par/%s"
#define OLD "/home/root/par/old/%s"


int flash_par_bitfile(char *name){
    char file_name[256];
    int fd;
    int ret;
    int size = 999999; //TODO get size of PBlock
    char buffer[size];

    if (name != '\0') {
        // compose file name
        sprintf(file_name, PATH, name);

        // open partial bitfile
        fd = open(file_name, O_RDONLY);
        if (fd < 0) {
            //printf("failed to open partial bitfile %s\n", file_name);
```

[5]https://github.com/json-c/json-c

```
            fflush(stdout);
            return -1;
        }

        // read partial bitfile into buffer
        ret = read(fd, buffer, size);
        if (ret < 0) {
            printf("failed to read partial bitfile %s\n", file_name);
            fflush(stdout);
            close(fd);
            return -1;
        }

        // close file handle
        close(fd);

                // Set is_partial_bitfile device attribute
                fd = open("/sys/devices/soc0/amba/f8007000.devcfg/is_partial_bitstream", O
                if (fd < 0) {
                        printf("failed to set xdevcfg attribute 'is_partial_bitstream'\n")
                        fflush(stdout);
                        return -1;
                }
                write(fd, "1", 2);
                close(fd);

                // Write partial bitfile to xdevcfg device
                fd = open("/dev/xdevcfg", O_RDWR);
                if (fd < 0) {
                        printf("failed to open xdevcfg device\n");
                        fflush(stdout);
                        return -1;
                }
                write(fd, buffer, size);

                char temp[256];
                sprintf(temp, OLD, name);
                rename(file_name, temp);
                close(fd);
    }

    return 0;
}
```

Listing 2: Partial flashing of bit files.

**libgpio**[6] The libgpio is used to handle all GPIO set/read operations. Every user has his own GPIO Block, as well as one reset line on the reset GPIO.

[6]`https://github.com/Digilent/libgpio`

**Web Handling** This part handles everything web-related and performs the user management. After a new user connects, it handles the right MUX position as well as the reset of the design.

#### 5.3.3 Build Instructions

To build the project, it is needed to be cloned on a compatible machine (see Section 1.5 on page 5). After that, it is possible to modify the source code of the webserver in sdk/gpio/src (IP, other webcam, etc).

```
source /path/to/vivado/settings64.sh
source /path/to/petalinux/settings64.sh

git clone --recursive https://github.com/braun-vogt/VHDL_Testing_Platform.git
cd VHDL_Testing_Platform
```

Listing 3: Setup

After an update of the hardware description or at first time use, it is necessary to run the following commands.

