ME461 Semester Project #3
I2C Serial Port, Interfacing the DAN28027 and the BQ32000
Due November 3rd 2:00PM

Make sure to read this entire document before getting started so you know all the parts of the project and can plan accordingly. Also if you would like, I have copied to the class Box folder four recorded lectures that I made Spring 2021 for my SE423 class. You can find them in the "I2CLectures_SE423_Spring21" folder. I will be lecturing on the I2C serial port this semester but you may find these additional lectures helpful.

Before we get started working with I2C, there is an additional component of this project, and that is to have you 3D print the parts you will need for Lab 6 and Lab 7. I have copied the .stl files for these parts to the course Box folder names "3DPrintParts". You need to print two of SegbotWheel_spoke.stl, one left and one right kick stand stand_large_wheel_left.stl, stand_large_wheel_right.stl and one of flatmount.stl. I would like you to try to 3D print these parts outside of the lab. But if you run into trouble talk to me and we can print with the lab's two 3D printers.

The goal of the rest of this project is to teach you how to use the I2C serial port to communicate with external sensors and other peripherals. You will be programming the F28379D to use its I2C serial port to communicate with the "DAN28027" chip and the BQ32000 Real-Time Clock chip. In this project, you will learn more how to use the I2C serial port compared to learning how to setup the registers of the F28379D's I2C peripheral like many of the labs have done for other peripherals. Below you are given the initialization code for the F28379D's I2C peripheral and some example functions using the I2C serial port. Following these examples, you will create "write" and "read" functions for both the DAN28027 and BQ32000.

Comparing SPI to I2C there are a number of differences you need to keep in mind when programming I2C. The biggest difference is that I2C has only one data line so it can be set to read data for a certain number of bytes or set to write a certain number of bytes. This makes I2C slower along with its maximum recommended bit rate of 400000 bits/second where SPI can go 20Mbits/second or higher. Additionally I2C does not use a chip select (or slave select) pin, but instead transmits a slave address to select the slave device. All this is done with the three wires, SCL, SDA and GND connecting the I2C master (F28379D) with many possible I2C slave devices.

Because the I2C serial port is usually used for slower devices, we will not be using the I2C interrupts. Instead, you will be "polling" the status of the I2C serial port to know when it is ready to send more data or has received new data. If you would like to learn more about the I2C serial port and use its interrupt capabilities, talk to me about using I2C in your class final project.

After studying the given examples and the datasheet of the DAN28027 and BG32000, create separate functions for reading and writing the DAN28027 and BQ32000. Because these are "polling" functions they will need to be called inside main()'s while(1) loop.

```
uint16_t WriteDAN28027RCServo(uint16_t RC1, uint16_t RC2)

uint16_t ReadDAN28027ADC(uint16_t  *ADC1, uint16_t *ADC2)

uint16_t WriteBQ32000(uint16_t second,uint16_t minute,uint16_t hour,uint16_t day,uint16_t date,uint16_t month,uint16_t year)

uint16_t ReadBQ32000(uint16_t *second,uint16_t *minute,uint16_t *hour,uint16_t *day,uint16_t *date,uint16_t *month,uint16_t *year)
```

The TMS320F28379D I2C users guide can be found at

http://coecsl.ece.illinois.edu/ME461/labs/I2CCondensed_TechRef.pdf

The DAN28027 Datasheet can be found at

http://coecsl.ece.illinois.edu/ME461/labs/DAN28027_I2C_Datasheet.pdf

The BQ32000 Datasheet with some ME461 additions can be found at

http://coecsl.ece.illinois.edu/ME461/labs/bq32000Condensed.pdf

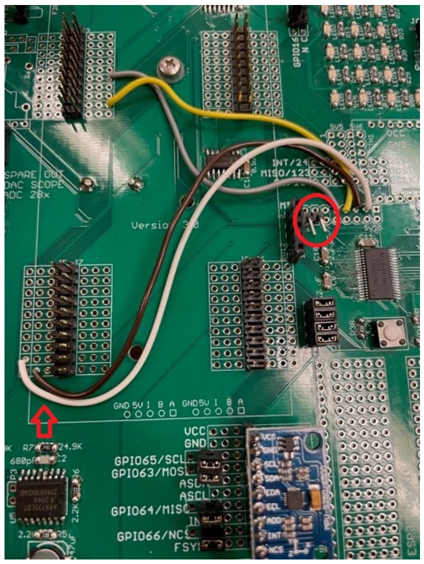
Items you need to solder.

