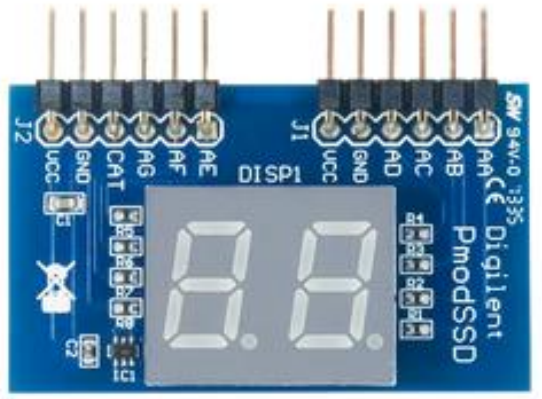
香港中文大學
The Chinese University of Hong Kong

CENG3430 Rapid Prototyping of Digital Systems

Lab06: Driving Pmods

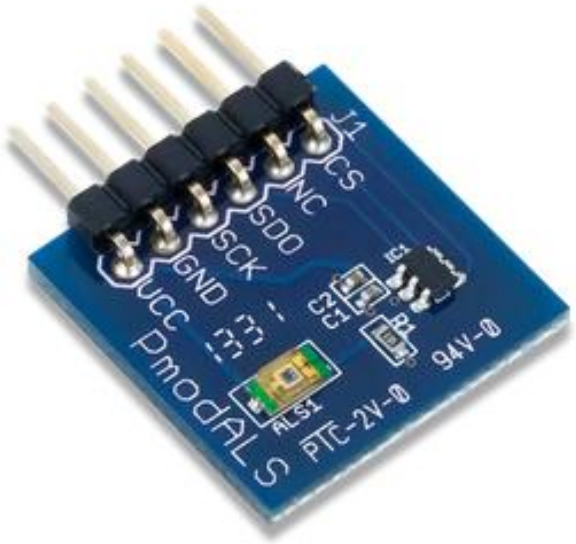
Required Equipment

- Make sure you've received the following equipment:



Pmod SSD

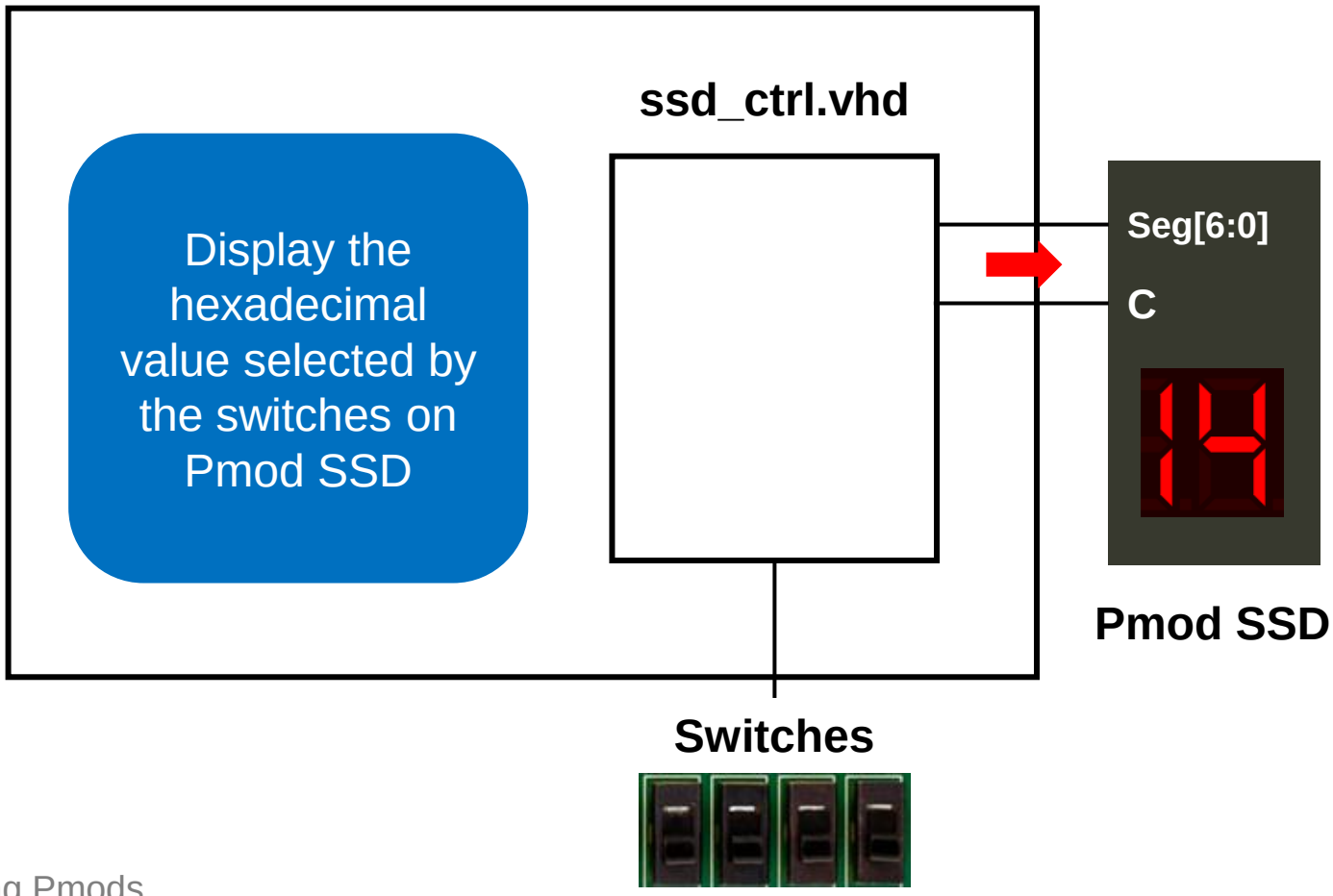
2-digit Seven-Segment Display



Pmod ALS

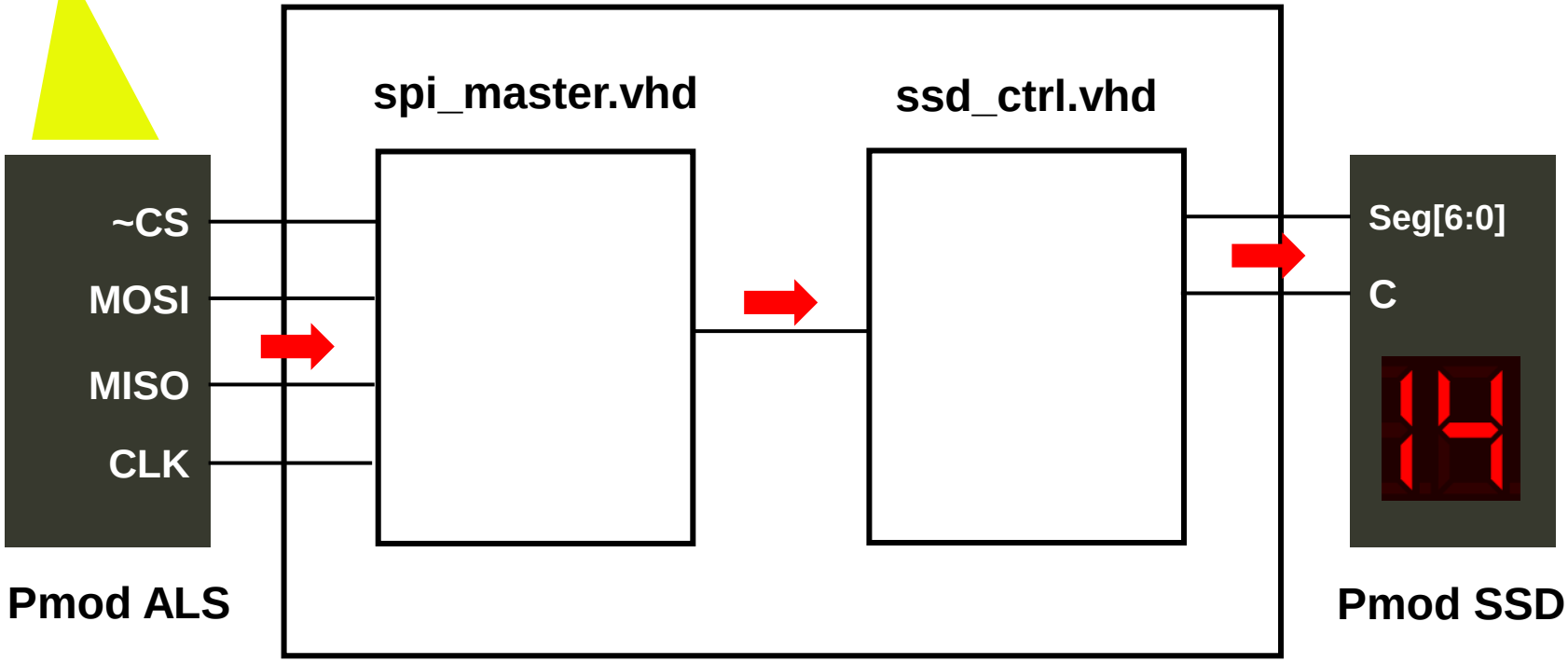
Ambient light sensor

Lab06: Integrating Pmod ALS & SSD



Lab06: Integrating Pmod ALS & SSD

lab06.vhd
(Task 2)

