# EE3220: System-On-Chip (SoC) Design

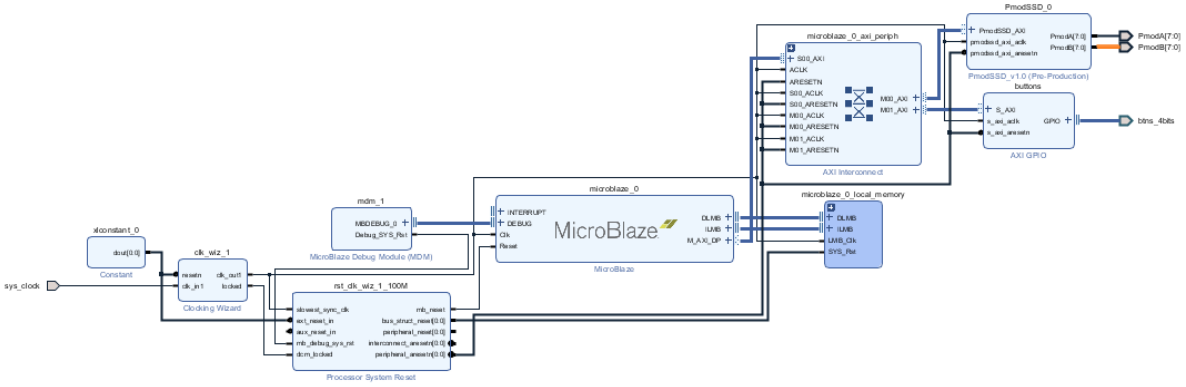
## Laboratory 5: Using Seven Segment Display and Pmod with MicroBlaze

## Objectives:

*At the end of this lab, students should be able to:*

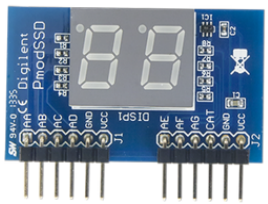
- Add third party IP to Vivado project
- Instantiate the third party PmodSSD IP in Vivado
- Integrate the IP with MicroBlaze soft processor
- Use seven-segment multiplexing in C MicroBlaze program to display numbers on the seven-segment display

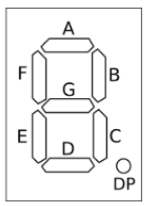
## System overview:

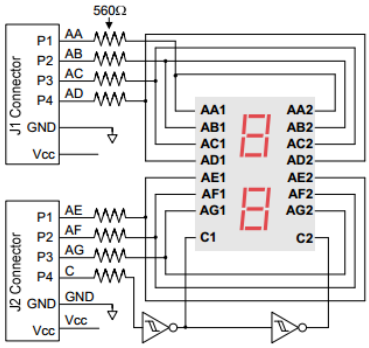



In this exercise, we are going to use the concept of seven segment multiplexing to display numbers on seven segment display (SSD). We are going to use Pmod connectors available on the FPGA board and a Pmod SSD. We provide you with a PmodSSD IP for using the Pmod connectors (PmodA and Pmod B) with seven segment. We are going to instantiate and use the Pmod IP together with MicroBlaze processor. The MicroBlaze C program will be written in Vitis IDE. Two digits SSD having Pmod interface and common cathode connection will be used. There will be three check points, for each check point, take snapshots as evendences to be included in your lab report.

A seven-segment display uses LEDs connected in common anode or common cathode to display numbers and signs. Multiplexing is a technique used to display more than one digits (seven segments) from one interface by using perception of vision. The seven segments are ON and OFF one at a time to display a multidigit number. Each display is repeated after at least 20 milli-second delay. Users may think the displays are ON all at the same time.






