```
/*******************************************************************************
 Module
    Round1HSM.c

 Description
    This is a template file for implementing state machines.

*******************************************************************************/
/*---------------------------- Include Files ----------------------------*/
// Basic includes for a program using the Events and Services Framework
#include "ES_Configure.h"
#include "ES_Framework.h"
#include <stdio.h>

/* 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 "Round1HSM.h"


/*---------------------------- Module Defines ----------------------------*/
// define constants for the states for this machine
// and any other local defines

/*---------------------------- Module Functions ----------------------------*/
/* prototypes for private functions for this machine, things like during
   functions, entry & exit functions.They should be functions relevant to the
   behavior of this state machine
*/
static ES_Event DuringCharging1HSM(ES_Event Event);


/*---------------------------- Module Variables ----------------------------*/
// everybody needs a state variable, you may need others as well
static Round1State_t CurrentState;

/*---------------------------- Module Code ----------------------------*/
/*******************************************************************************
 Function
    RunRound1HSM

 Parameters
    ES_Event: the event to process

 Returns
    ES_Event: an event to return

 Description
    add your description here

*******************************************************************************/
ES_Event RunRound1HSM(ES_Event CurrentEvent)
{
    unsigned char MakeTransition = false; // are we making a state
transition?
    Round1State_t NextState = CurrentState;
```

```
    ES_Event EntryEventKind = {ES_ENTRY,0}; // default to normal entry to new
state
    ES_Event ReturnEvent = CurrentEvent; // assume we are not consuming event
        ES_Event PostEvent;

    #ifdef PRINT_ROUND1_HSM_CALLS

        printf("RunRound1HSM\n\r");

    #endif

    switch(CurrentState)
    {
        case Charging1:

            #ifdef PRINT_ROUND1_HSM_STATES

                printf("Charging1\n\r");

            #endif

            // Execute During function for Charging. ES_ENTRY & ES_EXIT are
processed here allow the lower level state machines to re-map or consume the
event
            CurrentEvent = DuringCharging1HSM(CurrentEvent);
            // Process any events
            if(CurrentEvent.EventType != ES_NO_EVENT) // if an event is
active
            {
                switch(CurrentEvent.EventType)
                {
                    /*case Crossed_Slow_Down_Fwd_Offset:

                        #ifdef PRINT_ROUND1_HSM_EVENTS

                            printf("Crossed_Slow_Down_Fwd_Offset\n\r");

                        #endif

                        NextState = Completing1;
                        MakeTransition = true;
                        EntryEventKind.EventType = ES_ENTRY;
                        ReturnEvent = CurrentEvent;
                        break;*/

                                        case ES_TIMEOUT :

                                                if(CurrentEvent.EventParam ==
LANCE_DELAY_TIMER)

                                                {
                                                        PostEvent.EventType =
LowerLance;

        PostMasterHSM(PostEvent);

                                                }
```

```
                                        break;


                    /*case front_bumper_pressed:

                        #ifdef PRINT_ROUND1_HSM_EVENTS

                            printf("front_bumper_pressed\n\r");

                        #endif

                        NextState = Completing1;
                        MakeTransition = true;
                        EntryEventKind.EventType = ES_ENTRY;
                        EntryEventKind.EventParam = 1; // enter into
Completing substate FindingStartline
                        ReturnEvent = CurrentEvent;
                        break;*/
                }
            }
            break;

       /*case Soccering1:

            #ifdef PRINT_ROUND1_HSM_STATES

                printf("Soccering1\n\r");

            #endif

            // Execute During function for Soccering. ES_ENTRY & ES_EXIT are
processed here allow the lowere level state machines to re-map or consume the
event
            CurrentEvent = DuringSoccering1HSM(CurrentEvent);
            // Process any events
            if(CurrentEvent.EventType != ES_NO_EVENT) // if an event is
active
            {
                switch(CurrentEvent.EventType)
                {
                    case Out_of_Ammo: // event put into Completing state

                        #ifdef PRINT_ROUND1_HSM_EVENTS

                            printf("Out_of_Ammo\n\r");

                        #endif

                        NextState = Completing1;
                        MakeTransition = true;
                        EntryEventKind.EventType = ES_ENTRY;
                        ReturnEvent = CurrentEvent;
                        break;
                }
            }
            break;*/
```

```
       /* case Completing1:

            #ifdef PRINT_ROUND1_HSM_STATES

                printf("Completing1\n\r");

            #endif

            // Execute During function for Charging. ES_ENTRY & ES_EXIT are
processed here allow the lowere level state machines to re-map or consume the
event
            CurrentEvent = DuringCompleting1HSM(CurrentEvent);
            // Process any events

            break;*/

    }

    //   If we are making a state transition
    if (MakeTransition == true)
    {
        // Execute exit function for current state
        CurrentEvent.EventType = ES_EXIT;
        RunRound1HSM(CurrentEvent);

        CurrentState = NextState; //Modify state variable

        // Execute entry function for new state
        // this defaults to ES_ENTRY
        RunRound1HSM(EntryEventKind);
    }
    return(ReturnEvent);
}
/*****************************************************************************
 Function
     StartRound1HSM

 Parameters
     None

 Returns
     None

 Description
     Does any required initialization for this state machine
 Notes

 Author
     J. Edward Carryer, 2/18/99, 10:38AM
*****************************************************************************/
void StartRound1HSM(ES_Event CurrentEvent)
{

    // local variable to get debugger to display the value of CurrentEvent
    ES_Event LocalEvent = CurrentEvent;
```

```
    #ifdef PRINT_ROUND1_HSM_CALLS

        printf("StartRound1HSM\n\r");

    #endif
    // to implement entry to a history state or directly to a substate
    // you can modify the initialization of the CurrentState variable
    // otherwise just start in the entry state every time the state machine
    // is started
    if(ES_ENTRY_HISTORY != CurrentEvent.EventType)
    {
        CurrentState = Charging1;
    }
    // call the entry function (if any) for the ENTRY_STATE
    RunRound1HSM(CurrentEvent);
}

/*****************************************************************************
 Function
     QueryRound1HSM

 Parameters
     None

 Returns
     Round1State_t The current state of the Template state machine

 Description
     returns the current state of the Template state machine
 Notes

 Author
     J. Edward Carryer, 2/11/05, 10:38AM
*****************************************************************************/
Round1State_t QueryRound1HSM(void)
{

    #ifdef PRINT_ROUND1_HSM_CALLS

        printf("QueryRound1HSM\n\r");

    #endif

    return(CurrentState);
}

/*****************************************************************************
 private functions
 *****************************************************************************/
static ES_Event DuringCharging1HSM(ES_Event Event)
{
    ES_Event ReturnEvent = Event; // assmes no re-mapping or comsumption

    #ifdef PRINT_ROUND1_HSM_CALLS

        printf("DuringChargingHSM\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

        // after that start any lower level machines that run in this state
        Event.EventParam = 1;
            StartDrivingHSM(Event);
            StartJoustingSM(Event);
            //StartShootingSM(Event);

            ES_Timer_SetTimer(LANCE_DELAY_TIMER, LANCE_DELAY_INTERVAL_MS);
        ES_Timer_StartTimer(LANCE_DELAY_TIMER);

        // 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
        RunDrivingHSM(Event);
            StopJoustingSM(Event);
            //StopShootingSM(Event);

        // 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
        RunDrivingHSM(Event);
            RunJoustingSM(Event);
            //RunShootingSM(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);
}
```