

ME 461 Laboratory #5 (Three Week Lab)

SPI Serial Port and the MPU-9250 IMU

Goals

1. Write code to setup SPIB. We will initially just scope the SPI Pins to see that the transmission and the receive interrupts are working correctly.
2. Write code to communicate with the DAN28027 chip. Your code should send two PWM values and receive two 12-bit ADC conversions
3. Write code to communicate with the MPU-9250, reading three accelerometer readings and three gyro readings.

Exercise 1

Use your soldered board for this exercise since your board does not have the DAN28027 chip soldered to it. This exercise does not communicate with any chip / device, we just want to be able to scope the SPI outputs to get a better understanding of SPI. *If you would like to use the robot car you can, you will just be communicating incorrectly with the DAN28027, which is not a problem.*

The “DAN28027” chip has been soldered to your robot’s breakout board. Also, jumper wires have been soldered to connect DAN28027’s ADC1 pin to Joystick X and DAN28027’s ADC2 pin to Joystick Y.

In Exercise 2, scope DAN28027’s PWM1 and PWM2 pins at the connector in the left corner of the robot’s breakout board which has the labels “27_0” and “27_1.” Have your TA show you where these pins are on the robot.

For this exercise, I would like you to set up the SPI port for sending and receiving but you will not communicate with an actual chip. I would like you to set up the SPI and then transmit two bytes of data every 10ms in CPU timer 0’s interrupt. Since the SPI pins will not be connected to any chip, the transmit data is not doing anything, but it is allowing you to scope the four SPI pins and check that SPIB is set up correctly. Modify the following code after copying and pasting it into the specified locations.