```
cd petalinux
petalinux-config --get-hw-description=../hw_handoff
petalinux-build
petalinux-package --boot --fsbl image/linux/zynq.elf
--fpga image/linux/top.bit --u-boot
```

Listing 4: Petalinux Build

The output files can be found under petalinux/image/linux/ .

# 6 Discussion

## 6.1 Problems Encountered

### 6.1.1 Dynamic Pblock Allocation

We tried to find an automatic way to implement all designs in one big Pblock. The goal was to be as economical as possible with the limited space on the FPGA fabric and have multiple designs combined, without harming previously implemented designs. Unfortunately, this solution is not supported by Xilinx, because this will lead to a Pblock in Pblock solutions (See UG947 / UG909 Xilinx references). Due to this limitation, we used 8 independent Pblocks.

### 6.1.2 Petalinux 2018.x / Partial Reconfiguration

Due to changes in Petalinux 2018.x regarding PR [7], we were forced to use Petalinux 2017.4. Petalinux switched from the xdevcfg utility to the FPGA Manager, which caused problems with the flash of the partial bitstream.

## 6.2 Engineering Resources Used

Estimated time is 300h/per student which sums to approximately 600 hours in total.

[7] Partial Reconfiguration

## 6.3 Marketability

Our project could be used in a variety of application. For example, FPGA Board manufactures could use this platform to give an online demo access to their potential customers. Companies producing IPs could provide access to their customers to interact with the IP block remotely. This platform also provides new opportunities for cloud service providers to offer FPGA prototyping and testing services to smaller companies. This would be in particular interesting for smaller companies who do no wish or intend to invest in acquiring high-end FPGA boards. Last but not least, as the focus of our project was, it could be used for (online) educational programs.

It is important to consider that the cost for acquiring and using all the licences that are required is relatively high. Therefore, the end user does not save only on the FPGA hardware costs, but also on the acquisition of licenses. However, both the software and hardware costs are negligible for the service provider since it is spread across a large quantity of users.

## 6.4 Community Feedback

We have received many positive and enthusiastic comments from the academic community. However, we have not presented our work to the industrial/business community and thus have not received any comments from them. This will be in our future plans.

## 7 References

- `http://svenand.blogdrives.com/archive/197.html`
- `https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841851/XAPP1231+-+Partial+Reconfiguration-of+a+Hardware+Accelerator+with+Vivado+Design+Suite`
- `https://github.com/json-c/json-c`
- `https://github.com/autosub-team/autosub`

# 8 Appendix A: top.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer: Felix Georg Braun
--
-- Create Date: 15.02.2019 16:46:24
-- Design Name: top
-- Module Name: top - Behavioral
-- Project Name: vhdl testing platform
-- Target Devices: Zybo Z7-20 (xc7z020clg400-1)
-- Tool Versions: VIVADO 2017.4
-- Revision:
-- Revision 0.01 - File Created
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity top is
        port (
                --user pmod
                PMOD_JB : inout STD_LOGIC_VECTOR(7 downto 0);
                PMOD_JC : inout STD_LOGIC_VECTOR(7 downto 0);
                PMOD_JD : inout STD_LOGIC_VECTOR(7 downto 0);
                PMOD_JE : inout STD_LOGIC_VECTOR(7 downto 0);

                --clk
                sys_clock : in STD_LOGIC;

                --part RGB
                RGB_LED1 : out STD_LOGIC_VECTOR (2 downto 0);
                RGB_LED2 : out STD_LOGIC_VECTOR (2 downto 0);
                leds : out STD_LOGIC_VECTOR (3 downto 0);

                DDR_addr : inout STD_LOGIC_VECTOR (14 downto 0);
                DDR_ba : inout STD_LOGIC_VECTOR (2 downto 0);
                DDR_cas_n : inout STD_LOGIC;
                DDR_ck_n : inout STD_LOGIC;
                DDR_ck_p : inout STD_LOGIC;
                DDR_cke : inout STD_LOGIC;
                DDR_cs_n : inout STD_LOGIC;
                DDR_dm : inout STD_LOGIC_VECTOR (3 downto 0);
                DDR_dq : inout STD_LOGIC_VECTOR (31 downto 0);
                DDR_dqs_n : inout STD_LOGIC_VECTOR (3 downto 0);
                DDR_dqs_p : inout STD_LOGIC_VECTOR (3 downto 0);
                DDR_odt : inout STD_LOGIC;
                DDR_ras_n : inout STD_LOGIC;
                DDR_reset_n : inout STD_LOGIC;
                DDR_we_n : inout STD_LOGIC;
                FIXED_IO_ddr_vrn : inout STD_LOGIC;
                FIXED_IO_ddr_vrp : inout STD_LOGIC;
                FIXED_IO_mio : inout STD_LOGIC_VECTOR (53 downto 0);
                FIXED_IO_ps_clk : inout STD_LOGIC;
                FIXED_IO_ps_porb : inout STD_LOGIC;
                FIXED_IO_ps_srstb : inout STD_LOGIC
        );
end top;

architecture Behavioral of top is
        component MUX is
                port (
                        SYSCLK_125MHZ : in STD_LOGIC;
                        MUX_GPIO : in STD_LOGIC_VECTOR(28 downto 0);
                        RGB_LED1 : out STD_LOGIC_VECTOR(2 downto 0);
                        RGB_LED2 : out STD_LOGIC_VECTOR(2 downto 0);
                        LEDS : out STD_LOGIC_VECTOR(3 downto 0);

                        PMOD_JB_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JC_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JD_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JE_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JE_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JE_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        --par0
                        PMOD_JB_IN_0 : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OUT_0 : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OE_0 : in STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JC_IN_0 : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OUT_0 : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OE_0 : in STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JD_IN_0 : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OUT_0 : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OE_0 : in STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JE_IN_0 : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JE_OUT_0 : in STD_LOGIC_VECTOR(7 downto 0);
```

```
PMOD_JE_OE_0 : in STD_LOGIC_VECTOR(7 downto 0);

RGB_LED1_0 : in STD_LOGIC_VECTOR(2 downto 0);
RGB_LED2_0 : in STD_LOGIC_VECTOR(2 downto 0);
LEDS_0 : in STD_LOGIC_VECTOR(3 downto 0);

--par1
PMOD_JB_IN_1 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OUT_1 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OE_1 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JC_IN_1 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OUT_1 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OE_1 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JD_IN_1 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OUT_1 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OE_1 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JE_IN_1 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OUT_1 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OE_1 : in STD_LOGIC_VECTOR(7 downto 0);

RGB_LED1_1 : in STD_LOGIC_VECTOR(2 downto 0);
RGB_LED2_1 : in STD_LOGIC_VECTOR(2 downto 0);
LEDS_1 : in STD_LOGIC_VECTOR(3 downto 0);
--par2
PMOD_JB_IN_2 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OUT_2 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OE_2 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JC_IN_2 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OUT_2 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OE_2 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JD_IN_2 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OUT_2 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OE_2 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JE_IN_2 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OUT_2 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OE_2 : in STD_LOGIC_VECTOR(7 downto 0);

RGB_LED1_2 : in STD_LOGIC_VECTOR(2 downto 0);
RGB_LED2_2 : in STD_LOGIC_VECTOR(2 downto 0);
LEDS_2 : in STD_LOGIC_VECTOR(3 downto 0);

--par3
PMOD_JB_IN_3 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OUT_3 : in STD_LOGIC_VECTOR(7 downto 0);
```

```
PMOD_JB_OE_3 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JC_IN_3 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OUT_3 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OE_3 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JD_IN_3 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OUT_3 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OE_3 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JE_IN_3 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OUT_3 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OE_3 : in STD_LOGIC_VECTOR(7 downto 0);

RGB_LED1_3 : in STD_LOGIC_VECTOR(2 downto 0);
RGB_LED2_3 : in STD_LOGIC_VECTOR(2 downto 0);
LEDS_3 : in STD_LOGIC_VECTOR(3 downto 0);

--par4
PMOD_JB_IN_4 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OUT_4 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OE_4 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JC_IN_4 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OUT_4 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OE_4 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JD_IN_4 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OUT_4 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OE_4 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JE_IN_4 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OUT_4 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OE_4 : in STD_LOGIC_VECTOR(7 downto 0);

RGB_LED1_4 : in STD_LOGIC_VECTOR(2 downto 0);
RGB_LED2_4 : in STD_LOGIC_VECTOR(2 downto 0);
LEDS_4 : in STD_LOGIC_VECTOR(3 downto 0);

--par5
PMOD_JB_IN_5 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OUT_5 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OE_5 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JC_IN_5 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OUT_5 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OE_5 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JD_IN_5 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OUT_5 : in STD_LOGIC_VECTOR(7 downto 0);
```

```
PMOD_JD_OE_5 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JE_IN_5 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OUT_5 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OE_5 : in STD_LOGIC_VECTOR(7 downto 0);

RGB_LED1_5 : in STD_LOGIC_VECTOR(2 downto 0);
RGB_LED2_5 : in STD_LOGIC_VECTOR(2 downto 0);
LEDS_5 : in STD_LOGIC_VECTOR(3 downto 0);

--par6
