Software Section

Abstract:
The purpose of this code is to poll the sensors and then relay this data back to the main data base. This circuit is also smart enough to determine whether someone is entering or exiting.

Main class:
This class is the main code, This is a follow chart of the basic layout of the class:

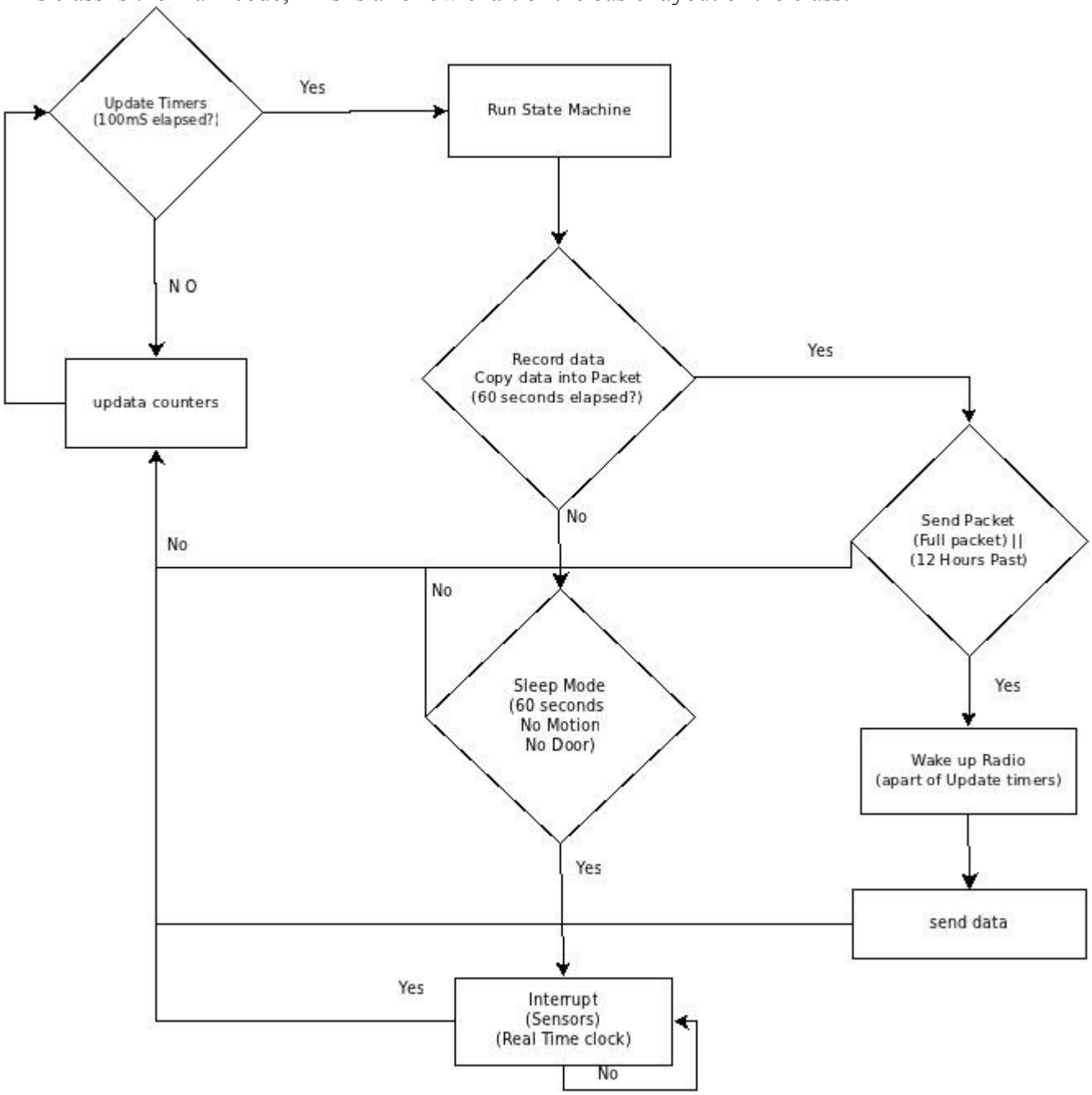

Figure 1: This is a basic layout of the main code.

