Lab#5 - SPI communication

Leonardo Fusser (1946995)

Objectives:

- Write a SPI HAL layer library.
- Use a logic analyzer to analyze a SPI frame.
- Write an OLED driver layer library.
- Write an application of the OLED lib.

Hardware: Logic analyser, Explorer 16/32 board, PMOD OLEDrgb board, pin header, 100 Ohm resistor.

Documents:

PmodOLEDrgb reference manual	pmodoledrgb_reference_manual.pdf
OLED controller datasheet	SSD1331_1.2.pdf
Explorer16/32 schematics	Explorer_16_32_Schematics_R6_3_with_comments_for_PIC32.pdf

To hand in on GitHub:

These sheets including answers to all questions and screenshots.
All screenshots must be properly identified. All values must be in hexadecimal or binary format.

C code :

i. Must be indented and commented properly.
ii. Function prolog header must be included.
iii. The use of macros and pre-processor directive (#ifdef) is now mandatory.
iv. main.c prolog header: Name and lab number, a short description, author, date, and version.
v. You must keep a version history of your project.

Use of macro:
E.g.

```
#define SS                  _LATD12
#define DRAWRECTANGLE       0x22
```

Table 1: Template file content

Files	Content
`initBoard.c`	Contain all functions related to initializing the board: oscillator, timers, IOs, etc.
`spi.c`	HAL layer: Contains implementations for low level SPI functions.
`oled.c`	Driver layer: Contains implementations for the OLED display functions.
`application.c`	Application layer: Contains implementations of the OLED library applications.
`util.c`	Contains blocking delay.

`spi.h`
`oled.h`
`initBoard.h`
`application.h`
`oled_init.h`
`util.h`
	Includes all dependencies, public macros, and function prototypes.

`main.c`	Initializes the resources used: IOs, libraries, etc. Executes the super loop.
`oled_init.a`	Static lib to initialize the OLED display.

Application Layer	`app_task1` `app_task2`

Driver Layer	`oled_clr()`
	`oled_drw_rect()`
	`oled_drw_line()`

Hardware Abstraction Layer	`spi_ld_buffer()` `spi_init()`

Lab Work

GitHub invitation: https://classroom.github.com/a/Je91SWiJ

Part 1: HAL layer - SPI library

For the explorer 16/32 board, only one of the many PIC32 microcontroller SPI ports is accessible through the PMOD JA connector.
Give its number. (e.g., SPI1, SPI2, SP3...).

- The SPI module that is accessible through the PMOD JA connector on the Explorer 16/32 board is the SPI2 module on PIC32. Refer to "Explorer_16_32_Schematics_R6_3_with_comments_for_PIC32.pdf" for complete details.

Assuming a system clock of 80MHz, write a function named `spi_init()` that initializes the SPI as follows:

SPI mode	Master
Communication mode	8 bits

The function passes 3 arguments:

```
void spi_init(int baud, int cpol, int cpha)
```

Write a function named `spi_ld_buffer()` that :

1. Loads a byte to the SPIxBUF (this data will be transferred automatically to the slave).
2. Wait for the transaction to complete.
3. Returns the value received in the SPIxBUF.

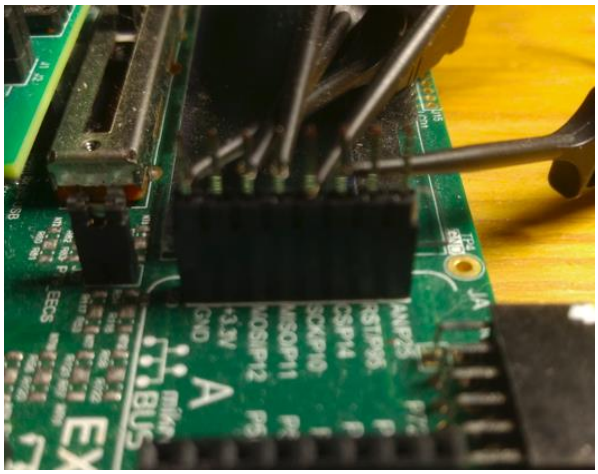
Without loopback

Test your newly created functions by repeatedly writing 0xA3. Set the baud rate and clock format as follows:

Clock format	Mode 0
Baud rate	100000 bits/sec

```
void main(void) {
    spi_init(…);
    while(1){
      /* Every 10 mS*/
      SS=1;
      dummy = spi_ld_buffer(0xA3);
      SS=0;
    }
}
```

Connect the logic analyzer to the SPI lines – see picture.



