AARHUS
UNIVERSITY
AARHUS UNIVERSITY SCHOOL OF ENGINEERING

I3GFV

Experiment

Communication busses

Purpose

The main purpose of these experiments is to understand the basic principles for commonly used inter-component busses, and to be able to use these busses in practice.

The I2C bus is examined by interfacing the temperature sensor LM75, and by using one PSoC as an I2C master device.

The SPI bus is examined by interfacing two PSoC devices, one as Master and the other as Slave.

The experiments should end up with a small journal and the PSoC creator projects.

Literature

- Data sheet for LM75.
- PSoC Manuals.

The relevant documents can be downloaded from BlackBoard.

General guidelines

Document the experiments in a journal.

Describe the experiment objective(s), results and reflect upon the results.

Document the test setup with photos and diagrams.

Note which components you use. Which type of motor, which sensor etc.

Document the electrical wiring and create oscilloscope/logic analyzer dumps, where you find it appropriate.

Include relevant parts of datasheets or other documentation. The relevant parts are often diagrams and illustrations.

Keep a good structure in the code and document the code.

Perform the experiments in a structured manner: Think -> Do -> Document -> Reflect. And possibly iterate.

Conclude upon the results:

- What worked?
- What didn't work?
- Did anything surprise you?
- What caused the most problems?

I2C

In this part ("I2C"), we will use the circuit below:

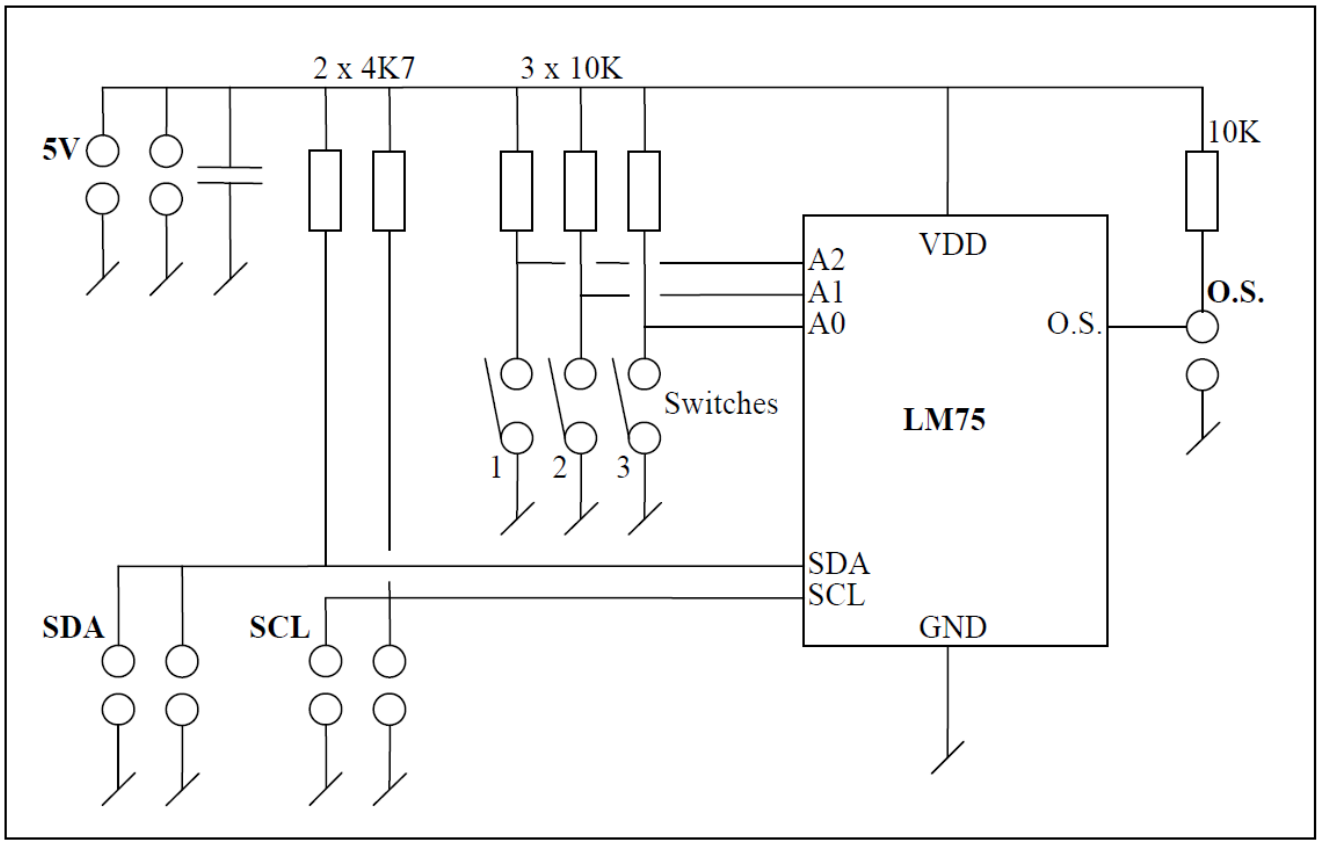


Figure 1 I2C Circuit

You will need two or more PCB's with the LM75 I2C slaves.

Connectors:

5 volt: Power supply (from PSoC) and for connecting power to the next slave unit (if any). Note that the LM75 supply voltage range is 3.0V to 5.0V.

SDA and SCL: The I2C bus (connect to the corresponding I2C port pins at PSoC) and to the next slave unit (if any).

O.S.: The alarm output from the LM75. It can be connected to an interrupt input at the PSoC if you want to experiment with the alarm feature of the LM75.

The 3 switches are used for setting up the LM75 local I2C address (A2-A0). Setting a switch ON, sets the corresponding address input LOW.

1. I2C Experiment: PSoC Master and LM75 Slave

Use the PSoC Creator to define a new empty project. Insert an I2C Master and a UART into the project.

Connect the SCL and SDA to some external pins on the PSoC.

Read the temperature from the LM75. To do this, you have to write a C program (see hints!) inside the PSoC Creator and use the I2C bus. You will need the PSoC I2C controller datasheet to find out how to use the I2C controller in the PSoC. Right-click the I2C component and select "Open Datasheet" to see the datasheet.

Implement an endless loop, where you read the temperature. You will need the LM75 datasheet (it is on Blackboard) to find out, how to communicate with the LM75.

Convert the bits received from LM75 to the real temperature value in Celsius and output the temperature on the UART, so you can see the output in a console on your PC.

Read the temperature from multiple LM75's on the same bus and output the temperatures to the UART.

Use an oscilloscope to examine the communication on the I2C bus:

- Where are the start and stop conditions for a single communication?
- How do you see which LM75 the PSoC communicates with?
- How do you see, if a data transfer is a read or write?'
- Where is the data, which is being transferred? In which direction does the data flow?
- Where are the ACK/NACK parts of the the communication?

You need to have a delay between multiple readings of the temperature from the same LM75. Why is that? How to you determine the required delay and what is the minimum delay?

Optional: You can experiment with the alarm features of the LM75.

Hints:

Read the LM75 datasheet and the I2C datasheet.

In a loop do (Pseudo code):

```
status = MasterSendStart(address, read)
if (status == NO_ERROR)
  {
  b1 = MasterReadByte with ACK
  b2 = MasterReadByte with NACK
  MasterSendStop

  Combine bytes to a temperature
  Send temperature to UART
  }
else {
  report error on UART
  MasterSendStop
}
```

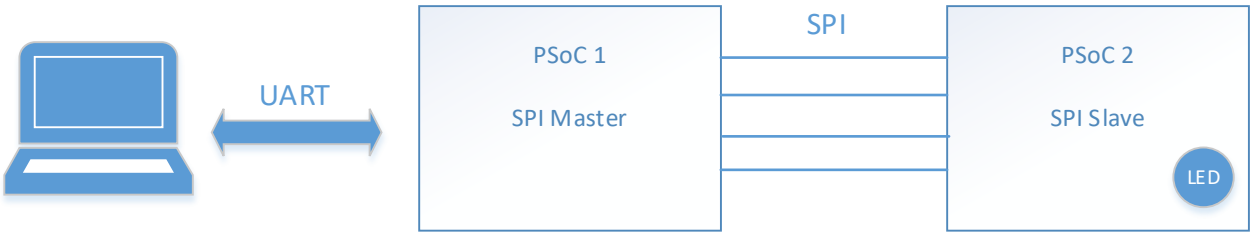
A delay can be implemented by the CyDelay() command, e.g `CyDelay(1000);` which gives a delay on 1000 ms.

2. SPI Experiment: PSoC Master and PSoC Slave

In this experiment, you will use two PSoC's: One shall be a SPI Master and the other shall be a SPI Slave.

Connect the master and the slave to each other.

Remember to make a common ground between the two boards (GND-GND)!



Write a program, where you can toggle the LED on the SPI Slave, by entering commands (i.e. pressing a key) in a console connected to the UART on the SPI Master.