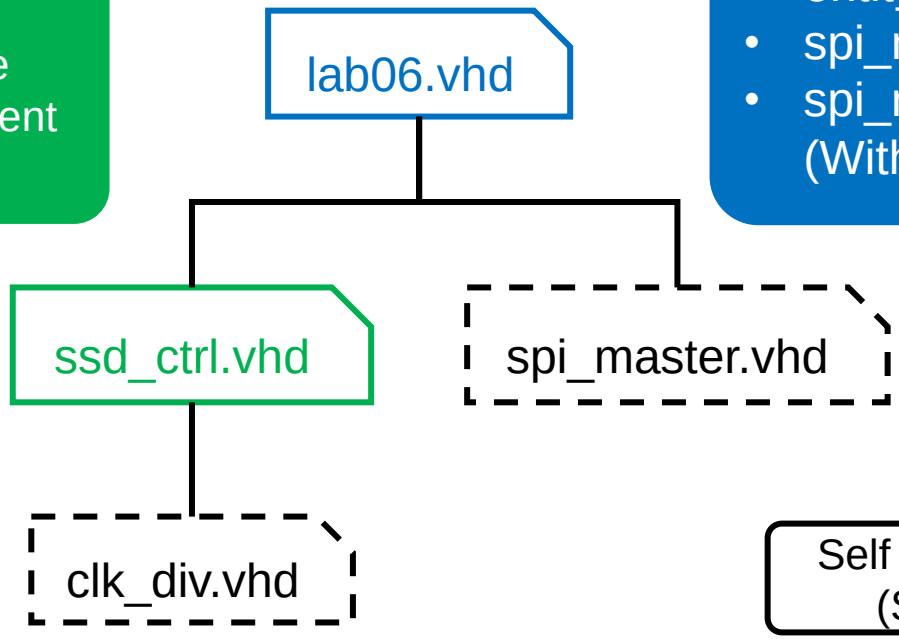


Display the light intensity on the PmodSSD in hexadecimal value

Lab06 Design Source Hierarchy

`ssd_ctrl.vhd` template
- simple digit assignment

`lab06.vhd` template
- entity
- spi_master component
- spi_master port map (Without value)

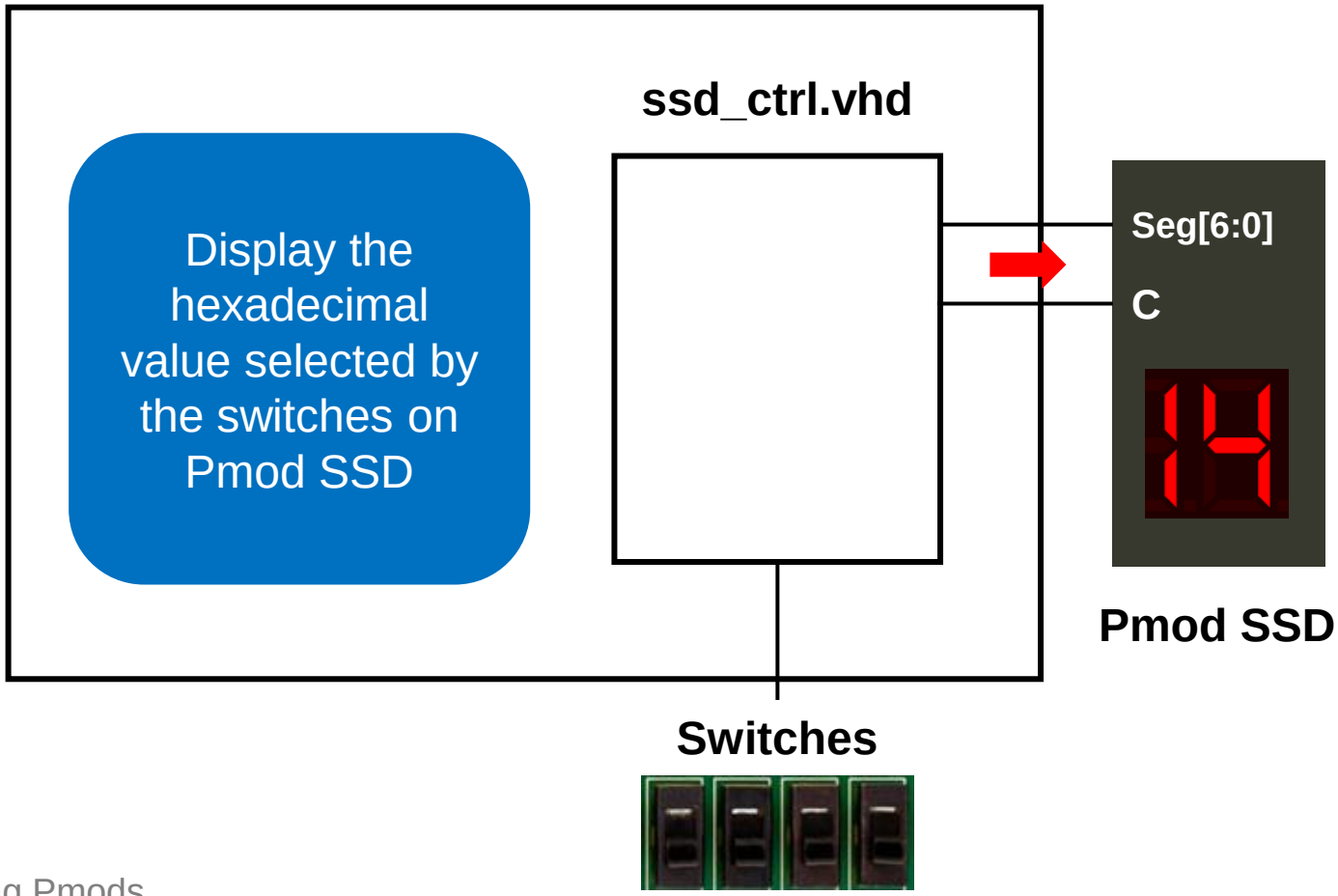


Self implement / Import
(See later slides)