PMOD_JB_IN_6 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OUT_6 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OE_6 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JC_IN_6 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OUT_6 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OE_6 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JD_IN_6 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OUT_6 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OE_6 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JE_IN_6 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OUT_6 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OE_6 : in STD_LOGIC_VECTOR(7 downto 0);

RGB_LED1_6 : in STD_LOGIC_VECTOR(2 downto 0);
RGB_LED2_6 : in STD_LOGIC_VECTOR(2 downto 0);
LEDS_6 : in STD_LOGIC_VECTOR(3 downto 0);

--par7
PMOD_JB_IN_7 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OUT_7 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OE_7 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JC_IN_7 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OUT_7 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OE_7 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JD_IN_7 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OUT_7 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OE_7 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JE_IN_7 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OUT_7 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OE_7 : in STD_LOGIC_VECTOR(7 downto 0);

RGB_LED1_7 : in STD_LOGIC_VECTOR(2 downto 0);
RGB_LED2_7 : in STD_LOGIC_VECTOR(2 downto 0);
```

```
                        LEDS_7 : in STD_LOGIC_VECTOR(3 downto 0)

                );
        end component;


        ----------------------------------------------------------------
        --ADD PARTIAL RECONFIGURATIONS 0 to 7
        ----------------------------------------------------------------
        component par0 is
                port (
                        CLK_125MHZ : in STD_LOGIC;
                        CLK_MMC : in STD_LOGIC_VECTOR (0 to 0);
                        CLK_PLL : in STD_LOGIC_VECTOR (0 to 0);

                        reset : in STD_LOGIC;

                        PAR_TEST_GPIO0_IN : in STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO0_OUT : out STD_LOGIC_VECTOR (31 downto 0);

                        PMOD_JB_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JC_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JD_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JE_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JE_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JE_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        leds : out STD_LOGIC_VECTOR (3 downto 0);

                        RGB_LED : out STD_LOGIC_VECTOR (2 downto 0);
                        RGB_LED2 : out STD_LOGIC_VECTOR (2 downto 0)
                );
        end component;

        component par1 is
                port (
                        CLK_125MHZ : in STD_LOGIC;
                        CLK_MMC : in STD_LOGIC_VECTOR (0 to 0);
                        CLK_PLL : in STD_LOGIC_VECTOR (0 to 0);

                        reset : in STD_LOGIC;
                        PAR_TEST_GPIO0_IN : in STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO0_OUT : out STD_LOGIC_VECTOR (31 downto 0);
                        PMOD_JB_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JC_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JD_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JE_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JE_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JE_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        leds : out STD_LOGIC_VECTOR (3 downto 0);

                        RGB_LED : out STD_LOGIC_VECTOR (2 downto 0);
                        RGB_LED2 : out STD_LOGIC_VECTOR (2 downto 0)
                );
        end component;

        component par2 is
                port (
                        CLK_125MHZ : in STD_LOGIC;
                        CLK_MMC : in STD_LOGIC_VECTOR (0 to 0);
                        CLK_PLL : in STD_LOGIC_VECTOR (0 to 0);

                        reset : in STD_LOGIC;

                        PAR_TEST_GPIO0_IN : in STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO0_OUT : out STD_LOGIC_VECTOR (31 downto 0);
                        PMOD_JB_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JC_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JD_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JE_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JE_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JE_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        leds : out STD_LOGIC_VECTOR (3 downto 0);

                        RGB_LED : out STD_LOGIC_VECTOR (2 downto 0);
                        RGB_LED2 : out STD_LOGIC_VECTOR (2 downto 0)
                );
        end component;

        component par3 is
                port (
                        CLK_125MHZ : in STD_LOGIC;
                        CLK_MMC : in STD_LOGIC_VECTOR (0 to 0);
                        CLK_PLL : in STD_LOGIC_VECTOR (0 to 0);

                        reset : in STD_LOGIC;

                        PAR_TEST_GPIO0_IN : in STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO0_OUT : out STD_LOGIC_VECTOR (31 downto 0);

                        PMOD_JB_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JC_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JD_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JE_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JE_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JE_OE : out STD_LOGIC_VECTOR(7 downto 0);
                        leds : out STD_LOGIC_VECTOR (3 downto 0);

                        RGB_LED : out STD_LOGIC_VECTOR (2 downto 0);
                        RGB_LED2 : out STD_LOGIC_VECTOR (2 downto 0)
                );
        end component;

        component par4 is
                port (
                        CLK_125MHZ : in STD_LOGIC;
                        CLK_MMC : in STD_LOGIC_VECTOR (0 to 0);
                        CLK_PLL : in STD_LOGIC_VECTOR (0 to 0);

                        reset : in STD_LOGIC;
                        PAR_TEST_GPIO0_IN : in STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO0_OUT : out STD_LOGIC_VECTOR (31 downto 0);

                        PMOD_JB_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JC_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JD_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JE_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JE_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JE_OE : out STD_LOGIC_VECTOR(7 downto 0);
                        leds : out STD_LOGIC_VECTOR (3 downto 0);

                        RGB_LED : out STD_LOGIC_VECTOR (2 downto 0);
                        RGB_LED2 : out STD_LOGIC_VECTOR (2 downto 0)
                );
        end component;

        component par5 is
                port (
                        CLK_125MHZ : in STD_LOGIC;
                        CLK_MMC : in STD_LOGIC_VECTOR (0 to 0);
                        CLK_PLL : in STD_LOGIC_VECTOR (0 to 0);

                        reset : in STD_LOGIC;

                        PAR_TEST_GPIO0_IN : in STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO0_OUT : out STD_LOGIC_VECTOR (31 downto 0);

                        PMOD_JB_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JC_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JD_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JE_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JE_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JE_OE : out STD_LOGIC_VECTOR(7 downto 0);
                        leds : out STD_LOGIC_VECTOR (3 downto 0);

                        RGB_LED : out STD_LOGIC_VECTOR (2 downto 0);
                        RGB_LED2 : out STD_LOGIC_VECTOR (2 downto 0)
                );
        end component;

        component par6 is
                port (
                        CLK_125MHZ : in STD_LOGIC;
                        CLK_MMC : in STD_LOGIC_VECTOR (0 to 0);
                        CLK_PLL : in STD_LOGIC_VECTOR (0 to 0);

                        reset : in STD_LOGIC;

                        PAR_TEST_GPIO0_IN : in STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO0_OUT : out STD_LOGIC_VECTOR (31 downto 0);

                        PMOD_JB_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JC_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JD_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JE_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JE_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JE_OE : out STD_LOGIC_VECTOR(7 downto 0);
                        leds : out STD_LOGIC_VECTOR (3 downto 0);

                        RGB_LED : out STD_LOGIC_VECTOR (2 downto 0);
                        RGB_LED2 : out STD_LOGIC_VECTOR (2 downto 0)
                );
        end component;


        component par7 is
                port (
                        CLK_125MHZ : in STD_LOGIC;
                        CLK_MMC : in STD_LOGIC_VECTOR (0 to 0);
                        CLK_PLL : in STD_LOGIC_VECTOR (0 to 0);

                        reset : in STD_LOGIC;
                        UART_ZYNQ_txd : in STD_LOGIC;
                        UART_ZYNQ_rxd : out STD_LOGIC;

                        PAR_TEST_GPIO0_IN : in STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO0_OUT : out STD_LOGIC_VECTOR (31 downto 0);

                        PMOD_JB_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JB_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JC_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JC_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JD_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JD_OE : out STD_LOGIC_VECTOR(7 downto 0);

                        PMOD_JE_IN : in STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JE_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                        PMOD_JE_OE : out STD_LOGIC_VECTOR(7 downto 0);
                        leds : out STD_LOGIC_VECTOR (3 downto 0);

                        RGB_LED : out STD_LOGIC_VECTOR (2 downto 0);
                        RGB_LED2 : out STD_LOGIC_VECTOR (2 downto 0)
                );
        end component;

        component bd_name_wrapper is
                port (
                        DDR_cas_n : inout STD_LOGIC;
                        DDR_cke : inout STD_LOGIC;
                        DDR_ck_n : inout STD_LOGIC;
                        DDR_ck_p : inout STD_LOGIC;
                        DDR_cs_n : inout STD_LOGIC;
                        DDR_reset_n : inout STD_LOGIC;
                        DDR_odt : inout STD_LOGIC;
                        DDR_ras_n : inout STD_LOGIC;
                        DDR_we_n : inout STD_LOGIC;
                        DDR_ba : inout STD_LOGIC_VECTOR (2 downto 0);
                        DDR_addr : inout STD_LOGIC_VECTOR (14 downto 0);
                        DDR_dm : inout STD_LOGIC_VECTOR (3 downto 0);
                        DDR_dq : inout STD_LOGIC_VECTOR (31 downto 0);
                        DDR_dqs_n : inout STD_LOGIC_VECTOR (3 downto 0);
                        DDR_dqs_p : inout STD_LOGIC_VECTOR (3 downto 0);
                        FIXED_IO_mio : inout STD_LOGIC_VECTOR (53 downto 0);
                        FIXED_IO_ddr_vrn : inout STD_LOGIC;