You will need to solder two wires and a two pin header to your green board connecting SDA and SCL to the DAN28027 chip.

The DAN28027 SDA pin is brought out at pin 5 of SV15. Connect it to GPIO40 which is F28379D Launchpad pin 50.

The DAN28027 SCL pin is brought out at pin 3 of SV15. Connect it to GPIO41 which is F28379D Launchpad pin 49.

Solder a "two pin" header to pins 14 and 16 of SV15. You will connect these two header pins to two digital inputs of the oscilloscope to see the RCServo PWM signal changing.



<u>I2C initialization function</u>, cut and paste into your C file and call it after the init_serial() function in main() and before the while(1) continuous loop (Not inside the loop).

```
void I2CB_Init(void)
{
   EALLOW;
   /* Enable internal pull-up for the selected I2C pins */
   GpioCtrlRegs.GPBPUD.bit.GPIO40 = 1;
   GpioCtrlRegs.GPBPUD.bit.GPIO41 = 1;

   /* Set qualification for the selected I2C pins */
   GpioCtrlRegs.GPBQSEL1.bit.GPIO40 = 3;
   GpioCtrlRegs.GPBQSEL1.bit.GPIO41 = 3;

   /* Configure which of the possible GPIO pins will be I2C_B pins using GPIO regs*/
   GpioCtrlRegs.GPBGMUX1.bit.GPIO40 = 1;
   GpioCtrlRegs.GPBMUX1.bit.GPIO40 = 2;

   GpioCtrlRegs.GPBGMUX1.bit.GPIO41 = 1;
   GpioCtrlRegs.GPBMUX1.bit.GPIO41 = 2;
   EDIS;

  // Initialize I2C
  I2cbRegs.I2CMDR.bit.IRS = 0;

  // 200MHz / 20 = 10MHz
  I2cbRegs.I2CPSC.all = 19;

  // 10MHz/40 = 250KHz
  I2cbRegs.I2CCLKL = 15;  //psc > 2 so d = 5  See Usersguide
  I2cbRegs.I2CCLKH = 15;  //psc > 2 so d = 5  See Usersguide

  I2cbRegs.I2CIER.all = 0x00;

  I2cbRegs.I2CMDR.bit.IRS = 1;
  DELAY_US(5000);
}
```