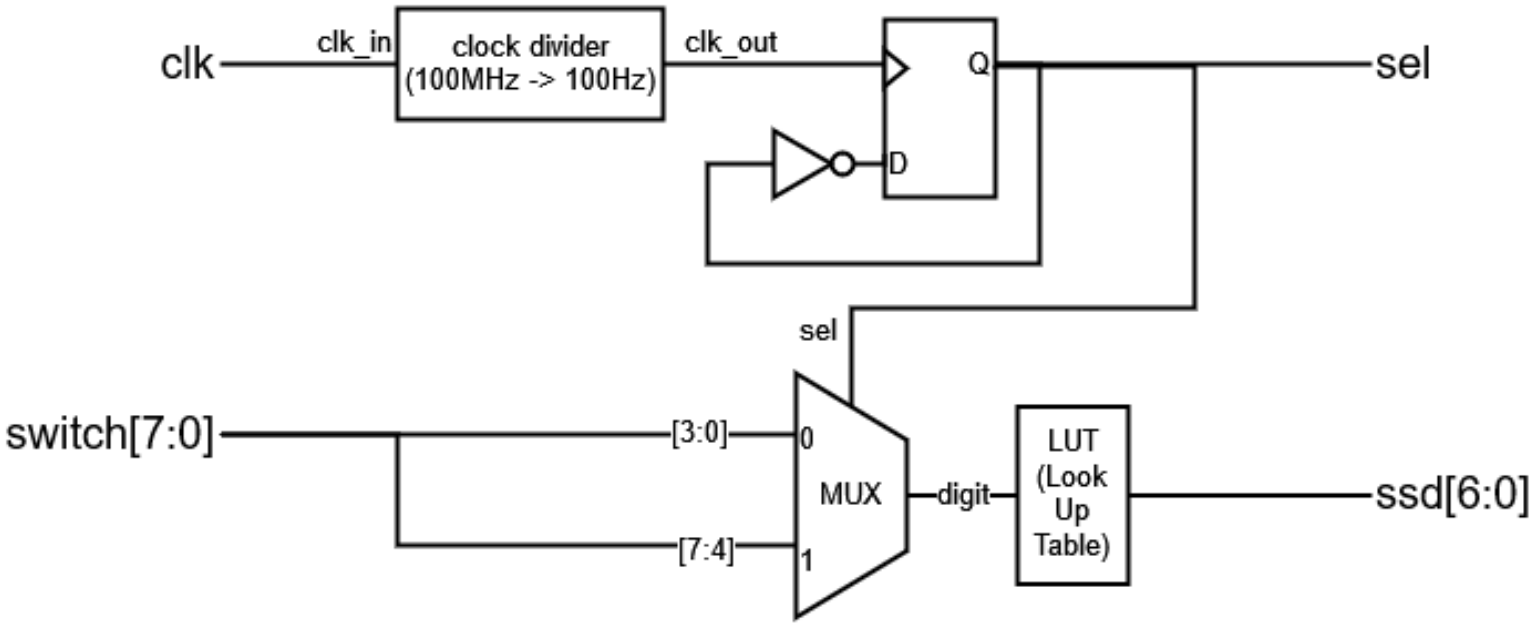
Task 1 - Driving Pmod SSD

Lab06: Integrating Pmod ALS & SSD

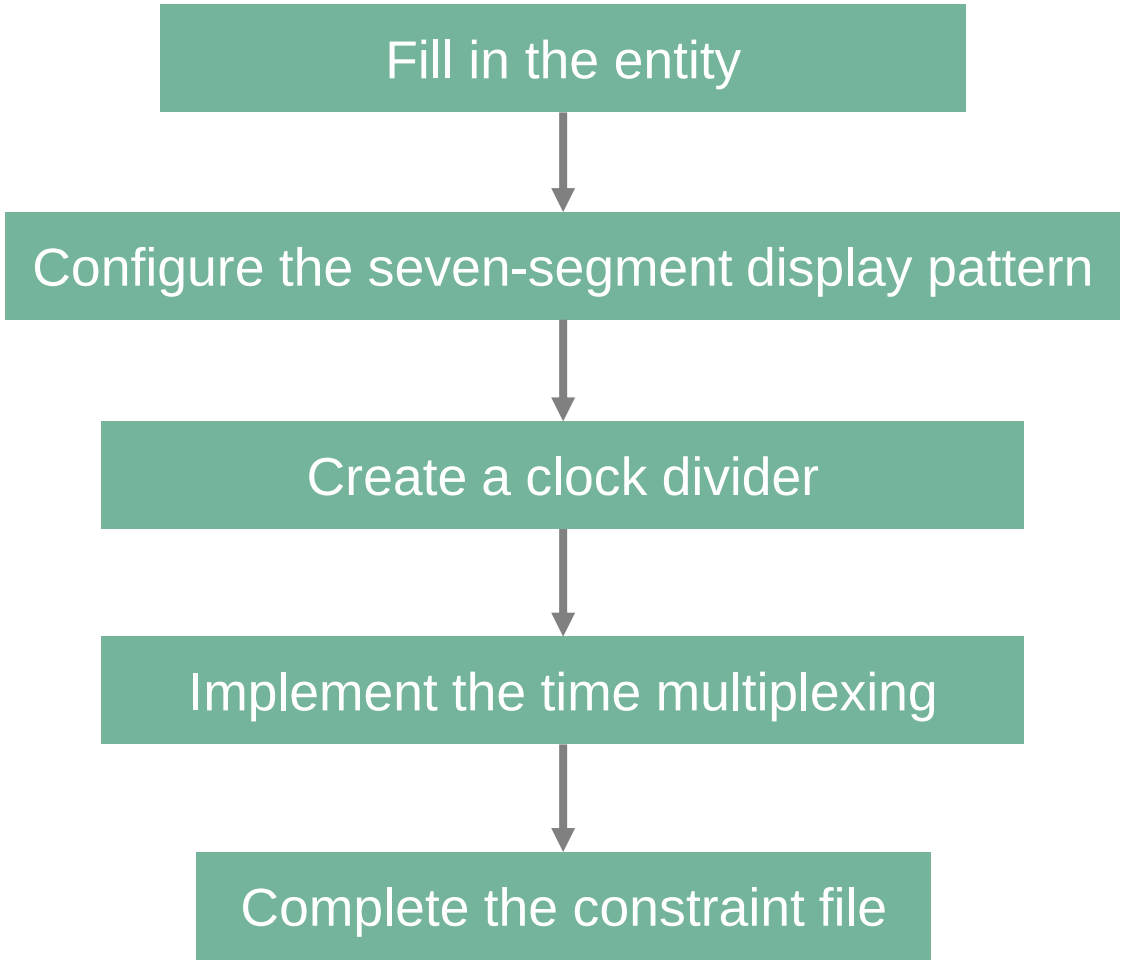
lab06.vhd
(Task 1)



Logic Diagram

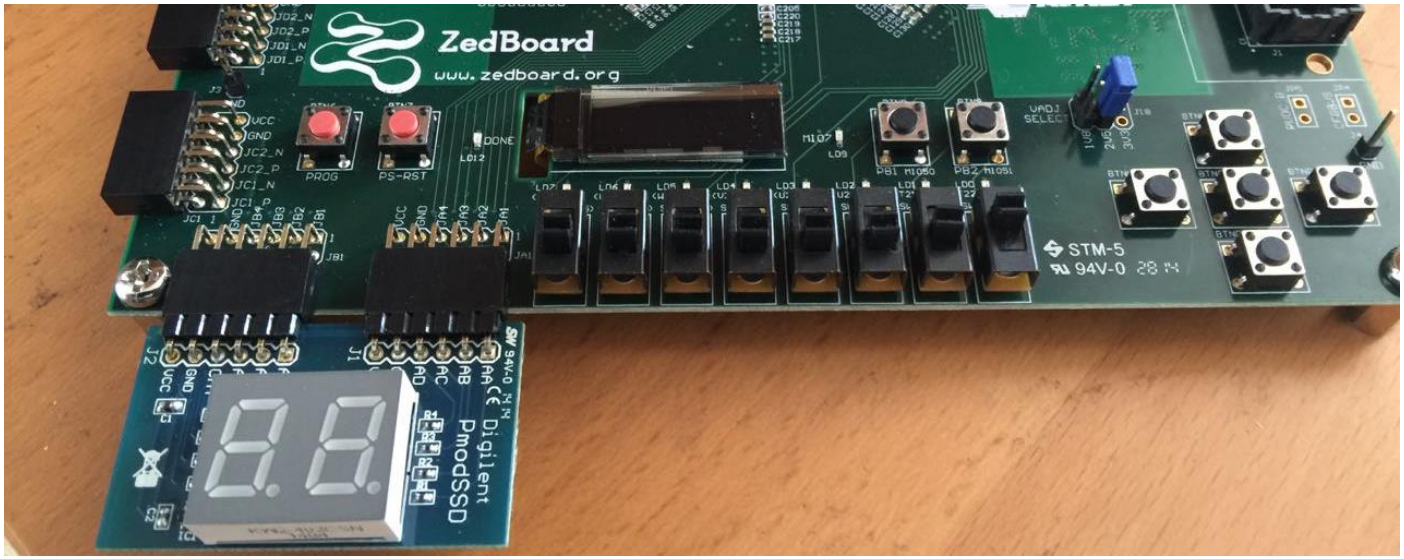


Task 1 – Design Flow



Driving Pmod SSD

- First, connect the Pmod via the top pins of JA & JB

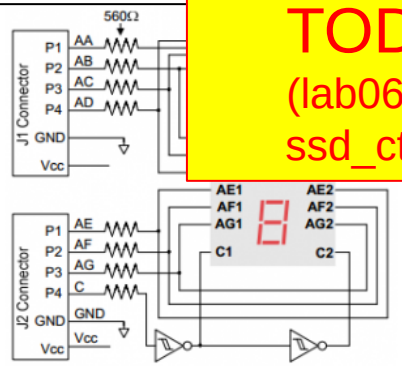


Driving Pmod SSD

- Fill in the entity of **lab06.vhd & ssd_ctrl.vhd**

Pinout Description Table

Header J1			Header J2		
Pin	Signal	Description	Pin	Signal	Description
1	AA	Segment A	1	AE	Segment E
2	AB	Segment B	2	AF	Segment F
3	AC	Segment C	3	AG	Segment G
4	AD	Segment D	4	C	Digit Selection pin
5	GND	Power Supply Ground	5	GND	Power Supply Ground
6	VCC	Positive Power Supply	6	VCC	Positive Power Supply



Seven-Segment Display Connection Diagram

PmodSSD Manual

TODO-1
(lab06.vhd & ssd_ctrl.vhd)

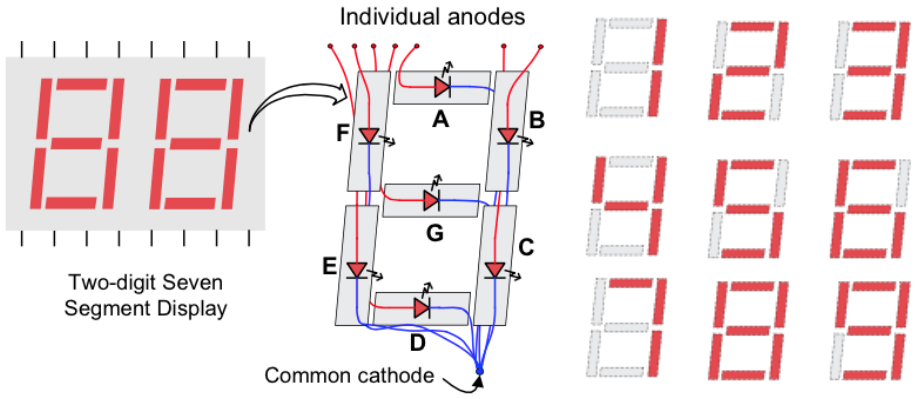
```
entity lab06 is
  Port (
    -- TODO-1: Fill in the input/output ports for Pmod SSD and switches
    clk         : in std_logic;
    switch      : in std_logic_vector (7 downto 0);
    sel         : buffer std_logic := '0';
    ssd         : out std_logic_vector (6 downto 0)
    -- TODO-10: Fill in the input/output ports for Pmod ALS
  );
end lab06;
```

```
entity ssd_ctrl is
  Port (
    -- TODO-1: Create the input/output ports
    clk         :in std_logic;
    data_in     :in std_logic_vector (7 downto 0);
    sel         :buffer std_logic := '0';
    ssd         :out std_logic_vector (6 downto 0)
  );
end ssd_ctrl;
```

Driving Pmod SSD

- Fill in the ssd configuration process

TODO-2
(ssd_ctrl.vhd)



Input Value	Segments Lit	Output Value
0 $(0000)_2$	A B C D E F	"1111110"
1 $(0001)_2$	B C	"0110000"
2 $(0010)_2$	A B D E G	"1101101"
3 $(0011)_2$	A B C D G	"1111001"
4 $(0100)_2$	B C F G	"0110011"
5 $(0101)_2$	A C D F G	"1011011"
6 $(0110)_2$	A C D E F G	"1011111"
7 $(0111)_2$	A B C	"1110000"
8 $(1000)_2$	…	Fill in the blank
9 $(1001)_2$	…	…

```
-- TODO-2: Fill in the blank
process(digit) begin
    case digit is
        when "0000" => ssd <= "1111110";    -- 0x0
        when "0001" => ssd <= "0110000";    -- 0x1
        when "0010" => ssd <= "1101101";    -- 0x2
        when "0011" => ssd <= "1111001";    -- 0x3
        when "0100" => ssd <= "0110011";    -- 0x4
        when "0101" => ssd <= "1011011";    -- 0x5
        when "0110" => ssd <= "1011111";    -- 0x6
        when "0111" => ssd <= "1110000";    -- 0x7
        when "1000" => ssd <= "";           -- 0x8
        when "1001" => ssd <= "";           -- 0x9
        when "1010" => ssd <= "";           -- 0xA
        when "1011" => ssd <= "";           -- 0xb (lowercase)
        when "1100" => ssd <= "";           -- 0xC
        when "1101" => ssd <= "";           -- 0xd (lowercase)
        when "1110" => ssd <= "";           -- 0xE
        when "1111" => ssd <= "";           -- 0xF
        when others => ssd <= "00000000";
    end case;
end process;
```