This class spends most of its time idling. This is to help keep the sampling of the sensors more uniform. The first thing this code looks for is a change in the second counter or the 100m second timer elapses. After a 100m second elapse the method counter_state runs, this gets the state of each sensor. After 60

seconds of data is collected, the data is put into the packet and sent to the coordinator. For the sleep mode, after zeros for both sensors is recorded for 60seconds the node puts itself to sleep.

StateMachineCounter:

This class is design to poll the sensors, as constantly as possible, with taking the minimum amount of time possible.

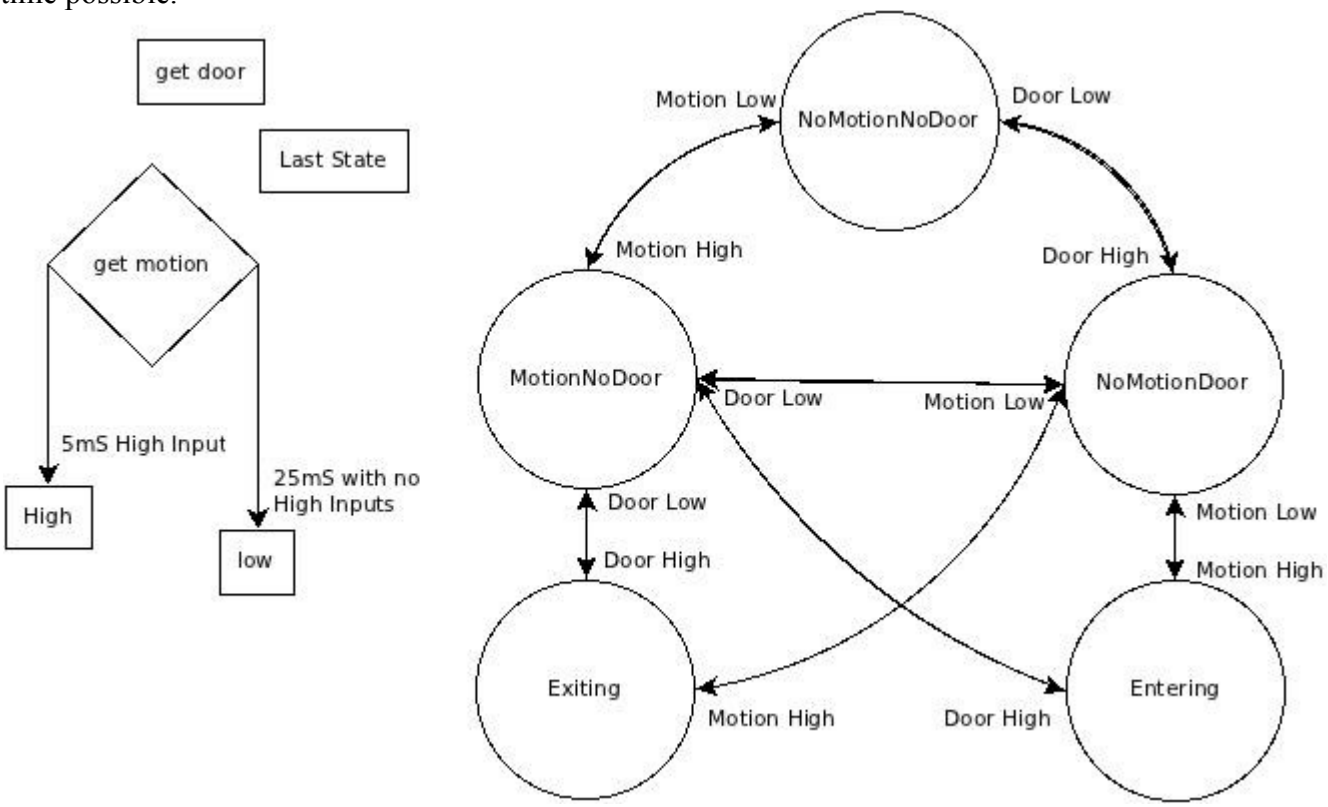

Figure 2: The StateMachineCounter, this is specifically two function counter_state() and debounce(), The left flow chart is the debounce() function and the right flow chart is the counter_state() function.

This class has three main function:

check_second_change_flag(), This function is a boolean function and will return one when 100mS has elapsed. Also this needs to be reset before it will read zero.

counter_state(), this gets the state of the sensors, this needs to be run every time check_second_change_flag() is high.

get_*(), this gets the amount of recorded time in seconds, this needs to be reset by reset_all_states().