1. Copy and paste this shell code in to your `main()` function below the `init_serialSCIA`, etc., function calls. Fill in the ??? with the correct value by reading the [SPI Condensed TechRef](#) and its register descriptions.

```
GPIO_SetupPinMux(9, GPIO_MUX_CPU1, 0); // Set as GPIO9 and used as DAN28027 SS
GPIO_SetupPinOptions(9, GPIO_OUTPUT, GPIO_PUSHPULL); // Make GPIO9 an Output Pin
GpioDataRegs.GPASET.bit.GPIO9 = 1; //Initially Set GPIO9/SS High so DAN28027 is not selected

GPIO_SetupPinMux(66, GPIO_MUX_CPU1, 0); // Set as GPIO66 and used as MPU-9250 SS
GPIO_SetupPinOptions(66, GPIO_OUTPUT, GPIO_PUSHPULL); // Make GPIO66 an Output Pin
GpioDataRegs.GPCSET.bit.GPIO66 = 1; //Initially Set GPIO66/SS High so MPU-9250 is not selected
```

```

GPIO_SetupPinMux(63, GPIO_MUX_CPU1, ???); //Set GPIO63 pin to SPISIMOB
GPIO_SetupPinMux(64, GPIO_MUX_CPU1, ???); //Set GPIO64 pin to SPISOMIB
GPIO_SetupPinMux(65, GPIO_MUX_CPU1, ???); //Set GPIO65 pin to SPICLK

EALLOW;
GpioCtrlRegs.GPBPUd.bit.GPIO63 = 0; // Enable Pull-ups on SPI PINS Recommended by TI for SPI Pins
GpioCtrlRegs.GPCPUd.bit.GPIO64 = 0;
GpioCtrlRegs.GPCPUd.bit.GPIO65 = 0;
GpioCtrlRegs.GPBQSEL2.bit.GPIO63 = 3; // Set I/O pin to asynchronous mode recommended for SPI
GpioCtrlRegs.GPCQSEL1.bit.GPIO64 = 3; // Set I/O pin to asynchronous mode recommended for SPI
GpioCtrlRegs.GPCQSEL1.bit.GPIO65 = 3; // Set I/O pin to asynchronous mode recommended for SPI
EDIS;

// -----
SpibRegs.SPICCR.bit.SPISWRESET = ???; // Put SPI in Reset

SpibRegs.SPICTL.bit.CLK_PHASE = 1; //This happens to be the mode for both the DAN28027 and
SpibRegs.SPICCR.bit.CLKPOLARITY = 0; //The MPU-9250, Mode 01.
SpibRegs.SPICTL.bit.MASTER_SLAVE = ???; // Set to SPI Master
SpibRegs.SPICCR.bit.SPICHR = ???; // Set to transmit and receive 16-bits each write to SPITXBUF
SpibRegs.SPICTL.bit.TALK = ???; // Enable transmission
SpibRegs.SPIPRI.bit.FREE = 1; // Free run, continue SPI operation
SpibRegs.SPICTL.bit.SPIINTENA = ???; // Disables the SPI interrupt

SpibRegs.SPIBRR.bit.SPI_BIT_RATE = ???; // Set SCLK bit rate to 1 MHz so 1us period. SPI base clock is
// 50MHZ. And this setting divides that base clock to create SCLK's period
SpibRegs.SPISTS.all = 0x0000; // Clear status flags just in case they are set for some reason

SpibRegs.SPIFFTX.bit.SPIRST = ???; // Pull SPI FIFO out of reset, SPI FIFO can resume transmit or receive.
SpibRegs.SPIFFTX.bit.SPIFFENA = ???; // Enable SPI FIFO enhancements
SpibRegs.SPIFFTX.bit.TXFIFO = 0; // Write 0 to reset the FIFO pointer to zero, and hold in reset
SpibRegs.SPIFFTX.bit.TXFFINTCLR = 1; // Write 1 to clear SPIFFTX[TXFFINT] flag just in case it is set

SpibRegs.SPIFFRX.bit.RXFIFORESET = 0; // Write 0 to reset the FIFO pointer to zero, and hold in reset
SpibRegs.SPIFFRX.bit.RXFFOVFLR = 1; // Write 1 to clear SPIFFRX[RXFFOVFLR] just in case it is set
SpibRegs.SPIFFRX.bit.RXFFINTCLR = ???; // Write 1 to clear SPIFFRX[RXFFINT] flag just in case it is set
SpibRegs.SPIFFRX.bit.RXFFIENA = ???; // Enable the RX FIFO Interrupt. RXFFST >= RXFFIL

SpibRegs.SPIFFCT.bit.TXDLY = ???; //Set delay between transmits to 16 spi clocks. Needed by DAN28027
chip

SpibRegs.SPICCR.bit.SPISWRESET = ???; // Pull the SPI out of reset

SpibRegs.SPIFFTX.bit.TXFIFO = ???; // Release transmit FIFO from reset.
SpibRegs.SPIFFRX.bit.RXFIFORESET = 1; // Re-enable receive FIFO operation
SpibRegs.SPICTL.bit.SPIINTENA = 1; // Enables SPI interrupt. !! I don't think this is needed. Need to
Test

SpibRegs.SPIFFRX.bit.RXFFIL = ???; //Interrupt Level to 16 words or more received into FIFO causes
interrupt. This is just the initial setting for the register. Will be changed below

```

2. Set up CPU Timer 0's interrupt function to be called every 10ms. This is the default so you may not need to change anything. Inside CPU Timer 0's interrupt function, call these three lines of code to tell the SPI to transmit two 16-bit values over the SPI. Because this is a SPI serial port, two 16-bit values will be received. Whenever you transmit data in a SPI serial port, you also receive. Once two 16-bit values are received into the FIFO the `SPIB_RX_INT` hardware interrupt function will be called.

```

Clear GPIO9 Low to act as a Slave Select. Right now, just to scope. Later to select DAN28027 chip
GpioDataRegs.???? = ???;
SpibRegs.SPIFFRX.bit.RXFFIL = 2; // Issue the SPIB_RX_INT when two values are in the RX FIFO
SpibRegs.SPITXBUF = 0x4A3B; // 0x4A3B and 0xB517 have no special meaning. Wanted to send
SpibRegs.SPITXBUF = 0xB517; // something so you can see the pattern on the Oscilloscope