[Guide: (https://reference.digilentinc.com/reference/pmod/pmodssd/reference-manual )]

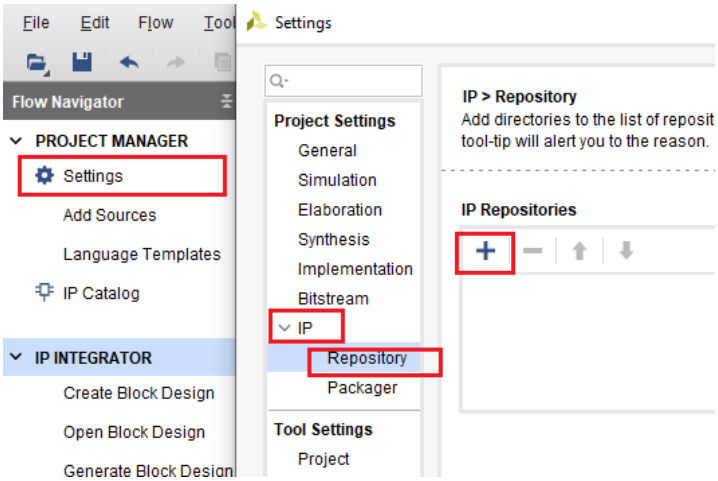
## Part 1: MicroBlaze and Pmod IP Instantiation in Vivado

In this section, students will use Vivado hlx to instantiate a MicroBlaze and Pmod IP for the seven-segment display. We need to add the third party PmodSSD IP to Vivado's library.

Follow the following steps to achieve part 1:

1. Add third party **PmodSSD** IP to Vivado

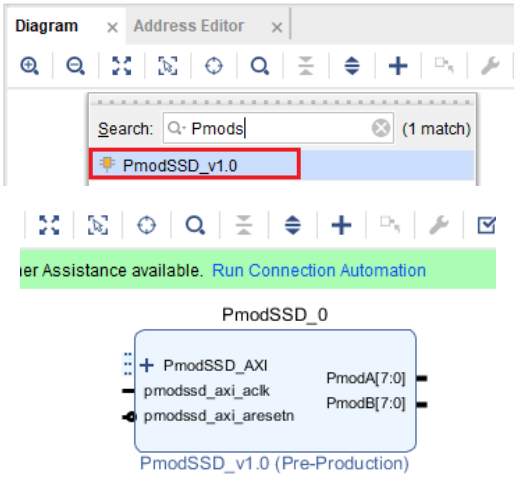
- Create new Project in Vivado. Name it **lab5**.
- Copy the **PmodSSD** IP from Canvas to a folder in the project directory or another place.
- Under **Project Manager** in Vivado, go to **Settings -> IP -> Repository**
- Click '**+**' to add repository. Browse and select the PmodSSD IP folder in the project directory, the PmodSSD IP repository will then be added to the IP repository of the project.
- Click apply then OK.



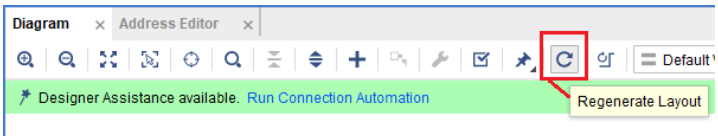

2. Create block design to instantiate MicroBlaze and PmodSSD IPs

- Click **Create Block Design** tab under **IP Integrator** in the flow navigator
- Name the new block design as **PmodSSD_lab5**, Click OK
- Click add IP button (**+**), add **MicroBlaze IP** and run the subsequent block and connection automations. Remember to set the Clocking Wizard to use **sys_clk** and have **active low** reset output as in the previous lab.
- Add a **constant IP** of value **1** and connect the resets to the constant
- Also delete the **Reset_tl** of the clock Wizard as well as the reset of the Processor System Reset and tie the two resets to the **constant IP** as done in the previous labs.