Remember that SPI is full-duplex communication. You can't send data without receiving data and you can't receive data without sending data. So a good way to solve this exercise is to create a continuous stream of data (e.g. have the master do a data transfer at a given time interval) and give some of the data special meaning, i.e. that the LED shall toggle.

Extend the program, so the SPI Master **continuously** receives the state of the push button from the SPI Slave and outputs the state to its UART. The push button is connected to P2.2.

If you can toggle the Slave LED from the Master and read the button state, you have now invented a simple protocol ☺

Describe the protocol you just invented.

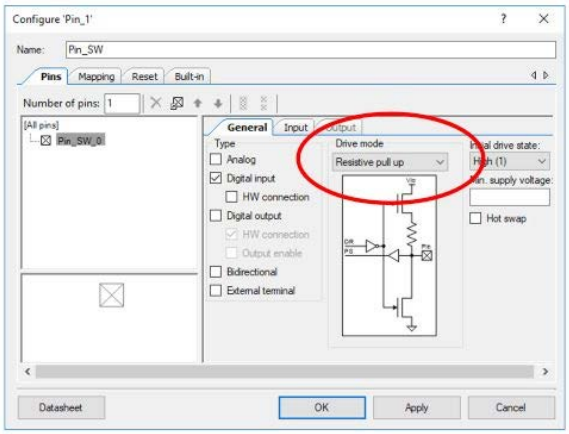
Use an oscilloscope to examine the communication on the SPI bus:

- How does the SPI communication work?
- Which SPI mode are you using? How can you see that on the oscilloscope?
- Explain the details of your protocol, using the oscilloscope plot(s).

Hints:

Read the SPI Master and SPI Slave datasheets. To access them, right-click on the component in PSoC Creator and choose: "Open Datasheet".

The push-button is just a switch, which connects to gnd. You have to configure it to be pulled up to 5V in the PSoC creator:



The SPI Slave has a 4 byte buffer. Read how it works on the next page and see the slides from the lecture for an animated example.

Connect an interrupt service routine (ISR) to the SPI RX interrupt, so you'll know when data is available. Remember to enable interrupt for "Rx FIFO Not Empty" (right-click on the SPI component and choose 'configure').

You can connect an ISR to the SPI Slave Tx interrupt, if you want to know when the slave has transmitted its data.

You may want to start by verifying, that the communication between Master and Slave works. You can do that, by having a variable with a number, you increment every time you send a byte to the SPI slave and send the number to the Slave (the byte you send will contain the number). You can do the same with the data you send form the slave to the master.

Variables holding a byte should be declared as `uint8_t` inside the PSoC Creator.

The xxx_`GetRxBufferSize()` method is used to read the number of bytes inside the RX buffer.

The xxx_`ReadRxData()` method is used to read a byte from the RX buffer.

The xxx_`WriteTxData()` method is used to write a byte to the TX buffer.

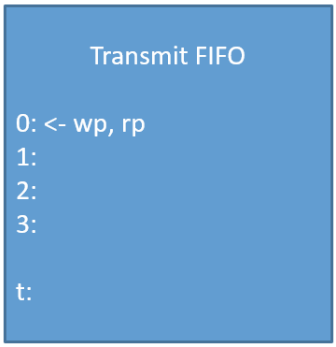
Note that the SPI Master controls the transactions by calling the xxx_`WriteTxData()` method.

(where xxx in the above methods should be replaced by the name you chose for the SPI component).

PSoC SPI Slave buffer

The PSoC 5LP has a 4 byte transmit-buffer (FIFO) implemented in hardware.

It also has a register, from which the actual transmission takes place.



Rules:

The buffer is circular.

Write pointer (wp) blocks writing if FIFO is full.

Read pointer (rp) will not go further than the write pointer.

Data is transfered from 't' register.

The time of transfer from rp to 't' depends on the CPHA mode.

CPHA 0:

Transfer from rp to t register is when a byte transfer is complete.

You can use the SPIS_WriteTxDataZero(byte) method to set the t register directly.

CPHA 1:

Transfer from **rp** to **t** register is before starting a byte transfer.

3. I2C Experiment: PSoC Master and PSoC Slave (Optional)

Use one PSoC as an I2C master, and use another PSoC as an I2C slave device. Connect the SCL and SDA between the PSoC devices. Make sure to handle the pull-up resistors!

Make experiments with read and write transactions between the master and slave devices.

Connect an oscilloscope to the SDA and SCL lines and verify the communication between the devices.