```

3. Set up the SPIB interrupt service routine:

- At the top of your C file add a predefinition of `__interrupt void SPIB_isr(void)`.
- Then add this function to the PieVectTable like you did in Lab 4 for the ADCD, ADCB, and ADCA interrupt service routines.
- Look up in the [PIE Channel Mapping Table](#) the interrupt number for SPIB_RX and implement this interrupt by enabling both the major interrupt by typing `IER |= M_INT?` and the PIE group and channel assignment using `PIEIER?.bit.???`.
- Finally, insert the `SPIB_isr(void)` function, below, and correct the ???

```
int16_t spival1 = 0;
int16_t spival2 = 0;
__interrupt void SPIB_isr(void) {
    spival1 = SpibRegs.???; // Read first 16-bit value off RX FIFO. Probably is zero since no chip
    spival2 = SpibRegs.???; // Read second 16-bit value off RX FIFO. Again probably zero
    GpioDataRegs.??? = ???; // Set GPIO9 high to end Slave Select. Now Scope. Later to deselect
    DAN28027
    // Later when actually communicating with the DAN28027 do something with the data. Now do nothing.

    SpibRegs.SPIFFRX.bit.RXFFOVFLR = 1; // Clear Overflow flag just in case of an overflow
    SpibRegs.SPIFFRX.bit.RXFFINTCLR = 1; // Clear RX FIFO Interrupt flag so next interrupt will happen

    PieCtrlRegs.PIEACK.all = PIEACK_GROUP6; // Acknowledge INT6 PIE interrupt
}
```

- Compile and run this code. Use the logic analyzer channels of the oscilloscope to scope SS, SCLK, MOSI and MISO. Trigger on SS. Verify that the correct clock rate and clock mode is being used. Verify that 0x4A3B and 0xB517 are being transmitted and probably zero is being received.

Exercise 2

Now use the robot car, which has the DAN28027 chip already soldered. For this exercise, I would like you to use the [DAN28027 datasheet](#) to figure out how to communicate with the DAN28027 over the SPIB serial port. All the setups you performed in Exercise 1 are correct for the DAN28027 chip, so you have most of the initialization code developed at this point. For example, the DAN28027 communicates using SPI Mode 1 (01) and that is what you set in Exercise 1. Also, you need to communicate 16 bits at a time to the DAN28027. You will be sending three 16-bit values and receiving three 16-bit values for one communication with the DAN28027 chip.

Modify your Exercise 1 code so that every 20 milliseconds it communicates two PWM values to the DAN28027 chip and receives the two ADC values from the DAN28027. Remember that when the SPI is transmitting it is also receiving just as the DAN28027 datasheet specifies. For the PWM values you send, note from the datasheet that the command value is from 0 to 3000. So just as you did in Lab 3, when gradually increasing and then decreasing the LED brightness, every 20 milliseconds increment the PWM commands by 10 until they reach 3000 and then start decrementing by 10 until they reach 0

and then repeat the pattern. To see if these values are sent correctly to the DAN28027 chip, use more channels of the logic analyzer to scope the DAN28027's PWM1 and PWM2 pins. You should see the PWM duty cycle gradually getting larger and then smaller. You can scope PWM1 and PWM2 at the bottom left corner of your green board at the connector that is labelled 27_0 and 27_1.

For the two ADC readings that the DAN28027 sends over SPI, print their value in units of volts to Tera Term every 100 milliseconds. Note that these ADC channels are 12-bit and have a range of 0V to 3.3V. So, 0 equates to 0V and 4095 equates to 3.3V. To test that the ADC readings are correct, move your joystick and see that the voltages change.

