```
/*
 * SeniorDesignI2C.c
 *
 * Created: 11/24/2014 6:54:26 PM
 * Author: Robert Bower / Alonzo Ubilla
 * Chip 2 control
 */

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include "TWI_slave.h"

//#define F_CPU 16000000UL
#define SEI()   sei()
#define SLEEP() sleep_cpu()
#define SAMPLES 10

unsigned char start_cycle = 0x1A;
unsigned char sys_mode;
unsigned char rx_data = 0, tx_data = 0, hex_command = 0;
unsigned char variable_data = 0, cooler_temp = 0, kettle_level = 0;
unsigned char VPUMP_INLET = 0, VMASH_INLET = 0, VSPARGE_INLET = 0, VSPARGE_DRAIN =
0, VMASH_DRAIN = 0, VCOOLING_INLET = 0, VCARBOY_FILL = 0;

uint8_t KETTLE_LOAD = 0;
uint8_t KETTLE_PERCENT = 0;
uint8_t KETTLE_OFFSET = 0;
uint8_t ADCREAD1 = 0;
uint8_t COOLING_LINE_TEMP = 0;
uint8_t FTEMP_COOL = 0;
uint8_t CTEMP_COOL = 0;
uint8_t ADCREAD3 = 0;
int step_nmbr = 0;
int step = 0;
int i = 0;
int k = 0;


int _ADCSample(uint8_t chnl);
int read_data(int);
int send_data(int);
void blinky(void);
void temp_control(int);
int evaluate_data(void);
void test_function(void);
void _initADC(void);
void control_loop(int);
```

```
int change_mode(int);



int main(void)
{

  unsigned char TWI_slaveAddress;

  //Slave address of chip being programmed
  TWI_slaveAddress = 0x20;

  DDRB = 0xFF;
  DDRD = 0xFF;

  //Initializes the TWI module
  TWI_Slave_Initialise((unsigned
char)((TWI_slaveAddress<<TWI_ADR_BITS)|(TRUE<<TWI_GEN_BIT)));

  SEI();

  //Starts the transceiver
  TWI_Start_Transceiver();

  _initADC();

  for (;;)
  {

        //Main Control Loop.  Code will be inserted here.
        i = read_data(i);

        if(i == 2)
        {
                step = evaluate_data();
                i = 0;
        }

        blinky();

        control_loop(step);
  }
}

int read_data(int i)
{
        if(!TWI_Transceiver_Busy())
        {
```

```
                if(TWI_statusReg.RxDataInBuf)
                {
                        switch(i)
                        {
                                case 0:
                                TWI_Get_Data_From_Transceiver(&hex_command, 1);
                                i++;
                                break;
                                case 1:
                                TWI_Get_Data_From_Transceiver(&variable_data, 1);
                                i++;
                                break;
                        }
                }
        }

        return i;
}

int evaluate_data(void)
{
        int h = hex_command;

        switch (h)
        {
                //Send start Acknowledge to raspberry pi
                case 0xA0:
                        k = 0;
                        send_data(k);
                        break;

                //Spare
                case 0xA1:

                        break;

                //Spare
                case 0xA2:

                        break;

                //Spare
                case 0xA3:

                        break;

                //Spare
                case 0xA4:
```

```
        break;

//Spare
case 0xA5:

        break;

//Spare
case 0xA6:

        break;

//Spare
case 0xA7:
        k = 1;
        send_data(k);
        break;

//Spare
case 0xA8:
        k = 2;
        PORTD = 0x03;
        break;

//Spare
case 0xA9:
        k = 3;
        send_data(k);
        break;

//Send back kettle level to Pi
case 0xB0:
        k = 4;
        send_data(k);
        break;

//Spare
case 0xB1:

        break;

//Send back cooler temperature
case 0xB2:
        k = 6;
        send_data(k);
        break;
```

```
		//Zero out the kettle level
		case 0xB3:
			KETTLE_LOAD = _ADCSample(0);
			KETTLE_OFFSET = ((double)KETTLE_LOAD - (double)47);
			break;

		//Spare
		case 0xB4:

			break;

		//Spare
		case 0xB5:

			break;

		//Spare
		case 0xB6:

			break;

		//Read all analog sensors
		case 0xB7:
			KETTLE_LOAD = _ADCSample(0);
			KETTLE_PERCENT = ((double)-2.12766*((double)KETTLE_LOAD -
(double)KETTLE_OFFSET)+(double)100);
			//KETTLE_PERCENT =
((double)(-0.0197*(double)KETTLE_LOAD)+(double)2.1674)*(double)100;
			ADCREAD1 = _ADCSample(1);
			COOLING_LINE_TEMP = _ADCSample(2);
			ADCREAD3 = _ADCSample(3);
			FTEMP_COOL =
((double)-0.0000001781*(double)COOLING_LINE_TEMP*(double)COOLING_LINE_TEMP*(double)C
OOLING_LINE_TEMP*(double)COOLING_LINE_TEMP)+((double)0.00009504*(double)COOLING_LINE
_TEMP*(double)COOLING_LINE_TEMP*(double)COOLING_LINE_TEMP)-((double)0.0192429*(doubl
e)COOLING_LINE_TEMP*(double)COOLING_LINE_TEMP)+((double)2.316097*COOLING_LINE_TEMP)+
((double)33.797378);
			CTEMP_COOL = ((double)FTEMP_COOL -
(double)32)*(double)0.5555556;
			break;

		//Initiate step 1 (heat mash water)
		case 0xC0:
			step_nmbr = 1;
			break;

		//Initiate step 2 (transfer mash water)
		case 0xC1:
```

```
        step_nmbr = 2;
        break;

//Initiate step 3 (transfer sparge water)
case 0xC2:
        step_nmbr = 3;
        break;

//Start mash soak
case 0xC3:
        step_nmbr = 4;
        break;

//Transfer wort back to kettle
case 0xC4:
        step_nmbr = 5;
        break;

//Start Boil
case 0xC5:
        step_nmbr = 6;
        break;

//Initiate cool down step
case 0xC6:
        step_nmbr = 7;
        break;

//Initiate wort drain into carboy
case 0xC7:
        step_nmbr = 8;
        break;

//End of cycle notification
case 0xC8:
        step_nmbr = 9;
        break;

//Spare
case 0xD0:

        break;

//Send back pump inlet valve status
case 0xE0:
        k = 11;
        send_data(k);
        break;
```

```
//Send back mash inlet valve status
case 0xE1:
        k = 12;
        send_data(k);
        break;

//Send back sparge inlet valve status
case 0xE2:
        k = 13;
        send_data(k);
        break;

//Send back sparge drain valve status
case 0xE3:
        k = 14;
        send_data(k);
        break;

//Send back mash drain valve status
case 0xE4:
        k = 15;
        send_data(k);
        break;

//Send back cooling inlet valve status
case 0xE5:
        k = 16;
        send_data(k);
        break;

//Send back carboy fill valve status
case 0xE6:
        k = 17;
        send_data(k);
        break;

//Spare
case 0xE7:

        break;

//Spare
case 0xE8:

        break;

//Receive system mode: Manual or Auto
```

```
                case 0xF0:
                        sys_mode = 0x01; //Automatic mode
                        break;

                case 0xF1:
                        sys_mode = 0x02; //Manual mode
                        break;
        }
        return step_nmbr;
}

int send_data(int k)
{
        if(!TWI_Transceiver_Busy())
        {
                switch (k)
                {
                        case 0:
                        TWI_Start_Transceiver_With_Data(&start_cycle, 1);
                        break;

                        case 1:
                        //Spare
                        break;

                        case 2:
                        //Spare
                        break;

                        case 3:
                        //Spare
                        break;

                        //Send back Kettle level
                        case 4:
                        TWI_Start_Transceiver_With_Data(&KETTLE_PERCENT, 1);
                        break;

                        case 5:
                        TWI_Start_Transceiver_With_Data(&ADCREAD1, 1);
                        break;

                        //Send back cooler temperature
                        case 6:
                        TWI_Start_Transceiver_With_Data(&CTEMP_COOL, 1);
                        break;

                        case 7:
```

```
TWI_Start_Transceiver_With_Data(&ADCREAD3, 1);
break;

case 8:
//Spare
break;

case 9:
//Spare

break;

case 10:
//Spare
break;


case 11:
TWI_Start_Transceiver_With_Data(&VPUMP_INLET, 1);
break;


case 12:
TWI_Start_Transceiver_With_Data(&VMASH_INLET, 1);
break;


case 13:
TWI_Start_Transceiver_With_Data(&VSPARGE_INLET, 1);
break;


case 14:
TWI_Start_Transceiver_With_Data(&VSPARGE_DRAIN, 1);
break;


case 15:
TWI_Start_Transceiver_With_Data(&VMASH_DRAIN, 1);
break;


case 16:
TWI_Start_Transceiver_With_Data(&VCOOLING_INLET, 1);
break;


case 17:
```

```
                        TWI_Start_Transceiver_With_Data(&VCARBOY_FILL, 1);
                        break;


                        case 18:
                        //Spare
                        break;


                        case 19:
                        //Spare
                        break;
                }
        }
}

void control_loop(int step)
{
        switch(step)
        {
                //Preheat for mash
                case 1:
                        //Idle during step 1 (heat mash water)
                        break;

                //Transfer water to mash tun
                case 2:
                        PORTD = 0x03; //Open kettle valve and mash valve
                        VPUMP_INLET = 0x01;
                        VMASH_INLET = 0x01;
                        break;

                //Transfer water to sparge tank
                case 3:
                        PORTD = 0x05; //Open valve to sparge tank
                        VMASH_INLET = 0x00;
                        VPUMP_INLET = 0x01;
                        VSPARGE_INLET = 0x01;
                        break;

                //Start mash (soak grain/timed)
                case 4:
                        PORTD = 0x00;
                        VPUMP_INLET = 0x00;
                        VSPARGE_INLET = 0x00;
                        break;

                //Transfer Wort back to kettle
```

```
                case 5:
                        PORTD = 0x18;
                        VSPARGE_DRAIN = 0x01;
                        VMASH_DRAIN = 0x01;
                        break;

                //Begin Boil(timed step)
                case 6:
                        PORTD = 0x00;
                        VSPARGE_DRAIN = 0x00;
                        VMASH_DRAIN = 0x00;
                        break;

                //Begin cooling step
                case 7:
                        PORTD = 0x21;
                        VPUMP_INLET = 0x01;
                        VCOOLING_INLET = 0x01;
                        break;

                //Transfer to Carboy
                case 8:
                        PORTD = 0x40;
                        VPUMP_INLET = 0x00;
                        VCOOLING_INLET = 0x00;
                        VCARBOY_FILL = 0x01;
                        break;

                case 9:
                        PORTD = 0x00;
                        VCARBOY_FILL = 0x00;
                        break;
        }
}

void blinky(void)
{
        if(TWI_statusReg.RxDataInBuf)
        {
                PORTB ^= 0x01;
        }
}

void temp_control(int setpoint)
{

}
```

```
void _initADC(){
        /*
        ADLAR is in the ADMUX Register, bit 5.