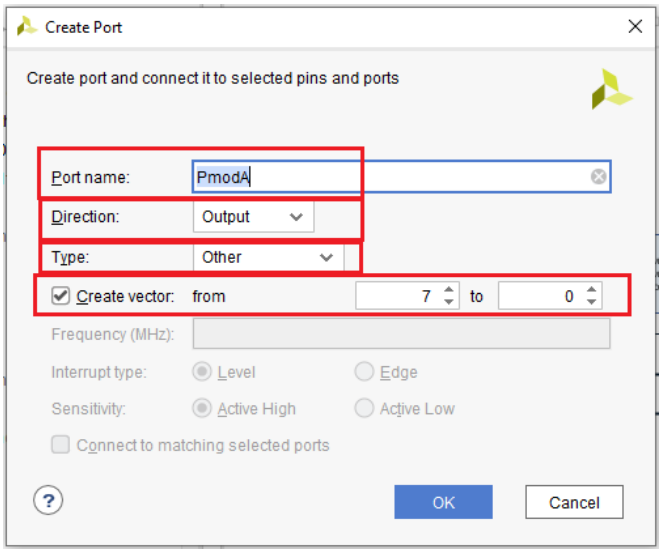
- Add **AXI_GPIO** IP, rename it as **buttons**, double click it and set its board interface to **btns 4bits**. Although the buttons are added in the hardware, however it is not used in the software for this lab.
- Click add IP button (**+**) again, search and add **PmodSSD**



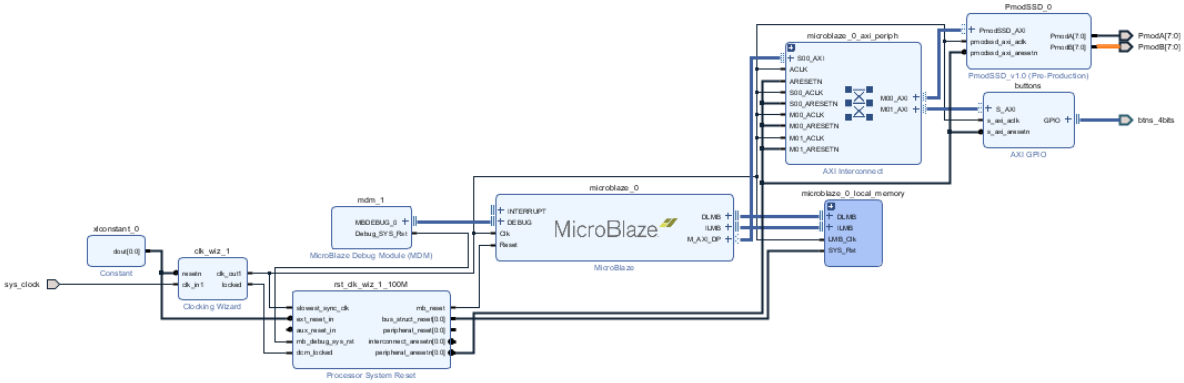

- Run connection automation, select all elements and click **OK**, click **Regenerate Layout**.




- Right click the block design space and select **Create Port**
- Specify the port name as **PmodA**, set direction output, tick create vector and set range 0 to 7
- Repeat port creation for **PmodB** too

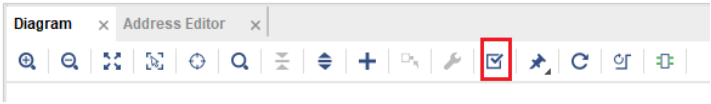



- Connect the **PmodA** and **PmodB** ports to the **PmodSSD**
- Click **Regenerate Layout**




3. Validate the design and create bitstream

- Click on validate design (a tick icon in the block design menu). After the validation, a dialog box shows validation successful. Click OK