Using the logic analyzer, take a screenshot of the frame.

Note: the logic analyzer mode must be set properly!

➢ Refer to “Figure 1” on next page.

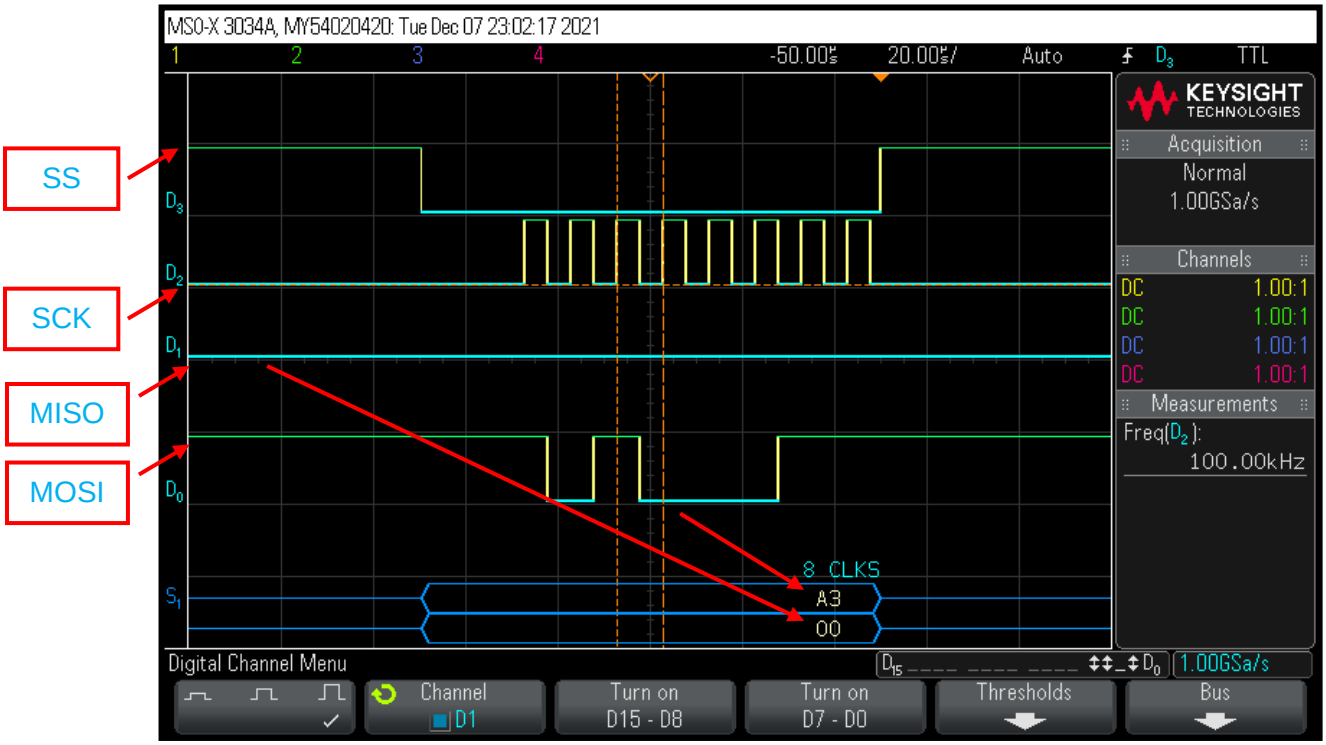


Figure 1. Screenshot for SPI frame that has a value of 0xA3 being written from the MOSI pin on PIC32. SPI mode is 0 (CPOL = 0 and CPHA = 0). SS is brought LOW and it takes 8 clock pulses to transmit the value of 0xA3 before SS is brought back HIGH. MISO shows low because there is no feedback.

