```c
/*******************************************************************************
 Module
    MasterHSM.c

 Description
    This is a template for the top level Hierarchical state machine

*******************************************************************************/
/*---------------------------- Include Files ----------------------------*/
/* include header files for this state machine as well as any machines at the
   next lower level in the hierarchy that are sub-machines to this machine
*/
#include "ES_Configure.h"
#include "ES_Framework.h"
#include <stdio.h>

#include "MasterHSM.h"
#include "Round1HSM.h"
#include "Intermission1HSM.h"
#include "Round2HSM.h"
#include "Intermission2HSM.h"
#include "Round3HSM.h"
#include "Intermission3HSM.h"
#include "SuddenDeathHSM.h"

/*---------------------------- Module Defines ----------------------------*/

/*---------------------------- Module Functions ----------------------------*/
static ES_Event DuringIdleHSM(ES_Event Event);
static ES_Event DuringRound1HSM(ES_Event Event);
static ES_Event DuringIntermission1HSM(ES_Event Event);
static ES_Event DuringRound2HSM(ES_Event Event);
static ES_Event DuringIntermission2HSM(ES_Event Event);
static ES_Event DuringRound3HSM(ES_Event Event);
static ES_Event DuringIntermission3HSM(ES_Event Event);
static ES_Event DuringSuddenDeathHSM(ES_Event Event);

void InitializeHardware(ES_Event CurrentEvent);
/*---------------------------- Module Variables ----------------------------*/
// everybody needs a state variable, though if the top level state machine
// is just a single state container for orthogonal regions, you could get
// away without it
static MasterState_t CurrentState;

// with the introduction of Gen2, we need a module level Priority var as well
static uint8_t MyPriority;

static unsigned char strategy_mode = 0;

/*---------------------------- Module Code ----------------------------*/
/*******************************************************************************
 Function
    InitMasterHSM

 Parameters
    uint8_t : the priorty of this service
```

```
 Returns
    boolean, false if error in initialization, true otherwise

 Description
    Saves away the priority,  and starts the top level state machine

*****************************************************************************/
bool InitMasterHSM(uint8_t Priority)
{
    ES_Event ThisEvent;

        #ifdef PRINT_MASTER_HSM_CALLS

                printf("InitMasterHSM\n\r");

        #endif

    MyPriority = Priority;  // save priority

    // Initialize Services
    //InitJSRService(); // initialize JSR service
    //InitMotorService(); // initialize motor service
    //InitNavigationSensorService(); // initialize navigation sensor service
    //InitKnightSensorService(); // initialize knight sensor service
    //InitGoalSensorService(); // initialize goal sensor service

    // Start MasterHSM
    ThisEvent.EventType = ES_ENTRY;
    StartMasterHSM(ThisEvent);

    return true;
}

/*****************************************************************************
 Function
    PostMasterHSM

 Parameters
    ES_Event ThisEvent , the event to post to the queue

 Returns
    boolean false if the post operation failed, true otherwise

 Description
    Posts an event to this state machine's queue

*****************************************************************************/
bool PostMasterHSM(ES_Event ThisEvent)
{

        #ifdef PRINT_MASTER_HSM_CALLS

                printf("PostMasterHSM\n\r");

        #endif
```

```
        return ES_PostToService(MyPriority,ThisEvent);
}

/****************************************************************************
 Function
    RunMasterHSM

 Parameters
    ES_Event: the event to process

 Returns
    ES_Event: an event to return

 Description
    the run function for the top level state machine

 Notes
   uses nested switch/case to implement the machine.