- On the sources panel, right the block design, choose **Create HDL Wrapper**, click OK.
- Choose add sources (**+**) sign under source to create a constraint file, click **Create File** tab, specify constraint file name and click OK
- Double click and open the constraint file you just created , i.e. pynq_z2_constr.xdc
- Enter the constraints we provide you below or in the Canvas by copying or typing. Note: When copying the code of file(.xdc), students need to manually revise it. Sometimes a comment or command in one line can jump to the next line which can lead to a syntax error.
- In the **Flow Navigator**, click **Generate Bitstream**

```
## This file is a constraint (.xdc) for the PYNQ-Z2 board
## Clock signal 125 MHz
set_property -dict {PACKAGE_PIN H16  IOSTANDARD LVCMOS33}[get_ports sys_clock];
##PmodA
set_property -dict {PACKAGE_PIN Y18  IOSTANDARD LVCMOS33} [ get_ports
{PmodA[0]}];
set_property -dict {PACKAGE_PIN Y19  IOSTANDARD LVCMOS33} [ get_ports
{PmodA[1]}];
set_property -dict {PACKAGE_PIN Y16  IOSTANDARD LVCMOS33} [ get_ports
{PmodA[2]}];
set_property -dict {PACKAGE_PIN Y17  IOSTANDARD LVCMOS33} [ get_ports
{PmodA[3]}];
set_property -dict {PACKAGE_PIN U18  IOSTANDARD LVCMOS33} [ get_ports
{PmodA[4]}];
set_property -dict {PACKAGE_PIN U19  IOSTANDARD LVCMOS33} [ get_ports
{PmodA[5]}];
set_property -dict {PACKAGE_PIN W18  IOSTANDARD LVCMOS33} [ get_ports
{PmodA[6]}];
set_property -dict {PACKAGE_PIN W19  IOSTANDARD LVCMOS33} [ get_ports
{PmodA[7]}];
##PmodB
set_property -dict {PACKAGE_PIN W14  IOSTANDARD LVCMOS33} [ get_ports
{PmodB[0]}];
set_property -dict {PACKAGE_PIN Y14  IOSTANDARD LVCMOS33} [ get_ports
{PmodB[1]}];
```

```
set_property -dict {PACKAGE_PIN T11  IOSTANDARD LVCMOS33} [ get_ports
{PmodB[2]}];
set_property -dict {PACKAGE_PIN T10  IOSTANDARD LVCMOS33} [ get_ports
{PmodB[3]}];
set_property -dict {PACKAGE_PIN V16  IOSTANDARD LVCMOS33} [ get_ports
{PmodB[4]}];
set_property -dict {PACKAGE_PIN W16  IOSTANDARD LVCMOS33} [ get_ports
{PmodB[5]}];
set_property -dict {PACKAGE_PIN V12  IOSTANDARD LVCMOS33} [ get_ports
{PmodB[6]}];
set_property -dict {PACKAGE_PIN W13  IOSTANDARD LVCMOS33} [ get_ports
{PmodB[7]}];
##Buttons
set_property -dict {PACKAGE_PIN D19  IOSTANDARD LVCMOS33} [ get_ports
{btns_4bits_tri_i[0]}];
set_property -dict {PACKAGE_PIN D20  IOSTANDARD LVCMOS33} [ get_ports
{btns_4bits_tri_i[1]}];
set_property -dict {PACKAGE_PIN L20  IOSTANDARD LVCMOS33} [ get_ports
{btns_4bits_tri_i[2]}];
set_property -dict {PACKAGE_PIN L19  IOSTANDARD LVCMOS33} [ get_ports
{btns_4bits_tri_i[3]}];
```

4.Export hardware.

After the bitstream file is successfully generated, then export the hardware.

- Go to File, under *Export*, click **Export Hardware**
- Select **Fixed** option and click next
- Choose **include bitstream** and click next
- Choose the lab5 project directory where the XSA file will be stored.
- Click Next and then Finish

### Check Point 1:

Take snapshots of your block design and the evidence of successful bitstream generation for your report.

## Part 2: Software Development using Vitis IDE

1. Create new Vitis application project

- Open Vitis,  choose a folder as your workspace location, then click launch.