Measure

Fill in the following table:

Clock frequency (Hz)	Clock Idle level (H or L)	Clock sampling edge (rising or falling)
100kHz.	Low.	Rising.

Using the debugger or the serial UART2 console, find value of dummy. Explain that value.

- The value of dummy is 0xFF. This is because there is no loopback connected to the MISO pin from MOSI pin on the PIC32. This means that whatever data that is sent on the MOSI pin won't be returned to the MISO pin. Instead, garbage values will be seen.

With loopback

Now, add a loopback using a resistor of 100 Ohms – see picture.

Using the logic analyzer, take a screenshot of the frame.

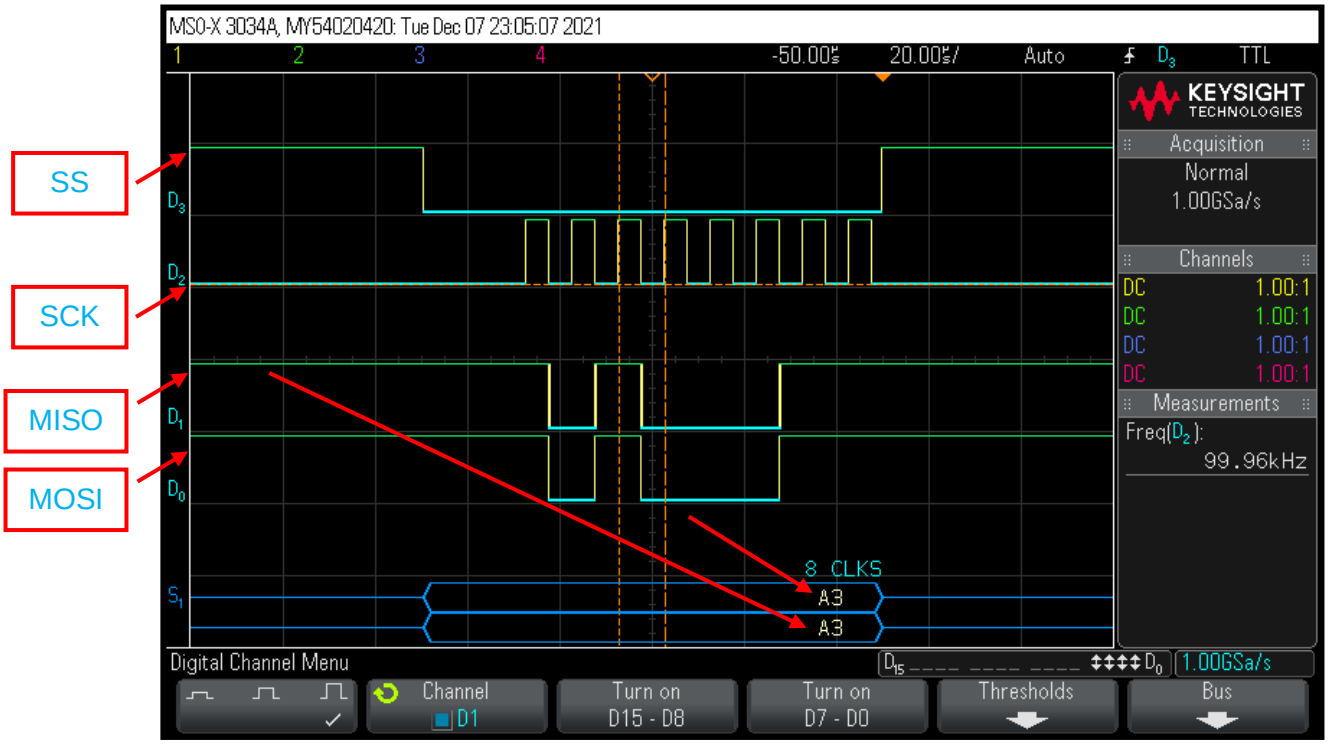


Figure 2. Screenshot for SPI frame that has a value of 0xA3 being written from the MOSI pin on PIC32. SPI mode is 0 (CPOL = 0 and CPHA = 0). SS is brought LOW and it takes 8 clock pulses to transmit the value of 0xA3 before SS is brought back HIGH. Value received by MISO is the same as value written from MOSI because there is feedback connected this time.

