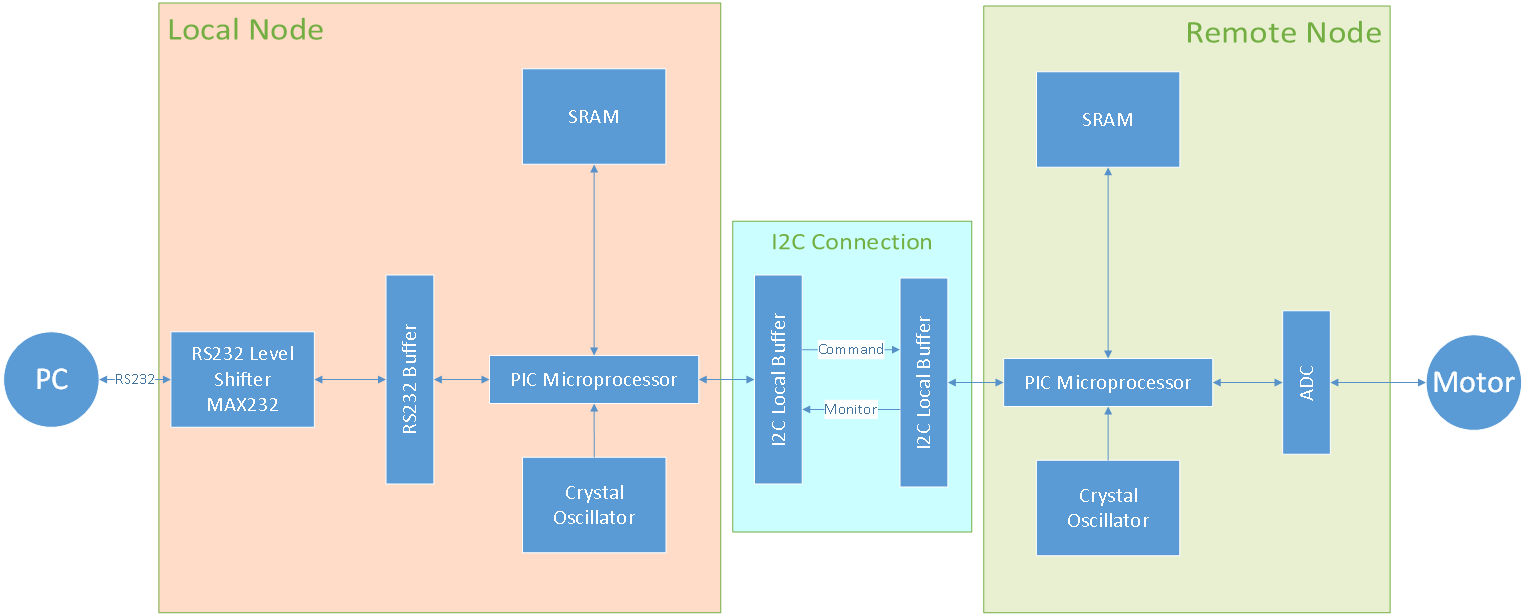
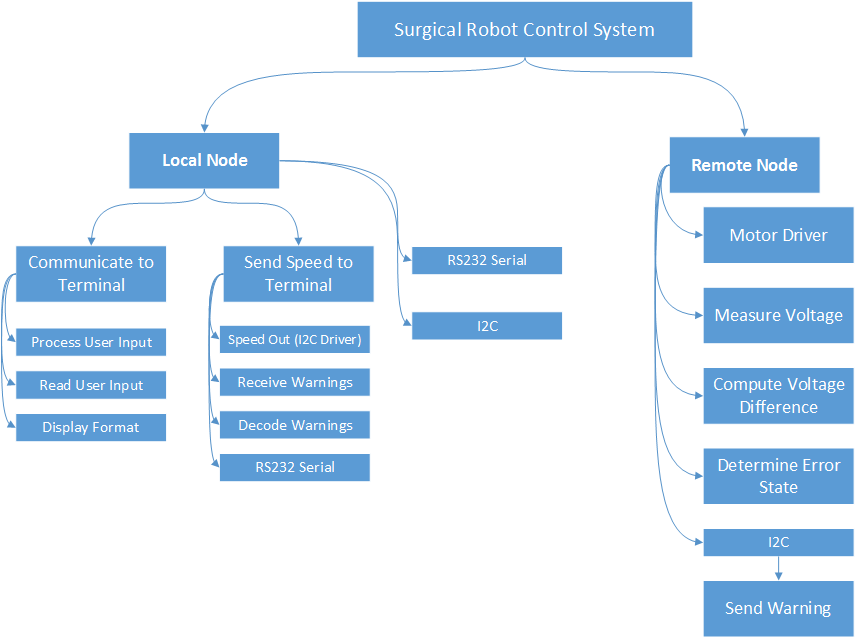
**Appendix A: Charts and Tables**