Interrupt class:

This class is to sleep the processor, some of the methods in this class need to be in StateMachineCounter. The method sleep() calls all of the functions of the class except init().

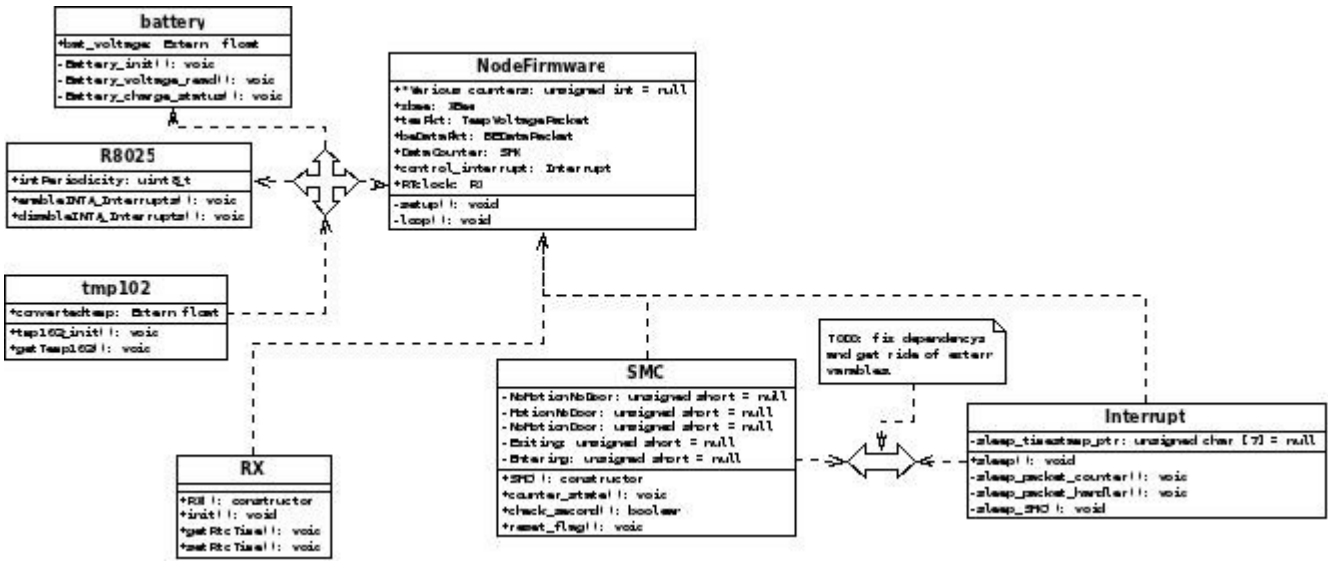

Figure 3: UML Flow chart of all of the code

Appendix 1: Further References:

Arduino IDE: http://www.arduino.cc/
Arduino Sleep code: http://www.arduino.cc/playground/Learning/ArduinoSleepCode
Seeeduino Wiki: http://seeedstudio.com/wiki/Seeeduino_Stalker_v2.0#Key_Technical_Specifications

Appendix 2: Code Classes, NodeFirmware, SMC, Interrupt :