Pointers to think about when developing this code:

- Figure out how many 16-bit values to write to the FIFO for one transmission to the DAN28027. As long as this number of values is less than 16 you can write all the values one after the other to the FIFO and have the FIFO take care of sending each value one at a time across the SPI serial port.
- Don't forget to set the `SpibRegs.SPIFFRX.bit.RXFFIL` (Receive FIFO interrupt level) to the number of 16-bit values you write to the TX FIFO every 20 milliseconds. Remember, the number of words you write to the TX FIFO will be the number of words you receive in the RX FIFO and therefore cause an interrupt when all the values have been received.
- Study the timing diagram of the DAN28027 datasheet and ask questions.
- Don't forget that some of the data you receive may not be a part of an ADC reading and should not be used.

Exercise 3

For this exercise you are going to initialize the MPU-9250 and then read its three accelerometer readings and its three gyro readings every 1 ms. Use the [MPU-9250 Datasheet](#), [MPU-9250 Register Reference](#) and especially my [MPU-9250 SPI Programming Tips](#) for the explanation on how to fill in the needed code below.

First finish the `setupSpib()` function given below. Much of the code is given to you but you will need to add to this function the parts described. Copy this function to the bottom of your project's C file and create a predefinition of the function at the top of your C file. Then make sure to call `setupSpib()` in `main()` after the `init_serialSCIA()`, etc. statements and before interrupts are enabled.

```
void setupSpib(void) //Call this function in main() somewhere after the DINT; line of code.
{
    int16_t temp = 0;
    Step 1.
```

```

// cut and paste here all the SpibRegs initializations you found for part 3. Make sure the TXdelay in
between each transfer to 0. Also don't forget to cut and paste the GPIO settings for GPIO9, 63, 64, 65,
66 which are also a part of the SPIB setup.

```

```

SpibRegs.SPICCR.bit.SPICHAR = 0xF;
SpibRegs.SPIFFCT.bit.TXDLY = 0x00;

```

```

//-----

```

Step 2.

```

// perform a multiple 16-bit transfer to initialize MPU-9250 registers 0x13,0x14,0x15,0x16
// 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C 0x1D, 0x1E, 0x1F. Use only one SS low to high for all these writes
// some code is given, most you have to fill you yourself.
GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1; // Slave Select Low

```

```

// Perform the number of needed writes to SPITXBUF to write to all 13 registers. Remember we are sending
16-bit transfers, so two registers at a time after the first 16-bit transfer.

```

```

// To address 00x13 write 0x00
// To address 00x14 write 0x00
// To address 00x15 write 0x00
// To address 00x16 write 0x00
// To address 00x17 write 0x00
// To address 00x18 write 0x00
// To address 00x19 write 0x13
// To address 00x1A write 0x02
// To address 00x1B write 0x00
// To address 00x1C write 0x08
// To address 00x1D write 0x06
// To address 00x1E write 0x00
// To address 00x1F write 0x00

```

```

// wait for the correct number of 16-bit values to be received into the RX FIFO

```

```

while(SpibRegs.SPIFFRX.bit.RXFFST !=???);
GpioDataRegs.GPCSET.bit.GPIO66 = 1; // Slave Select High
temp = SpibRegs.SPIRXBUF;
// ??? read the additional number of garbage receive values off the RX FIFO to clear out the RX FIFO
DELAY_US(10); // Delay 10us to allow time for the MPU-2950 to get ready for next transfer.

```

Step 3.

```

// perform a multiple 16-bit transfer to initialize MPU-9250 registers 0x23,0x24,0x25,0x26
// 0x27, 0x28, 0x29. Use only one SS low to high for all these writes
// some code is given, most you have to fill you yourself.
GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1; // Slave Select Low

```

```

// Perform the number of needed writes to SPITXBUF to write to all 7 registers

```

```

// To address 00x23 write 0x00
// To address 00x24 write 0x40
// To address 00x25 write 0x8C
// To address 00x26 write 0x02
// To address 00x27 write 0x88
// To address 00x28 write 0x0C
// To address 00x29 write 0x0A

```

```

// wait for the correct number of 16-bit values to be received into the RX FIFO

```

```

while(SpibRegs.SPIFFRX.bit.RXFFST !=???);
GpioDataRegs.GPCSET.bit.GPIO66 = 1; // Slave Select High
temp = SpibRegs.SPIRXBUF;
// ??? read the additional number of garbage receive values off the RX FIFO to clear out the RX FIFO
DELAY_US(10); // Delay 10us to allow time for the MPU-2950 to get ready for next transfer.