                        FIXED_IO_ddr_vrp : inout STD_LOGIC;
                        FIXED_IO_ps_srstb : inout STD_LOGIC;
                        FIXED_IO_ps_clk : inout STD_LOGIC;
                        FIXED_IO_ps_porb : inout STD_LOGIC;
                        UART_ZYNQ_txd : out STD_LOGIC;
                        UART_ZYNQ_rxd : in STD_LOGIC;
                        CLK_MMC : out STD_LOGIC_VECTOR (0 to 0);
                        CLK_PLL : out STD_LOGIC_VECTOR (0 to 0);
                        reset_par : out STD_LOGIC_VECTOR (15 downto 0);
                        PAR_TEST_GPIO0_IN : in STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO0_OUT : out STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO1_IN : in STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO2_IN : in STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO3_IN : in STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO4_IN : in STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO5_IN : in STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO6_IN : in STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO7_IN : in STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO1_OUT : out STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO2_OUT : out STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO3_OUT : out STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO5_OUT : out STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO6_OUT : out STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO7_OUT : out STD_LOGIC_VECTOR (31 downto 0);
                        PAR_TEST_GPIO4_OUT : out STD_LOGIC_VECTOR (31 downto 0);
                        MUX_GPO : out STD_LOGIC_VECTOR (28 downto 0 )
                );
        end component;


    ----------------------------------------------------------------------------
    --IOBUF for threestate Signals
    ---------------------------------------------------------------------------
        component IOBUF is
                port (
                        I : in STD_LOGIC;
                        O : out STD_LOGIC;
                        T : in STD_LOGIC;
                        IO : inout STD_LOGIC
                );
        end component;

        --gpio signals
        signal reset_par : STD_LOGIC_VECTOR (15 downto 0);

        signal PAR_TEST_GPIO0_IN_s : STD_LOGIC_VECTOR (31 downto 0);
        signal PAR_TEST_GPIO0_OUT_s : STD_LOGIC_VECTOR (31 downto 0);

        signal PAR_TEST_GPIO1_IN : STD_LOGIC_VECTOR (31 downto 0);
        signal PAR_TEST_GPIO1_OUT : STD_LOGIC_VECTOR (31 downto 0);

        signal PAR_TEST_GPIO2_IN : STD_LOGIC_VECTOR (31 downto 0);
        signal PAR_TEST_GPIO2_OUT : STD_LOGIC_VECTOR (31 downto 0);
```

```
signal PAR_TEST_GPIO3_IN_S : STD_LOGIC_VECTOR (31 downto 0);
signal PAR_TEST_GPIO3_OUT_S : STD_LOGIC_VECTOR (31 downto 0);

signal PAR_TEST_GPIO4_IN : STD_LOGIC_VECTOR (31 downto 0);
signal PAR_TEST_GPIO4_OUT : STD_LOGIC_VECTOR (31 downto 0);

signal PAR_TEST_GPIO5_IN : STD_LOGIC_VECTOR (31 downto 0);
signal PAR_TEST_GPIO5_OUT : STD_LOGIC_VECTOR (31 downto 0);

signal PAR_TEST_GPIO6_IN : STD_LOGIC_VECTOR (31 downto 0);
signal PAR_TEST_GPIO6_OUT : STD_LOGIC_VECTOR (31 downto 0);

signal PAR_TEST_GPIO7_IN : STD_LOGIC_VECTOR (31 downto 0);
signal PAR_TEST_GPIO7_OUT : STD_LOGIC_VECTOR (31 downto 0);
signal CLK_MMC : STD_LOGIC_VECTOR(0 downto 0);
signal CLK_PLL : STD_LOGIC_VECTOR(0 downto 0);
signal sys_clk : STD_LOGIC_VECTOR(0 downto 0);

signal UART_ZYNQ_rxd : STD_LOGIC;
signal UART_ZYNQ_txd : STD_LOGIC;

signal PMOD_JB_IN : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JB_OUT : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JB_OE : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JC_IN : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JC_OUT : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JC_OE : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_IN : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_OUT : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_OE : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_IN : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_OUT : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_OE : STD_LOGIC_VECTOR(7 downto 0);

--par0
signal PMOD_JB_IN_0 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JB_OUT_0 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JB_OE_0 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JC_IN_0 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JC_OUT_0 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JC_OE_0 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_IN_0 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_OUT_0 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_OE_0 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_IN_0 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_OUT_0 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_OE_0 : STD_LOGIC_VECTOR(7 downto 0);
signal RGB_LED1_0 : STD_LOGIC_VECTOR(2 downto 0);
signal RGB_LED2_0 : STD_LOGIC_VECTOR(2 downto 0);
signal LEDS_0 : STD_LOGIC_VECTOR(3 downto 0);

--par1
signal PMOD_JB_IN_1 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JB_OUT_1 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JB_OE_1 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JC_IN_1 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JC_OUT_1 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JC_OE_1 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JD_IN_1 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_OUT_1 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_OE_1 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JE_IN_1 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_OUT_1 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_OE_1 : STD_LOGIC_VECTOR(7 downto 0);

signal RGB_LED1_1 : STD_LOGIC_VECTOR(2 downto 0);
signal RGB_LED2_1 : STD_LOGIC_VECTOR(2 downto 0);
signal LEDS_1 : STD_LOGIC_VECTOR(3 downto 0);
--par2
signal PMOD_JB_IN_2 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JB_OUT_2 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JB_OE_2 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JC_IN_2 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JC_OUT_2 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JC_OE_2 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JD_IN_2 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_OUT_2 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_OE_2 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JE_IN_2 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_OUT_2 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_OE_2 : STD_LOGIC_VECTOR(7 downto 0);

signal RGB_LED1_2 : STD_LOGIC_VECTOR(2 downto 0);
signal RGB_LED2_2 : STD_LOGIC_VECTOR(2 downto 0);
signal LEDS_2 : STD_LOGIC_VECTOR(3 downto 0);

--par3
signal PMOD_JB_IN_3 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JB_OUT_3 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JB_OE_3 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JC_IN_3 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JC_OUT_3 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JC_OE_3 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JD_IN_3 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_OUT_3 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_OE_3 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JE_IN_3 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_OUT_3 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_OE_3 : STD_LOGIC_VECTOR(7 downto 0);

signal RGB_LED1_3 : STD_LOGIC_VECTOR(2 downto 0);
signal RGB_LED2_3 : STD_LOGIC_VECTOR(2 downto 0);
signal LEDS_3 : STD_LOGIC_VECTOR(3 downto 0);

--par4
signal PMOD_JB_IN_4 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JB_OUT_4 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JB_OE_4 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JC_IN_4 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JC_OUT_4 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JC_OE_4 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JD_IN_4 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_OUT_4 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_OE_4 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JE_IN_4 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_OUT_4 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_OE_4 : STD_LOGIC_VECTOR(7 downto 0);

signal RGB_LED1_4 : STD_LOGIC_VECTOR(2 downto 0);
signal RGB_LED2_4 : STD_LOGIC_VECTOR(2 downto 0);
signal LEDS_4 : STD_LOGIC_VECTOR(3 downto 0);

--par5
signal PMOD_JB_IN_5 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JB_OUT_5 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JB_OE_5 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JC_IN_5 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JC_OUT_5 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JC_OE_5 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JD_IN_5 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_OUT_5 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_OE_5 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_IN_5 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_OUT_5 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_OE_5 : STD_LOGIC_VECTOR(7 downto 0);

signal RGB_LED1_5 : STD_LOGIC_VECTOR(2 downto 0);
signal RGB_LED2_5 : STD_LOGIC_VECTOR(2 downto 0);
signal LEDS_5 : STD_LOGIC_VECTOR(3 downto 0);

--par6
signal PMOD_JB_IN_6 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JB_OUT_6 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JB_OE_6 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JC_IN_6 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JC_OUT_6 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JC_OE_6 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JD_IN_6 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_OUT_6 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_OE_6 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JE_IN_6 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_OUT_6 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_OE_6 : STD_LOGIC_VECTOR(7 downto 0);

signal RGB_LED1_6 : STD_LOGIC_VECTOR(2 downto 0);
signal RGB_LED2_6 : STD_LOGIC_VECTOR(2 downto 0);
signal LEDS_6 : STD_LOGIC_VECTOR(3 downto 0);

--par7
signal PMOD_JB_IN_7 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JB_OUT_7 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JB_OE_7 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JC_IN_7 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JC_OUT_7 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JC_OE_7 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JD_IN_7 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_OUT_7 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JD_OE_7 : STD_LOGIC_VECTOR(7 downto 0);

signal PMOD_JE_IN_7 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_OUT_7 : STD_LOGIC_VECTOR(7 downto 0);
signal PMOD_JE_OE_7 : STD_LOGIC_VECTOR(7 downto 0);

signal RGB_LED1_7 : STD_LOGIC_VECTOR(2 downto 0);
signal RGB_LED2_7 : STD_LOGIC_VECTOR(2 downto 0);