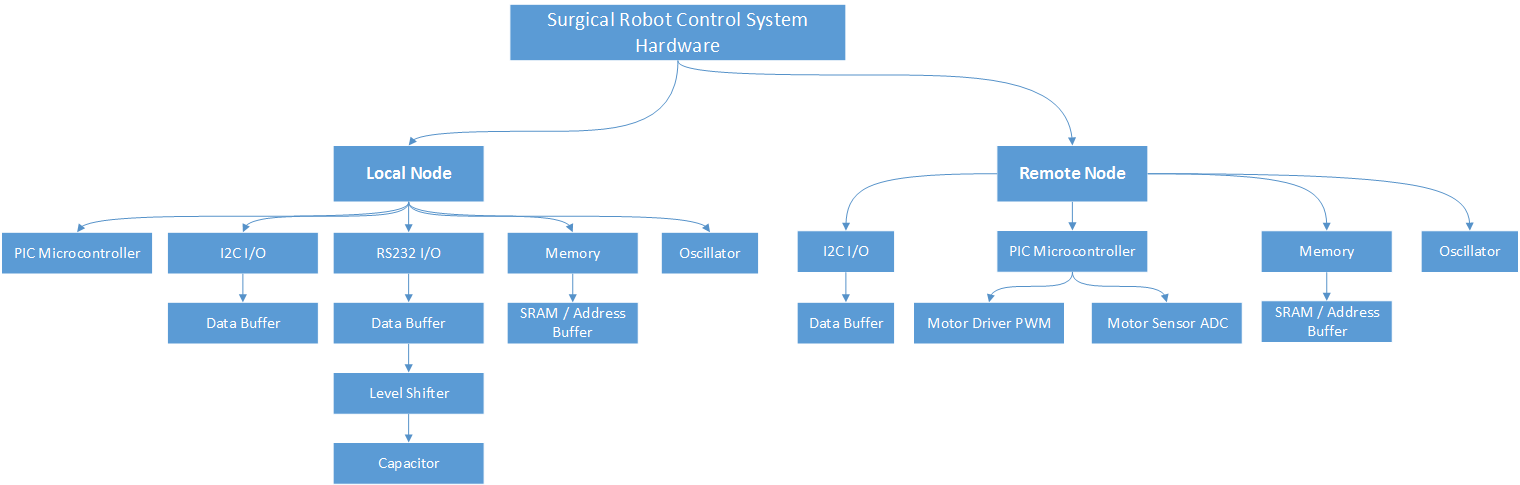
A1. Block Diagram



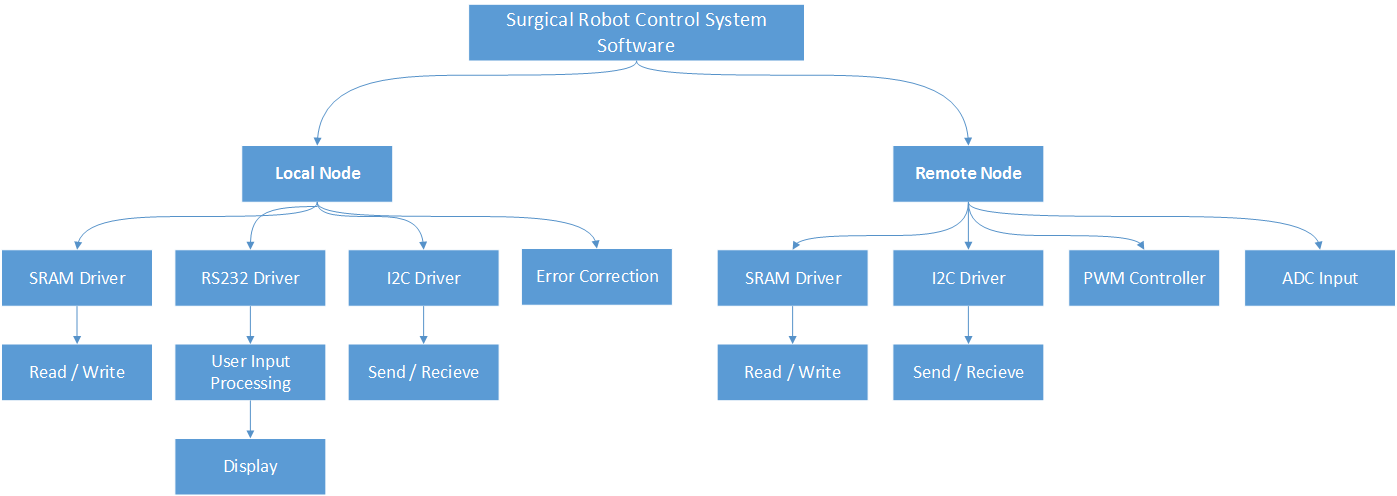
A2. Functional Decomposition of Overall System



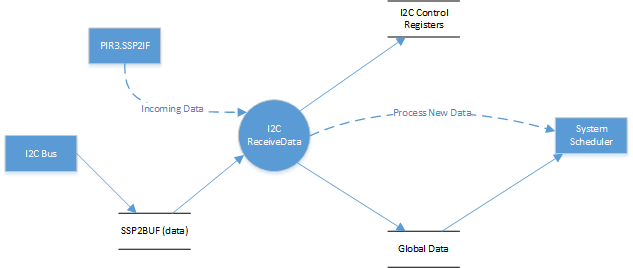
A3. Functional Decomposition of Hardware



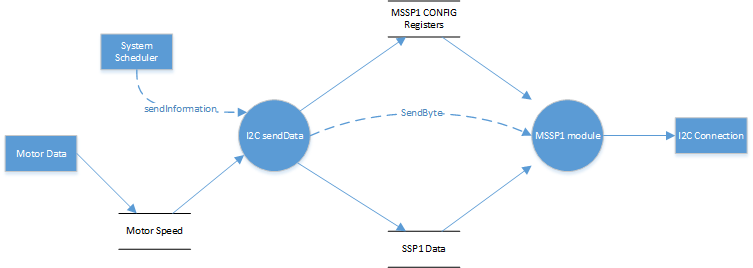
A4. Functional Decomposition of Software