****************************************************************************/
// make recursive call warning into info
#pragma MESSAGE INFORMATION C1855

ES_Event RunMasterHSM(ES_Event CurrentEvent)
{
    unsigned char MakeTransition = false;
    MasterState_t NextState = CurrentState;
    ES_Event EntryEventKind = {ES_ENTRY,0};
        ES_Event PostEvent;
        ES_Event ReturnEvent;

        #ifdef PRINT_MASTER_HSM_CALLS

                printf("RunMasterHSM\n\r");

        #endif

    switch(CurrentState)
    {
        case Idle:


                        #ifdef PRINT_MASTER_HSM_STATES

                                printf("Idle\n\r");

                        #endif

                        // Execute During function for state. ES_ENTRY &
ES_EXIT are processed here allow the lower leve state machines to re-map or
consume the event
                        CurrentEvent = DuringIdleHSM(CurrentEvent);
                        // Process any events
                        if(CurrentEvent.EventType != ES_NO_EVENT)
                        {
                                switch(CurrentEvent.EventType)
                                {
```

```
                                        case pas_d_armes:
                                                #ifdef PRINT_MASTER_HSM_EVENTS

        printf("pas_d_armes\n\r");

                                                #endif

                                                NextState = Round1;
                                                MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;
                                                break;
                                    }

                            }
                            break;


        case Round1:

                            #ifdef PRINT_MASTER_HSM_STATES

                                    printf("Round1\n\r");

                            #endif

            // Execute During function for state. ES_ENTRY & ES_EXIT are
processed here allow the lower leve state machines to re-map or consume the
event
            CurrentEvent = DuringRound1HSM(CurrentEvent);
            // Process any events
            if (CurrentEvent.EventType != ES_NO_EVENT)
            {
                switch(CurrentEvent.EventType)
                {
                    case recess:
                                                #ifdef PRINT_MASTER_HSM_EVENTS

                                                        printf("recess\n\r");

                                                #endif

                        NextState = Intermission1;
                        MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;
                        break;

                    case end_of_match:
                                                #ifdef PRINT_MASTER_HSM_EVENTS
```

```
        printf("end_of_match\n\r");

                                                #endif

                        NextState = Idle;
                        MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;
                        break;

                    case unhorsed:

                                                #ifdef PRINT_MASTER_HSM_EVENTS

                                                        printf("unhorsed\n\r");

                                                #endif

                        NextState = Idle;
                        MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;
                        break;
                }
            }
            break;

        case Intermission1:

                            #ifdef PRINT_MASTER_HSM_STATES


        printf("Intermission1\n\r");

                            #endif

                    // Execute During function for state. ES_ENTRY &
ES_EXIT are processed here allow the lower leve state machines to re-map or
consume the event
                        CurrentEvent = DuringIntermission1HSM(CurrentEvent);
            // no during function
                        if (CurrentEvent.EventType != ES_NO_EVENT)
                        {
                                switch(CurrentEvent.EventType)
                                {
                                        case pas_d_armes:

                                                #ifdef PRINT_MASTER_HSM_EVENTS


        printf("pas_d_armes\n\r");

                                                #endif
```

```
                                                NextState = Round2;
                                                MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;
                                                break;

                                        case unhorsed:

                                                #ifdef PRINT_MASTER_HSM_EVENTS

                                                        printf("unhorsed\n\r");

                                                #endif

                                NextState = Idle;
                                MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;
                                break;

                                        case end_of_match:

                                                #ifdef PRINT_MASTER_HSM_EVENTS


        printf("end_of_match\n\r");

                                                #endif

                                                NextState = Idle;
                                                MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;
                                                break;
                                        }
                                }
                        break;

                case Round2:

                                #ifdef PRINT_MASTER_HSM_STATES

                                        printf("Round2\n\r");

                                #endif

                        // Execute During function for state. ES_ENTRY & ES_EXIT are
processed here allow the lower leve state machines to re-map or consume the
event
                        CurrentEvent = DuringRound2HSM(CurrentEvent);
                        // Process any events
                        if (CurrentEvent.EventType != ES_NO_EVENT)
                        {
```

```
                switch(CurrentEvent.EventType)
                {
                    case recess:

                                                #ifdef PRINT_MASTER_HSM_EVENTS

                                                        printf("recess\n\r");

                                                #endif

                        NextState = Intermission2;