signal LEDS_7 : STD_LOGIC_VECTOR(3 downto 0);
```

```
        signal MUX_GPIO : STD_LOGIC_VECTOR(28 downto 0);
        signal RGB_LED_obuf : STD_LOGIC_VECTOR(2 downto 0);

begin
        bd_name_wrapper_port_map : bd_name_wrapper
        port map(
                --CLK
                CLK_MMC => CLK_MMC,
                CLK_PLL => CLK_PLL,

                --DDR
                DDR_cas_n => DDR_cas_n,
                DDR_cke => DDR_cke,
                DDR_ck_n => DDR_ck_n,
                DDR_ck_p => DDR_ck_p,
                DDR_cs_n => DDR_cs_n,
                DDR_reset_n => DDR_reset_n,
                DDR_odt => DDR_odt,
                DDR_ras_n => DDR_ras_n,
                DDR_we_n => DDR_we_n,
                DDR_ba => DDR_ba,
                DDR_addr => DDR_addr,
                DDR_dm => DDR_dm,
                DDR_dq => DDR_dq,
                DDR_dqs_n => DDR_dqs_n,
                DDR_dqs_p => DDR_dqs_p,
                --FIXEDIO
                FIXED_IO_mio => FIXED_IO_mio,
                FIXED_IO_ddr_vrn => FIXED_IO_ddr_vrn,
                FIXED_IO_ddr_vrp => FIXED_IO_ddr_vrp,
                FIXED_IO_ps_srstb => FIXED_IO_ps_srstb,
                FIXED_IO_ps_clk => FIXED_IO_ps_clk,
                FIXED_IO_ps_porb => FIXED_IO_ps_porb,

                --UART
                UART_ZYNQ_txd => UART_ZYNQ_txd,
                UART_ZYNQ_rxd => UART_ZYNQ_rxd,

                --Shared Reset
                reset_par => reset_par,
                PAR_TEST_GPIO0_IN => PAR_TEST_GPIO0_IN_s,
                PAR_TEST_GPIO1_IN => PAR_TEST_GPIO1_IN,
                PAR_TEST_GPIO2_IN => PAR_TEST_GPIO2_IN,
                PAR_TEST_GPIO3_IN => PAR_TEST_GPIO3_IN_S,
                PAR_TEST_GPIO4_IN => PAR_TEST_GPIO4_IN,
                PAR_TEST_GPIO5_IN => PAR_TEST_GPIO5_IN,
                PAR_TEST_GPIO6_IN => PAR_TEST_GPIO6_IN,
                PAR_TEST_GPIO7_IN => PAR_TEST_GPIO7_IN,
                PAR_TEST_GPIO0_OUT => PAR_TEST_GPIO0_OUT_s,
                PAR_TEST_GPIO1_OUT => PAR_TEST_GPIO1_OUT,
                PAR_TEST_GPIO2_OUT => PAR_TEST_GPIO2_OUT,
                PAR_TEST_GPIO3_OUT => PAR_TEST_GPIO3_OUT_S,
                PAR_TEST_GPIO4_OUT => PAR_TEST_GPIO4_OUT,
                PAR_TEST_GPIO5_OUT => PAR_TEST_GPIO5_OUT,
                PAR_TEST_GPIO6_OUT => PAR_TEST_GPIO6_OUT,
                PAR_TEST_GPIO7_OUT => PAR_TEST_GPIO7_OUT,

                MUX_GPO => MUX_GPIO
        );

        par0pm : par0
        port map(
                CLK_125MHZ => sys_clock ,
                CLK_MMC => CLK_MMC,
                CLK_PLL => CLK_PLL,

                reset => reset_par(0),

                PAR_TEST_GPIO0_IN => PAR_TEST_GPIO0_OUT_S,
                PAR_TEST_GPIO0_OUT => PAR_TEST_GPIO0_IN_S,

                PMOD_JB_IN => PMOD_JB_IN_0,
                PMOD_JB_OUT => PMOD_JB_OUT_0,
                PMOD_JB_OE => PMOD_JB_OE_0,

                PMOD_JC_IN => PMOD_JC_IN_0,
                PMOD_JC_OUT => PMOD_JC_OUT_0,
                PMOD_JC_OE => PMOD_JC_OE_0,

                PMOD_JD_IN => PMOD_JD_IN_0,
                PMOD_JD_OUT => PMOD_JD_OUT_0,
                PMOD_JD_OE => PMOD_JD_OE_0,

                PMOD_JE_IN => PMOD_JE_IN_0,
                PMOD_JE_OUT => PMOD_JE_OUT_0,
                PMOD_JE_OE => PMOD_JE_OE_0,

                leds => LEDS_0,

                RGB_LED => RGB_LED1_0,
                RGB_LED2 => RGB_LED2_0
        );

        par1pm : par1
        port map(
                CLK_125MHZ => sys_clock ,
                CLK_MMC => CLK_MMC,
                CLK_PLL => CLK_PLL,
                reset => reset_par(1),

                PAR_TEST_GPIO0_IN => PAR_TEST_GPIO1_OUT,
                PAR_TEST_GPIO0_OUT => PAR_TEST_GPIO1_IN,

                PMOD_JB_IN => PMOD_JB_IN_1,
                PMOD_JB_OUT => PMOD_JB_OUT_1,
                PMOD_JB_OE => PMOD_JB_OE_1,

                PMOD_JC_IN => PMOD_JC_IN_1,
                PMOD_JC_OUT => PMOD_JC_OUT_1,
                PMOD_JC_OE => PMOD_JC_OE_1,

                PMOD_JD_IN => PMOD_JD_IN_1,
                PMOD_JD_OUT => PMOD_JD_OUT_1,
                PMOD_JD_OE => PMOD_JD_OE_1,

                PMOD_JE_IN => PMOD_JE_IN_1,
                PMOD_JE_OUT => PMOD_JE_OUT_1,
                PMOD_JE_OE => PMOD_JE_OE_1,

                leds => LEDS_1,

                RGB_LED => RGB_LED1_1,
                RGB_LED2 => RGB_LED2_1
        );
        par2pm : par2
        port map(
                CLK_125MHZ => sys_clock,
                CLK_MMC => CLK_MMC,
                CLK_PLL => CLK_PLL,

                reset => reset_par(2),

                PAR_TEST_GPIO0_IN => PAR_TEST_GPIO2_OUT,
                PAR_TEST_GPIO0_OUT => PAR_TEST_GPIO2_IN,

                PMOD_JB_IN => PMOD_JB_IN_2,
                PMOD_JB_OUT => PMOD_JB_OUT_2,
                PMOD_JB_OE => PMOD_JB_OE_2,

                PMOD_JC_IN => PMOD_JC_IN_2,
                PMOD_JC_OUT => PMOD_JC_OUT_2,
                PMOD_JC_OE => PMOD_JC_OE_2,

                PMOD_JD_IN => PMOD_JD_IN_2,
                PMOD_JD_OUT => PMOD_JD_OUT_2,
                PMOD_JD_OE => PMOD_JD_OE_2,
                PMOD_JE_IN => PMOD_JE_IN_2,
                PMOD_JE_OUT => PMOD_JE_OUT_2,
                PMOD_JE_OE => PMOD_JE_OE_2,

                leds => LEDS_2,

                RGB_LED => RGB_LED1_2,
                RGB_LED2 => RGB_LED2_2
        );

        par3pm : par3
        port map(
                CLK_125MHZ => sys_clock ,
                CLK_MMC => CLK_MMC,
                CLK_PLL => CLK_PLL,

                reset => reset_par(3),

                PAR_TEST_GPIO0_IN => PAR_TEST_GPIO3_OUT_S,
                PAR_TEST_GPIO0_OUT => PAR_TEST_GPIO3_IN_S,

                PMOD_JB_IN => PMOD_JB_IN_3,
                PMOD_JB_OUT => PMOD_JB_OUT_3,
                PMOD_JB_OE => PMOD_JB_OE_3,

                PMOD_JC_IN => PMOD_JC_IN_3,
                PMOD_JC_OUT => PMOD_JC_OUT_3,
                PMOD_JC_OE => PMOD_JC_OE_3,

                PMOD_JD_IN => PMOD_JD_IN_3,
                PMOD_JD_OUT => PMOD_JD_OUT_3,
                PMOD_JD_OE => PMOD_JD_OE_3,

                PMOD_JE_IN => PMOD_JE_IN_3,
                PMOD_JE_OUT => PMOD_JE_OUT_3,
                PMOD_JE_OE => PMOD_JE_OE_3,

                leds => LEDS_3,

                RGB_LED => RGB_LED1_3,
                RGB_LED2 => RGB_LED2_3
        );

        par4pm : par4
        port map(
                CLK_125MHZ => sys_clock ,
                CLK_MMC => CLK_MMC,
                CLK_PLL => CLK_PLL,

                reset => reset_par(4),
                PAR_TEST_GPIO0_IN => PAR_TEST_GPIO4_OUT,
                PAR_TEST_GPIO0_OUT => PAR_TEST_GPIO4_In,

                PMOD_JB_IN => PMOD_JB_IN_4,
                PMOD_JB_OUT => PMOD_JB_OUT_4,
                PMOD_JB_OE => PMOD_JB_OE_4,

                PMOD_JC_IN => PMOD_JC_IN_4,
                PMOD_JC_OUT => PMOD_JC_OUT_4,
                PMOD_JC_OE => PMOD_JC_OE_4,

                PMOD_JD_IN => PMOD_JD_IN_4,
                PMOD_JD_OUT => PMOD_JD_OUT_4,
                PMOD_JD_OE => PMOD_JD_OE_4,

                PMOD_JE_IN => PMOD_JE_IN_4,
                PMOD_JE_OUT => PMOD_JE_OUT_4,
                PMOD_JE_OE => PMOD_JE_OE_4,

                leds => LEDS_4,

                RGB_LED => RGB_LED1_4,
                RGB_LED2 => RGB_LED2_4
        );

        par5pm : par5
        port map(
                CLK_125MHZ => sys_clock ,
                CLK_MMC => CLK_MMC,
                CLK_PLL => CLK_PLL,

                reset => reset_par(5),

                PAR_TEST_GPIO0_IN => PAR_TEST_GPIO5_OUT,
                PAR_TEST_GPIO0_OUT => PAR_TEST_GPIO5_IN,

                PMOD_JB_IN => PMOD_JB_IN_5,
                PMOD_JB_OUT => PMOD_JB_OUT_5,
                PMOD_JB_OE => PMOD_JB_OE_5
                ,
                PMOD_JC_IN => PMOD_JC_IN_5,
                PMOD_JC_OUT => PMOD_JC_OUT_5,
                PMOD_JC_OE => PMOD_JC_OE_5,

                PMOD_JD_IN => PMOD_JD_IN_5,
                PMOD_JD_OUT => PMOD_JD_OUT_5,
                PMOD_JD_OE => PMOD_JD_OE_5,

                PMOD_JE_IN => PMOD_JE_IN_5,
                PMOD_JE_OUT => PMOD_JE_OUT_5,
                PMOD_JE_OE => PMOD_JE_OE_5,

                leds => LEDS_5,

                RGB_LED => RGB_LED1_5,
                RGB_LED2 => RGB_LED2_5
        );

        par6pm : par6
        port map(
                CLK_125MHZ => sys_clock ,
                CLK_MMC => CLK_MMC,
                CLK_PLL => CLK_PLL,

                reset => reset_par(6),

                PAR_TEST_GPIO0_IN => PAR_TEST_GPIO6_OUT,
                PAR_TEST_GPIO0_OUT => PAR_TEST_GPIO6_IN,

                PMOD_JB_IN => PMOD_JB_IN_6,
                PMOD_JB_OUT => PMOD_JB_OUT_6,
                PMOD_JB_OE => PMOD_JB_OE_6,

                PMOD_JC_IN => PMOD_JC_IN_6,
                PMOD_JC_OUT => PMOD_JC_OUT_6,
                PMOD_JC_OE => PMOD_JC_OE_6,

                PMOD_JD_IN => PMOD_JD_IN_6,
                PMOD_JD_OUT => PMOD_JD_OUT_6,
                PMOD_JD_OE => PMOD_JD_OE_6,

                PMOD_JE_IN => PMOD_JE_IN_6,
                PMOD_JE_OUT => PMOD_JE_OUT_6,
                PMOD_JE_OE => PMOD_JE_OE_6,

                leds => LEDS_6,

                RGB_LED => RGB_LED1_6,
                RGB_LED2 => RGB_LED2_6
        );
        par7pm : par7
        port map(
                CLK_125MHZ => sys_clock ,
                CLK_MMC => CLK_MMC,
                CLK_PLL => CLK_PLL,

                uart_zynq_rxd => uart_zynq_rxd ,
                uart_zynq_txd => uart_zynq_txd ,
                reset => reset_par(7),
                PAR_TEST_GPIO0_IN => PAR_TEST_GPIO7_OUT,
                PAR_TEST_GPIO0_OUT => PAR_TEST_GPIO7_IN,

                PMOD_JB_IN => PMOD_JB_IN_7,
                PMOD_JB_OUT => PMOD_JB_OUT_7,
                PMOD_JB_OE => PMOD_JB_OE_7,

                PMOD_JC_IN => PMOD_JC_IN_7,
                PMOD_JC_OUT => PMOD_JC_OUT_7,
                PMOD_JC_OE => PMOD_JC_OE_7,

                PMOD_JD_IN => PMOD_JD_IN_7,
                PMOD_JD_OUT => PMOD_JD_OUT_7,
                PMOD_JD_OE => PMOD_JD_OE_7,

                PMOD_JE_IN => PMOD_JE_IN_7,
                PMOD_JE_OUT => PMOD_JE_OUT_7,
                PMOD_JE_OE => PMOD_JE_OE_7,

                leds => LEDS_7,

                RGB_LED => RGB_LED1_7,
                RGB_LED2 => RGB_LED2_7
        );


        MUX0 : MUX
        port map(
                SYSCLK_125MHZ => sys_clock ,
                MUX_GPIO => MUX_GPIO,
                RGB_LED1 => RGB_LED1,
                RGB_LED2 => RGB_LED2,
                LEDS => LEDS,

                PMOD_JB_IN => PMOD_JB_IN,
                PMOD_JB_OUT => PMOD_JB_OUT,
                PMOD_JB_OE => PMOD_JB_OE,

                PMOD_JC_IN => PMOD_JC_IN,
                PMOD_JC_OUT => PMOD_JC_OUT,
                PMOD_JC_OE => PMOD_JC_OE,

                PMOD_JD_IN => PMOD_JD_IN,
                PMOD_JD_OUT => PMOD_JD_OUT,
                PMOD_JD_OE => PMOD_JD_OE,

                PMOD_JE_IN => PMOD_JE_IN,
                PMOD_JE_OUT => PMOD_JE_OUT,
                PMOD_JE_OE => PMOD_JE_OE,
                --par0
                PMOD_JB_IN_0 => PMOD_JB_IN_0,
                PMOD_JB_OUT_0 => PMOD_JB_OUT_0,