NodeFirmware:

```
#include <Wire.h>
#include <math.h>
#include <XBee.h>
#include <TimerOne.h>

#include "Battery.h"
#include "RX8025.h"
#include "tmp102.h"

#include "Configuration.h"
#include "StateMachineCounter.h"
#include "BEDataPacket.h"
#include "ProtocolBase.h"
#include "TempVoltagePacket.h"
#include "Interrupt.h"

extern void sleep();
unsigned int counter;  // packet counter

// data packet variables
int collect_data_point_counter; // counter for creating a data point
boolean send_data_packet;  // flag to send a data packet - only set within block that prepares the packet
```

```
int send_data_packet_counter;  // counter for sending data packets
boolean initial_data_point;  // need to know which data point is first for offset purposes
boolean old_packet; // use to send yesterday's packets

// temp voltage variables
float convertedtemp;
int tmp102_val;
unsigned int bat_read;
float bat_voltage;
unsigned char charge_status;
int send_temp_packet_counter;
boolean send_temp_packet;  // flag to send a temp packet - only set within block that prepares the packet

//Configuration variables
//unsigned char KEEP_PROCESSOR_AWAKE, USE_LED;
//unsigned char DATA_POINT_INTERVAL, SLEEP_WAIT_TIME,
SEND_DATA_PACKET_INTERVAL, SEND_TEMP_PACKET_INTERVAL;

// sleep and interrupt variables
int XBee_wakeup_counter;  // counter for waiting until you can send a packet after waking up XBee
boolean XBee_wakeup; // set this flag when you wake up the XBee and leave set until XBee goes to sleep
boolean sleep_processor;  // set to 1 when you want to sleep Arduino
int sleep_counter; // counter for putting processor to sleep

// create the XBee object
XBee xbee;

// create packets
TempVoltagePacket tempPkt;
BEDataPacket beDataPkt;

// SH + SL Address of receiving XBee -- 0x00, 0x00 === Send to coordinator
XBeeAddress64 addr64(0x00, 0x00);
ZBTxRequest zbTxTemp(addr64, (uint8_t *)&tempPkt, sizeof(tempPkt));
ZBTxRequest zbTxBEData(addr64, (uint8_t *)&beDataPkt, sizeof(beDataPkt));

// Data counter object to collect data
SMC DataCounter;

// Interrupt control
Interrupt control_interrupt;

// Clock
RX RTclock;

void setup()
{
```

```
  counter = 0x10;
  XBee_wakeup = 0;
  XBee_wakeup_counter = 0;
  xbee.begin(9600);

  Battery_init();
  RX8025.init();
  tmp102_init();
  DataCounter.SMC_init();

  collect_data_point_counter = 0;
  send_data_packet_counter = 0;
  send_temp_packet_counter = 0;
  initial_data_point = 1;
  old_packet = 0;

  pinMode(XBee_sleep_port, OUTPUT);

  if(TURN_ON_SERIAL_PORT) Serial.begin(9600);

  // Set the RTC
  unsigned char clock[7] = {0x00,0x00,0x00,0x00,0x00,0x00,0x0000};  //second, minute, hour, day of
week, date, month, year (BCD format)
  RTclock.setRtcTime(clock);

  if(TURN_ON_SERIAL_PORT) Serial.print("debug");
  if(TURN_ON_SERIAL_PORT) Serial.print(KEEP_PROCESSOR_AWAKE,DEC);
  if(TURN_ON_SERIAL_PORT) Serial.print(USE_LED,DEC);

}

void loop()
{
  // increment one-second counters
  if(DataCounter.check_second_change_flag() == 1)  // DataCounter.change_flag == 1 after each timer
interrupt
  {
   collect_data_point_counter++;
   send_data_packet_counter++;
   send_temp_packet_counter++;
   if(! ROUTER) DataCounter.counter_state(); // read sensor inputs, increment state counters
   if(send_data_packet || send_temp_packet) XBee_wakeup_counter++;
   DataCounter.reset_flag(); // DataCounter.change_flag = 0
