1. Source files are attached to email as separate files
2. Part 2 Service Routine Explained:
   The Read and Write Functions when called will raise our interrupt flag to start servicing and communicating over the SPI4 communication with an external EEProm.
   At the high level design overview, we want to check the status and make sure that a write data sequence is not in progress, if it is then we keep checking until its not. Next we begin the write or read sequence depending on which function raised the interrupt flag. If its a write then we want to send a write enable command before we begin writing and then send the address, and then keep looping until each byte has been written and that completes the write sequence. If its a read then we send the address and start reading each byte in a loop and that completes the read sequence.
   One thing to note is that each sequence have many interrupts before completing so right before each interrupt we want to set the new state and leave the service routine so that when it returns in continues in the next steps. Compared to part 1, each time in part 1 where we have "while(SPI4STATbits.SPIRBF != 1);" is when there is an interrupt waiting to happen. So in part 2 we want to set the next state leave the ISR so it can interrupt again. Next I lay out and explain each step of the State machine.
   a. State 0
      i. Begin the Check Status instructions: Send the read status command, wait until the buffer is empty and send a dummy for the buffer is empty data that was received.
      ii. Set state to 1
   b. State 1
      i. Midpoint of Check Status: we just read the dummy that was clocked in while sending the read status command that was written in state 0.
      ii. Set state to 2
   c. State 2
      i. End the Check Status, Begin either Read or Write Sequence
      ii. Read the Status values into variable
      iii. Check if a Write is Still in Progress(go to state 1)
      iv. If no write is in progress and we want to write then send the Write Enable command and set the state to 3
      v. If no write is in progress and we want to read then write the Read Data command, wait for the empty buffer, write the MSA, set the state to 4
   d. State 3
      i. Finish Write Enable and Begin Write Data
      ii. End the write enable process by reading a dummy that was clocked in when we sent the write enable command.
      iii. Write the write data command and wait for the empty buffer and write the MSA.
      iv. Set State 6
   e. State 4

    i. Continue reading and read the dummy that was clocked in when writing the MSA, then writing the LSA.
    ii. Set State 5

f. State 5
    i. Read and Write a dummy needs to be done before reading bytes
    ii. Set State 9

g. State 6
    i. Continue Writing
    ii. Read the dummy that was clocked in when sending the MSA, then send the LSA.
    iii. Set State 7

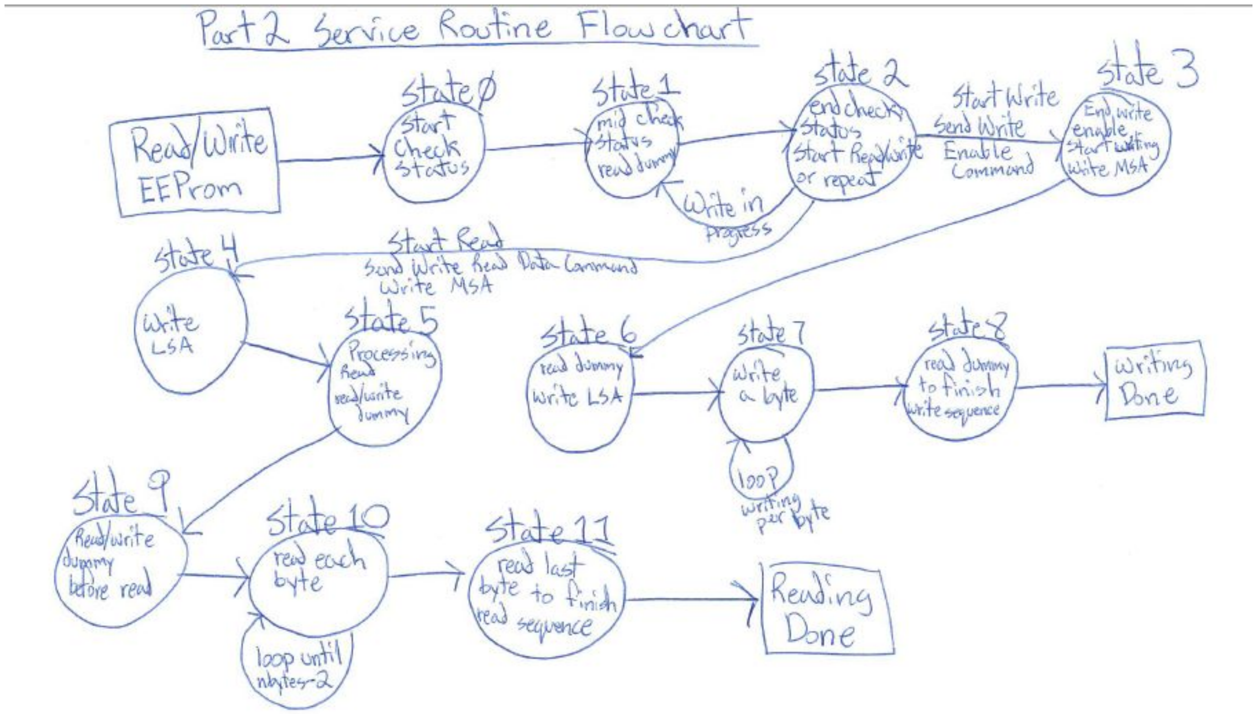
h. State 7
    i. Continue Writing
    ii. Read the dummy that was clocked in when sending the LSA and when writing a byte, then write a byte to the buffer. Repeat State 7 until the total bytes has been written and then read the second to last dummy.
    iii. Set State 8