                PMOD_JB_OE_0 => PMOD_JB_OE_0,

                PMOD_JC_IN_0 => PMOD_JC_IN_0,
                PMOD_JC_OUT_0 => PMOD_JC_OUT_0,
                PMOD_JC_OE_0 => PMOD_JC_OE_0,

                PMOD_JD_IN_0 => PMOD_JD_IN_0,
                PMOD_JD_OUT_0 => PMOD_JD_OUT_0,
                PMOD_JD_OE_0 => PMOD_JD_OE_0,

                PMOD_JE_IN_0 => PMOD_JE_IN_0,
                PMOD_JE_OUT_0 => PMOD_JE_OUT_0,
                PMOD_JE_OE_0 => PMOD_JE_OE_0,

                RGB_LED1_0 => RGB_LED1_0,
                RGB_LED2_0 => RGB_LED2_0,
                LEDS_0 => LEDS_0,

                --par1
                PMOD_JB_IN_1 => PMOD_JB_IN_1,
                PMOD_JB_OUT_1 => PMOD_JB_OUT_1,
                PMOD_JB_OE_1 => PMOD_JB_OE_1,

                PMOD_JC_IN_1 => PMOD_JC_IN_1,
                PMOD_JC_OUT_1 => PMOD_JC_OUT_1,
                PMOD_JC_OE_1 => PMOD_JC_OE_1,

                PMOD_JD_IN_1 => PMOD_JD_IN_1,
                PMOD_JD_OUT_1 => PMOD_JD_OUT_1,
                PMOD_JD_OE_1 => PMOD_JD_OE_1,

                PMOD_JE_IN_1 => PMOD_JE_IN_1,
                PMOD_JE_OUT_1 => PMOD_JE_OUT_1,
                PMOD_JE_OE_1 => PMOD_JE_OE_1,

                RGB_LED1_1 => RGB_LED1_1,
                RGB_LED2_1 => RGB_LED2_1,
                LEDS_1 => LEDS_1,
                --par2
                PMOD_JB_IN_2 => PMOD_JB_IN_2,
                PMOD_JB_OUT_2 => PMOD_JB_OUT_2,
                PMOD_JB_OE_2 => PMOD_JB_OE_2,

                PMOD_JC_IN_2 => PMOD_JC_IN_2,
                PMOD_JC_OUT_2 => PMOD_JC_OUT_2,
                PMOD_JC_OE_2 => PMOD_JC_OE_2,
                PMOD_JD_IN_2 => PMOD_JD_IN_2,
                PMOD_JD_OUT_2 => PMOD_JD_OUT_2,
                PMOD_JD_OE_2 => PMOD_JD_OE_2,

                PMOD_JE_IN_2 => PMOD_JE_IN_2,
                PMOD_JE_OUT_2 => PMOD_JE_OUT_2,
                PMOD_JE_OE_2 => PMOD_JE_OE_2,

                RGB_LED1_2 => RGB_LED1_2,
                RGB_LED2_2 => RGB_LED2_2,
                LEDS_2 => LEDS_2,

                --par3
                PMOD_JB_IN_3 => PMOD_JB_IN_3,
                PMOD_JB_OUT_3 => PMOD_JB_OUT_3,
                PMOD_JB_OE_3 => PMOD_JB_OE_3,

                PMOD_JC_IN_3 => PMOD_JC_IN_3,
                PMOD_JC_OUT_3 => PMOD_JC_OUT_3,
                PMOD_JC_OE_3 => PMOD_JC_OE_3,

                PMOD_JD_IN_3 => PMOD_JD_IN_3,
                PMOD_JD_OUT_3 => PMOD_JD_OUT_3,
                PMOD_JD_OE_3 => PMOD_JD_OE_3,

                PMOD_JE_IN_3 => PMOD_JE_IN_3,
                PMOD_JE_OUT_3 => PMOD_JE_OUT_3,
                PMOD_JE_OE_3 => PMOD_JE_OE_3,

                RGB_LED1_3 => RGB_LED1_3,
                RGB_LED2_3 => RGB_LED2_3,
                LEDS_3 => LEDS_3,

                --par4
                PMOD_JB_IN_4 => PMOD_JB_IN_4,
                PMOD_JB_OUT_4 => PMOD_JB_OUT_4,
                PMOD_JB_OE_4 => PMOD_JB_OE_4,

                PMOD_JC_IN_4 => PMOD_JC_IN_4,
                PMOD_JC_OUT_4 => PMOD_JC_OUT_4,
                PMOD_JC_OE_4 => PMOD_JC_OE_4,

                PMOD_JD_IN_4 => PMOD_JD_IN_4,
                PMOD_JD_OUT_4 => PMOD_JD_OUT_4,
                PMOD_JD_OE_4 => PMOD_JD_OE_4,

                PMOD_JE_IN_4 => PMOD_JE_IN_4,
                PMOD_JE_OUT_4 => PMOD_JE_OUT_4,
                PMOD_JE_OE_4 => PMOD_JE_OE_4,
                RGB_LED1_4 => RGB_LED1_4,
                RGB_LED2_4 => RGB_LED2_4,
                LEDS_4 => LEDS_4,

                --par5
                PMOD_JB_IN_5 => PMOD_JB_IN_5,
                PMOD_JB_OUT_5 => PMOD_JB_OUT_5,
                PMOD_JB_OE_5 => PMOD_JB_OE_5,

                PMOD_JC_IN_5 => PMOD_JC_IN_5,
                PMOD_JC_OUT_5 => PMOD_JC_OUT_5,
                PMOD_JC_OE_5 => PMOD_JC_OE_5,

                PMOD_JD_IN_5 => PMOD_JD_IN_5,
                PMOD_JD_OUT_5 => PMOD_JD_OUT_5,
                PMOD_JD_OE_5 => PMOD_JD_OE_5,

                PMOD_JE_IN_5 => PMOD_JE_IN_5,
                PMOD_JE_OUT_5 => PMOD_JE_OUT_5,
                PMOD_JE_OE_5 => PMOD_JE_OE_5,

                RGB_LED1_5 => RGB_LED1_5,
                RGB_LED2_5 => RGB_LED2_5,
                LEDS_5 => LEDS_5,

                --par6
                PMOD_JB_IN_6 => PMOD_JB_IN_6,
                PMOD_JB_OUT_6 => PMOD_JB_OUT_6,
                PMOD_JB_OE_6 => PMOD_JB_OE_6,

                PMOD_JC_IN_6 => PMOD_JC_IN_6,
                PMOD_JC_OUT_6 => PMOD_JC_OUT_6,
                PMOD_JC_OE_6 => PMOD_JC_OE_6,

                PMOD_JD_IN_6 => PMOD_JD_IN_6,
                PMOD_JD_OUT_6 => PMOD_JD_OUT_6,
                PMOD_JD_OE_6 => PMOD_JD_OE_6,

                PMOD_JE_IN_6 => PMOD_JE_IN_6,
                PMOD_JE_OUT_6 => PMOD_JE_OUT_6,
                PMOD_JE_OE_6 => PMOD_JE_OE_6,

                RGB_LED1_6 => RGB_LED1_6,
                RGB_LED2_6 => RGB_LED2_6,
                LEDS_6 => LEDS_6,

                --par7
                PMOD_JB_IN_7 => PMOD_JB_IN_7,
                PMOD_JB_OUT_7 => PMOD_JB_OUT_7,
                PMOD_JB_OE_7 => PMOD_JB_OE_7,

                PMOD_JC_IN_7 => PMOD_JC_IN_7,
                PMOD_JC_OUT_7 => PMOD_JC_OUT_7,
                PMOD_JC_OE_7 => PMOD_JC_OE_7,

                PMOD_JD_IN_7 => PMOD_JD_IN_7,
                PMOD_JD_OUT_7 => PMOD_JD_OUT_7,
                PMOD_JD_OE_7 => PMOD_JD_OE_7,

                PMOD_JE_IN_7 => PMOD_JE_IN_7,
                PMOD_JE_OUT_7 => PMOD_JE_OUT_7,
                PMOD_JE_OE_7 => PMOD_JE_OE_7,

                RGB_LED1_7 => RGB_LED1_7,
                RGB_LED2_7 => RGB_LED2_7,
                LEDS_7 => LEDS_7
        );


        IOBUF0 : IOBUF
        port map(
                I => PMOD_JB_OUT(0),
                O => PMOD_JB_IN(0),
                T => PMOD_JB_OE(0),
                IO => PMOD_JB(0)
        );

        IOBUF1 : IOBUF
        port map(
                I => PMOD_JB_OUT(1),
                O => PMOD_JB_IN(1),
                T => PMOD_JB_OE(1),
                IO => PMOD_JB(1)
        );

        IOBUF2 : IOBUF
        port map(
                I => PMOD_JB_OUT(2),
                O => PMOD_JB_IN(2),
                T => PMOD_JB_OE(2),
                IO => PMOD_JB(2)
        );

        IOBUF3 : IOBUF
        port map(
                I => PMOD_JB_OUT(3),
                O => PMOD_JB_IN(3),
                T => PMOD_JB_OE(3),
                IO => PMOD_JB(3)
        );

        IOBUF4 : IOBUF
        port map(
                I => PMOD_JB_OUT(4),
                O => PMOD_JB_IN(4),
                T => PMOD_JB_OE(4),
                IO => PMOD_JB(4)
        );

        IOBUF5 : IOBUF
        port map(
                I => PMOD_JB_OUT(5),
                O => PMOD_JB_IN(5),
                T => PMOD_JB_OE(5),
                IO => PMOD_JB(5)
        );

        IOBUF6 : IOBUF
        port map(
                I => PMOD_JB_OUT(6),
                O => PMOD_JB_IN(6),
                T => PMOD_JB_OE(6),
                IO => PMOD_JB(6)
        );

        IOBUF7 : IOBUF
        port map(
                I => PMOD_JB_OUT(7),
                O => PMOD_JB_IN(7),
                T => PMOD_JB_OE(7),
                IO => PMOD_JB(7)
        );
        IOBUF0C : IOBUF
        port map(
                I => PMOD_JC_OUT(0),
                O => PMOD_JC_IN(0),
                T => PMOD_JC_OE(0),
                IO => PMOD_JC(0)
        );

        IOBUF1C : IOBUF
        port map(
                I => PMOD_JC_OUT(1),
                O => PMOD_JC_IN(1),
                T => PMOD_JC_OE(1),
                IO => PMOD_JC(1)
        );

        IOBUF2C : IOBUF
```

```
    port map(
            I => PMOD_JC_OUT(2),
            O => PMOD_JC_IN(2),
            T => PMOD_JC_OE(2),
            IO => PMOD_JC(2)
    );

    IOBUF3C : IOBUF
    port map(
            I => PMOD_JC_OUT(3),
            O => PMOD_JC_IN(3),
            T => PMOD_JC_OE(3),
            IO => PMOD_JC(3)
    );

    IOBUF4C : IOBUF
    port map(
            I => PMOD_JC_OUT(4),
            O => PMOD_JC_IN(4),
            T => PMOD_JC_OE(4),
            IO => PMOD_JC(4)
    );

    IOBUF5C : IOBUF
    port map(
            I => PMOD_JC_OUT(5),
            O => PMOD_JC_IN(5),
            T => PMOD_JC_OE(5),
            IO => PMOD_JC(5)
    );

    IOBUF6C : IOBUF
    port map(
            I => PMOD_JC_OUT(6),
            O => PMOD_JC_IN(6),
            T => PMOD_JC_OE(6),
            IO => PMOD_JC(6)
    );

    IOBUF7C : IOBUF
    port map(
            I => PMOD_JC_OUT(7),
            O => PMOD_JC_IN(7),
            T => PMOD_JC_OE(7),
            IO => PMOD_JC(7)
    );
    IOBUF0D : IOBUF
    port map(
            I => PMOD_JD_OUT(0),
            O => PMOD_JD_IN(0),
            T => PMOD_JD_OE(0),
            IO => PMOD_JD(0)
    );

    IOBUF1D : IOBUF
    port map(
            I => PMOD_JD_OUT(1),
            O => PMOD_JD_IN(1),
            T => PMOD_JD_OE(1),
            IO => PMOD_JD(1)
    );

    IOBUF2D : IOBUF
    port map(
            I => PMOD_JD_OUT(2),
            O => PMOD_JD_IN(2),
            T => PMOD_JD_OE(2),
            IO => PMOD_JD(2)
    );

    IOBUF3D : IOBUF
    port map(
            I => PMOD_JD_OUT(3),
            O => PMOD_JD_IN(3),
            T => PMOD_JD_OE(3),
            IO => PMOD_JD(3)
    );

    IOBUF4D : IOBUF
    port map(
            I => PMOD_JD_OUT(4),
            O => PMOD_JD_IN(4),
            T => PMOD_JD_OE(4),
            IO => PMOD_JD(4)
    );

    IOBUF5D : IOBUF
    port map(
            I => PMOD_JD_OUT(5),
            O => PMOD_JD_IN(5),
            T => PMOD_JD_OE(5),
            IO => PMOD_JD(5)
    );

    IOBUF6D : IOBUF
    port map(
            I => PMOD_JD_OUT(6),
            O => PMOD_JD_IN(6),
            T => PMOD_JD_OE(6),
            IO => PMOD_JD(6)
    );

    IOBUF7D : IOBUF
    port map(
            I => PMOD_JD_OUT(7),
            O => PMOD_JD_IN(7),
            T => PMOD_JD_OE(7),
            IO => PMOD_JD(7)
    );

    IOBUF0E : IOBUF
    port map(
            I => PMOD_JE_OUT(0),
            O => PMOD_JE_IN(0),
            T => PMOD_JE_OE(0),
            IO => PMOD_JE(0)
    );

    IOBUF1E : IOBUF
    port map(
            I => PMOD_JE_OUT(1),
            O => PMOD_JE_IN(1),
            T => PMOD_JE_OE(1),
            IO => PMOD_JE(1)
    );

    IOBUF2E : IOBUF
    port map(
            I => PMOD_JE_OUT(2),
            O => PMOD_JE_IN(2),
            T => PMOD_JE_OE(2),
            IO => PMOD_JE(2)
    );

    IOBUF3E : IOBUF
    port map(
            I => PMOD_JE_OUT(3),
            O => PMOD_JE_IN(3),
            T => PMOD_JE_OE(3),
            IO => PMOD_JE(3)
    );

    IOBUF4E : IOBUF
    port map(
            I => PMOD_JE_OUT(4),
            O => PMOD_JE_IN(4),
            T => PMOD_JE_OE(4),
            IO => PMOD_JE(4)
    );
    IOBUF5E : IOBUF
    port map(
            I => PMOD_JE_OUT(5),
            O => PMOD_JE_IN(5),
            T => PMOD_JE_OE(5),
            IO => PMOD_JE(5)
    );

    IOBUF6E : IOBUF
    port map(
            I => PMOD_JE_OUT(6),
            O => PMOD_JE_IN(6),
            T => PMOD_JE_OE(6),
            IO => PMOD_JE(6)
    );

    IOBUF7E : IOBUF
    port map(
            I => PMOD_JE_OUT(7),
            O => PMOD_JE_IN(7),
            T => PMOD_JE_OE(7),
            IO => PMOD_JE(7)
    );

end Behavioral;
```