  }

  // save data to data point
  if((collect_data_point_counter/TIME_DIVISOR) >= (DATA_POINT_INTERVAL*60) && !
ROUTER)
  {
```

```
    if(initial_data_point)  // first data point in packet
    {
      RX8025.getRtcTime(beDataPkt.getTimestamp().getTS());
      RX8025.getRtcTime(beDataPkt.getSeqTimestamp().getTS());
    }
    BEDataPoint dp;

    dp.offset = DataCounter.get_offset();
    dp.timeExiting = DataCounter.get_Exiting();
    dp.timeEntering = DataCounter.get_Entering();
    dp.timeMotionNoDoor = DataCounter.get_MotionNoDoor();
    dp.timeDoorNoMotion = DataCounter.get_NoMotionDoor();
    dp.timeNoMotion = DataCounter.get_NoMotionNoDoor();

    beDataPkt.addDatapoint(dp);
    collect_data_point_counter = 0;
    DataCounter.reset_all_states();
    initial_data_point = 0;
    if(WRITE_SERIAL_DATAPOINTS)  DataCounter.print_datapoint(dp);
  }

  // prepare to send data packet
  if((((double)send_data_packet_counter/TIME_DIVISOR/60 >=
SEND_DATA_PACKET_INTERVAL) || beDataPkt.isFull() ||
    old_packet) && ! beDataPkt.isEmpty() && ! send_data_packet && USE_DATA_PACKET && !
ROUTER)
  {
    if(! XBee_wakeup)
    {
      digitalWrite(XBee_sleep_port,LOW);
      XBee_wakeup = 1;
    }
    send_data_packet = 1;
    send_data_packet_counter = 0;
  }

  // prepare to send temp packet
  if((((double)send_temp_packet_counter/TIME_DIVISOR/60 >=
SEND_TEMP_PACKET_INTERVAL)  || old_packet)
    && ! send_temp_packet && USE_TEMP_VOLTAGE_PACKET)
  {
    getTemp102();
    RX8025.getRtcTime(tempPkt.getTimestamp().getTS());
    tempPkt.setTemp(convertedtemp);
    tempPkt.setBatteryVoltage(bat_voltage);

    if(! XBee_wakeup)
    {
      digitalWrite(XBee_sleep_port,LOW);
```

```
    XBee_wakeup = 1;
   }
   send_temp_packet = 1;
   send_temp_packet_counter = 0;
   old_packet = 0;
  }

  // send packets
  if((send_data_packet || send_temp_packet) && ((double)XBee_wakeup_counter/TIME_DIVISOR >=
XBEE_WAKEUP_TIME))
  {
   // send data packet
   if(send_data_packet)
   {
    beDataPkt.setCounter(counter++);
    if(WRITE_SERIAL_DATAPOINTS)  Serial.print("---------- Data Packet ---------\n");
    /* To do
     * scroll through data points to send multiple packets - use data point pointer to keep track
     */
    xbee.send(zbTxBEData);
    beDataPkt.reset();
    if(WRITE_SERIAL_DATAPOINTS)  Serial.print("\n--------------------------------\n\n\n");

    send_data_packet = 0;
    initial_data_point = 1; // next data point is first in packet
   }

   // send temp voltage packet
   if(send_temp_packet)
   {
    tempPkt.setCounter(counter++);
    if(WRITE_SERIAL_DATAPOINTS)  Serial.print("\n\n----- Temp Voltage Packet ------\n");
    xbee.send(zbTxTemp);  // send temp voltage packet
    if(WRITE_SERIAL_DATAPOINTS)  Serial.print("\n--------------------------------\n");

    send_temp_packet = 0;
   }

   // To-do: check if there is an incoming packet
   // check xbee class functions for incoming data
   // put XBee to sleep
   delay(5);  // wait 5 msec after XBee sends data
   digitalWrite(XBee_sleep_port,HIGH);
   XBee_wakeup_counter = 0;
   XBee_wakeup = 0;
  }

  if(ROUTER)
   sleep_processor = 1;
```

```
  // put processor to sleep
  if(send_data_packet == 0 && send_temp_packet == 0 && sleep_processor == 1 && !
KEEP_PROCESSOR_AWAKE)
     control_interrupt.sleep();
}
```