```

Step 4.

```

// perform a single 16-bit transfer to initialize MPU-9250 register 0x2A

```

```

GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
// Write to address 0x2A the value 0x81

```

```

// wait for one byte to be received

```

```

while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
GpioDataRegs.GPCSET.bit.GPIO66 = 1;
temp = SpibRegs.SPIRXBUF;
DELAY_US(10);

```

```

// The Remainder of this code is given to you and you do not need to make any changes.

```

```

GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
SpibRegs.SPITXBUF = (0x3800 | 0x0001); // 0x3800

```

```

while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
GpioDataRegs.GPCSET.bit.GPIO66 = 1;
temp = SpibRegs.SPIRXBUF;
DELAY_US(10);
GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
SpibRegs.SPITXBUF = (0x3A00 | 0x0001); // 0x3A00
while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
GpioDataRegs.GPCSET.bit.GPIO66 = 1;
temp = SpibRegs.SPIRXBUF;
DELAY_US(10);
GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
SpibRegs.SPITXBUF = (0x6400 | 0x0001); // 0x6400
while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
GpioDataRegs.GPCSET.bit.GPIO66 = 1;
temp = SpibRegs.SPIRXBUF;
DELAY_US(10);
GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
SpibRegs.SPITXBUF = (0x6700 | 0x0003); // 0x6700
while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
GpioDataRegs.GPCSET.bit.GPIO66 = 1;
temp = SpibRegs.SPIRXBUF;
DELAY_US(10);
GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
SpibRegs.SPITXBUF = (0x6A00 | 0x0020); // 0x6A00
while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
GpioDataRegs.GPCSET.bit.GPIO66 = 1;
temp = SpibRegs.SPIRXBUF;
DELAY_US(10);
GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
SpibRegs.SPITXBUF = (0x6B00 | 0x0001); // 0x6B00
while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
GpioDataRegs.GPCSET.bit.GPIO66 = 1;
temp = SpibRegs.SPIRXBUF;
DELAY_US(10);
GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
SpibRegs.SPITXBUF = (0x7500 | 0x0071); // 0x7500
while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
GpioDataRegs.GPCSET.bit.GPIO66 = 1;
temp = SpibRegs.SPIRXBUF;
DELAY_US(10);
GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
SpibRegs.SPITXBUF = (0x7700 | 0x00EB); // 0x7700
while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
GpioDataRegs.GPCSET.bit.GPIO66 = 1;
temp = SpibRegs.SPIRXBUF;
DELAY_US(10);
GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
SpibRegs.SPITXBUF = (0x7800 | 0x0012); // 0x7800
while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
GpioDataRegs.GPCSET.bit.GPIO66 = 1;
temp = SpibRegs.SPIRXBUF;
DELAY_US(10);
GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
SpibRegs.SPITXBUF = (0x7A00 | 0x0010); // 0x7A00
while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
GpioDataRegs.GPCSET.bit.GPIO66 = 1;
temp = SpibRegs.SPIRXBUF;
DELAY_US(10);
GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
SpibRegs.SPITXBUF = (0x7B00 | 0x00FA); // 0x7B00
while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
GpioDataRegs.GPCSET.bit.GPIO66 = 1;
temp = SpibRegs.SPIRXBUF;
DELAY_US(10);
GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
SpibRegs.SPITXBUF = (0x7D00 | 0x0021); // 0x7D00
while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
GpioDataRegs.GPCSET.bit.GPIO66 = 1;
temp = SpibRegs.SPIRXBUF;
DELAY_US(10);
GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
SpibRegs.SPITXBUF = (0x7E00 | 0x0050); // 0x7E00
while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
GpioDataRegs.GPCSET.bit.GPIO66 = 1;