- Click **Create application project** and click Next
- Click **create a new platform from hardware(xsa)**, browse and go to the lab5 project folder and select the **PmodSSD_lab5_wrapper.xsa**, click Next and leave other setting as default, then click Next.
- Specify the project name **(lab5_vitis)**, click Next.
- Click Next
- Choose Empty Application and click Finish. Our new application project will be created

2. Create new source code file **(PmodSSD.c)**

- Right click **Src**, select **New -> File** to create a new source code file.
- Name the source file as **PmodSSD.c**, click finish. A new source file will be created

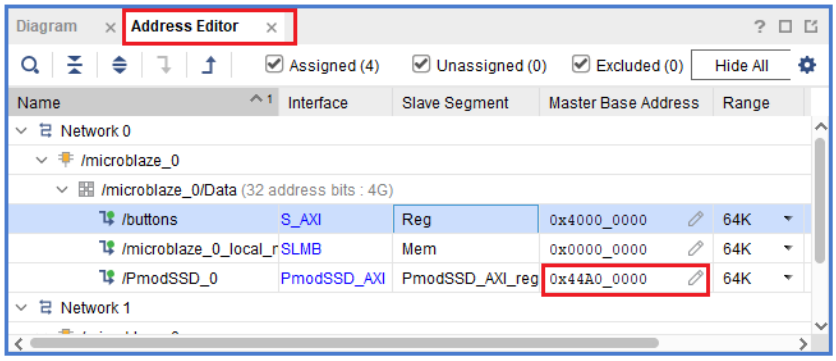
3. Write a program to display different numbers on the seven-segment display, build project and program the FPGA

- Open PmodSSD.c file, copy and paste the source code provided. Change the number to the last two digits of your student ID.

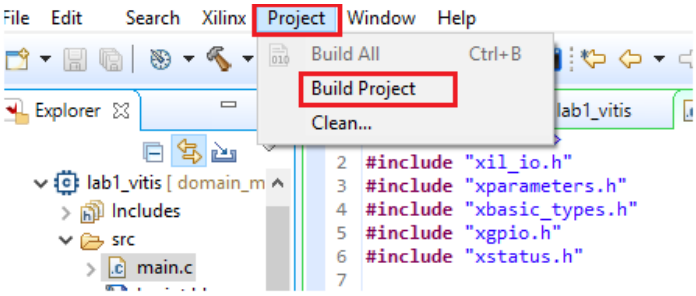
```
#include "sleep.h"
#include <stdio.h>
#include "xil_io.h"
#include "xparameters.h"
#include "xbasic_types.h"
#include "xgpio.h"
#include "xstatus.h"

#define base_addr 0x44A00000 //BASEADDR of PmodSSD IP register value obtained
from                                    //xparameters.h or the Address Editor in
Vivado
/*----- Variable declarations -----------*/

u8  first_digit;
u8  sec_digit;
int number;

/*----- function prototypes -------------*/

void  display_digit(u8 value);
void  display(void);
u8    decode(u8 digit);
void  on_digit(u8 value);
void  extract_digit(int number);

/*----- The main function -------------*/

int main() {

   number      = 27;              //the number to be displayed
   extract_digit(number);
   first_digit = decode(first_digit);
   sec_digit   = decode(sec_digit);

   while(1){
        display();
   };
   return 0;
}

/*----- Other functions  -------------*/

void display(void){
  on_digit(1);                  //ON the first digit SSD
  display_digit(first_digit);   //display first_digit
  usleep(10000);                //10 milliseconds delay
  on_digit(2);                  //ON the second digit SSD
```