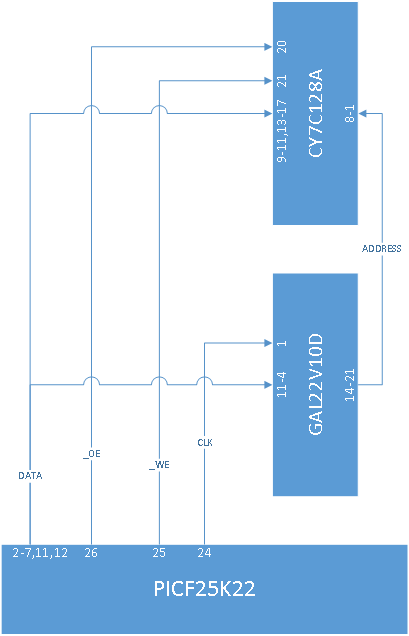
A5. I2C Receive Control Flow Diagram



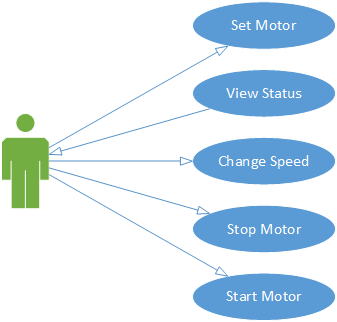
A6. I2C Send Control Flow Diagram



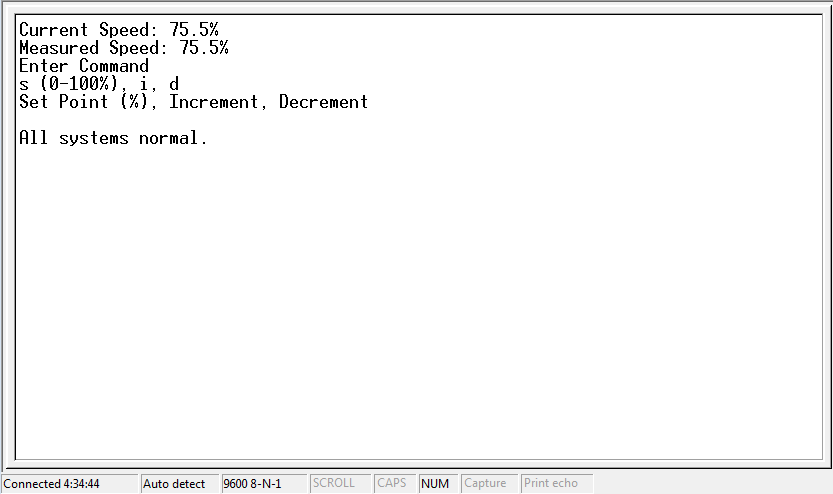
A7. SRAM Hardware Block Diagram



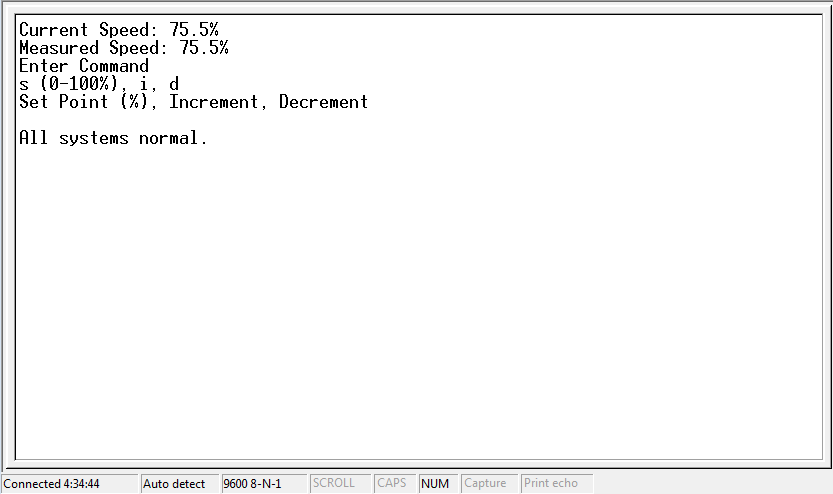
A8. Use Case Diagram



A9. User Interface Display via Hyper Terminal



A10.User Interface Display Showing Error



Appendix B: Graphs

B1.

B2.

B2.

B3.

B4.

Appendix C: Code

C1. SRAM Header File

|  |
| --- |
| /\*  \* File: SRAM.h  \* Author: castia  \*  \* Created on April 24, 2014, 1:59 AM  \*/  #ifndef SRAM\_H  #define SRAM\_H  #ifdef \_\_cplusplus  extern "C" {  #endif  void SRAMsetUp();  void setUpIn();  void setUpOut();  int readData(int adx);  void writeData(int adx, int data);  #ifdef \_\_cplusplus  }  #endif  #endif /\* SRAM\_H \*/ |

C2. SRAM Driver

|  |
| --- |
| //#include "GLOBALS\_H"  #include <p18f25k22.h>  #include "SRAM.h"  #include <delays.h>  unsigned short OE;  unsigned short WE;  unsigned short store;  void SRAMsetUp() {  // Write Enable  TRISBbits.TRISB4 = 0;  // Output Enable  TRISBbits.TRISB5 = 0;  // Store  TRISBbits.TRISB3 = 0;  // Default Values  PORTBbits.RB3 = 1;  PORTBbits.RB4 = 1;  PORTBbits.RB5 = 1;  ANSELA = 0x00;  ANSELBbits.ANSB3 = 0;  ANSELBbits.ANSB4 = 0;  ANSELBbits.ANSB5 = 0;  setUpIn();  }  void setUpOut () {  // Set data outputs  TRISAbits.TRISA0 = 0;  TRISAbits.TRISA1 = 0;  TRISAbits.TRISA2 = 0;  TRISAbits.TRISA3 = 0;  TRISAbits.TRISA4 = 0;  TRISAbits.TRISA5 = 0;  TRISCbits.TRISC0 = 0;  TRISCbits.TRISC1 = 0;  }  void setUpIn () {  // Set data inputs  TRISAbits.TRISA0 = 1;  TRISAbits.TRISA1 = 1;  TRISAbits.TRISA2 = 1;  TRISAbits.TRISA3 = 1;  TRISAbits.TRISA4 = 1;  TRISAbits.TRISA5 = 1;  TRISCbits.TRISC0 = 1;  TRISCbits.TRISC1 = 1;  }  // Reading data  int readData (int adx) {  int myRead = 0;  SRAMsetUp();  setUpOut();  // Setting up Address  PORTAbits.RA0 = (adx >> 0) & 0x01;  PORTAbits.RA1 = (adx >> 1) & 0x01;  PORTAbits.RA2 = (adx >> 2) & 0x01;  PORTAbits.RA3 = (adx >> 3) & 0x01;  PORTAbits.RA4 = (adx >> 4) & 0x01;  PORTAbits.RA5 = (adx >> 5) & 0x01;  PORTCbits.RC0 = (adx >> 6) & 0x01;  PORTCbits.RC1 = (adx >> 7) & 0x01;  // Store in MAR  PORTBbits.RB3 = 0;  PORTBbits.RB3 = 1;  // I/O are inputs  setUpIn();  // Output Enable  PORTBbits.RB5 = 0;  Delay10TCYx(5);  // Get the first 6 bits of Port A and the first 2 bits of port C shifted  myRead = (PORTA & 0x3F) | ((PORTC << 6) & 0xC0);  // Delay10TCYx(5);  // Output Enable  PORTBbits.RB5 = 1;  return myRead;  }  // Writing data  void writeData(int adx, int data) {  SRAMsetUp();  setUpOut();  // Setting up Address  PORTAbits.RA0 = (adx >> 0) & 0x01;  PORTAbits.RA1 = (adx >> 1) & 0x01;  PORTAbits.RA2 = (adx >> 2) & 0x01;  PORTAbits.RA3 = (adx >> 3) & 0x01;  PORTAbits.RA4 = (adx >> 4) & 0x01;  PORTAbits.RA5 = (adx >> 5) & 0x01;  PORTCbits.RC0 = (adx >> 6) & 0x01;  PORTCbits.RC1 = (adx >> 7) & 0x01;  // Store in MAR  PORTBbits.RB3 = 0;  PORTBbits.RB3 = 1;  // Send Data  PORTAbits.RA0 = (data >> 0) & 0x01;  PORTAbits.RA1 = (data >> 1) & 0x01;  PORTAbits.RA2 = (data >> 2) & 0x01;  PORTAbits.RA3 = (data >> 3) & 0x01;  PORTAbits.RA4 = (data >> 4) & 0x01;  PORTAbits.RA5 = (data >> 5) & 0x01;  PORTCbits.RC0 = (data >> 6) & 0x01;  PORTCbits.RC1 = (data >> 7) & 0x01;  // Write Enable  PORTBbits.RB4 = 0;  //Delay10TCYx(10);  PORTBbits.RB4 = 1;  } |