        Setting ADLAR = 0 stores 8 MSB's in ADCH (ADC High Byte)
        and the 2 LSB's in bit 7 and 6 of ADCL.
           15    14    13    12    11    10    9     8
        |  -  |  -  |  -  |  -  |  -  |  -  | ADC9| ADC8|              <---ADCH
        | ADC7| ADC6| ADC5| ADC4| ADC3| ADC2| ADC1| ADC0|              <---ADCL
           7     6     5     4     3     2     1     0

        Setting ADLAR = 1 stores 8 LSB's in ADCL (ADC Low Byte)
        and the 2 MSB's in bit 1 and 0 of ADCH.
           15    14    13    12    11    10    9     8
        | ADC9| ADC8| ADC7| ADC6| ADC5| ADC4| ADC3| ADC2|               <---ADCH
        | ADC1| ADC0|  -
 |  -  |  -  |  -  |  -  |  -  |                <---ADCL
           7     6     5     4     3     2     1     0
        More info on Pg.265
        */

        ADMUX |= (1<<ADLAR);    //Turn on left adujust so that ADCH contains the 8
High bits of the output.
        //ADMUX &= ~(1<<ADLAR); //Turn on right adjust so that ADCH contains 2 high
bits, ADCL contains 8 low bits.
        ADMUX &= ~(1<<MUX0 | 1<<MUX1 | 1<<MUX2); //Set bits 0, 1, and 2, to 0
enabling ADC0.
        ADMUX &= ~(1<<REFS1);
        ADMUX |= 1<<REFS0;              //Turn off bit 7 and on bit 6 of ADMUX to
enable the AVCC REF.
        ADCSRA |= (1<<ADPS0 | 1<<ADPS1 | 1<<ADPS2);     //Turn on bit 2/1/0 of ADC
Control Register to set prescaler to 128.
        sei();                  //Pg.264 Enable Global Interrupts.
        //ADCSRA |= (1<<ADATE); //Turn on bit 5 of ADC Control Register, ADC Auto
Trigger.
        //ADCSRA |= (1<<ADIF);  //Turn on bit 3 of ADC Control Register, enabling
ADC interrupts.
        ADCSRA |= (1<<ADEN);    //Turn on bit 7 of ADC Control register.  Enable
ADC.
                                                          //Pg.263
        ADCSRA |= (1<<ADSC);    //Turn on bit 6 of ADC Control Register (START
CONVERTING).
        _delay_ms(20);
        return;
}

int _ADCSample(uint8_t chnl){
        uint8_t i = 0;
        uint16_t val = 0;
```

```
        chnl &= 0x07;           //Make sure only bits 0, 1, and 2 hold data in
"chnl" variable.
        ADMUX = (ADMUX & 0xF8)|chnl;    //ADMUX & 0xF8 clears bits 0, 1, and 2 on
ADMUX (these are the MUX0,
                                                                        //MUX1, and
MUX2, bits that choose which ADC channel.
                                                                        //ADMUX is
then OR'd with 'chnl' to set the proper bits for the channel selected.

        //This for loop will loop through and collect samples (#define SAMPLES at
top), pausing during sata collection.
        //All of the samples are aggregated to variable 'val'.
        for (i = 0; i < SAMPLES; i++) {
                ADCSRA |= (1<<ADSC);                    //Initiate sampling
                while (bit_is_set(ADCSRA, 6));  //Pause during sampling.
                val += ADCH;                                    //Aggregate sample.
        }
        //The samples are then averaged and returned..
        val = val/SAMPLES;
        return val;
}
```