Driving Pmod SSD

TODO-3

Create the clk_div.vhd

```
34  entity clk_div is
35    generic(
36      N :          integer
37    );
38    Port (
39      clk_in :     in std_logic;
40      clk_out :    buffer std_logic := '0'
41    );
42  end clk_div;
43
44  architecture Behavioral of clk_div is
45      signal counter :     integer := 0;
46  begin
47
48      process(clk_in)
49      begin
50          if rising_edge(clk_in) then
51              if counter = N - 1 then
52                  clk_out <= not clk_out;
53                  counter <= 0;
54              else
55                  counter <= counter + 1;
56              end if;
57          end if;
58      end process;
59
60  end Behavioral;
```

TODO-4&5
(ssd_ctrl.vhd)

- **4. Create the component of clk_div on ssd_ctrl.vhd**
- **5. Port map the clk_div**

(From onboard clock source **100MHz to 100Hz**)

(Create any signal if you needed)

Driving Pmod SSD

TODO-6
(ssd_ctrl.vhd)

Digit Selection Pin (sel or C)

Header J1			Header J2		
Pin	Signal	Description	Pin	Signal	Description
1	AA	Segment A	1	AE	Segment E
2	AB	Segment B	2	AF	Segment F
3	AC	Segment C	3	AG	Segment G
4	AD	Segment D	4	C	Digit Selection pin

PmodSSD Manual

- Display the digit on the **right when sel = 0**
- Display the digit on the **left when sel = 1**
- **Hence**
  - **Toggle sel by the 100Hz clock**
  - **Set digit as data[3:0] when sel = 0, and data[7:4] when sel = 1**

Driving Pmod SSD

TODO-7&8
(lab06.vhd)

- Switch back to lab06.vhd

6. Create the component of ssd_ctrl

7. Port map the ssd_ctrl

- Finally, add the lab06.xdc

We use the top pins only

Pmod	Signal Name	Zynq pin	Pmod	Signal Name	Zynq pin
JA1	JA1	Y11	JB1	JB1	W12
	JA2	AA11		JB2	W11
	JA3	Y10		JB3	V10
	JA4	AA9		JB4	W8
	JA7	AB11		JB7	V12
	JA8	AB10		JB8	W10
	JA9	AB9		JB9	V9
	JA10	AA8		JB10	V8

Zedboard Hardware User Guide

Header J1			Header J2		
Pin	Signal	Description	Pin	Signal	Description
1	AA	Segment A	1	AE	Segment E
2	AB	Segment B	2	AF	Segment F
3	AC	Segment C	3	AG	Segment G
4	AD	Segment D	4	C	Digit Selection pin

PmodSSD Manual

Driving Pmod SSD

TODO-9
(lab06.xdc)

- Constraint file configuration
  - **Fill in the {ssd} and {sel} constraint**

Pmod	Signal Name	Zynq pin	Pmod	Signal Name	Zynq pin
JA1	JA1	Y11	JB1	JB1	W12
	JA2	AA11		JB2	W11
	JA3	Y10		JB3	V10
	JA4	AA9		JB4	W8

Header J1			Header J2		
Pin	Signal	Description	Pin	Signal	Description
1	AA	Segment A	1	AE	Segment E
2	AB	Segment B	2	AF	Segment F
3	AC	Segment C	3	AG	Segment G
4	AD	Segment D	4	C	Digit Selection pin

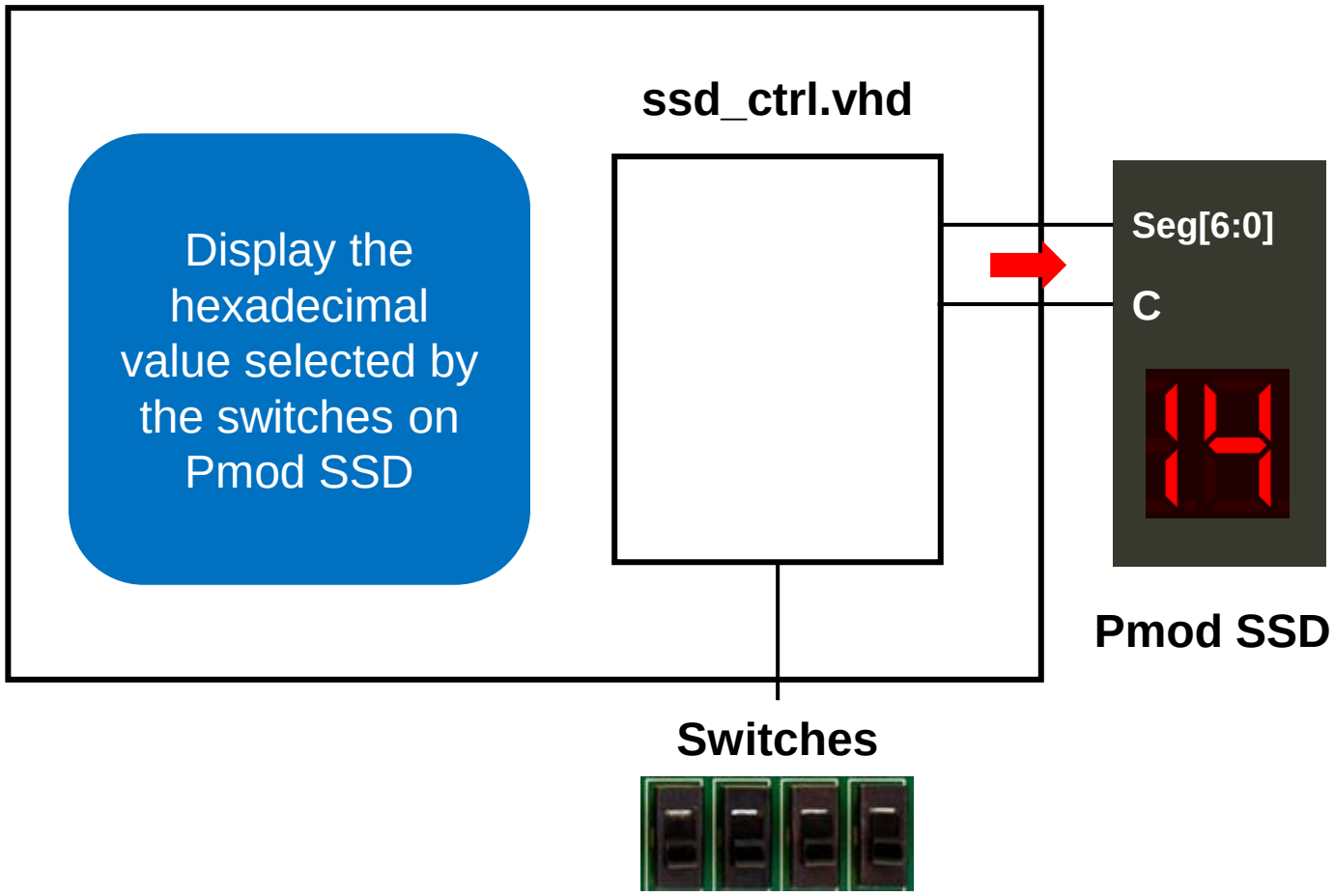
Input Value	Segments Lit	Output Value
0 $(0000)_2$	A B C D E F	"1111110"
1 $(0001)_2$	B C	"0110000"
...	...	...

'A' should be ssd[6] at Y11

```
# TODO-9: Fill in the constraint of {ssd} and {sel}
set_property PACKAGE_PIN Y11 [get_ports {ssd[6]}];
set_property IOSTANDARD LVCMOS33 [get_ports ssd];

#TODO: Step 6 - Complete the constraint file for pmod als


# Do not need to modify below
set_property PACKAGE_PIN Y9 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
create_clock -period 10 [get_ports clk]

set_property PACKAGE_PIN F22 [get_ports {switch[0]}];
set_property PACKAGE_PIN G22 [get_ports {switch[1]}];
set_property PACKAGE_PIN H22 [get_ports {switch[2]}];
set_property PACKAGE_PIN F21 [get_ports {switch[3]}];
set_property PACKAGE_PIN H19 [get_ports {switch[4]}];
set_property PACKAGE_PIN H18 [get_ports {switch[5]}];
```

