**Basheer Ammar’s PSoC 5LP SPI Implementation (with an addressing proof of concept)**

**Code**

Initial Setup

All of the code is done within the main of the PSoC. Due to the configurable nature of PSoCs and the number of pins, it is possible to have multiple masters and/or slaves on one board!

It should be noted that in my experience reliability issues occasionally arose from testing with multiple boards but only one slave select, and I found myself having to plug in the wires to connect AFTER connecting the boards to the computer, likely due to clock issues.

Before the for loop, you’ll want to initialize your masters/slaves with a statement like the following:

SPIM\_Start();

SPIS\_Start();

Within the for loop:

It’s a good idea to clear the Tx and Rx buffers at the start of each loop.

SPIM\_ClearTxBuffer(); //good practice

SPIS\_ClearTxBuffer();

SPIM\_ClearRxBuffer();

SPIS\_ClearRxBuffer();

Sending Messages

These will usually be hexadecimal messages.

Before sending a message from the master specifically, you’ll want to double check that no current processes are taking place. Your statement should look something like this:

uint8 temp = SPIM\_ReadTxStatus();

//Ensure that previous SPI read is done, or SPI is idle before sending data

if(temp & (SPIM\_STS\_SPI\_DONE | SPIM\_STS\_SPI\_IDLE))

{

SPIM\_WriteTxData(0x34u); //data request sent to slave 1 //no address

}

With “SPIM\_1\_WriteTxData();” being the specific function that sends the message. The slave has the equivalent function, and slave messages can be sent without a request by the master, but typically, you’d have a master send out a request and have a slave respond. The master can also respond to a slave’s message if desired!

For example:

if (SPIS\_ReadRxData() == 0x34u) //ideal slave send message //no addresses

{

//put whatever specific functions you want here in response to the request

SPIS\_WriteTxData(0xAAu); //response message to master

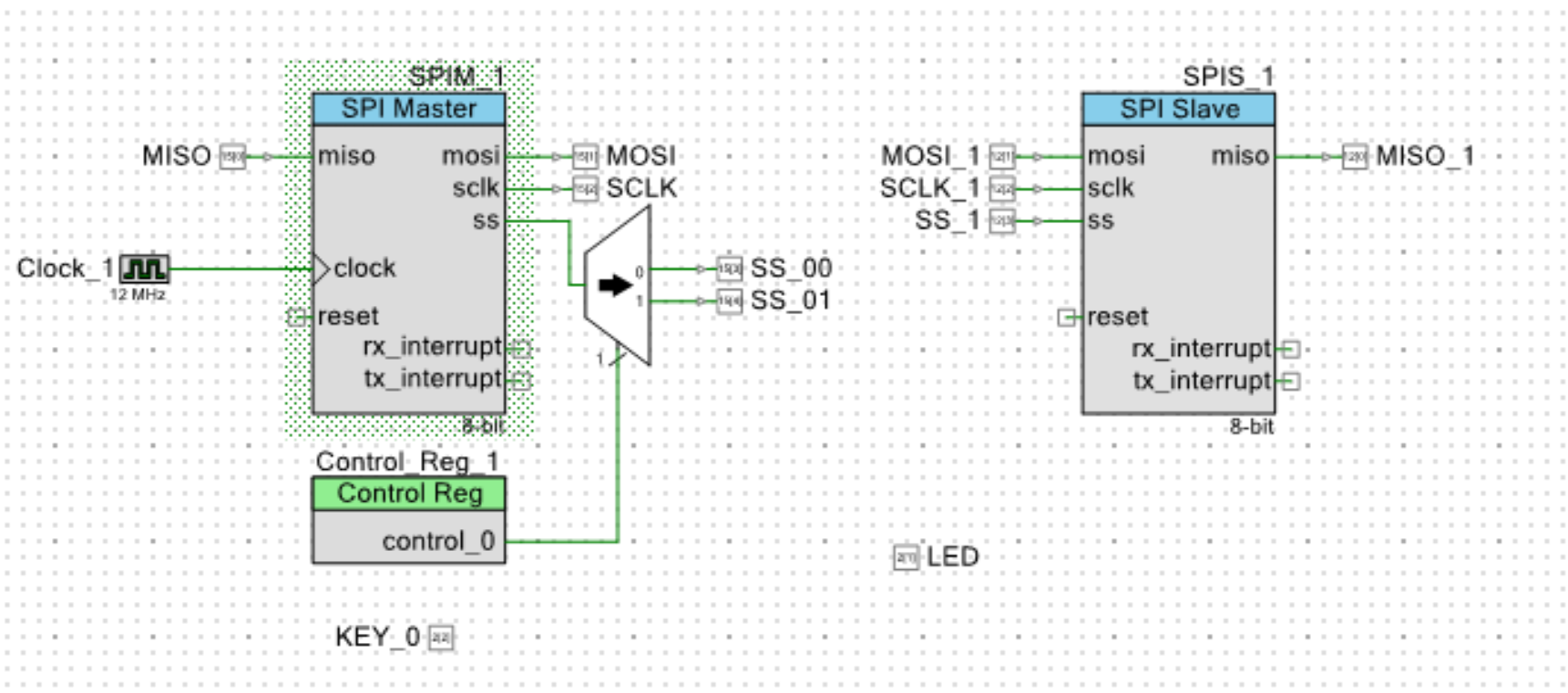
}

Then, if the master were to respond to that message…

if (SPIM\_ReadRxData() == 0xAAu) //if master receives the response...

LED\_Write(1); //it works!!!

Traditional Slave Select



Using a 1:2 Demultiplexer and a control register, you’re able to have two different slave selects to address two different slaves with one master (of course more can be added with more parts). Though this is the most reliable way for one master to communicate with multiple slaves, it requires more wiring and more work on the part of the programmer when the time comes to develop a more complex system.

In this system, if one were to enable a certain slave select, you would use the function “Control\_Reg\_1\_Write(X);”, with X being either 0 for the first slave select or 1 for the second slave select.

Slave Select via Addressing

The other way to do this, which allows a master to address one slave out of several all connected to the same slave select is to create a system with sent messages involving an addressing bit and a data bit. Though, like other decimals, this would typically be done with hexadecimals (though it would likely depend on the hardware that the PSoC is communicating with), I’ve made a proof of concept that is easy to understand which involves the use of two-digit decimal numbers.

For instance, in a system with one master and two slaves, let’s have the master send out a message like

SPIM\_1\_WriteTxData(21); //data request sent to slave 2

The first slave responds to an address of 1 and the second slave responds to an address of 2, so this would be a message directed to the second slave.

The check for the second slave (and the first slave) would involve “decoding” the message, and acquiring the address bit, represented by the leftmost (tens) digit, and the data bit, represented by the rightmost (ones) digit.

int address = SPIS\_2\_ReadRxData() / 10;

int data = SPIS\_2\_ReadRxData() % 10;

Once this check is done, the message response would look something like this:

if (address == 2) //the second slave’s preferred address

{

if (data == 1) //the message that was sent by the master

{

//put your response here...

}

}