Listing 5: topvhd

# 9 Appendix B: mux.vhd

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer: Felix Georg Braun
--
-- Create Date: 15.02.2019 16:46:24
-- Design Name: top
-- Module Name: top - Behavioral
-- Project Name: vhdl testing platform
-- Target Devices: Zybo Z7-20 (xc7z020clg400-1)
-- Tool Versions: VIVADO 2017.4
-- Revision:
-- Revision 0.01 - File Created
--
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity MUX is
        port (
            --Sysclk
                SYSCLK_125MHZ : in STD_LOGIC;

                --IO Controll
                MUX_GPIO : in STD_LOGIC_VECTOR(28 downto 0);

                --Leds
                RGB_LED1 : out STD_LOGIC_VECTOR(2 downto 0);
                RGB_LED2 : out STD_LOGIC_VECTOR(2 downto 0);
                LEDS : out STD_LOGIC_VECTOR(3 downto 0);

                PMOD_JB_IN : in STD_LOGIC_VECTOR(7 downto 0);
                PMOD_JB_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                PMOD_JB_OE : out STD_LOGIC_VECTOR(7 downto 0);

                PMOD_JC_IN : in STD_LOGIC_VECTOR(7 downto 0);
                PMOD_JC_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                PMOD_JC_OE : out STD_LOGIC_VECTOR(7 downto 0);

                PMOD_JD_IN : in STD_LOGIC_VECTOR(7 downto 0);
                PMOD_JD_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                PMOD_JD_OE : out STD_LOGIC_VECTOR(7 downto 0);

                PMOD_JE_IN : in STD_LOGIC_VECTOR(7 downto 0);
                PMOD_JE_OUT : out STD_LOGIC_VECTOR(7 downto 0);
                PMOD_JE_OE : out STD_LOGIC_VECTOR(7 downto 0);

                --par0
```

```
PMOD_JB_IN_0 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OUT_0 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OE_0 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JC_IN_0 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OUT_0 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OE_0 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JD_IN_0 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OUT_0 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OE_0 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JE_IN_0 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OUT_0 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OE_0 : in STD_LOGIC_VECTOR(7 downto 0);

RGB_LED1_0 : in STD_LOGIC_VECTOR(2 downto 0);
RGB_LED2_0 : in STD_LOGIC_VECTOR(2 downto 0);
LEDS_0 : in STD_LOGIC_VECTOR(3 downto 0);

--par1
PMOD_JB_IN_1 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OUT_1 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OE_1 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JC_IN_1 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OUT_1 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OE_1 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JD_IN_1 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OUT_1 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OE_1 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JE_IN_1 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OUT_1 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OE_1 : in STD_LOGIC_VECTOR(7 downto 0);

RGB_LED1_1 : in STD_LOGIC_VECTOR(2 downto 0);
RGB_LED2_1 : in STD_LOGIC_VECTOR(2 downto 0);
LEDS_1 : in STD_LOGIC_VECTOR(3 downto 0);
--par2
PMOD_JB_IN_2 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OUT_2 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OE_2 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JC_IN_2 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OUT_2 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OE_2 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JD_IN_2 : out STD_LOGIC_VECTOR(7 downto 0);
```

```
PMOD_JD_OUT_2 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OE_2 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JE_IN_2 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OUT_2 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OE_2 : in STD_LOGIC_VECTOR(7 downto 0);

RGB_LED1_2 : in STD_LOGIC_VECTOR(2 downto 0);
RGB_LED2_2 : in STD_LOGIC_VECTOR(2 downto 0);
LEDS_2 : in STD_LOGIC_VECTOR(3 downto 0);

--par3
PMOD_JB_IN_3 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OUT_3 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OE_3 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JC_IN_3 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OUT_3 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OE_3 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JD_IN_3 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OUT_3 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OE_3 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JE_IN_3 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OUT_3 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OE_3 : in STD_LOGIC_VECTOR(7 downto 0);

RGB_LED1_3 : in STD_LOGIC_VECTOR(2 downto 0);
RGB_LED2_3 : in STD_LOGIC_VECTOR(2 downto 0);
LEDS_3 : in STD_LOGIC_VECTOR(3 downto 0);

--par4
PMOD_JB_IN_4 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OUT_4 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OE_4 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JC_IN_4 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OUT_4 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OE_4 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JD_IN_4 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OUT_4 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OE_4 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JE_IN_4 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OUT_4 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OE_4 : in STD_LOGIC_VECTOR(7 downto 0);

RGB_LED1_4 : in STD_LOGIC_VECTOR(2 downto 0);
```

```
RGB_LED2_4 : in STD_LOGIC_VECTOR(2 downto 0);
LEDS_4 : in STD_LOGIC_VECTOR(3 downto 0);

--par5
PMOD_JB_IN_5 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OUT_5 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OE_5 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JC_IN_5 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OUT_5 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OE_5 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JD_IN_5 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OUT_5 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OE_5 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JE_IN_5 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OUT_5 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OE_5 : in STD_LOGIC_VECTOR(7 downto 0);

RGB_LED1_5 : in STD_LOGIC_VECTOR(2 downto 0);
RGB_LED2_5 : in STD_LOGIC_VECTOR(2 downto 0);
LEDS_5 : in STD_LOGIC_VECTOR(3 downto 0);

--par6
PMOD_JB_IN_6 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OUT_6 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OE_6 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JC_IN_6 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OUT_6 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JC_OE_6 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JD_IN_6 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OUT_6 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JD_OE_6 : in STD_LOGIC_VECTOR(7 downto 0);

PMOD_JE_IN_6 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OUT_6 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JE_OE_6 : in STD_LOGIC_VECTOR(7 downto 0);

RGB_LED1_6 : in STD_LOGIC_VECTOR(2 downto 0);
RGB_LED2_6 : in STD_LOGIC_VECTOR(2 downto 0);
LEDS_6 : in STD_LOGIC_VECTOR(3 downto 0);

--par7
PMOD_JB_IN_7 : out STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OUT_7 : in STD_LOGIC_VECTOR(7 downto 0);
PMOD_JB_OE_7 : in STD_LOGIC_VECTOR(7 downto 0);
```

```
                PMOD_JC_IN_7 : out STD_LOGIC_VECTOR(7 downto 0);
                PMOD_JC_OUT_7 : in STD_LOGIC_VECTOR(7 downto 0);
                PMOD_JC_OE_7 : in STD_LOGIC_VECTOR(7 downto 0);

                PMOD_JD_IN_7 : out STD_LOGIC_VECTOR(7 downto 0);
                PMOD_JD_OUT_7 : in STD_LOGIC_VECTOR(7 downto 0);
                PMOD_JD_OE_7 : in STD_LOGIC_VECTOR(7 downto 0);

                PMOD_JE_IN_7 : out STD_LOGIC_VECTOR(7 downto 0);
                PMOD_JE_OUT_7 : in STD_LOGIC_VECTOR(7 downto 0);
                PMOD_JE_OE_7 : in STD_LOGIC_VECTOR(7 downto 0);

                RGB_LED1_7 : in STD_LOGIC_VECTOR(2 downto 0);
                RGB_LED2_7 : in STD_LOGIC_VECTOR(2 downto 0);
                LEDS_7 : in STD_LOGIC_VECTOR(3 downto 0)

        );
end MUX;

architecture Behavioral of MUX is
    alias Enable :std_logic_vector (0 downto 0) is MUX_GPIO(0 downto 0);
        alias MUX_PMODB : std_logic_vector(3 downto 0) is MUX_GPIO(4 downto 1);
        alias MUX_PMODC : std_logic_vector(3 downto 0) is MUX_GPIO(8 downto 5);
        alias MUX_PMODD : std_logic_vector(3 downto 0) is MUX_GPIO(12 downto 9);
        alias MUX_PMODE : std_logic_vector(3 downto 0) is MUX_GPIO(16 downto 13);
        alias MUX_RGBLED1 : std_logic_vector(3 downto 0) is MUX_GPIO(20 downto 17);
        alias MUX_RGBLED2 : std_logic_vector(3 downto 0) is MUX_GPIO(24 downto 21);
        alias MUX_LEDS : std_logic_vector(3 downto 0) is MUX_GPIO(28 downto 25);

begin
        process (SYSCLK_125MHZ)
        begin
                if (rising_edge(SYSCLK_125MHZ)) then
                  if(Enable="1") then
                        case MUX_PMODB is
                                when "0000" =>
                                        PMOD_JB_IN_0 <= PMOD_JB_IN;
                                        PMOD_JB_OUT <= PMOD_JB_OUT_0;
                                        PMOD_JB_OE <= PMOD_JB_OE_0;
                                when "0001" =>
                                        PMOD_JB_IN_1 <= PMOD_JB_IN;
                                        PMOD_JB_OUT <= PMOD_JB_OUT_1;
                                        PMOD_JB_OE <= PMOD_JB_OE_1;
                                when "0010" =>
                                        PMOD_JB_IN_2 <= PMOD_JB_IN;
                                        PMOD_JB_OUT <= PMOD_JB_OUT_2;
                                        PMOD_JB_OE <= PMOD_JB_OE_2;
                                when "0011" =>
                                        PMOD_JB_IN_3 <= PMOD_JB_IN;
                                        PMOD_JB_OUT <= PMOD_JB_OUT_3;
                                        PMOD_JB_OE <= PMOD_JB_OE_3;
                                when "0100" =>
                                        PMOD_JB_IN_4 <= PMOD_JB_IN;
                                        PMOD_JB_OUT <= PMOD_JB_OUT_4;
                                        PMOD_JB_OE <= PMOD_JB_OE_4;
                                when "0101" =>
                                        PMOD_JB_IN_5 <= PMOD_JB_IN;
                                        PMOD_JB_OUT <= PMOD_JB_OUT_5;
                                        PMOD_JB_OE <= PMOD_JB_OE_5;
                                when "0110" =>
                                        PMOD_JB_IN_6 <= PMOD_JB_IN;
                                        PMOD_JB_OUT <= PMOD_JB_OUT_6;
                                        PMOD_JB_OE <= PMOD_JB_OE_6;
                                when "0111" =>
                                        PMOD_JB_IN_7 <= PMOD_JB_IN;
                                        PMOD_JB_OUT <= PMOD_JB_OUT_7;
                                        PMOD_JB_OE <= PMOD_JB_OE_7;
                                when others =>
                                        PMOD_JB_IN_0 <= (others => '0');
                                        PMOD_JB_OUT <= (others => '0');
                                        PMOD_JB_OE <= (others => '0');
                        end case;