C3. Local Main

|  |
| --- |
| /\*  \* File: main.c  \* Author: castia  \*  \* Created on April 24, 2014, 4:22 PM  \*/  #include <stdio.h>  #include <stdlib.h>  #include "globals.h"  #include "i2c\_local.h"  #include "SRAM.h"  #include "rs232.h"  #include "adc.h"  //#include "display.h"  #include <i2c.h> // i2c library  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*USART set up \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #pragma config FCMEN = OFF  #pragma config IESO = OFF  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Clocking set up \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #pragma config WDTEN = OFF // turn off watch dog timer  #pragma config FOSC = ECHP // Ext. Clk, Hi Pwr  #pragma config PRICLKEN = OFF // disable primary clock  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #define IN\_BUF\_SZ 64  #define DEF\_PWM 0  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Global Setup\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  // System variables  unsigned int setSpeed = DEF\_PWM; // the user defined motor speed  unsigned int actualSpeed = 0; // Motor speed given from the remote (was motorSpeed)  unsigned int controllerSpeed = DEF\_PWM; // Motor speed send to the remote node by the local  unsigned int errorState = 4; // The current motor state is off  // User Input  unsigned char myInput[IN\_BUF\_SZ];  int inputSpot = 0;  short inputFinished = 0;  // Flags  short displayFlag = 1;  short SRAMflag = 0;  short i2cFlag = 0;  short processFlag = 0;  int myCommand = 0;  int myOp = 0;  Global ourGlobal = {&setSpeed, &actualSpeed, &controllerSpeed, &errorState, &myInput, &inputSpot, &inputFinished, &displayFlag, &SRAMflag, &i2cFlag, &processFlag, &myCommand, &myOp};  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Interrupt Stuff\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* Global Controls for I/O banks\*/  #pragma config PBADEN = OFF // turn off bank B ADCs  #pragma code high\_vector=0x08  void interrupt\_at\_high\_vector(void) {  \_asm GOTO rcISR \_endasm  }  #pragma code  #pragma interrupt rcISR  void rcISR(void) {  unsigned char input;  // Don't have to wait for data available if we are in ISR  input = getc1USART();  // Terminate string on enter or max size  // Allow for only regular keyboard characters and delete character  if (input >= ' ' && input <= 'z' || input == '\r' || input == '\n' || input == '\b') {  if (input == '\r' || input == '\n' || inputSpot == (IN\_BUF\_SZ - 1)) {  myInput[inputSpot] = '\0';  // Reset input, declare finished  inputSpot = 0;  inputFinished = 1;  \*ourGlobal.processFlag = 1;  // Delete a value  } else if (input == '\b' && inputSpot > 0) {  myInput[inputSpot] = '\0';  inputSpot--;  putc1USART(input);  } else if (input != '\b') {  // Put character in the input buffer  myInput[inputSpot] = input;  inputSpot++;  // Print current character  putc1USART(input);  inputFinished = 0;  }  }  // Clear interrupt  PIR1bits.RCIF = 0;  }  #pragma code low\_vector=0x18  void interrupt\_at\_low\_vector(void) {  \_asm GOTO i2cISR \_endasm  }  #pragma code  #pragma interrupt i2cISR  void i2cISR(void) {  unsigned int temp = 0;  static int byteNum = 0;  if (SSP2STATbits.D\_A == 0 && SSP2STATbits.BF == 1) {  temp = SSP2BUF;  }  if (SSP2STATbits.D\_A == 1 && SSP2STATbits.BF == 1) {  if (byteNum == 0) {  \*ourGlobal.actualSpeed = SSP2BUF;  byteNum++;  } else {  \*ourGlobal.errorState = SSP2BUF;  byteNum = 0;  }  }  \*ourGlobal.displayFlag = 1;  PIR3bits.SSP2IF = 0;  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* setsup the pwm to output the voltage  \*/  void setupPWM() {  // configure PWM  TRISBbits.RB0 = 1; // disable PWM output  CCPTMRS1 = 0b00000001; // set C4TSEL = 0b01  PR4 = 0xF9; // PR = 2 for 20kHz  T4CON = 0b00000100; // set timer prescale 1:1, turn on timer4  CCP4CON = 0b00011100; // set LSB of duty cyle, select pwm mode  CCPR4L = 0x3E; // set MSB of duty cycle  PIR5 = 0b00000000; // clear timer interrupt flag  TRISBbits.RB0 = 0; //enable PWM output  }  /\*  \*  \*/  void main() {  unsigned int temp = 0;  char errorMsg[] = "Error: Input again.\n\r";  // I2c Setup  setupOutgoing();  setupIncoming();  // Rs232 setup and interrupt  rs232Setup();  // Enable SRAM pins correctly  SRAMsetUp();  // Setup ADC  OpenADC(ADC\_FOSC\_64 & ADC\_RIGHT\_JUST & ADC\_12\_TAD,  ADC\_CH14 & ADC\_INT\_OFF, 15);  // setup PWM (debug purposes only)  setupPWM();  // Set Default PWM  SetDCPWM4(5 \* (\*ourGlobal.controllerSpeed));  writeData(1, \*ourGlobal.controllerSpeed);  while (1) {  displayFrontPanel(&ourGlobal);  dataProcess(&ourGlobal);  if (\*ourGlobal.SRAMflag == 1) {  \*ourGlobal.SRAMflag = 0;  writeData(0, \*ourGlobal.myCommand);  switch (\*ourGlobal.myCommand) {  case 1:  writeData(1, \*ourGlobal.setSpeed);  \*ourGlobal.i2cFlag = 1;  break;  case 2:  temp = readData(1) + 1;  if (temp <= 202 && temp >= 0) {  writeData(1, temp);  } else {  writeData(1, 202);  }  \*ourGlobal.i2cFlag = 1;  break;  case 3:  temp = readData(1) - 1;  if (temp <= 202 && temp >= 0) {  writeData(1, temp);  } else {  writeData(1, 0);  }  \*ourGlobal.i2cFlag = 1;  break;  case 4:  puts1USART(errorMsg);  \*ourGlobal.displayFlag = 1;  break;  }  }  if (\*ourGlobal.i2cFlag == 1) {  \*ourGlobal.controllerSpeed = readData(1);  SetDCPWM4(5 \* (\*ourGlobal.controllerSpeed));  runLocalI2C(ourGlobal.controllerSpeed);  \*ourGlobal.i2cFlag = 0;  \*ourGlobal.displayFlag = 1;  }  Delay1KTCYx(1);  }  }  // as the name says  int readADC() {  ADCON0bits.GO\_DONE = 1;  while (ADCON0bits.GO\_DONE);  ADCON0bits.ADON = 0;  return ADRESL;  } |