StateMachineCounter:

```
#include "WProgram.h"
#include "BEDataPacket.h"

#ifndef StateMachineCounter_h
#define StateMachineCounter_h
#define logic_high 1
#define logic_low 0
#define interrupt0 0
#define interrupt1 1

class SMC
{
          public:
            SMC();
        void set_time(int c_sec,int c_min,int c_hr);
        void counter_state();
        void SMC_init();
        unsigned char get_NoMotionNoDoor();
        unsigned char get_MotionNoDoor();
        unsigned char get_NoMotionDoor();
        unsigned char get_Exiting();
        unsigned char get_Entering();
        boolean check_second_change_flag();
        void reset_flag();
        short unsigned int get_offset();
        void reset_offset();
        void reset_all_states();
        void timer_init();
        void print_datapoint(BEDataPoint dp);
          private:
        unsigned short NoMotionNoDoor, MotionNoDoor, NoMotionDoor, Exiting, Entering;
        static void current_time();
           void set_local_varables();
        boolean debounce(char port);
};

extern int collect_data_point_counter;
extern int send_temp_packet_counter;
extern int send_data_packet_counter;
extern int sleep_counter;
```

```
extern boolean sleep_processor;
extern BEDataPacket beDataPkt;
extern boolean old_packet;
extern boolean initial_data_point;

#endif

/*
*Author: Kevin Premo, et al.
*Program: Motion interupt
*/

#include "Configuration.h"
#include "StateMachineCounter.h"
#include "WProgram.h"
#include "BEDataPacket.h"
#include "RX8025.h"
#include "R8025.h"
#include <TimerOne.h>

R8025 RTC;
static bool change_flag = 0;
unsigned char * packet_timestamp_ptr;
unsigned char current_timestamp_ptr [7];

SMC::SMC()
{}

/*
* sets up the controller
*/
void SMC::SMC_init()
{
 pinMode(motion_sensor1_port,INPUT);
 pinMode(door_sensor1_port,INPUT);
 pinMode(sensor_interrupt_port,INPUT);

 digitalWrite(clock_interrupt_port,HIGH);//interrupt code
 pinMode(clock_interrupt_port,INPUT);//interrupt code
 RTC.begin(); //interrupt code
 RTC.enableINTA_Interrupts(RTC_INTERRUPT); //interrupt at  EverySecond, EveryMinute,
EveryHour or EveryMonth

 pinMode(latch_control_port,OUTPUT); //interrupt code
 digitalWrite(latch_control_port,HIGH); //interrupt code

 if(USE_LED)
 {
   pinMode(LED_port,OUTPUT);
```

```
  digitalWrite(LED_port,HIGH);
 }

 // initialize timer interrupt (smc timer this is to give a simi arcrate sampling of the sensors)
 Timer1.initialize(1000000/TIME_DIVISOR);
 Timer1.attachInterrupt(&current_time,1000000/TIME_DIVISOR);

}

void SMC::timer_init()
{
 Timer1.attachInterrupt(&current_time,1000000/TIME_DIVISOR);
}

/*
* this happends every second to update time
*/
void SMC::current_time()
{
 change_flag = 1;
}

boolean SMC::check_second_change_flag()
{
 return change_flag;
}

void SMC::reset_flag()
{
 change_flag = 0;
}

/*
* This is design to take 25ms, the motion signal with all set to low for 50ms is defined as a low signal. Anything else is considered high.
* This function is to minimise the bouncing signal miss read. The sensor has a 500mS gate delay (from movement to digital high) and 10ms min signal length (if triggered)
* How long the input is high is unpredictable, but between 10ms to several seconds. (large motion sensor,pg 33, Q15)
*/
boolean SMC::debounce(char port)
{
  boolean tmp_port = 0;
  for(char debounce_time = 0; debounce_time != 5; debounce_time++)
  {
    delay(5);
    tmp_port = digitalRead(port);
   if(logic_high == tmp_port)
     return tmp_port;
```

```
  }
  return tmp_port;
}

/*
* This caculates what state the door and motion sensor are in and increments the counter for the
respected state.
*/
void SMC::counter_state()
{
  boolean current_motion;
  boolean current_door;
  static int state;
  current_motion = debounce(motion_sensor1_port);
  current_door = digitalRead(door_sensor1_port);

  if(state == 3 && current_motion == logic_high && current_door == logic_high)
  {
   if(WRITE_SERIAL_SENSORS && collect_data_point_counter%TIME_DIVISOR == 0)
Serial.print("Exiting and motion \n");
   Exiting++;
   state = 3;
   set_local_varables();
  }
  else if(state == 4 && current_motion == logic_high && current_door == logic_high)
  {
   if(WRITE_SERIAL_SENSORS && collect_data_point_counter%TIME_DIVISOR == 0)