```

```

temp = SpibRegs.SPIRXBUF;
DELAY_US(50);

// Clear SPIB interrupt source just in case it was issued due to any of the above initializations.
SpibRegs.SPIFFRX.bit.RXFFOVFLR=1; // Clear Overflow flag
SpibRegs.SPIFFRX.bit.RXFFINTCLR=1; // Clear Interrupt flag
PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;
}

```

Answer to your TA. In the above initialization of the MPU-9250 you were given the values to write to certain registers. I would like you to read the [Register Map document](#) and explain how these following register assignments setup the MPU-9250. Setting CONFIG (address 0x1A) to 0x2, GYRO_CONFIG (0x1B) to 0x0, ACCEL_CONFIG (0x1C) to 0x8 and ACCEL_CONFIG2 (0x1D) to 0x6.

Use your DAN28027 code as a guide to complete these next steps along with studying my [MPU-9250 SPI Programming Tips](#).

Now every 1ms inside your CPU Timer 0 interrupt function, transmit the correct 16-bit values and correct number of 16-bit values to the MPU-9250 so that it will transmit back the three accelerometer readings and the three gyro readings. *I am going to leave reading and processing the magnetometer readings as a possible final project for the class, so we will not worry about them for this lab.* Make sure to set `SpibRegs.SPIFFRX.bit.RXFFIL` to the correct value so that the SPIB interrupt function will be called when the SPI transmission from the master to the slave and also from the slave to the master is complete. Remember these transmissions happen at the same time.

Inside the SPIB interrupt function, make sure to pull Slave Select high. Then, read the three accelerometer integer readings and the three gyro integer readings. *To read these three 16-bit accelerometer readings and the three 16-bit gyro readings in one chip select cycle, you should notice that you also have to read the 16-bit temperature reading which falls inbetween.* Scale the accelerometer readings to units of g. Remember the initialization chose the range of -4g to 4g for the accelerometers. Also, scale the gyro readings to units of degrees/second. The initialization chose the range of -250 to 250 degrees per second. Print these six sensor readings to Tera Term every 200ms. **Demo to your TA.** With your IMU readings, you will probably see that the resting values of the accelerometer are not at zero and possibly saturated at 4g or -4g. Every once in a while we find that the accelerometer axis that is saturated is broken. But most of the time the initial offset value needs to be adjusted to get the accelerometer axis out of saturation and operational. Look at the register map for XA_OFFS, YA_OFFS and ZA_OFFS and notice that these are 15-bit offsets that allow for an integer offset between -16384 and 16383. As a first exercise show your TA that the default offsets set to the IMU in the given `setupSpib` function are XA offset equal to -2679, YA offset equal to 2173, and ZA offset equal to 4264. Adjust these offsets so that the resting value of each accelerometer axis is somewhat close to zero. The [MPU-9250 Register Reference](#) on pages 44-46 specifies the resolution of this offset number to be .00098g steps. Note that this offset is a 15 bit signed integer so (-16384 to 16383) or (-16g to 16g).

Demo to your TA.

Lab Checklist

1. Demonstrate your Exercise 1 code working that sends and receives two bytes but only used to scope SPI signals.
2. Demonstrate your Exercise 2 code communicating with the DAN28027 chip.
3. Demonstrate your Exercise 3 code communicating with the MPU9250 and printing accelerometer and gyro readings. Make sure you have adjusted the accelerometer's offset so that the resting value of all accelerometers is pretty close to zero.
4. Submit all your written code after adding comments explaining your code and what you learned. Also be clear what code is for what exercise.