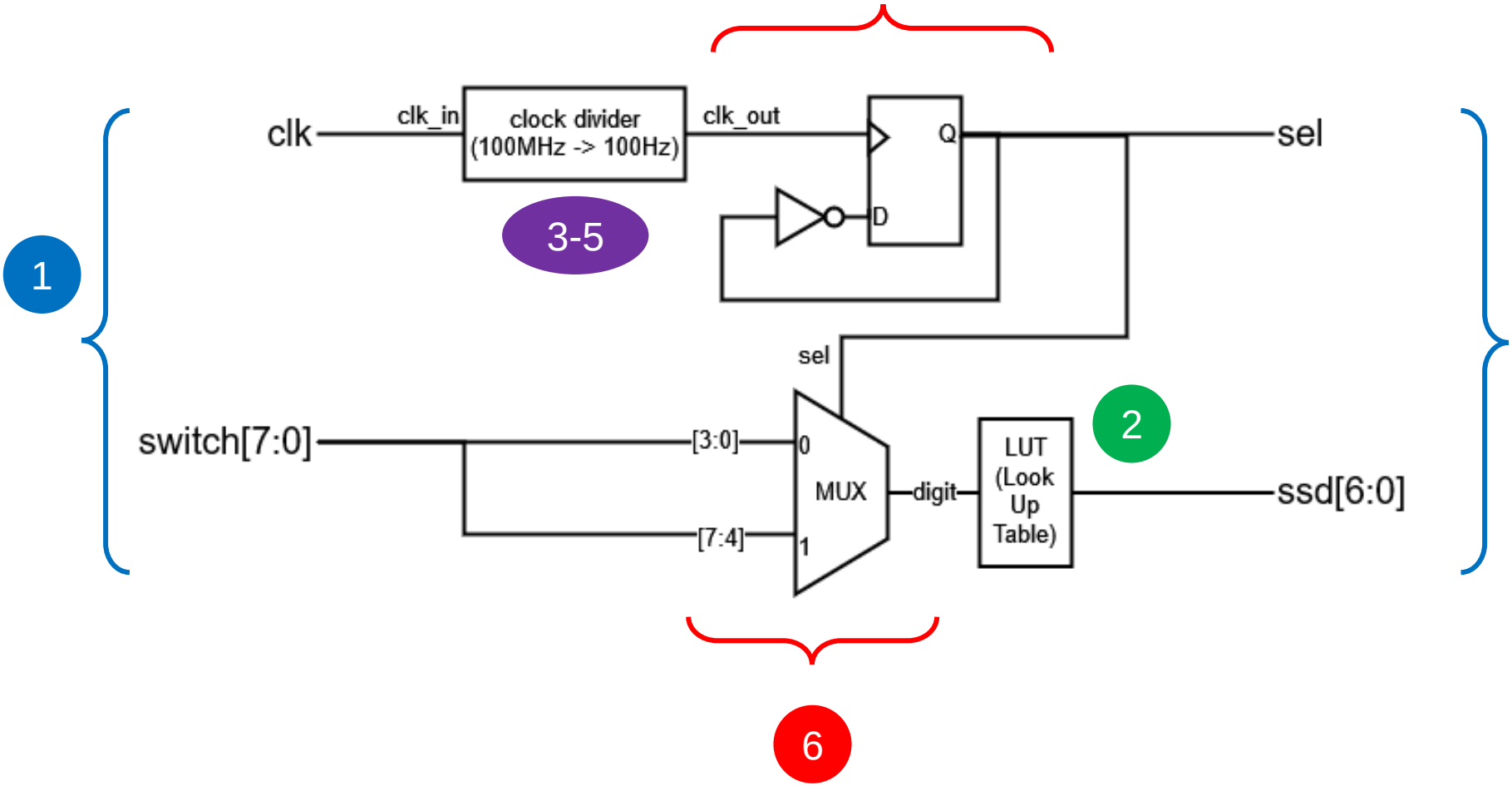
Task 1 Done!

- Test your implementation



Driving Pmod SSD

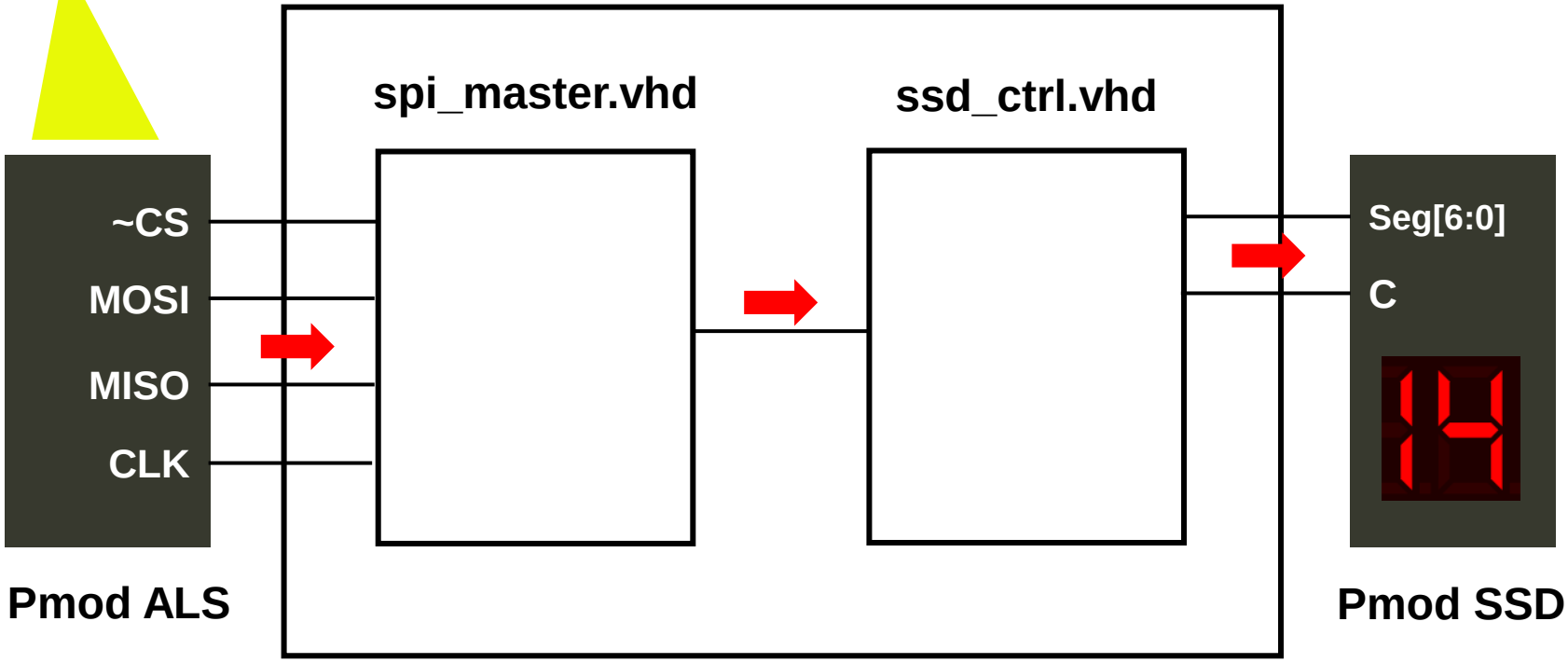
- Recall all the steps and verify your design



Task 2 - Driving Pmod ALS

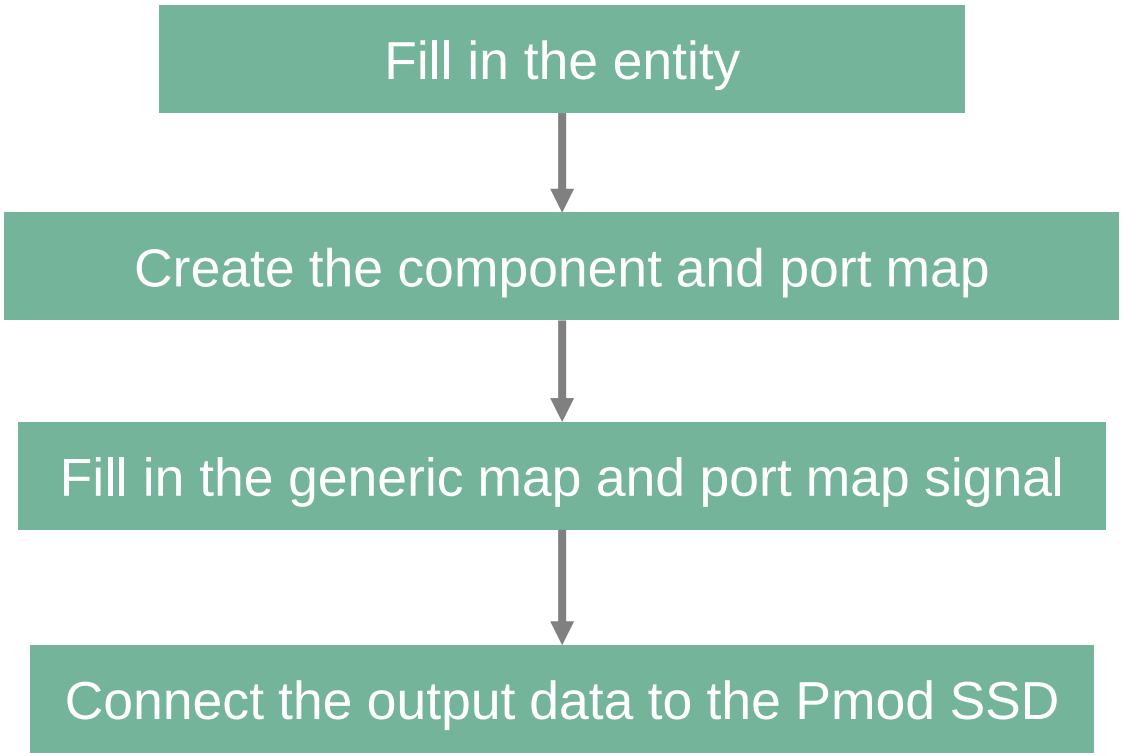
Lab06: Integrating Pmod ALS & SSD

lab06.vhd
(Task 2)



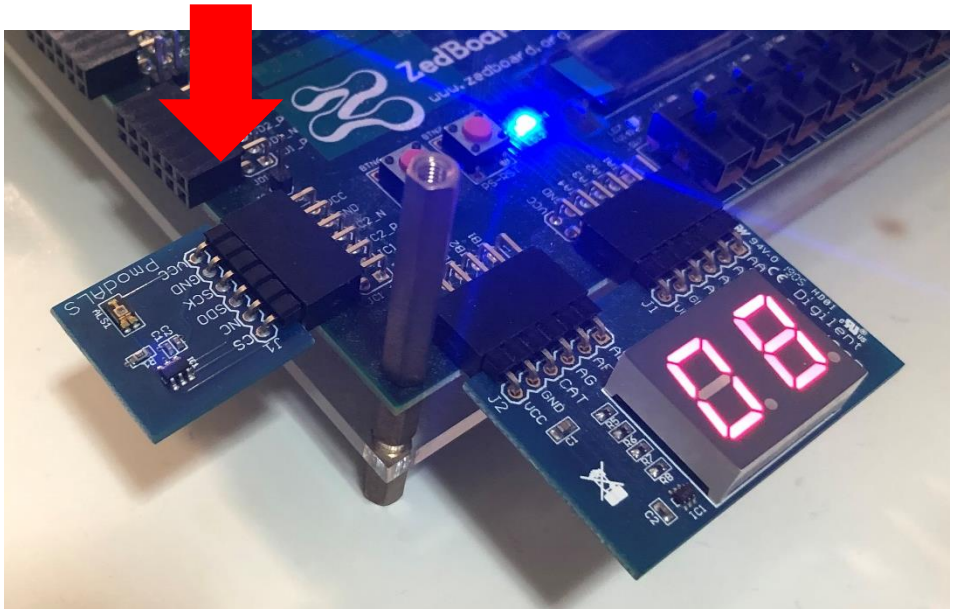
Display the light intensity on the PmodSSD in hexadecimal value

Task 2 – Design Flow



Driving Pmod ALS

- Connect the Pmod via the top pins of JC1



Driving Pmod ALS

- Complete the entity for Pmod ALS

TODO-10
(lab06.vdh)

Interfacing with the Pmod

The PmodALS communicates with the host board via the SPI protocol in SPI Mode 3. Since the on-board analog-to-digital converter is a read-only module, the only wires in the SPI protocol that are required are the Chip Select, Master-In-Slave-Out, and Serial Clock lines. The location of each of these lines on the Pmod header are shown in the table below.