                        MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;
                        break;

                    case end_of_match:

                                                #ifdef PRINT_MASTER_HSM_EVENTS


        printf("end_of_match\n\r");

                                                #endif

                        NextState = Idle;
                        MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;
                        break;

                    case unhorsed:

                                                #ifdef PRINT_MASTER_HSM_EVENTS

                                                        printf("unhorsed\n\r");

                                                #endif

                        NextState = Idle;
                        MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;
                        break;
                }
            }
            break;

        case Intermission2:

                        #ifdef PRINT_MASTER_HSM_STATES

                                printf("Intermission2\n\r");
```

```
                    #endif

            // Execute During function for state. ES_ENTRY & ES_EXIT are
processed here allow the lower leve state machines to re-map or consume the
event
            CurrentEvent = DuringIntermission2HSM(CurrentEvent);
            // Process any events
            if (CurrentEvent.EventType != ES_NO_EVENT)
            {
                switch(CurrentEvent.EventType)
                {
                    case pas_d_armes:

                                                #ifdef PRINT_MASTER_HSM_EVENTS


        printf("pas_d_armes\n\r");

                                                #endif

                                                NextState = Round3;
                        MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;

                        break;

                                        case unhorsed:

                                                #ifdef PRINT_MASTER_HSM_EVENTS

                                                        printf("unhorsed\n\r");

                                                #endif

                        NextState = Idle;
                        MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;
                        break;

                    case end_of_match:

                                                #ifdef PRINT_MASTER_HSM_EVENTS


        printf("end_of_match\n\r");

                                                #endif

                        NextState = Idle;
                        MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;
```

```
                        break;
                }
            }
            break;

        case Round3:

                        #ifdef PRINT_MASTER_HSM_STATES

                                printf("Round3\n\r");

                        #endif

            // Execute During function for state. ES_ENTRY & ES_EXIT are
processed here allow the lower leve state machines to re-map or consume the
event
            CurrentEvent = DuringRound3HSM(CurrentEvent);
            // Process any events
            if (CurrentEvent.EventType != ES_NO_EVENT)
            {
                switch(CurrentEvent.EventType)
                {
                    case recess:

                                                #ifdef PRINT_MASTER_HSM_EVENTS

                                                        printf("recess\n\r");

                                                #endif

                        NextState = Intermission3;
                        MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;
                        break;

                    case end_of_match:

                                                #ifdef PRINT_MASTER_HSM_EVENTS


        printf("end_of_match\n\r");

                                                #endif

                        NextState = Idle;
                        MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;
                        break;

                    case unhorsed:

                                                #ifdef PRINT_MASTER_HSM_EVENTS
```

```
                                                printf("unhorsed\n\r");

                                        #endif

                        NextState = Idle;
                        MakeTransition = true;
                                        EntryEventKind.EventType =
ES_ENTRY;
                                        ReturnEvent = CurrentEvent;
                        break;
                }
            }
            break;

        case Intermission3:

                        #ifdef PRINT_MASTER_HSM_STATES

                                printf("Intermission3\n\r");

                        #endif

            // Execute During function for state. ES_ENTRY & ES_EXIT are
processed here allow the lower leve state machines to re-map or consume the
event
                        CurrentEvent = DuringIntermission3HSM(CurrentEvent);
            // Process any events
                        if (CurrentEvent.EventType != ES_NO_EVENT)
                        {
                                switch(CurrentEvent.EventType)
                                {
                                        case pas_d_armes:

                                        #ifdef PRINT_MASTER_HSM_EVENTS


        printf("pas_d_armes\n\r");

                                        #endif

                                        NextState = SuddenDeath;
                                        MakeTransition = true;
                                        EntryEventKind.EventType =
ES_ENTRY;
                                        ReturnEvent = CurrentEvent;
                                        break;

                                case sudden_death:

                                        #ifdef PRINT_MASTER_HSM_EVENTS


        printf("sudden_death\n\r");

                                        #endif

                                        NextState = SuddenDeath;
```

```
                                                MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;
                                                break;

                                        case unhorsed:

                                                #ifdef PRINT_MASTER_HSM_EVENTS

                                                        printf("unhorsed\n\r");

                                                #endif