C4. Local Node I2C Header File

|  |
| --- |
| /\*  \* File: i2c\_local.h  \* Author: Patrick  \*  \* Created on April 23, 2014, 8:20 PM  \*/  #ifndef I2C\_LOCAL\_H  #define I2C\_LOCAL\_H  void runLocalI2C(unsigned int \*setSpeed);  void runReceiveI2C(unsigned int \*setSpeed);  int setupOutgoing();  int setupIncoming();  int sendSpeed(unsigned int \*slaveAddr, unsigned int \*speed);  int getRemoteData();  #endif /\* I2C\_LOCAL\_H \*/ |

C5. Local Node with I2C

|  |
| --- |
| /\*  \* File: i2c\_local.c  \* Author: Patrick Ma  \*  \* Created on April 23, 2014, 8:20 PM  \*  \* Controls the I2C behavior for the local control node.  \* Outgoing communication occurs on the I2C1 channel.  \* Incoming communication occurs on the I2C2 channel.  \*/  /\*\*\*\*\*\*\*\*\*\*\*\* I2C set globals \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include "globals.h"  #include "i2c\_local.h"  #include <i2c.h> // i2c library  /\* Local defines \*/  #define BAUD\_RATE 100000 // The transmit baud rate  #define BD\_RT ((20000000 / (4 \* BAUD\_RATE)) -1)// calculated SSPADD  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* local variables\*/  unsigned char data, addr;  int status, index;  /\* method declarations\*/  // See header file  /\* TX & RX interrupt handlers\*/  /\*  \* configures and starts outgoing data communication  \* The outgoing I2C uses MSSP1  \*/  int setupOutgoing() {  // set pins RC14, RC15 as inputs  TRISCbits.TRISC3 = 1; // SCL1  ANSELCbits.ANSC3 = 0;  TRISCbits.TRISC4 = 1; // SDA1  ANSELCbits.ANSC4 = 0;  // configure i2c1 for master mode @ 100 kHz  CloseI2C1();  OpenI2C1(MASTER, SLEW\_OFF);  SSP1ADD = BD\_RT;  return (1);  }  /\*  \* configures and starts incoming data communication  \* The incoming I2C uses MSSP2  \*/  int setupIncoming() {  int SLAVE\_ADDRESS = 0x00; // the slave address  // set pins RC14, RC15 as inputs  TRISBbits.TRISB1 = 1; // SCL2  ANSELBbits.ANSB1 = 0;  TRISBbits.TRISB2 = 1; // SDA2  ANSELBbits.ANSB2 = 0;  // Enable Priority  RCONbits.IPEN = 1;  // Low priority receive interrupt  IPR3bits.SSP2IP = 0;  // Enable all low priority interrupts  INTCONbits.GIEL = 1;  INTCONbits.PEIE = 1;  PIE3bits.SSP2IE = 1;  // configure MSSP2 for i2c slave operation.  CloseI2C2();  OpenI2C2(SLAVE\_7, SLEW\_OFF);  SSP2ADD = SLAVE\_ADDRESS;  return (1);  }  /\* Task to send the current setpoint to a remote node\*/  void runLocalI2C(unsigned int \*setSpeed) {  addr = 0x0;  sendSpeed(&addr, setSpeed);  // Delay10TCYx(20);  }  /\*  \* send data to a slave device  \* takes the slave address, writes the given data byte  \*/  int sendSpeed(unsigned int \*slaveAddr, unsigned int \*speed) {  // get the data from global variable  char newSpeed[3] = {\*speed, 'X', '\0'};  // wait until idle - not actually needed for single-master bus  IdleI2C1();  StartI2C1(); // send start  data = SSP1BUF;  WriteI2C1(\*slaveAddr & 0xFE);  // do { // send address until ack'd  // status = WriteI2C1(\*slaveAddr || 0x00);  // if (!(0 == status)) { // write collision  // data = SSP1BUF;  // SSP1CON1bits.WCOL = 0;  // }  // } while (!(0 == status));  // send bytes  WriteI2C1(newSpeed[0]); // TODO: send multiple bytes  StopI2C1(); // stop transmission  return 1;  }  // Read data -> handled by interrupt |

C6. Local Node Interrupts

|  |
| --- |
| /\*  \* File: interrupts.h  \* Author: castia  \*  \* Created on April 24, 2014, 10:16 PM  \*/  #ifndef INTERRUPTS\_H  #define INTERRUPTS\_H  #ifdef \_\_cplusplus  extern "C" {  #endif  void rcISR(void);  void i2cISR(void);  #ifdef \_\_cplusplus  }  #endif  #endif /\* INTERRUPTS\_H \*/ |

C7. Local Node Process

|  |
| --- |
| #include "globals.h"  #include "string.h"  int stringToNum(char\* myStr);  void dataProcess(Global\* globalData) {  unsigned int userSetPoint = 0;  // myCommand key  // 1 = set point  // 2 = increment  // 3 = decrement  // 4 = error  if (\*globalData->processFlag == 1) {  \*globalData->processFlag = 0;  // Normal state  // if (\*globalData->errorState == 4) {  // Only for command "s "  if (globalData->myInput[0] == 's' && globalData->myInput[1] == ' ') {  \*globalData->myCommand = 1;  // Get data  userSetPoint = 2 \* stringToNum((char\*) &globalData->myInput[2]);  if (userSetPoint <= 202 && userSetPoint >= 0) { // Only allow inputs of 0 to 101% (101% is for testing purposes)  \*globalData->setSpeed = userSetPoint;  } else {  \*globalData->myCommand = 4;  }  } else if (globalData->myInput[0] == 'i') {  \*globalData->myCommand = 2;  } else if (globalData->myInput[0] == 'd') {  \*globalData->myCommand = 3;  } else {  \*globalData->myCommand = 4;  }  // States where speed should be set to 0 (crit, warning, off)  //} else if (\*globalData->errorState != 2) {  //\*globalData->setSpeed = 0;  //\*globalData->controllerSpeed = 0;  //writeData(1, 0);  // Attention  // } else {  //\*globalData->setSpeed /= 2;  //\*globalData->controllerSpeed /= 2;  //writeData(1, \*globalData->setSpeed);  //}  \*globalData->SRAMflag = 1;  }  }  int stringToNum(char\* myStr) {  int i = 0;  int sum = 0;  while (myStr[i] != '\0') {  sum \*= 10;  sum += myStr[i] - '0';  i++;  }  return sum;  } |

C8. Local Node RS232 Header File

|  |
| --- |
| */\**  *\* File: rs232.h*  *\* Author: castia*  *\**  *\* Created on April 24, 2014, 10:10 PM*  *\*/*  *void rs232Setup(void);*  *void readBytesUntil(char\* myStorage, char stopChar, int size);* |