Pinout Description Table

Pin	Signal	Description
1	~CS	Chip Select
2	NC	Not Connected
3	SDO	Master-In-Slave-Out
4	SCK	Serial Clock
5	GND	Power Supply Ground
6	VCC	Power Supply (3.3V/5V)

MISO

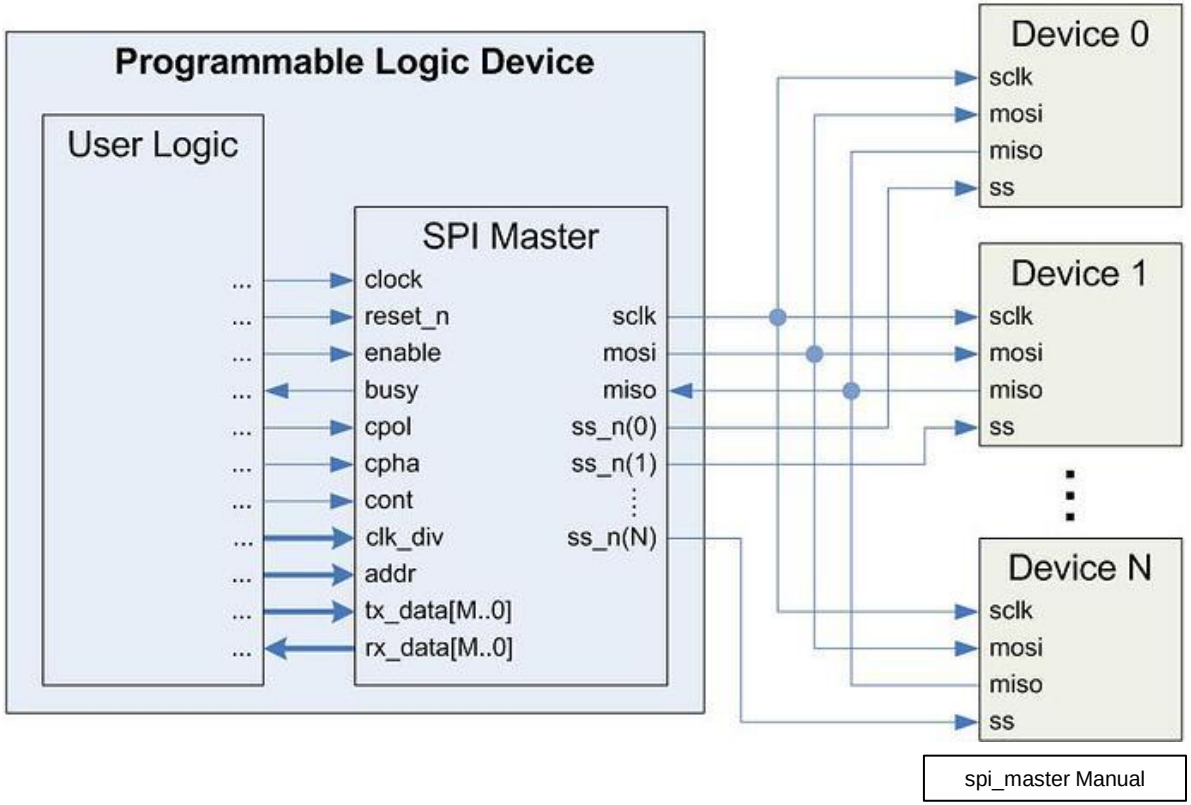
MOSI is not used

PmodALS Manual

```
-- TODO-10: Fill in the input/output ports for Pmod ALS
cs          : buffer std_logic_vector (0 downto 0);
-- mosi     : out std_logic; -- We don't use it in this lab
miso        : in std_logic;
sclk        : buffer std_logic
```

Driving Pmod ALS

We are going to use the library called spi_master (https://forum.digikey.com/t/spi-master-vhdl/12717)

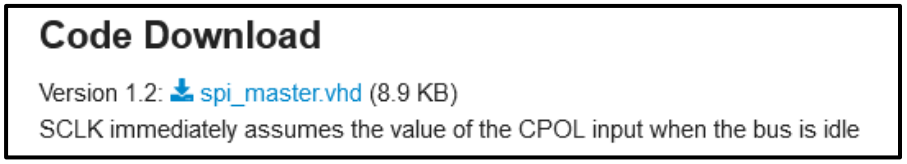


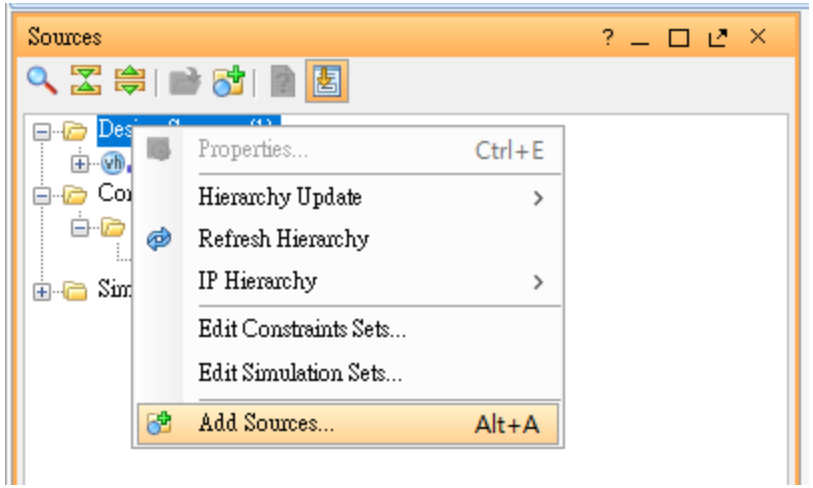
Driving Pmod ALS

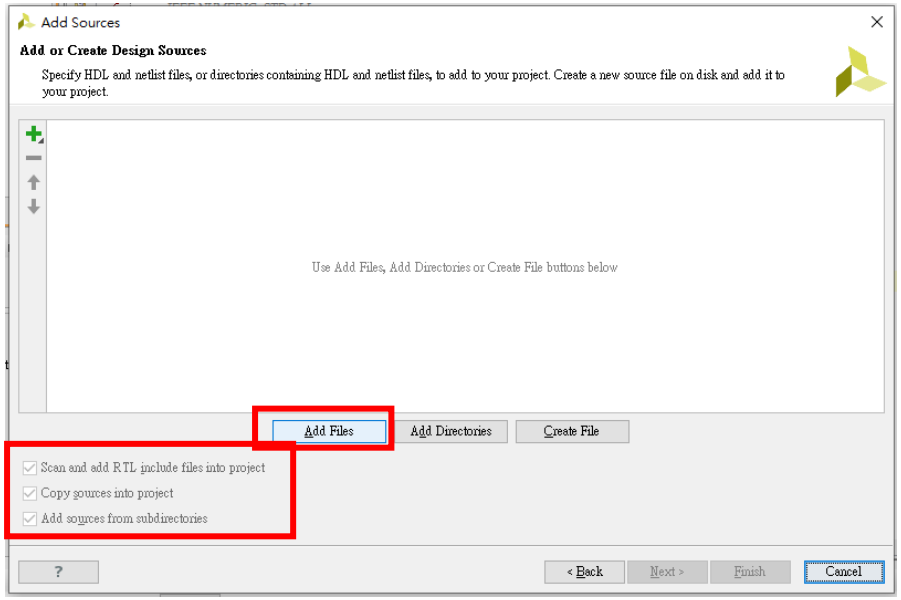
- **Add design source**

Download the spi_master.vhd, and import the file to our project

(Right click Design Source -> Click “Add Source…”







Tick all three checkbox

Driving Pmod ALS

TODO-11&12
(lab06.vdh)

- **Uncomment the component and port map** of spi_master at lab06.vhd
- Now, we are going to **go through the ports one by one**

```
-- TODO-10: Uncomment the component
component spi_master
  generic(
    slaves  : INTEGER;  --number of spi slaves
    d_width : INTEGER); --data bus width
  port(
    clock   : IN     STD_LOGIC;                             --system clock
    reset_n : IN     STD_LOGIC;                             --asynchronous reset
    enable  : IN     STD_LOGIC;                             --initiate transaction
    cpol    : IN     STD_LOGIC;                             --spi clock polarity
    cpha    : IN     STD_LOGIC;                             --spi clock phase
    cont    : IN     STD_LOGIC;                             --continuous mode command
    clk_div : IN     INTEGER;                               --system clock cycles per 1/2 period of sclk
    addr    : IN     INTEGER;                               --address of slave
    tx_data : IN     STD_LOGIC_VECTOR(d_width-1 DOWNTO 0);  --data to transmit
    miso    : IN     STD_LOGIC;                             --master in, slave out
    sclk    : BUFFER STD_LOGIC;                             --spi clock
    ss_n    : BUFFER STD_LOGIC_VECTOR(slaves-1 DOWNTO 0);   --slave select
    mosi    : OUT    STD_LOGIC;                             --master out, slave in
    busy    : OUT    STD_LOGIC;                             --busy / data ready signal
    rx_data : OUT    STD_LOGIC_VECTOR(d_width-1 DOWNTO 0)); --data received
end component;
```

```
-- TODO-11: Uncomment the component
  spi_master_0: spi_master
  generic map(
      slaves =>
      d_width =>
  )
  port map(
      clock =>
      clk_div =>
      sclk =>
      miso =>
      ss_n =>
      mosi =>
      reset_n =>
      addr =>
      cpol =>
      cpha =>
      tx_data =>
      rx_data =>
      enable =>
      busy =>
      cont =>
  );
```

Driving Pmod ALS

- spi_master could driver multiple SPI devices (Slaves)
  - In task 2 we drive the Pmod ALS only

Thus, 1 slave

- How about data width (`d_width`)?