Again, using the debugger or the serial UART2 console, find value of dummy.

Explain that value.

- This time, the value of dummy is 0xA3. This is because a loopback is connected from the MOSI pin to the MISO pin on the PIC32. Since 0xA3 is being written on the MOSI pin, the MISO pin will receive the same value. This can be seen in the screenshot found in "Figure 2" above.

Comment the experiment

- There is an impact when there is and without a feedback loop connected between the MOSI and MISO pins on PIC32. When there is a feedback connected, the value being written out on the MOSI pin will be the same value being read on the MISO pin. When there isn't any feedback connected, the opposite occurs. Instead, the value being written out on the MOSI pin won't be the same value being read on the MISO pin. The value that will be read on the MISO pin would be garbage.

  Regardless of if feedback is connected or not, SS has to be asserted low in order to transmit data on the MOSI pin, and it will take 8 clock pulses before it will be completed. The SPI mode does not matter in this case.

Mode 3 clock format

Redo the previous experiment with the clock format in mode 3.

Fill in the following table:

Clock frequency (Hz)	Clock Idle level (H or L)	Clock sampling edge (rising or falling)
100kHz.	High.	Rising.

Using the logic analyzer, take a screenshot of the frame.

- Refer to "Figure 3" and "Figure 4" on next page.

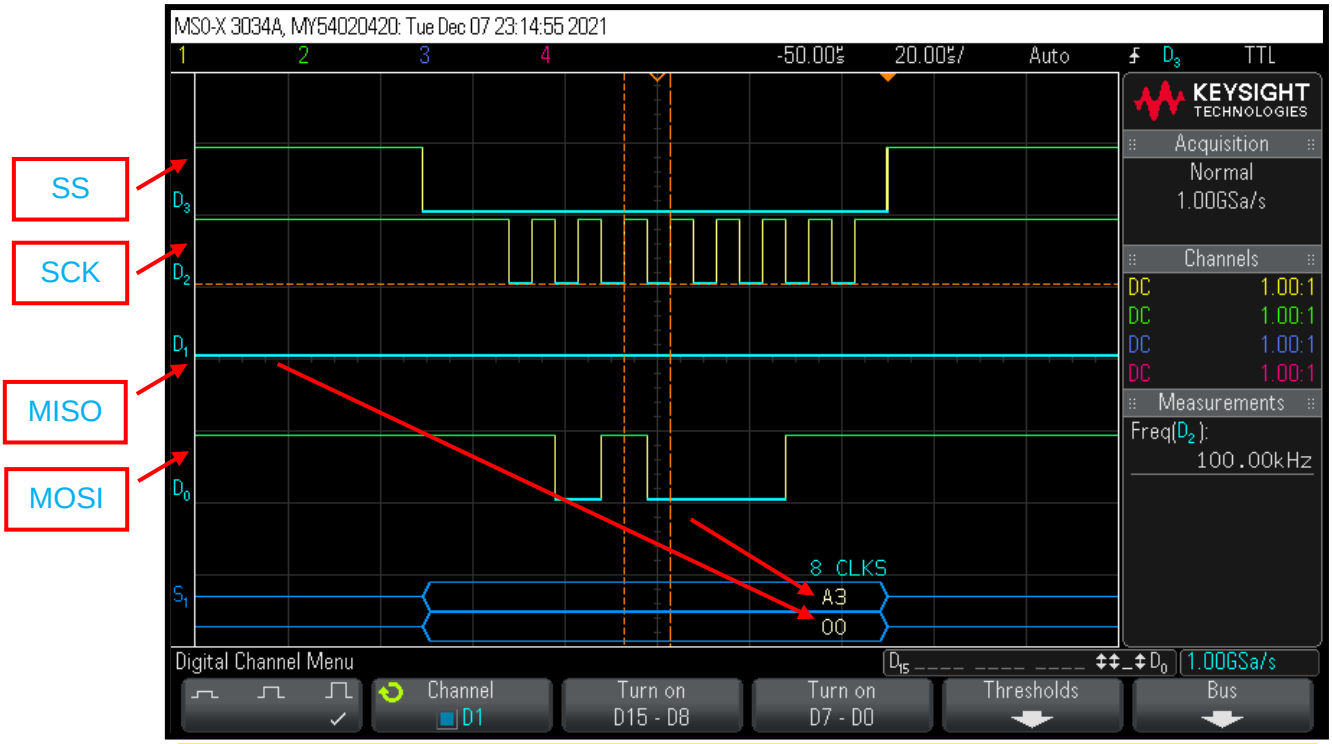


Figure 3. Screenshot for SPI frame that has a value of 0xA3 being written from the MOSI pin on PIC32. SPI mode is 3 (CPOL = 1 and CPHA = 1). SS is brought LOW and it takes 8 clock pulses to transmit the value of 0xA3 before SS is brought back HIGH. MISO shows low because there is no feedback.

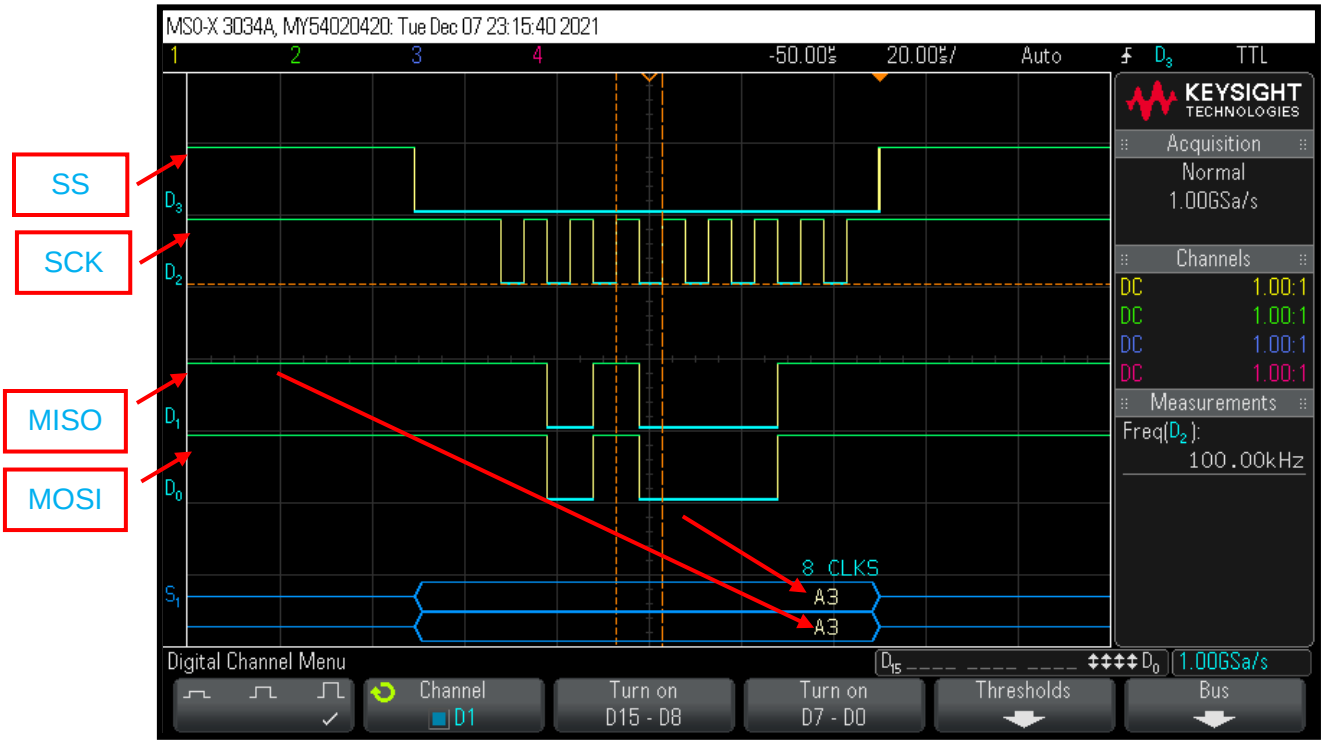
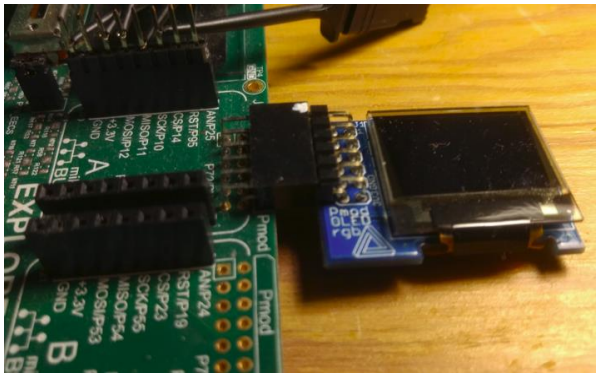