                        case MUX_PMODC is
                                when "0000" =>
                                        PMOD_JC_IN_0 <= PMOD_JC_IN;
                                        PMOD_JC_OUT <= PMOD_JC_OUT_0;
                                        PMOD_JC_OE <= PMOD_JC_OE_0;
                                when "0001" =>
                                        PMOD_JC_IN_1 <= PMOD_JC_IN;
                                        PMOD_JC_OUT <= PMOD_JC_OUT_1;
                                        PMOD_JC_OE <= PMOD_JC_OE_1;
                                when "0010" =>
                                        PMOD_JC_IN_2 <= PMOD_JC_IN;
                                        PMOD_JC_OUT <= PMOD_JC_OUT_2;
                                        PMOD_JC_OE <= PMOD_JC_OE_2;
                                when "0011" =>
                                        PMOD_JC_IN_3 <= PMOD_JC_IN;
                                        PMOD_JC_OUT <= PMOD_JC_OUT_3;
                                        PMOD_JC_OE <= PMOD_JC_OE_3;
                                when "0100" =>
                                        PMOD_JC_IN_4 <= PMOD_JC_IN;
                                        PMOD_JC_OUT <= PMOD_JC_OUT_4;
                                        PMOD_JC_OE <= PMOD_JC_OE_4;
                                when "0101" =>
                                        PMOD_JC_IN_5 <= PMOD_JC_IN;
                                        PMOD_JC_OUT <= PMOD_JC_OUT_5;
                                        PMOD_JC_OE <= PMOD_JC_OE_5;
                                when "0110" =>
                                        PMOD_JC_IN_6 <= PMOD_JC_IN;
```

```
                    PMOD_JC_OUT <= PMOD_JC_OUT_6;
                    PMOD_JC_OE <= PMOD_JC_OE_6;
            when "0111" =>
                    PMOD_JC_IN_7 <= PMOD_JC_IN;
                    PMOD_JC_OUT <= PMOD_JC_OUT_7;
                    PMOD_JC_OE <= PMOD_JC_OE_7;
            when others =>
                    PMOD_JC_IN_0 <= (others => '0');
                    PMOD_JC_OUT <= (others => '0');
                    PMOD_JC_OE <= (others => '0');
    end case;

    case MUX_PMODD is
            when "0000" =>
                    PMOD_JD_IN_0 <= PMOD_JD_IN;
                    PMOD_JD_OUT <= PMOD_JD_OUT_0;
                    PMOD_JD_OE <= PMOD_JD_OE_0;
            when "0001" =>
                    PMOD_JD_IN_1 <= PMOD_JD_IN;
                    PMOD_JD_OUT <= PMOD_JD_OUT_1;
                    PMOD_JD_OE <= PMOD_JD_OE_1;
            when "0010" =>
                    PMOD_JD_IN_2 <= PMOD_JD_IN;
                    PMOD_JD_OUT <= PMOD_JD_OUT_2;
                    PMOD_JD_OE <= PMOD_JD_OE_2;
            when "0011" =>
                    PMOD_JD_IN_3 <= PMOD_JD_IN;
                    PMOD_JD_OUT <= PMOD_JD_OUT_3;
                    PMOD_JD_OE <= PMOD_JD_OE_3;
            when "0100" =>
                    PMOD_JD_IN_4 <= PMOD_JD_IN;
                    PMOD_JD_OUT <= PMOD_JD_OUT_4;
                    PMOD_JD_OE <= PMOD_JD_OE_4;
            when "0101" =>
                    PMOD_JD_IN_5 <= PMOD_JD_IN;
                    PMOD_JD_OUT <= PMOD_JD_OUT_5;
                    PMOD_JD_OE <= PMOD_JD_OE_5;
            when "0110" =>
                    PMOD_JD_IN_6 <= PMOD_JD_IN;
                    PMOD_JD_OUT <= PMOD_JD_OUT_6;
                    PMOD_JD_OE <= PMOD_JD_OE_6;
            when "0111" =>
                    PMOD_JD_IN_7 <= PMOD_JD_IN;
                    PMOD_JD_OUT <= PMOD_JD_OUT_7;
                    PMOD_JD_OE <= PMOD_JD_OE_7;
            when others =>
                    PMOD_JD_IN_0 <= (others => '0');
                    PMOD_JD_OUT <= (others => '0');
                    PMOD_JD_OE <= (others => '0');
    end case;
```

```
case MUX_PMODE is
        when "0000" =>
                PMOD_JE_IN_0 <= PMOD_JE_IN;
                PMOD_JE_OUT <= PMOD_JE_OUT_0;
                PMOD_JE_OE <= PMOD_JE_OE_0;
        when "0001" =>
                PMOD_JE_IN_1 <= PMOD_JE_IN;
                PMOD_JE_OUT <= PMOD_JE_OUT_1;
                PMOD_JE_OE <= PMOD_JE_OE_1;
        when "0010" =>
                PMOD_JE_IN_2 <= PMOD_JE_IN;
                PMOD_JE_OUT <= PMOD_JE_OUT_2;
                PMOD_JE_OE <= PMOD_JE_OE_2;
        when "0011" =>
                PMOD_JE_IN_3 <= PMOD_JE_IN;
                PMOD_JE_OUT <= PMOD_JE_OUT_3;
                PMOD_JE_OE <= PMOD_JE_OE_3;
        when "0100" =>
                PMOD_JE_IN_4 <= PMOD_JE_IN;
                PMOD_JE_OUT <= PMOD_JE_OUT_4;
                PMOD_JE_OE <= PMOD_JE_OE_4;
        when "0101" =>
                PMOD_JE_IN_5 <= PMOD_JE_IN;
                PMOD_JE_OUT <= PMOD_JE_OUT_5;
                PMOD_JE_OE <= PMOD_JE_OE_5;
        when "0110" =>
                PMOD_JE_IN_6 <= PMOD_JE_IN;
                PMOD_JE_OUT <= PMOD_JE_OUT_6;
                PMOD_JE_OE <= PMOD_JE_OE_6;
        when "0111" =>
                PMOD_JE_IN_7 <= PMOD_JE_IN;
                PMOD_JE_OUT <= PMOD_JE_OUT_7;
                PMOD_JE_OE <= PMOD_JE_OE_7;
        when others =>
                PMOD_JE_IN_0 <= (others => '0');
                PMOD_JE_OUT <= (others => '0');
                PMOD_JE_OE <= (others => '0');
end case;

case MUX_RGBLED1 is
        when "0000" =>
                RGB_LED1 <= RGB_LED1_0;
        when "0001" =>
                RGB_LED1 <= RGB_LED1_1;
        when "0010" =>
                RGB_LED1 <= RGB_LED1_2;
        when "0011" =>
                RGB_LED1 <= RGB_LED1_3;
        when "0100" =>
```

```
                        RGB_LED1 <= RGB_LED1_4;
                when "0101" =>
                        RGB_LED1 <= RGB_LED1_5;
                when "0110" =>
                        RGB_LED1 <= RGB_LED1_6;
                when "0111" =>
                        RGB_LED1 <= RGB_LED1_7;
                when others =>
                        RGB_LED1 <= (others => '0');
        end case;

        case MUX_RGBLED2 is
                when "0000" =>
                        RGB_LED2 <= RGB_LED2_0;
                when "0001" =>
                        RGB_LED2 <= RGB_LED2_1;
                when "0010" =>
                        RGB_LED2 <= RGB_LED2_2;
                when "0011" =>
                        RGB_LED2 <= RGB_LED2_3;
                when "0100" =>
                        RGB_LED2 <= RGB_LED2_4;
                when "0101" =>
                        RGB_LED2 <= RGB_LED2_5;
                when "0110" =>
                        RGB_LED2 <= RGB_LED2_6;
                when "0111" =>
                        RGB_LED2 <= RGB_LED2_7;
                when others =>
                        RGB_LED2 <= (others => '0');
        end case;

        case MUX_LEDS is
                when "0000" =>
                        LEDS <= LEDS_0;
                when "0001" =>
                        LEDS <= LEDS_1;
                when "0010" =>
                        LEDS <= LEDS_2;
                when "0011" =>
                        LEDS <= LEDS_3;
                when "0100" =>
                        LEDS <= LEDS_4;
                when "0101" =>
                        LEDS <= LEDS_5;
                when "0110" =>
                        LEDS <= LEDS_6;
                when "0111" =>
                        LEDS <= LEDS_7;
                when others =>
```

```
                                        LEDS <= (others => '0');
                        end case;

            end if;
                end if;
        end process;
end Behavioral;
```

Listing 6: muxvhd