i. State 8
    i. Finish Writing
    ii. Read one last dummy to show that we are done writing.

j. State 9
    i. Another Read and Write needs to be done before reading bytes
    ii. Set State 10

k. State 10
    i. Process the number of bytes that we are going to read until < NBYTES-2 and write a dummy byte for each one.
    ii. Read the byte NBYTES-2 with no write dummy
    iii. Set State 11

l. State 11
    i. Finish the Reading Sequence
    ii. Read the byte NBYTES-1 with no write dummy to show that we are done reading.
    iii. We are Done Reading here.



3. Code Based Analysis Parameter 3

a.

```
!         while(SPI4STATbits.SPIRBF != 1);
0x9D0007CC: NOP
0x9D0007D0: LUI V0, -16510
0x9D0007D4: LW V0, 5648(V0)
0x9D0007D8: ANDI V0, V0, 1
0x9D0007DC: BEQ V0, ZERO, 0x9D0007D0
0x9D0007E0: NOP
!         // read the byte
!         byteRead[nbytes-1] = SPI4BUF;
0x9D0007E4: LW V1, 44(FP)
0x9D0007E8: LUI V0, 16383
0x9D0007EC: ORI V0, V0, -1
0x9D0007F0: ADDU V0, V1, V0
0x9D0007F4: SLL V0, V0, 2
0x9D0007F8: LW V1, 40(FP)
0x9D0007FC: ADDU V0, V1, V0
0x9D000800: LUI V1, -16510
0x9D000804: LW V1, 5664(V1)
0x9D000808: SW V1, 0(V0)
!     }
!     asm volatile("nop");
0x9D00080C: NOP
!     asm volatile("nop");
0x9D000810: NOP
!     asm volatile("nop");
0x9D000814: NOP
!     asm volatile("nop");
0x9D000818: NOP
!     asm volatile("nop");
0x9D00081C: NOP
!     asm volatile("nop");
0x9D000820: NOP
!     //negate CS
!     PORTFbits.RF8 = 1;
0x9D000824: LUI V1, -16506
0x9D000828: LHU V0, 1312(V1)
0x9D00082C: ADDIU A0, ZERO, 1
0x9D000830: INS V0, A0, 8, 1
0x9D000834: SH V0, 1312(V1)
```

b. In this example of my code there are 16 instructions between waiting for RBF and negating CS so my Parameter 3 requirements were met. We see that most of the code in between the two its for reading a byte of code from the buffer. In this case I didn't have to include to many nop instructions because read byte is 10 instructions and is enough to fulfill the requirement.