Figure 4. Screenshot for SPI frame that has a value of 0xA3 being written from the MOSI pin on PIC32. SPI mode is 3 (CPOL = 1 and CPHA = 1). SS is brought LOW and it takes 8 clock pulses to transmit the value of 0xA3 before SS is brought back HIGH. Value received by MISO is the same as value written from MOSI because there is feedback connected this time.

Part 2: Driver Layer - OLED library

In part2, you must write a library to communicate with the OLED display.



Using the Explorer16/32 schematics and PmodOLEDrgb reference manual, fill in the following table:

	Explorer16/32 Board Pmod JA			**PmodOLEDrgb display**
	PIC32 pin #	PIC32 I/O port	JA header pin#	J1 header pin#
MISO	P11		3	N/A
MOSI	P12		2	2
SCK	P10		4	4
CS	P14	Port G	1	1
D/C	P44	Port B	7	7

From the command table 10 along with section 9.2 of the SSD1331 controller datasheet, you must create the following function inside your library:

```
void oled_drw_line(int x1, int y1, int x2, int y2, int color)
void oled_clr(void)
void oled_drw_rect(int x1, int y1, int x2, int y2, int color)
```

From the SSD1331 datasheet, find the SPI mode.

Hint: see section 7.1.3 figure 8

- The SPI mode that the SSD1331 supports is SPI mode 2 (where CPOL = 1 and CPHA = 0, as indicated in the timing diagram/SPI frame example in the datasheet under section 7.1.3 – figure 8).

Remarks:

When setting a color, red, green, and blue must be sent separately.

Red color contains 5 bits
Green color contains 6 bits
Blue color contains 5 bits

Red					Green						Blue				
R4	R3	R2	R1	R0	G5	G4	G3	G2	G1	G0	B4	B3	B2	B1	B0

In your code, you must create a list of macros for all the commands.

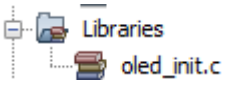
Example of macro:

```
#define   DRAWRECTANGLE  0x22
```

Since it takes about 1000uS to erase all ram internally, you must add a delay before returning from the `oled_clr` function.

The function `oled_init()` to initialize the display is already provided and you don't need to create it. This function must be the first being called by the main.

Hence, you must include the static lib:



Give a demo to the teacher.

Part 3: Application Layer

You must write the application of your choice.

It must be a dynamic application – something that moves.

Examples of applications:

Applications
A rotating line at a constant speed.
A bar graph representing the slider position.
A rotating line representing the slider position.
Applications - bonus
A bar graph representing the slider position. And… A line rotating at a variable speed.

To get a real time response, the baud rate must be increased to 3Mbits/sec.

Your solution must use tasks.

Give a demo to the teacher.

After lab questions

Q1- Is the MISO pin connected to the OLEDrgb display controller? Explain.

- The MISO pin is not connected to the OLEDrgb display controller. This is because the display does not have the capability of writing data back to a SPI master. As specified in the OLEDrgb datasheet, it can only read data that has been written by a SPI master. Therefore, there is no need for a MISO pin to be connected to the OLEDrgb display controller.