> The PmodALS reports to the host board when the ADC081S021 is placed in normal mode by bringing the CS pin low, and delivers a single reading in 16 SCLK clock cycles. The PmodALS requires the frequency of the SCLK to be between 1 MHz and 4 MHz. The bits of information, placed on the falling edge of the SCLK and valid on the subsequent rising edge of SCLK, consist of three leading zeroes, the eight bits of information with the MSB first, and four trailing zeroes. An occasional 4th leading zero may be captured if CS goes low to initiate another data transfer before the rising edge SCLK.

PmodALS Manual

3 leading zeroes **8-bits data** **4 trailing zeroes**

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	X	X	X	X	X	X	X	X	0	0	0	0

In total 15 bits

Driving Pmod ALS

- According to the document of spi_master, the clock to sclk would be
  - (The same as the N of our clk_div)

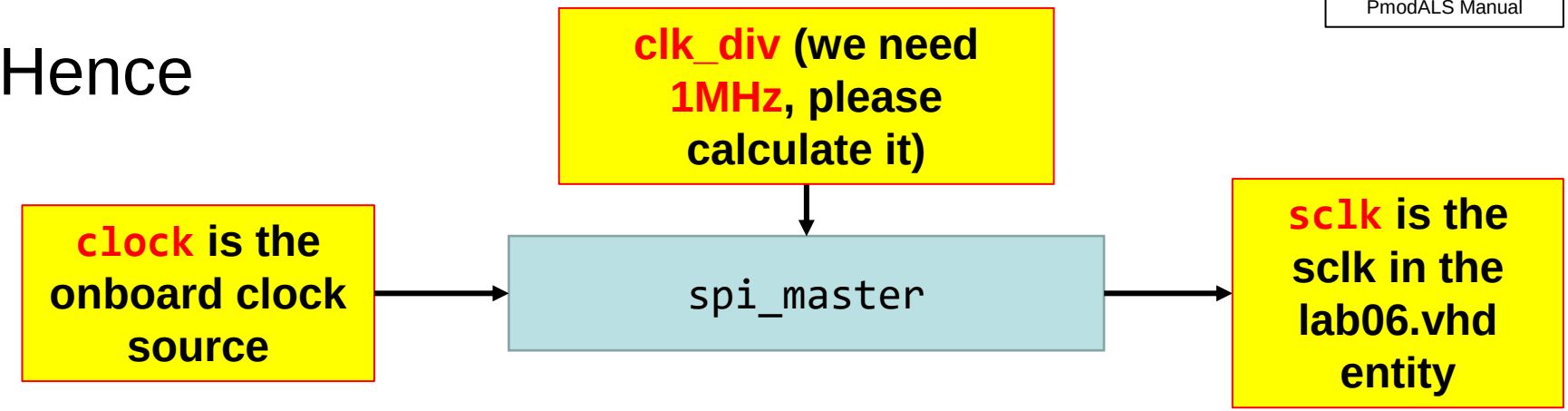
$$(1) \qquad f_{SCLK} = \frac{f_{clock}}{2 \cdot clk\_div}$$

spi_master Manual

The PmodALS reports to the host board when the ADC081S021 is placed in normal mode by bringing the CS pin low, and delivers a single reading in 16 SCLK clock cycles. The PmodALS requires the frequency of the SCLK to be between 1 MHz and 4 MHz. The bits of information, placed on the falling edge of the SCLK and valid on the subsequent rising edge of SCLK, consist of three leading zeroes, the eight bits of information with the MSB first, and four trailing zeroes. An occasional 4th leading zero may be captured if CS goes low to initiate another data transfer before the rising edge SCLK.

PmodALS Manual

- Hence



Driving Pmod ALS

PmodALS

```
-- TODO-10: Fill in the input/outp
cs      : buffer STD_LOG
mosi    : out    STD_LOGI
miso    : in     STD_LOGI
sclk    : buffer STD_LOG
```

miso	1	in	standard logic	slave devices	Master in, slave out data line.
ss_n	N^	buffer	standard logic vector	slave devices	Slave select signals.
mosi	1	out	standard logic	slave devices	Master out, slave in data line.

spi_master Manual

Connect to the ports at the entity of lab06.vhd

- **ss_n is cs (chip select)**

Since N = 1, the ss_n would be 1-bit std_logic_vector (0 downto 0)

- **miso => miso**
- **mosi => open** `-- Explain in a few pages later`

reset_n	1	in	standard logic	user logic	Asynchronous active low reset.

spi_master Manual

Active low means activate when signal is low

reset_n = '1' (Never reset)

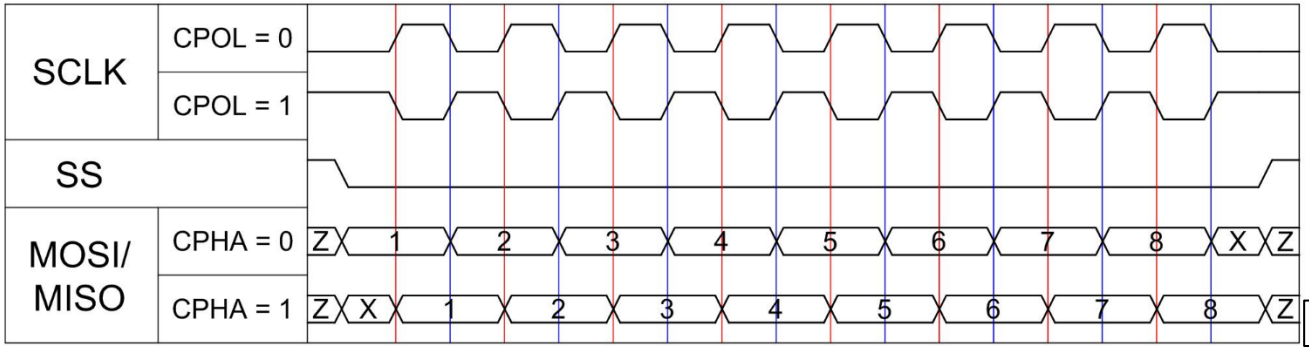
addr	32	in	integer	user logic	Address of target slave. The slaves are assigned addresses starting with 0.

spi_master Manual

address = 0 since we only have one slave

Driving Pmod ALS

- CPOL (Clock Polarity), CPHA (Clock Phase)



spi_master Manual

  - If CPOL = 0, then SCLK is normally low, and the first clock edge is a rising edge.
  - If CPOL = 1, SCLK is normally high, and the first clock edge is a falling edge.

  - If CPHA = 0, read on the first SCLK edge
  - If CPHA = 1, data is written on the first SCLK edge

The PmodALS reports to the host board when the ADC081S021 is placed in normal mode by bringing the CS pin low, and delivers a single reading in 16 SCLK clock cycles. The PmodALS requires the frequency of the SCLK to be between 1 MHz and 4 MHz. The bits of information, placed on the falling edge of the SCLK and valid on the subsequent rising edge of SCLK, consist of three leading zeroes, the eight bits of information with the MSB first, and four trailing zeroes. An occasional 4th leading zero may be captured if CS goes low to initiate another data transfer before the rising edge SCLK.

PmodALS Manual

CPOL = '1' and CPHA = '1'

Driving Pmod ALS

TODO-13
(lab06.vdh)

- Tx and Rx data
  - rx_data (receive) would be the 15-bit we want to receive
  - tx_data (transmit) would be the data you want to transfer (Not used in PmodALS)

tx_data => (OTHERS => '0')
Create a signal for rx_data

Unneeded Inputs on Module Instantiation

This final case occurs if you are instantiating a module that performs some functionality that you don't need for your purpose. If there is a port map signal that is not needed, you can simply wire it to *'0'* (for *std_logic* types) or *(others => '0')* for *std_logic_vector* types. For example:

```
1  Test_inst : Test_Module
2    port map (
3      i_Clk   => w_Clock,
4      i_Data  => (others => '0'),   -- i_Data  is type std_logic_vector
5      i_Valid => '0',               -- i_Valid is type std_logic
6      o_Data  => w_Data_Out,
7      o_Done  => open
8      );
```