C9. Local Node RS232

|  |
| --- |
| #include "globals.h"  #include "rs232.h"  void rs232Setup() {  // Set RX as input, TX as output  TRISCbits.TRISC7 = 1;  TRISCbits.TRISC6 = 0;  // Enable digital for all c pins  ANSELC=0x00;  // Configure UART, 9600 baud with 20MHz clock.  Open1USART(USART\_TX\_INT\_OFF & USART\_RX\_INT\_ON & USART\_ASYNCH\_MODE & USART\_EIGHT\_BIT & USART\_BRGH\_HIGH, 129);  // Enable Priority  RCONbits.IPEN = 1;  // High priority receive interrupt  IPR1bits.RCIP = 1;  // Enable all high priority interrupts  INTCONbits.GIEH = 1;  }  // Precondition: USART 1 is open & configured  void readBytesUntil(char\* myStorage, char stopChar, int size) {  int i = 0;  char message;  while(!DataRdy1USART());  message = getc1USART();  putc1USART(message);  while(message != stopChar && i < (size - 1)) {  myStorage[i] = message;  i++;  while(!DataRdy1USART());  message = getc1USART();  putc1USART(message);  }  myStorage[i] = '\0';  i = 0;  } |

C10. Local Node Global Header File

|  |
| --- |
| /\*  \* File: globals.h  \* Author: castia  \*  \* Created on April 23, 2014, 9:30 PM  \*  \* global header file for lab2 project. Put any externs used by other files here  \*  \*/  #ifndef GLOBALS\_H  #define GLOBALS\_H  /\* includes for the whole project\*/  #include <stdio.h>  #include <stdlib.h>  #include <usart.h>  #include <i2c.h>  #include <p18f25k22.h>  #include <delays.h>  #include <pwm.h>  #include "SRAM.h"  #include "rs232.h"  #include "string.h"  #include "interrupts.h"  /\* DEFINES for the whole project \*/  #define USE\_AND\_OR // allows use of compiler libs  typedef struct globalStruct {  // System variables  unsigned int\* setSpeed; // the user defined motor speed  unsigned int\* actualSpeed; // Motor speed given from the remote (was motorSpeed)  unsigned int\* controllerSpeed; // Motor speed send to the remote node by the local  unsigned int\* errorState; // The current motor state is off  // User Input  unsigned char\* myInput;  int\* inputSpot;  short\* inputFinished;  // Flags  short\* displayFlag;  short\* SRAMflag;  short\* i2cFlag;  short\* processFlag;  int\* myCommand;  int\* myOp;  } Global;  void displayFrontPanel(Global\* globalData);  void dataProcess(Global\* globalData);  void errorCheck(Global\* gData);  #endif /\* GLOBALS\_H \*/ |

C11. Local Node Display

|  |
| --- |
| #include "globals.h"  void usartPrint(const char\* myString);  void displayFrontPanel(Global\* globalData) {  char errorDisplay[64];  if (\*globalData->displayFlag == 1) {  \*globalData->displayFlag = 0;  putc1USART(0x0C);  if (\*globalData->controllerSpeed % 2 == 0) {  sprintf(errorDisplay, (char\*) "Current Speed: %s.0%%\r\n", itoa(\* (globalData->controllerSpeed) / 2, 16));  } else {  sprintf(errorDisplay, (char\*) "Current Speed: %s.5%%\r\n", itoa(\* (globalData->controllerSpeed) / 2, 16));  }  puts1USART(errorDisplay);  if (\*globalData->actualSpeed % 2 == 0) {  sprintf(errorDisplay, (char\*) "Measured Speed: %s.0%%\r\n", itoa(\*(globalData->actualSpeed) / 2, 16));  } else {  sprintf(errorDisplay, (char\*) "Measured Speed: %s.5%%\r\n", itoa(\*(globalData->actualSpeed) / 2, 16));  }  puts1USART(errorDisplay);  putrs1USART("Enter Command\r\n");  putrs1USART("s (0-100%), i, d\r\nSet Point (%), Increment, Decrement\r\n");  // \*globalData->displayFlag = 0;  switch (\*globalData->errorState) {  // Level 0: Severe  case 0:  putrs1USART("\r\nSevere error! Motor speed 5% out of range.\r\n");  \*globalData->processFlag = 1;  break;  // Level 1: Moderate  case 1:  putrs1USART("\r\nError! Motor speed 2% out of range.\r\n");  \*globalData->processFlag = 1;  break;  // Level 2: Warn (Of Concern)  case 2:  putrs1USART("\r\nWarning! Motor speed 1% out of requested speed\r\n");  \*globalData->processFlag = 1;  break;  // Level 3: Off  case 3:  putrs1USART("\r\nSystem currently off.\r\n");  break;  // Normal State  case 4:  putrs1USART("\r\nAll systems normal.\r\n");  break;  default:  putrs1USART("\r\nUnable to get motor status.\r\n");  break;  }  }  }  void usartPrint(const char\* myString) {  int i = 0;  while (myString[i] == '\0') {  putc1USART(myString[i]);  }  } |