Example of an I2C write function. This is for a fictitious chip named CHIPXYZ. It has 16 registers inside the chip. Here the function is writing to registers 4, 5, 6 and 7. Registers 4 and 5 are each 8bits and make up a 16bit command to CHIPXYZ. Register 4 is the LSB (least significant byte) and Register 5 is the MSB (most significant byte). In the same way registers 6 and 7 make up another command to CHIPXYZ and register 6 is LSB and register 7 is MSB. CHIPXYZ's slave address is **0x25**.

```
//Write 2 16bit commands (LSB then MSB) to I2C Slave CHIPXYZ starting at CHIPXYZ's register 4,
uint16_t WriteTwo16BitValuesToCHIPXYZ(uint16_t Cmd16bit_1, uint16_t Cmd16bit_2) {
    uint16_t Cmd1LSB = 0;
    uint16_t Cmd1MSB = 0;
    uint16_t Cmd2LSB = 0;
    uint16_t Cmd2MSB = 0;
    Cmd1LSB = Cmd16bit_1 & 0xFF;  //Bottom 8 bits of command
    Cmd1MSB = (Cmd16bit_1 >> 8) & 0xFF; //Top 8 bits of command
    Cmd2LSB = Cmd16bit_2 & 0xFF;  //Bottom 8 bits of command
    Cmd2MSB = (Cmd16bit_2 >> 8) & 0xFF; //Top 8 bits of command

    DELAY_US(200);  // Allow time for I2C to finish up previous commands.  It pains me to have this
                    // delay here but I have not had time to figure out what status bit to poll on to
                    // to check if the I2C peripheral is ready of the next command.  I have tried the busy
                    // bit but that gave me some issues especially at startup.  This would be a great
                    // choice for a part of your final project if you would like to learn more about I2C.

    if (I2cbRegs.I2CSTR.bit.BB == 1) {  // Check if I2C busy, if it is better
        return 2;      // to Exit and try again next sample
    }                  // This should not happen too often
    while(!I2cbRegs.I2CSTR.bit.XRDY);  //Poll until I2C ready to Transmit

    I2cbRegs.I2CSAR.all = 0x25;  // I2C address of ChipXYZ
    I2cbRegs.I2CCNT      =  5;  //Num Values plus Start Register 4 + 1
    I2cbRegs.I2CDXR.all  =  4;  // First need to transfer the Register value
                               // to start writing data
    // I2C in master mode (MST), I2C is in transmit mode (TRX) with start and stop
    I2cbRegs.I2CMDR.all  =  0x6E20;
    while(!I2cbRegs.I2CSTR.bit.XRDY);  //Poll until I2C ready to Transmit

    I2cbRegs.I2CDXR.all = Cmd1LSB;  // Write Command 1 LSB
    if (I2cbRegs.I2CSTR.bit.NACK == 1){  // Check for No Acknowledgement
        return 3;          // This should not happen
    }
    while(!I2cbRegs.I2CSTR.bit.XRDY);  //Poll until I2C ready to Transmit

    I2cbRegs.I2CDXR.all = Cmd1MSB;   // Write Command 1 MSB
    if (I2cbRegs.I2CSTR.bit.NACK == 1){  // Check for No Acknowledgement
        return 3;          // This should not happen
    }
    while(!I2cbRegs.I2CSTR.bit.XRDY);  //Poll until I2C ready to Transmit
```

```
   I2cbRegs.I2CDXR.all = Cmd2LSB;   // Write Command 2 LSB
   if (I2cbRegs.I2CSTR.bit.NACK == 1){  // Check for No Acknowledgement
      return 3;          // This should not happen
   }
   while(!I2cbRegs.I2CSTR.bit.XRDY);  //Poll until I2C ready to Transmit

   I2cbRegs.I2CDXR.all = Cmd2MSB;   // Write Command 2 MSB
                              // After this write since I2CCNT = 0
                              // A Stop condition will be issued
   if (I2cbRegs.I2CSTR.bit.NACK == 1){
      return 3;
   }
   return 0;
}
```