                                NextState = Idle;
                                MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;
                                break;

                                        case end_of_match:

                                                #ifdef PRINT_MASTER_HSM_EVENTS


        printf("end_of_match\n\r");

                                                #endif

                                                NextState = Idle;
                                                MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;
                                                break;
                                        }
                                }
                        break;

                case SuddenDeath:

                                #ifdef PRINT_MASTER_HSM_STATES

                                        printf("SuddenDeath\n\r");

                                #endif

                        // Execute During function for state. ES_ENTRY & ES_EXIT are
processed here allow the lower leve state machines to re-map or consume the
event
                        CurrentEvent = DuringSuddenDeathHSM(CurrentEvent);
                        // Process any events
                        if (CurrentEvent.EventType != ES_NO_EVENT)
                        {
                            switch(CurrentEvent.EventType)
                            {
```

```
                    case end_of_match:
                                                #ifdef PRINT_MASTER_HSM_EVENTS

        printf("end_of_match\n\r");
                                                #endif
                            NextState = Idle;
                            MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;
                            break;
                                        case unhorsed:
                                                #ifdef PRINT_MASTER_HSM_EVENTS
                                                        printf("unhorsed\n\r");
                                                #endif
                            NextState = Idle;
                            MakeTransition = true;
                                                EntryEventKind.EventType =
ES_ENTRY;
                                                ReturnEvent = CurrentEvent;
                            break;

                }
            }
            break;
    }

    if(MakeTransition == true)
    {
        // Execute exit function for current state
        CurrentEvent.EventType = ES_EXIT;
        RunMasterHSM(CurrentEvent);

        // Modify state variable
        CurrentState = NextState;

        // Execute entry function for new state
        RunMasterHSM(EntryEventKind);
    }

    // in the absence of an error the top level state machine should
    // always return ES_NO_EVENT
    CurrentEvent.EventType = ES_NO_EVENT;
    return(CurrentEvent);
}
/*****************************************************************************
 Function
     StartMasterHSM
```

```
 Parameters
     ES_Event CurrentEvent

 Returns
     nothing

 Description
     Does any required initialization for this state machine
 Notes

 Author
     J. Edward Carryer, 02/06/12, 22:15
*****************************************************************************/
void StartMasterHSM(ES_Event CurrentEvent)
{
        // local variable to get debugger to display the value of CurrentEvent
    volatile ES_Event LocalEvent = CurrentEvent;

        #ifdef PRINT_MASTER_HSM_CALLS

                printf("StartMasterHSM\n\r");

        #endif

    // if there is more than 1 state to the top level machine you will need
    // to initialize the state variable
    CurrentState = Idle;

        InitializeHardware(CurrentEvent); //call initialization functions as
necessary;

    // now we need to let the Run function init the lower level state
machines
    // use LocalEvent to keep the compiler from complaining about unused var
    RunMasterHSM(LocalEvent);

    return;
}


/*****************************************************************************
 private functions
 *****************************************************************************/

void InitializeHardware(ES_Event CurrentEvent) //call initialization
functions as necessary;
{

        ES_Event PostEvent;

        #ifdef PRINT_MASTER_HSM_CALLS

                printf("InitializeHardware\n\r");

        #endif
```

```
        ADS12_Init("IIIIAOOO"); //initialize analog/digital ports
        UpdateLEDs(1, 0, 0);

        StartShootingSM(CurrentEvent); //start then stop shooting service to
initialize motor port and set servo positions
        StopShootingSM(CurrentEvent);

        UpdateServo(LANCE_SERVO_CHANNEL, LANCE_SERVO_UP_POSITION);

        PostEvent.EventType = STOP;
        PostMotorService(PostEvent);

        //set strategy mode for this match
        switch (get_dipswitch())
        {
                case 0:
                        strategy_mode = 0;

                        break;

                case 1:
                        strategy_mode = 1;

                        break;

                default:
                        strategy_mode = 0;

                        break;

        }


}

 static ES_Event DuringIdleHSM(ES_Event Event)
{
    ES_Event ReturnEvent = Event; // assmes no re-mapping or comsumption
        ES_Event PostEvent;

        #ifdef PRINT_MASTER_HSM_CALLS

                printf("DuringIdleHSM\n\r");

        #endif

    // Process ES_ENTRY, ES_ENTRY_HISTORY, and ES_EXIT events
    if((Event.EventType == ES_ENTRY) || (Event.EventType ==
ES_ENTRY_HISTORY))
    {

                // implement any entry actions required for this state machine

                UpdateLEDs(1, 0, 0);  //Update LED display

                PostEvent.EventType = STOP;
```

```
                PostMotorService(PostEvent); // stop drive motors

        // repeat the StartxxxSM() functions for concurrent state machines on
the lower level
    }
    else if(Event.EventType == ES_EXIT)
    {


        // repeat for any concurrently running state machines

        // now do any local exit functionality
    }
    else // do the 'during' function for this state
    {
        // run any lower level state machine


        // repeat for any concurrent lower level machines

        // do any activity that is repeated as long as we are in this state
    }
    // return either Event, if you don't want to allow the lower level
machine
    // to remap the current event, or ReturnEvent if you do want to allow it.