Serial.print("Entering and motion\n");
   Entering++;
   state = 4;
   set_local_varables();
  }
  else if(current_motion == logic_low && current_door == logic_low)
  {
   if(WRITE_SERIAL_SENSORS && collect_data_point_counter%TIME_DIVISOR == 0)
Serial.print("no motion, no door \n");
   NoMotionNoDoor++;
   state = 0;
   sleep_counter++;
   digitalWrite(latch_control_port,HIGH); // latch first sensor detected (?)
  }
  else if(current_motion == logic_high && current_door == logic_low)
  {
   if(WRITE_SERIAL_SENSORS && collect_data_point_counter%TIME_DIVISOR == 0)
Serial.print("motion, no door \n");
   MotionNoDoor++;
   state = 3;
   set_local_varables();
  }
```

```
  else if(current_motion == logic_low && current_door == logic_high)
  {
   if(WRITE_SERIAL_SENSORS && collect_data_point_counter%TIME_DIVISOR == 0)
Serial.print("no motion, door \n");
   NoMotionDoor++;
   state = 4;
   set_local_varables();
  }
  else // motion and door at the same time - didn't see which was first - just count motion
  {
    if(WRITE_SERIAL_SENSORS && collect_data_point_counter%TIME_DIVISOR == 0)
Serial.print("Error - motion and door - didn't see which was first. \n");
    state = 0;
    set_local_varables();
  }

  // check if it is time to go to sleep
  if((sleep_counter/TIME_DIVISOR) >= SLEEP_WAIT_TIME)
    sleep_processor = 1;
}

void SMC::set_local_varables()
{
    sleep_counter = 0;
    sleep_processor = 0;
    digitalWrite(latch_control_port,LOW); // pass inputs through
}

/*
* gets the NoMotionNoDoor value and sets it to 0
*/
unsigned char SMC::get_NoMotionNoDoor()
{
 return round((double)NoMotionNoDoor/TIME_DIVISOR);
}

/*
* gets the MotionNoDoor value and sets it to 0
*/
unsigned char SMC::get_MotionNoDoor()
{
 return round((double)MotionNoDoor/TIME_DIVISOR);
}

/*
* gets the NoMotionDoor value and sets it to 0
*/
unsigned char SMC::get_NoMotionDoor()
{
```

```
 return round((double)NoMotionDoor/TIME_DIVISOR);
}

/*
* gets the Exiting value and sets it to 0
*/
unsigned char SMC::get_Exiting()
{
 return round((double)Exiting/TIME_DIVISOR);
}

/*
* gets the Entering value and sets it to 0
*/
unsigned char SMC::get_Entering()
{
 return round((double)Entering/TIME_DIVISOR);
}

/*
* return time offset - time difference between packet time and datapoint time
*/
short unsigned int SMC::get_offset()
{
  if(initial_data_point == 1)
    return 0;
  else
  {
    packet_timestamp_ptr = beDataPkt.getSeqTimestamp().getTS();
    RX8025.getRtcTime(current_timestamp_ptr);

    return ( ((current_timestamp_ptr[2] - packet_timestamp_ptr[2]) * 60 * 60) +
    ((current_timestamp_ptr[1] - packet_timestamp_ptr[1]) * 60) +
    ((current_timestamp_ptr[0] - packet_timestamp_ptr[0])) ); // find out how long you have been asleep
and add it to counters
  }
}

/*
* Reset all of the states: NoMotionNoDoor, MotionNoDoor, NoMotionDoor, Exiting, Entering.
*/
void SMC::reset_all_states()
{
  NoMotionNoDoor = MotionNoDoor = NoMotionDoor = Exiting = Entering = 0;
}

/*
* Output the data point that when into a packet, for testing only.
*/
```

```
void SMC::print_datapoint(BEDataPoint dp)
{
  packet_timestamp_ptr = beDataPkt.getSeqTimestamp().getTS();
  RX8025.getRtcTime(current_timestamp_ptr);

  Serial.print("\n\n---------- Data Point ----------");
  Serial.print("\nPacket timestamp = ");
  for(int i=6; i>=0; i--)
  {
    Serial.print((unsigned int) packet_timestamp_ptr[i]);
    Serial.print(":");
  }
  Serial.print("\nCurrent timestamp = ");
  for(int i=6; i>=0; i--)
  {
    Serial.print((unsigned int) current_timestamp_ptr[i]);
    Serial.print(":");
  }
  Serial.print("\nTime offset = ");
  Serial.print(dp.offset);
  Serial.print("\nTime exiting = ");
  Serial.print((unsigned int)dp.timeExiting);
  Serial.print("\nTime entering = ");
  Serial.print((unsigned int)dp.timeEntering);
  Serial.print("\nTime with motion but no door = ");
  Serial.print((unsigned int)dp.timeMotionNoDoor);
  Serial.print("\nTime with door but no motion = ");
  Serial.print((unsigned int)dp.timeDoorNoMotion);
  Serial.print("\nTime with no motion and no door = ");
  Serial.print((unsigned int)dp.timeNoMotion);
  Serial.print("\n--------------------------------\n\n\n");
}
```