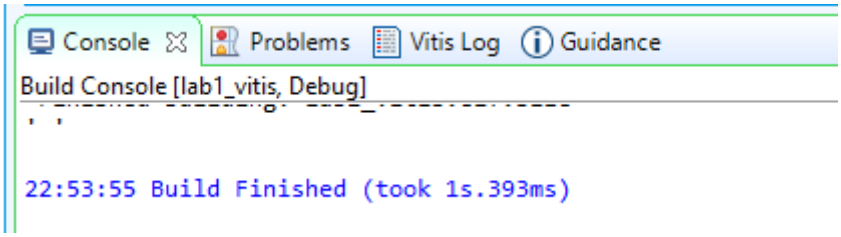
```
  display_digit(sec_digit);     //display second_digit
  usleep(10000);                //10 milliseconds delay
}

void display_digit(u8 value){
  Xil_Out32(base_addr, value);
};

u8 decode(u8 digit){
  u8 res;

  switch(digit){
     case 0: res = 0x3F; break;   // SSD value for 0
     case 1: res = 0x06; break;   // SSD value for 1
     case 2: res = 0x5B; break;   // SSD value for 2
     case 3: res = 0x4F; break;   // SSD value for 3
     case 4: res = 0x66; break;   // SSD value for 4
     case 5: res = 0x6D; break;   // SSD value for 5
     case 6: res = 0x7D; break;   // SSD value for 6
     case 7: res = 0x07; break;   // SSD value for 7
     case 8: res = 0x7F; break;   // SSD value for 8
     case 9: res = 0x6F; break;   // SSD value for 9
  }
  return res;
}

void on_digit(u8 value){
  if (value == 1) {
      first_digit= (first_digit & 0x7F);
  } else{
      sec_digit = (sec_digit | 0x80);
  }
}

void extract_digit(int number){
    first_digit =0;
    sec_digit   =0;

    if(number>0)
    {
       sec_digit   = number/10;
       first_digit = number%10;
    }
}
```

- Ensure the value of the base address **(base_addr)** used in the code is the same as the base address of your hardware shown in your Address Editor window.

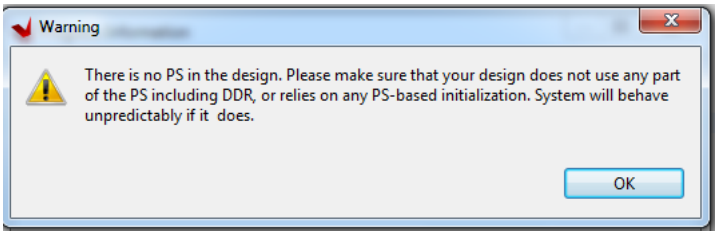


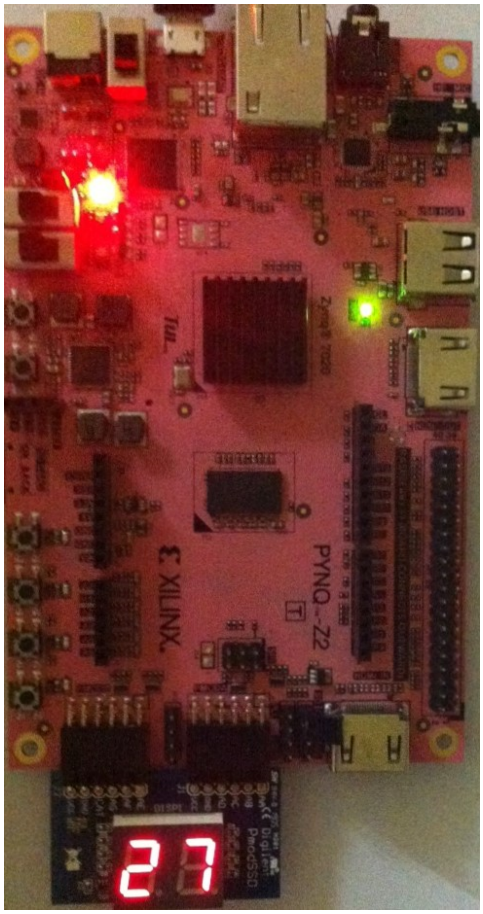

- Open the project menu, choose build project




- Open the Xilinx menu, choose **Program FPGA**. You should connect your board to your computer and turn it on before you can program the board.
- Browse the bitstream, mmi and the .elf for the project as done in the previous lab.
- **Ensure the SSD is connected to the upper slots of the PmodA and Pmod B as in the figure below**
- Click **Program**.
- Wait and click OK.






## Check Point 2:

Modify the program to display the last two digits of your student ID. Take snapshots of your Vitis environment and the numbers displayed on the board for your report as the evidence for this checkpoint.

## Check Point 3:

Modify the program to display a count of numbers from 0 to 5. Take snapshots as evidence of achieving this checkpoint. [Note: You can use each call of display() function as 20ms. Show your work to your lab supervisor. ]

~ END ~