<u>Example of an I2C read function.</u> This is for a fictitious chip named CHIPXYZ. It has 16 registers inside the chip. Here the function is reading registers 10, 11, 12 and 13. Registers 10 and 11 are each 8bits and make up a 16bit sensor reading from CHIPXYZ. Register 10 is the LSB (least significant byte) and Register 11 is the MSB (most significant byte). In the same way registers 12 and 13 make up another 16bit sensor reading from CHIPXYZ and register 12 is LSB and register 13 is MSB. CHIPXYZ's slave address is **0x25**. Notice how the function first writes the start register with one I2C start condition and then re-issues a second start condition to switch the I2C communication to a read command.

```
//Read Two 16 Bit values from I2C Slave CHIPXYZ starting at CHIPXYZ's register 10
//Notice the Rvalue1 and Rvalue2 passed as pointers (passed by reference)
//So pass address of uint16_t variable when using this function for example:
// uint16_t Rval1 = 0;
// uint16_t Rval2 = 0;
// err = ReadTwo8BitValuesFromCHIPXYZ(&Rval1,&Rval2);
// This allows Rval1 and Rval2 to be changed inside the function and return
// the values read inside the function.
uint16_t ReadTwo16BitValuesFromCHIPXYZ(uint16_t *Rvalue1,uint16_t *Rvalue2) {

   uint16_t Val1LSB = 0;
   uint16_t Val1MSB = 0;
   uint16_t Val2LSB = 0;
   uint16_t Val2MSB = 0;

   DELAY_US(200);  // Allow time for I2C to finish up previous commands.  It pains me to have this
                   // delay here but I have not had time to figure out what status bit to poll on to
                   // to check if the I2C peripheral is ready of the next command.  I have tried the busy
                   // bit but that gave me some issues especially at startup.  This would be a great
                   // choice for a part of your final project if you would like to learn more about I2C.
```

```
if (I2cbRegs.I2CSTR.bit.BB == 1) {
   return 2;
}
while(!I2cbRegs.I2CSTR.bit.XRDY);  //Poll until I2C ready to Transmit

I2cbRegs.I2CSAR.all = 0x25;  // I2C address of ChipXYZ

I2cbRegs.I2CCNT      = 1;  //  Just Sending Address to start reading from

I2cbRegs.I2CDXR.all  = 10;  // Start Reading at this Register location

// I2C in master mode (MST), I2C is in transmit mode (TRX) with start
I2cbRegs.I2CMDR.all  =  0x6620;

if (I2cbRegs.I2CSTR.bit.NACK == 1) {
   return 3;
}
while(!I2cbRegs.I2CSTR.bit.XRDY);  //Poll until I2C ready to Transmit

I2cbRegs.I2CSAR.all = 0x25;  // I2C address of ChipXYZ

I2cbRegs.I2CCNT      = 4;

// I2C in master mode (MST), TRX=0, receive mode start stop
I2cbRegs.I2CMDR.all   = 0x6C20;  // Reissuing another Start with Read

if (I2cbRegs.I2CSTR.bit.NACK == 1) {
   return 3;
}
while(!I2cbRegs.I2CSTR.bit.RRDY);  //Poll until I2C has Recieved 8 bit value

Val1LSB = I2cbRegs.I2CDRR.all;  // Read CHIPXYZ
if (I2cbRegs.I2CSTR.bit.NACK == 1) {
   return 3;
}
while(!I2cbRegs.I2CSTR.bit.RRDY);  //Poll until I2C has Recieved 8 bit value

Val1MSB = I2cbRegs.I2CDRR.all;  // Read CHIPXYZ
if (I2cbRegs.I2CSTR.bit.NACK == 1) {
   return 3;
}
while(!I2cbRegs.I2CSTR.bit.RRDY);  //Poll until I2C has Recieved 8 bit value

Val2LSB = I2cbRegs.I2CDRR.all;  // Read CHIPXYZ
if (I2cbRegs.I2CSTR.bit.NACK == 1) {
   return 3;
}
while(!I2cbRegs.I2CSTR.bit.RRDY);  //Poll until I2C has Recieved 8 bit value
```

```
  Val2MSB = I2cbRegs.I2CDRR.all;  // Read CHIPXYZ
  if (I2cbRegs.I2CSTR.bit.NACK == 1) {
     return 3;
  }
                        // After this read since I2CCNT = 0
                        // A Stop condition will be issued

  *Rvalue1 = (Val1MSB << 8) | (Val1LSB & 0xFF);
  *Rvalue2 = (Val2MSB << 8) | (Val2LSB & 0xFF);
  return 0;
}
```

<u>Items to Complete by Due Date. Your job is to pace yourself so you get everything done by the due date.:</u>

1. 3D print your two wheels, two kick stand and flat mount plate.
2. Demo/Checkoff in the lab, both your DAN28027 and BQ32000 code working together calling the three functions WriteDAN28027RCServo, ReadDAN28027ADC and ReadBQ32000 one right after the other. Call these three functions in a row every 20ms. You will need to create a flag variable like "UARTPrint" to flag when you should run these three functions in a row inside your main() function's "while(1)" loop. Print to Teraterm, every 100 ms, the two DAN28027 ADC readings and the current date and time in the format "day mm/dd/yy hour:minute:second." i.e. "Wednesday 03/24/21 23:16:34". In addition, scope the two RC servo PWM outputs and have your code gradually increase and decrease these PWM values.
3. In addition, demonstrate in lab that your WriteBQ32000() works to set a new day, date and time.
4. Submit to your Box folder:
   a. Your commented main C file. Should include the I2C functions you wrote.
   b. Tera Term screen shot showing the printout of your program working.
   c. Two pictures (with your phone) of the Oscilloscope displaying different PWM duty cycles produced by your I2C code transmitting new duty cycle commands. At your check of you will demonstrate the PWM duty cycle continuously varying. Here you will just take two pictures.