C12. Remote Node Main

|  |
| --- |
| /\*  \* File: main.c  \* Author: castia  \*  \* Created on April 24, 2014, 4:22 PM  \*/  #include <stdio.h>  #include <stdlib.h>  #include "globals.h"  #include "i2c\_local.h"  #include "SRAM.h"  //#include "display.h"  #include <i2c.h> // i2c library  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*USART set up \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #pragma config FCMEN = OFF  #pragma config IESO = OFF  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Clocking set up \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #pragma config WDTEN = OFF // turn off watch dog timer  #pragma config FOSC = ECHP // Ext. Clk, Hi Pwr  #pragma config PRICLKEN = OFF // disable primary clock  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #define IN\_BUF\_SZ 64  #define DEF\_PWM 00  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Global Setup\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  // System variables  unsigned int setSpeed = DEF\_PWM; // the user defined motor speed (from local node)  unsigned int actualSpeed = 0; // Motor speed given from the remote (was motorSpeed)  unsigned int controllerSpeed = DEF\_PWM; // PWM motor speed  unsigned int errorState = 4; // The current motor state is off  // User Input  unsigned char myInput[IN\_BUF\_SZ];  int inputSpot = 0;  short inputFinished = 0;  // Flags  short displayFlag = 1;  short SRAMflag = 0;  short i2cFlag = 0;  short processFlag = 0;  int myCommand = 0;  int myOp = 0;  Global ourGlobal = {&setSpeed, &actualSpeed, &controllerSpeed, &errorState, &myInput, &inputSpot, &inputFinished, &displayFlag, &SRAMflag, &i2cFlag, &processFlag, &myCommand, &myOp};  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Interrupt Stuff\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* Global Controls for I/O banks\*/  #pragma config PBADEN = OFF // turn off bank B ADCs  #pragma code high\_vector=0x08  void interrupt\_at\_high\_vector(void) {  \_asm GOTO i2cISR \_endasm  }  #pragma code  #pragma interrupt i2cISR  void i2cISR(void) {  int temp = 0;  if (SSP2STATbits.D\_A == 0 && SSP2STATbits.BF == 1) {  temp = SSP2BUF;  }  if (SSP2STATbits.D\_A == 1 && SSP2STATbits.BF == 1) {  \*ourGlobal.setSpeed = SSP2BUF; // new motor set speed  }  PIR3bits.SSP2IF = 0;  \*ourGlobal.SRAMflag = 1;  }  ///\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  int readADC();  /\* setsup the pwm to output the voltage  \*/  void setupPWM() {  // configure PWM  TRISBbits.RB0 = 1; // disable PWM output  CCPTMRS1 = 0b00000001; // set C4TSEL = 0b01  PR4 = 0xF9; // PR = 2 for 20kHz  T4CON = 0b00000100; // set timer prescale 1:1, turn on timer4  CCP4CON = 0b00011100; // set LSB of duty cyle, select pwm mode  CCPR4L = 0x3E; // set MSB of duty cycle  PIR5 = 0b00000000; // clear timer interrupt flag  TRISBbits.RB0 = 0; //enable PWM output  // Rs232 setup and interrupt  }  /\*  \*  \*/  void main() {  unsigned int temp = 0;  unsigned int correct = 0;  unsigned int delay = 0;  // setup i2c  setupIncoming();  setupOutgoing();  //setup PWM  setupPWM();  // Setup ADC  TRISCbits.RC2 = 1;  TRISCbits.RC5 = 0;  ANSELCbits.ANSC5 = 0;  ANSELCbits.ANSC2 = 1; //set as input  OpenADC(ADC\_FOSC\_64 & ADC\_RIGHT\_JUST & ADC\_12\_TAD,  ADC\_CH14 & ADC\_INT\_OFF, 15);  // Enable SRAM pins correctly  SRAMsetUp();  // Set Default PWM  SetDCPWM4(5 \* (\*ourGlobal.controllerSpeed));  writeData(1, \*ourGlobal.controllerSpeed);  while (1) {  // Store a set speed  if (\*ourGlobal.SRAMflag == 1) {  writeData(1, \*ourGlobal.setSpeed);  \*ourGlobal.processFlag = 1;  \*ourGlobal.SRAMflag = 0;  }  // Measure Motor  Delay10KTCYx(8);  PORTCbits.RC5 = 1; // conversion indicator  ConvertADC();  while (BusyADC());  temp = ReadADC();  temp = ((50 \* (temp)) >> 8) + 1;  PORTCbits.RC5 = 0; // conversion indicator  // only makes sense to update ADC if significantly different than  // prior reading  if ((temp - 1 > \*ourGlobal.actualSpeed) || (\*ourGlobal.actualSpeed > temp + 1)) {  correct = temp;  \*ourGlobal.actualSpeed = correct;  \*ourGlobal.i2cFlag = 1;  }  // Get error state  errorCheck(&ourGlobal);  // enable auto correct  //if (\*ourGlobal.actualSpeed < (\*ourGlobal.setSpeed - 1) ||  //(\*ourGlobal.setSpeed + 1) < \*ourGlobal.actualSpeed) {  controller(&ourGlobal); // compute the control function and adjust pwm  //}    // Process updated controller speed  if (\*ourGlobal.processFlag == 1) {  \*ourGlobal.processFlag = 0;  writeData(2, \*ourGlobal.controllerSpeed); // update the motor output  }  SetDCPWM4(5 \* readData(2));  // Transfer actual speed to local node  if (\*ourGlobal.i2cFlag == 1) {  \*ourGlobal.i2cFlag = 0;  // Adjust actual speed for tranmittal as 8 bits. We can't go above  // 65535 in th emath or it breaks 16 bit max int.  // So 100 \* 65 is the max doing it a conventional way  //correct = 50 \* ((\*ourGlobal.actualSpeed) >> 8); //correct = 2 \* (2\*((50 \* \*ourGlobal.actualSpeed) / 1023));  Delay1KTCYx(1);  runLocalI2C(ourGlobal.actualSpeed);  // TODO: Calculate and send error  }  Delay1KTCYx(1);  }  }  int readADC() {  ADCON0bits.GO\_DONE = 1;  while (ADCON0bits.GO\_DONE);  ADCON0bits.ADON = 0;  return ADRESH;  } |

C13. Remote Node Control Header File

|  |
| --- |
| /\*  \* header file for control.h  \*/  void controller(Global \*gData); |

C14. Remote Node Control

|  |
| --- |
| /\*  \* Created by: Patrick Ma  \* Date: April 29, 2014  \*  \* Control.c implements the local side of the controller controller.  \* It adjusts the controller speed value sent to the remote node according to the  \* difference between the expected set speed and the actual controller speed.  \*/  #include <stdlib.h>  #include <stdio.h>  #include "globals.h"  //#include "control.h"  #define TEST\_CONTROL 0  #ifdef TEST\_CONTROL  Global gData;  int i = 0;  #endif  void controller(Global \*gData) {    // Add the difference between the desired and measured speeds  // back to the controller speed sent to the remote  if (\*gData->setSpeed == 0) { // zero case  \*gData->controllerSpeed = 0;  } else if (\*gData->setSpeed > \*gData->actualSpeed) {  \*gData->controllerSpeed += \*gData->setSpeed - \*gData->actualSpeed;  } else if (\*gData->setSpeed < \*gData->actualSpeed) {  \*gData->controllerSpeed -= \*gData->actualSpeed - \*gData->setSpeed;  }    // store the updated speed in SRAM  \*gData->processFlag = 1;    //\*gData->controllerSpeed = \*gData->setSpeed;  return;  }  //#ifdef TEST\_CONTROL  //// a quick test program for the control logic  ////  //// 4/29/2014  //void printData() {  // printf("%d) set: %f controller: %f actual: %f\n", i, gData.setSpeed,  // gData.controllerSpeed, gData.actualSpeed);  //}  //  //void main () {  // gData.setSpeed = 10;  // gData.actualSpeed = 2;  //  //// for (int i = 0; i < 10; i++) {  // while(gData.setSpeed != gData.actualSpeed){  // i++;  // printData();  // control(&gData);  // gData.actualSpeed = (0.75) \* (gData.controllerSpeed); // model motor response  // printData();  // }  //  // return;  //}  //#endif |

C15. I2C Remote Header

|  |
| --- |
| /\*  \* File: i2c\_remote.h  \* Author: castia  \*  \* Created on April 25, 2014, 4:46 AM  \*/  #ifndef I2C\_REMOTE\_H  #define I2C\_REMOTE\_H  void runRemoteI2C(unsigned int \*setSpeed);  void runReceiveI2C(unsigned int \*setSpeed);  int setupOutgoing();  int setupIncoming();  int sendSpeed(unsigned int \*slaveAddr, unsigned int \*speed);  int getRemoteData();  #endif /\* I2C\_REMOTE\_H \*/ |