    return(ReturnEvent);
}


 static ES_Event DuringRound1HSM(ES_Event Event)
{
    ES_Event ReturnEvent = Event; // assmes no re-mapping or comsumption
        ES_Event PostEvent;

        #ifdef PRINT_MASTER_HSM_CALLS

                printf("DuringRound1HSM\n\r");

        #endif

    // Process ES_ENTRY, ES_ENTRY_HISTORY, and ES_EXIT events
    if((Event.EventType == ES_ENTRY) || (Event.EventType ==
ES_ENTRY_HISTORY))
    {
        // implement any entry actions required for this state machine

        UpdateLEDs(0, 1, 0); //Update LED display

        // after that start any lower level machines that run in this state
        StartRound1HSM(Event);

        // repeat the StartxxxSM() functions for concurrent state machines on
the lower level
    }
    else if(Event.EventType == ES_EXIT)
    {
        // on exit, give the lower levels a chance to clean up first
```

```
        RunRound1HSM(Event);

                PostEvent.EventType = STOP;
                PostMotorService(PostEvent);

        // repeat for any concurrently running state machines

        // now do any local exit functionality
    }
    else // do the 'during' function for this state
    {
        // run any lower level state machine
        ReturnEvent = RunRound1HSM(Event);

        // repeat for any concurrent lower level machines

        // do any activity that is repeated as long as we are in this state
    }
    // return either Event, if you don't want to allow the lower level
machine
    // to remap the current event, or ReturnEvent if you do want to allow it.
    return(ReturnEvent);
}

static ES_Event DuringIntermission1HSM(ES_Event Event)
{
    ES_Event ReturnEvent = Event; // assmes no re-mapping or comsumption
        ES_Event PostEvent;
        #ifdef PRINT_MASTER_HSM_CALLS

                printf("DuringIntermission1HSM\n\r");

        #endif

    // Process ES_ENTRY, ES_ENTRY_HISTORY, and ES_EXIT events
    if((Event.EventType == ES_ENTRY) || (Event.EventType ==
ES_ENTRY_HISTORY))
    {
        // implement any entry actions required for this state machine

        UpdateLEDs(0, 0, 1); //Update LED display

        // after that start any lower level machines that run in this state
        StartIntermission1HSM(Event);

        // repeat the StartxxxSM() functions for concurrent state machines on
the lower level
    }
    else if(Event.EventType == ES_EXIT)
    {
        // on exit, give the lower levels a chance to clean up first
        RunIntermission1HSM(Event);

                PostEvent.EventType = STOP;
                PostMotorService(PostEvent);
```

```
        // repeat for any concurrently running state machines

        // now do any local exit functionality
    }
    else // do the 'during' function for this state
    {
        // run any lower level state machine
        ReturnEvent = RunIntermission1HSM(Event);

        // repeat for any concurrent lower level machines

        // do any activity that is repeated as long as we are in this state
    }
    // return either Event, if you don't want to allow the lower level
machine
    // to remap the current event, or ReturnEvent if you do want to allow it.