4. Code Based Analysis Parameter 4

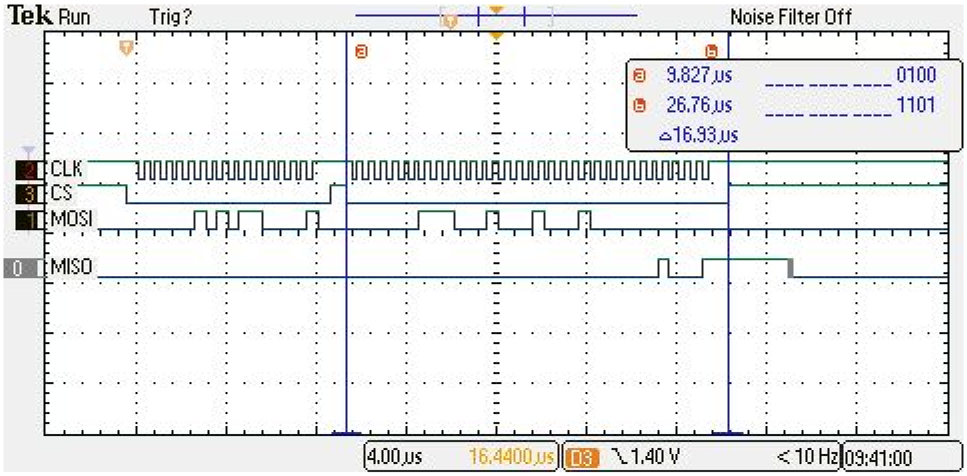
a.
```
!     // 9) Negate CS
!     PORTFbits.RF8 = 1;
0x9D000514: LUI V1, -16506
0x9D000518: LHU V0, 1312(V1)
0x9D00051C: ADDIU A0, ZERO, 1
0x9D000520: INS V0, A0, 8, 1
0x9D000524: SH V0, 1312(V1)
!
!     return byte;
0x9D000528: LW V0, 0(FP)
!}
0x9D00052C: ADDU SP, FP, ZERO
0x9D000530: LW FP, 12(SP)
0x9D000534: ADDIU SP, SP, 16
0x9D000538: JR RA
0x9D00053C: NOP
```

b.
```
!void writeEnableCommand(){
0x9D000540: ADDIU SP, SP, -16
0x9D000544: SW FP, 12(SP)
0x9D000548: ADDU FP, SP, ZERO
!
!     PORTFbits.RF8 = 0;
0x9D00054C: LUI V1, -16506
0x9D000550: LHU V0, 1312(V1)
0x9D000554: INS V0, ZERO, 8, 1
0x9D000558: SH V0, 1312(V1)
```

c. This example is from the end of my read status function and the beginning of my write enable function. Theres 9 instructions between the end of negate CS and the beginning of Assert CS so this example does pass the parameter 4 requirement. No code needed to be adjusted.

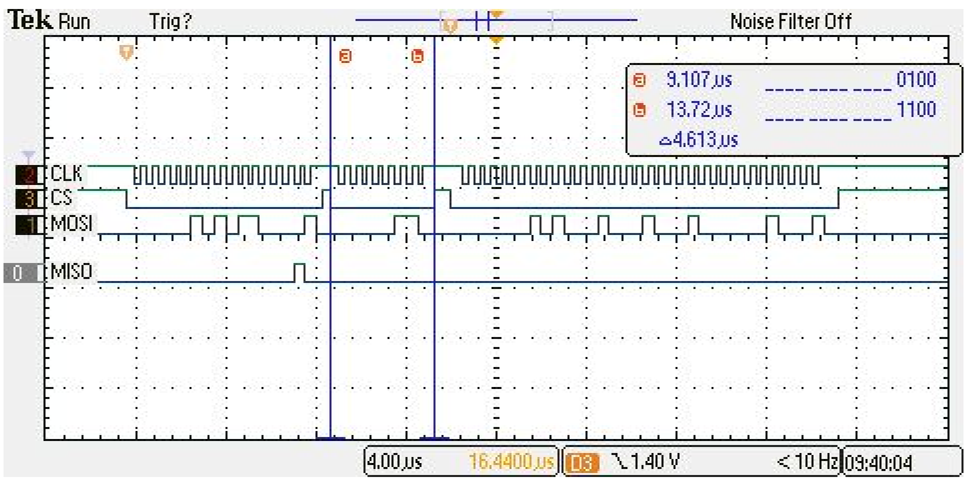
5. System Clocks Report
    a. I was only able to see 6 instructions or 6 system clocks. As I was debugging in the dissassembly the TBE while loop did not loop but only did those instructions once. So the number of system clocks could be a lot less.
6. SPI Clock Period Report
    a. The minimum SPI clock period to bring it down to 210ns minimum which we can operate in is 4.2MHz which gives us a BRG value of 9. The equation is SCLK = fPB/2(BRG+1) = 84MHz/20 = 4.2MHz.
7. MSO Part 1 Read and Write

a. Read



i. Includes read status and read data sequences

b. Write



i. Includes read status, write enable, and write data commands.