C16. I2C Remote

|  |
| --- |
| /\*  \* File: i2c\_local.c  \* Author: Patrick Ma  \*  \* Created on April 23, 2014, 8:20 PM  \*  \* Controls the I2C behavior for the local control node.  \* Outgoing communication occurs on the I2C1 channel.  \* Incoming communication occurs on the I2C2 channel.  \*/  /\*\*\*\*\*\*\*\*\*\*\*\* I2C set globals \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include "globals.h"  #include "i2c\_local.h"  #include <i2c.h> // i2c library  /\* Local defines \*/  #define BAUD\_RATE 100000 // The transmit baud rate  #define BD\_RT ((20000000 / (4 \* BAUD\_RATE)) -1)// calculated SSPADD  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* local variables\*/  extern Global ourGlobal;  unsigned char i2cData, addr1;  int status, index;  /\* method declarations\*/  // See header file  /\* TX & RX interrupt handlers\*/  /\*  \* configures and starts outgoing data communication  \* The outgoing I2C uses MSSP1  \*/  int setupOutgoing() {  // set pins RC14, RC15 as inputs  TRISCbits.TRISC3 = 1; // SCL1  ANSELCbits.ANSC3 = 0;  TRISCbits.TRISC4 = 1; // SDA1  ANSELCbits.ANSC4 = 0;  // configure i2c1 for master mode @ 100 kHz  CloseI2C1();  OpenI2C1(MASTER, SLEW\_OFF);  SSP1ADD = BD\_RT;  return (1);  }  /\*  \* configures and starts incoming data communication  \* The incoming I2C uses MSSP2  \*/  int setupIncoming() {  int SLAVE\_ADDRESS = 0x0; // the slave address  // set pins as inputs  TRISBbits.TRISB1 = 1; // SCL2  ANSELBbits.ANSB1 = 0;  TRISBbits.TRISB2 = 1; // SDA2  ANSELBbits.ANSB2 = 0;  // Enable Priority  RCONbits.IPEN = 1;  // High priority receive interrupt  IPR3bits.SSP2IP = 1;  // Enable all high priority interrupts  INTCONbits.GIEH = 1;  INTCONbits.PEIE = 1;  IPR3bits.SSP2IP = 1;  PIE3bits.SSP2IE = 1;  // configure MSSP2 for i2c slave operation.  CloseI2C2();  OpenI2C2(SLAVE\_7, SLEW\_OFF);  SSP2ADD = SLAVE\_ADDRESS;  return (1);  }  /\* Task to send the current setpoint to a remote node\*/  void runLocalI2C(unsigned int \*setSpeed) {  addr1 = 0x00;  sendSpeed(&addr1, setSpeed);  // Delay10TCYx(20);  }  /\* Task to read from the bus\*/  void runRemoteI2C(unsigned int \*setSpeed) {  receiveData();  }  /\*  \* send data to a slave device  \* takes the slave address, writes the given data byte  \*/  int sendSpeed(unsigned int \*slaveAddr, unsigned int \*speed) {  // get the data from global variable  char newSpeed[5];  newSpeed[0] = \*speed;  newSpeed[1] = \*ourGlobal.errorState;  newSpeed[4] = '\0';  // wait until idle - not actually needed for single-master bus  IdleI2C1();  StartI2C1(); // send start  WriteI2C1(\*slaveAddr & 0xFE);  // do { // send address until ack'd  // status = WriteI2C1(\*slaveAddr || 0x00);  // if (!(0 == status)) { // write collision  // data = SSP1BUF;  // SSP1CON1bits.WCOL = 0;  // }  // } while (!(0 == status));  // send bytes  WriteI2C1(newSpeed[0]);  WriteI2C1(newSpeed[1]);// TODO: send multiple bytes  StopI2C1(); // stop transmission  return 1;  }  // read data  int receiveData() {  // get here after interrupt  // dumb remote just waits for a call  while (0 == DataRdyI2C2());  //AckI2C2(); // do we need to ack?  i2cData = SSPBUF; // clear buffer  while (0 == DataRdyI2C2()); // wait for next byte  \*ourGlobal.setSpeed = getcI2C2(); // store buffer value to memory  Delay10TCYx(1);  } |

C17. Remote Node Error Check

|  |
| --- |
| /\*  \* Created by: Alyanna Castillo  \* Date: April 29, 2014  \*  \* errorCheck.c checks for errors within the monitor channel.  \* Errors are organized into three classes:  \* +/- 5% - Level 0 - Severe  \* +/- 2% - Level 1 - Moderate  \* +/- 1% - Level 2 - Of Concern  \*  \* There is a separate state for when the motor is off and  \* when the system is normal.  \*/  #include "globals.h"  int computeLow(Global\* gData, int range);  int computeHigh(Global\* gData, int range);  // Compare actual speed to set speed to see if error exists  void errorCheck(Global\* gData) {  static short lastState = 4;  if (\*gData->actualSpeed == 0 || CCP4CON == 0) {  // Check if motor is off  \*gData->errorState = 3;  } else if (\*gData->actualSpeed < computeLow(gData, 5) ||  \*gData->actualSpeed > computeHigh(gData, 5)) {  // Level 0 - Severe  \*gData->errorState = 0;  // \*gData->setSpeed = 0;  // writeData(1, 0);  // CCP4CON = 0; // disable pwm output  } else if (\*gData->actualSpeed < computeLow(gData, 2) ||  \*gData->actualSpeed > computeHigh(gData, 2)) {  // Level 1 - Warning  \*gData->errorState = 1;  //\*gData->setSpeed = 0;  //writeData(1, 0);  } else if (\*gData->actualSpeed < computeLow(gData, 1) ||  \*gData->actualSpeed > computeHigh(gData, 1)) {  // Level 2 - Attention  \*gData->errorState = 2;  //\*gData->setSpeed /= 2; // \*ourGlobal.setSpeed / 2  //writeData(1, \*gData->setSpeed);  } else {  // System is normal  \*gData->errorState = 4;  }  if (lastState != \*gData->errorState) { // send new state to local  \*gData->i2cFlag = 1;  }  lastState = \*gData->errorState;  }  // returns low bound, with floor of zero  int computeLow(Global\* gData, int range) {  if (2 \* range > \*gData->setSpeed) {  return 0;  }  return ((\*gData->setSpeed) - 2 \* range);  }  int computeHigh(Global\* gData, int range) {  return ((\*gData->setSpeed) + 2 \* range);  } |