https://www.nandland.com/vhdl/tutorials/dealing-with-unused-signals.html

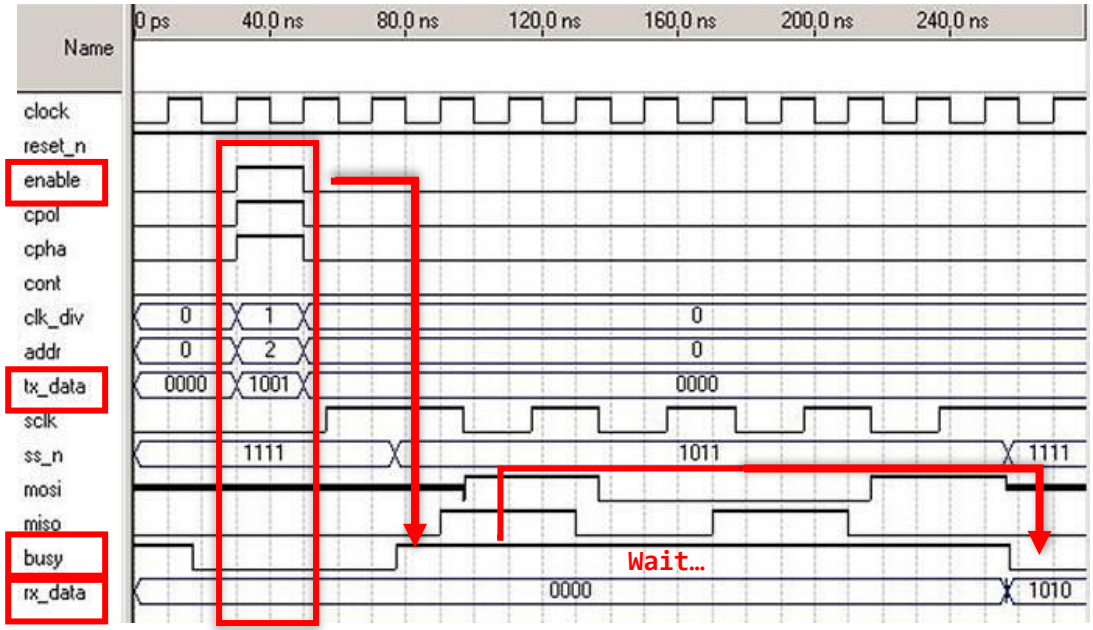
Driving Pmod ALS

spi_mater Manual

Port	Width	Mode	Data Type	Interface	Description
enable	1	in	standard logic	user logic	H: latches in settings, address, and data to initiate a transaction, L: no transaction is initiated.
cont	1	in	standard logic	user logic	Continuous mode flag.
tx_data	M*	in	standard logic vector	user logic	Data to transmit.
busy	1	out	standard logic	user logic	Busy / data ready signal.
rx_data	M*	out	standard logic vector	user logic	Data received from target slave.

Notes
* M is the specified data width, set by the *d_width* generic
^ N is the specified number of slaves, set by the *slaves* generic

spi_mater Manual



enable start a transaction, the `tx_data` would be accepted by the spi_master

The spi_master turn into **busy** (busy = 1), until the receive data is ready, then it would set the busy off (busy = 0, `rx_data` is ready)

For simplicity, we simply set **enable => '1', busy => open** *(open means no connection)*

Unused Input/Output Instantiation

Unused Outputs on Module Instantiation

This situation occurs when you instantiate a module that has an output that is not needed. There is a reserved keyword in VHDL, **open** which can be used in place of a signal name when you do the port mapping. Consider the example below:

```
1  Test_inst : Test_Module
2    port map (
3      i_Clk   => w_Clock,
4      i_Data  => w_Data_In,
5      i_Valid => w_Valid_In,
6      o_Data  => w_Data_Out,
7      o_Done  => open
8      );
```

In the instantiation above, the signal *o_Done* is not required in our higher level module, so we can leave it open using the VHDL reserved word. The synthesis tool will remove any logic inside of Test_Module that is used to drive the signal o_Done, since it is not being used by the higher-level module.

Unneeded Inputs on Module Instantiation

This final case occurs if you are instantiating a module that performs some functionality that you don't need for your purpose. If there is a port map signal that is not needed, you can simply wire it to *'0'* (for *std_logic* types) or *(others => '0')* for *std_logic_vector* types. For example:

```
1  Test_inst : Test_Module
2    port map (
3      i_Clk   => w_Clock,
4      i_Data  => (others => '0'),  -- i_Data  is type std_logic_vector
5      i_Valid => '0',              -- i_Valid is type std_logic
6      o_Data  => w_Data_Out,
7      o_Done  => open
8      );
```

https://www.nandland.com/vhdl/tutorials/dealing-with-unused-signals.html

Driving Pmod ALS

spi_mater Manual

Port	Width	Mode	Data Type	Interface	Description
enable	1	in	standard logic	user logic	H: latches in settings, address, and data to initiate a transaction, L: no transaction is initiated.
cont	1	in	standard logic	user logic	Continuous mode flag.
tx_data	M*	in	standard logic vector	user logic	Data to transmit.
busy	1	out	standard logic	user logic	Busy / data ready signal.
rx_data	M*	out	standard logic vector	user logic	Data received from target slave.

Notes
* M is the specified data width, set by the *d_width* generic
^ N is the specified number of slaves, set by the *slaves* generic

spi_mater Manual

cont means continuous mode. That means performing the transaction continuously.

e.g.
Send '10' first and then '01' at the tx_data
The spi_master receive '01' first, the receive '10' second

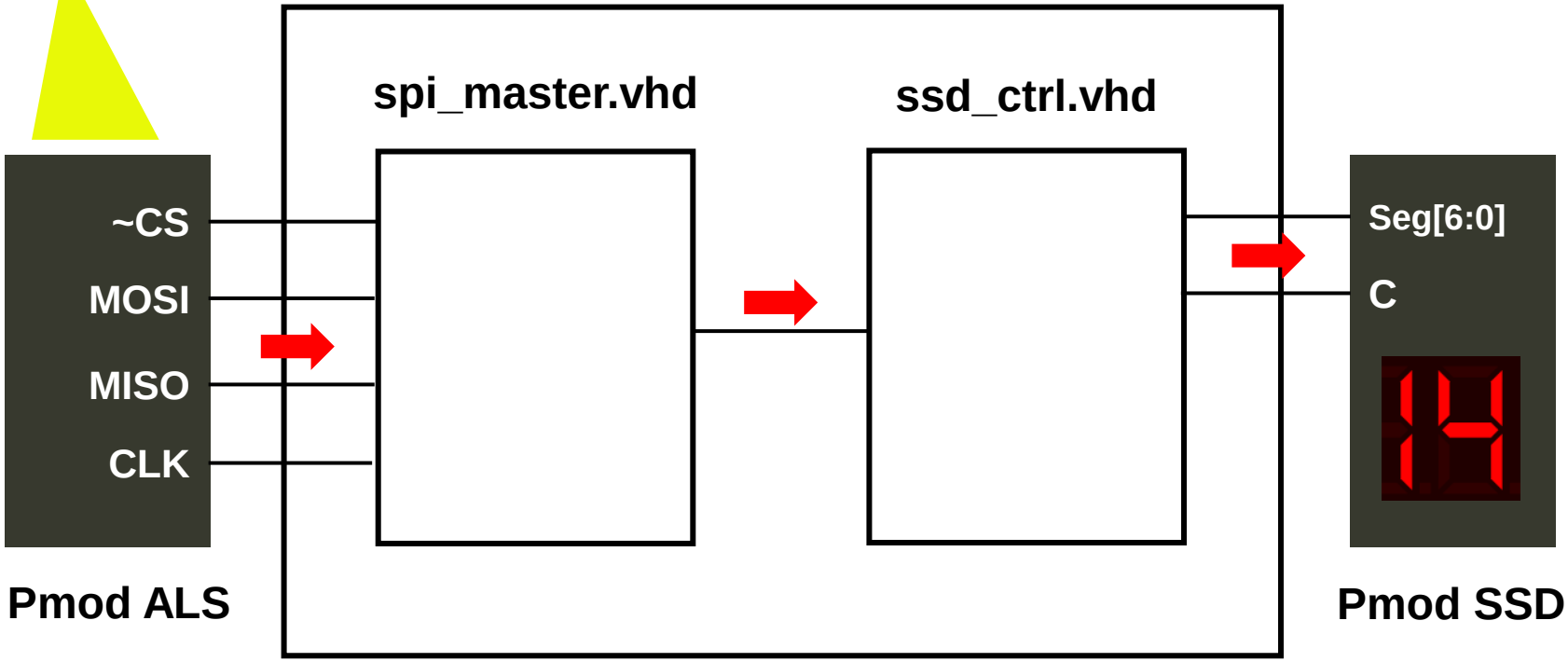
We do not use the continuous mode, so set **cont to 0**

1st rx_data is ready! 2nd rx_data is ready!

Driving the light sensor

TODO-14
(lab06.vdh)

lab06.vhd
(Task 2)



Send the rx_data from spi_master to ssd_ctrl
(Select the correct range of rx_data, see here)

Driving Pmod ALS

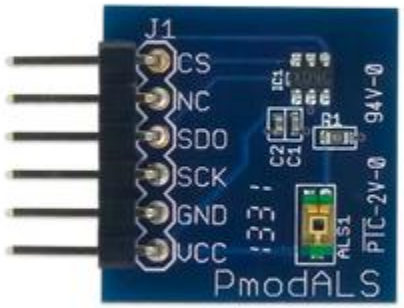
TODO-14
(lab06.xdc)

- Constraint file

Pmod	Signal Name	Zynq pin	
JC1 Differential	JC1_N	AB6	~~MOSI~~
	JC1_P	AB7	CS
	JC2_N	AA4	SCLK
	JC2_P	Y4	MISO
	JC3_N	T6	
	JC3_P	R6	
	JC4_N	U4	
	JC4_P	T4	

Zedboard Hardware User Guide

Any external power applied to the PmodALS must be within 2.7V and 5.25V; however it is recommended that Pmod is operated at 3.3V.



Lab06: Integrating Pmod ALS & SSD

- Finish the Task 1 & 2 by displaying the light intensity on the Pmod SSD
  - Pmod ALS on JC1-Top, Pmod SSD on JA1 and JB1 - Top
- In the demo video, you should clearly display
  1. The environment light intensity (about 3 to 5 sec.)
  2. Low intensity by covering the PmodALS (about 3 to 5 sec.)
  3. High intensity by approaching the light source to the PmodALS (about 3 to 5 sec.)
- Submit the zip file of lab06.vhd, ssd_ctrl.vhd, clk_div.vhd and the verification video to Blackboard.
  - Deadline: **12:30 on 12 Mar 2025**
    - Late submission is NOT acceptable unless otherwise approved.

References

- PmodSSD Manual
  - https://digilent.com/reference/pmod/pmodssd/reference-manual
- Zedboard Hardware User Guide
  - https://www.farnell.com/datasheets/2549188.pdf
- PmodALS Manual
  - https://digilent.com/reference/pmod/pmodals/reference-manual
- spi_master Manual
  - https://forum.digikey.com/t/spi-master-vhdl/12717

香港中文大學
The Chinese University of Hong Kong

Thank You!