Interrupt:

```
#ifndef Interrupt_h
#define Interrupt_h

#include "Configuration.h"
#include "StateMachineCounter.h"
#include "BEDataPacket.h"
#include "RX8025.h"
#include "R8025.h"
#include <TimerOne.h>
#include <avr/sleep.h>

#define interrupt0 0
#define interrupt1 1
```

```
  //attachInterrupt function looks for function in the global namespace.

class Interrupt
{
          public:
          Interrupt();
          void sleep();
        void init();


        protected:
        unsigned char sleep_timestamp_ptr [7];
        void sleep_packet_counter();
        void sleep_packet_handler();
        void sleep_SMC();

};

extern SMC DataCounter;
extern int sleep_counter;
extern boolean sleep_processor;
extern BEDataPacket beDataPkt;
extern boolean old_packet;
extern unsigned char * packet_timestamp_ptr;
extern unsigned char current_timestamp_ptr [7];
extern int collect_data_point_counter;
extern int send_data_packet_counter;
extern int send_temp_packet_counter;
extern R8025 RTC;
extern boolean RTC_interrupt;

#endif

#include "Configuration.h"
#include "StateMachineCounter.h"
#include "BEDataPacket.h"
#include "RX8025.h"
#include "R8025.h"
#include <TimerOne.h>
#include <avr/sleep.h>
#include "Interrupt.h"

  extern void sensor_interupt_ISR() {} //sensor interupt function
  extern void clock_interupt_ISR() {}  //rtc interupt function

Interrupt::Interrupt()
{
}
```

```
/*
* Initiate the object
*/
void Interrupt::init()
{

}

/*
* set the microcontroller to sleep.
*/
void Interrupt::sleep()
{
  RX8025.getRtcTime(sleep_timestamp_ptr);
  Timer1.detachInterrupt();
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
  if(WRITE_INTERRUPT_INFO) Serial.print("\nProcessor going to sleep.\n");
  if(USE_LED)  digitalWrite(LED_port,LOW);
  digitalWrite(latch_control_port,HIGH);
  RTC.refreshINTA();
  sleep_enable();
  attachInterrupt(interrupt0, clock_interupt_ISR, LOW);
  if(! ROUTER) attachInterrupt(interrupt1, sensor_interupt_ISR, RISING);
  // if the sensor interrupt goes off between attaching the interrupt and sleep mode finishing, that
interrupt is locked out until the RTC interrupt occurs.
  // The same does not apply to RTC interrupt occuring between attaching the interrupt and sleep mode
finishing.
  sleep_mode();

  // Processor is sleeping //

  sleep_disable();
  detachInterrupt(interrupt0);
  if(! ROUTER) detachInterrupt(interrupt1);
  if(WRITE_INTERRUPT_INFO)  Serial.print("\nProcessor waking up.\n\n");
  if(USE_LED)  digitalWrite(LED_port,HIGH);

  sleep_SMC();
  sleep_packet_handler();
  sleep_packet_counter();
  digitalWrite(latch_control_port,LOW); // read sensors
}

/*
* This restarts the SMC to poll the sensors again
*/
void Interrupt::sleep_SMC()
{
  DataCounter.timer_init();
```

```
  sleep_processor = 0;
  sleep_counter = 0;
  collect_data_point_counter = 0;
  DataCounter.reset_all_states();

  // see who woke up the processor
  bool current_motion = digitalRead(motion_sensor1_port);
  bool current_door = digitalRead(door_sensor1_port);
  if(current_motion == logic_high || current_door == logic_high) // Sensors woke up processor
    DataCounter.counter_state();
}

/*
* Get packet timestamp - if it was from yesterday, send packet
* This is to make sure that the offset in BEDataPacket is not overflow.
*/
void Interrupt::sleep_packet_handler()
{
  packet_timestamp_ptr = beDataPkt.getSeqTimestamp().getTS();
  RX8025.getRtcTime(current_timestamp_ptr);
  if(current_timestamp_ptr[4] != packet_timestamp_ptr[4] || (current_timestamp_ptr[2] -
packet_timestamp_ptr[2] > 12))
  {
    // send data packet from yesterday or if more than 12 hours old
    old_packet = 1;
  }
}

/*
* find out how long you have been asleep and add it to counters
*/
void Interrupt::sleep_packet_counter()
{
  int sleep_time = ( ((current_timestamp_ptr[2] - sleep_timestamp_ptr[2]) * 60 * 60) +
    ((current_timestamp_ptr[1] - sleep_timestamp_ptr[1]) * 60) +
    ((current_timestamp_ptr[0] - sleep_timestamp_ptr[0])) ); //caculating sleep time

   send_temp_packet_counter += (sleep_time * TIME_DIVISOR);
   send_data_packet_counter += (sleep_time * TIME_DIVISOR);

}
```