    return(ReturnEvent);
}

static ES_Event DuringRound2HSM(ES_Event Event)
{
    ES_Event ReturnEvent = Event; // assmes no re-mapping or comsumption
    ES_Event PostEvent;

        #ifdef PRINT_MASTER_HSM_CALLS

                printf("DuringRound2HSM\n\r");

        #endif

    // Process ES_ENTRY, ES_ENTRY_HISTORY, and ES_EXIT events
    if((Event.EventType == ES_ENTRY) || (Event.EventType ==
ES_ENTRY_HISTORY))
    {
        // implement any entry actions required for this state machine

        UpdateLEDs(0, 1, 0); //Update LED display

        // after that start any lower level machines that run in this state
        StartRound2HSM(Event);

        // repeat the StartxxxSM() functions for concurrent state machines on
the lower level
    }
    else if(Event.EventType == ES_EXIT)
    {
        // on exit, give the lower levels a chance to clean up first
        RunRound2HSM(Event);

                PostEvent.EventType = STOP;
                PostMotorService(PostEvent);

        // repeat for any concurrently running state machines

        // now do any local exit functionality
    }
    else // do the 'during' function for this state
```

```
    {
        // run any lower level state machine
        ReturnEvent = RunRound2HSM(Event);

        // repeat for any concurrent lower level machines

        // do any activity that is repeated as long as we are in this state
    }
    // return either Event, if you don't want to allow the lower level
machine
    // to remap the current event, or ReturnEvent if you do want to allow it.
    return(ReturnEvent);
}

static ES_Event DuringIntermission2HSM(ES_Event Event)
{
    ES_Event ReturnEvent = Event; // assmes no re-mapping or comsumption
        ES_Event PostEvent;

        #ifdef PRINT_MASTER_HSM_CALLS

                printf("DuringIntermission2HSM\n\r");

        #endif

    // Process ES_ENTRY, ES_ENTRY_HISTORY, and ES_EXIT events
    if((Event.EventType == ES_ENTRY) || (Event.EventType ==
ES_ENTRY_HISTORY))
    {
        // implement any entry actions required for this state machine
        UpdateLEDs(0, 0, 1); //Update LED display

        // after that start any lower level machines that run in this state
        StartIntermission2HSM(Event);

        // repeat the StartxxxSM() functions for concurrent state machines on
the lower level
    }
    else if(Event.EventType == ES_EXIT)
    {
        // on exit, give the lower levels a chance to clean up first
        RunIntermission2HSM(Event);

                PostEvent.EventType = STOP;
                PostMotorService(PostEvent);

        // repeat for any concurrently running state machines

        // now do any local exit functionality
    }
    else // do the 'during' function for this state
    {
        // run any lower level state machine
        ReturnEvent = RunIntermission2HSM(Event);

        // repeat for any concurrent lower level machines
```

```
        // do any activity that is repeated as long as we are in this state
    }
    // return either Event, if you don't want to allow the lower level
machine
    // to remap the current event, or ReturnEvent if you do want to allow it.
    return(ReturnEvent);
}

static ES_Event DuringRound3HSM(ES_Event Event)
{
    ES_Event ReturnEvent = Event; // assmes no re-mapping or comsumption
        ES_Event PostEvent;

        #ifdef PRINT_MASTER_HSM_CALLS

                printf("DuringRound3HSM\n\r");

        #endif

    // Process ES_ENTRY, ES_ENTRY_HISTORY, and ES_EXIT events
    if((Event.EventType == ES_ENTRY) || (Event.EventType ==
ES_ENTRY_HISTORY))
    {
        // implement any entry actions required for this state machine
        UpdateLEDs(0, 1, 0); //Update LED display

        // after that start any lower level machines that run in this state
        StartRound3HSM(Event);

        // repeat the StartxxxSM() functions for concurrent state machines on
the lower level
    }
    else if(Event.EventType == ES_EXIT)
    {
        // on exit, give the lower levels a chance to clean up first
        RunRound3HSM(Event);

                PostEvent.EventType = STOP;
                PostMotorService(PostEvent);

        // repeat for any concurrently running state machines

        // now do any local exit functionality
    }
    else // do the 'during' function for this state
    {
    // run any lower level state machine
```