```c
/****************************************************************************
 Module
   MasterVehidle.c (based on TopHSMTemplate.c)

 Description
   This is a code 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 "MasterVehicle.h"
#include "ES_DeferRecall.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "inc/hw_sysctl.h"
#include "driverlib/sysctl.h"
#include "driverlib/pin_map.h"  // Define PART_TM4C123GH6PM in project
#include "driverlib/gpio.h"
#include "inc/hw_timer.h"
#include "inc/hw_nvic.h"
#include "StageSM.h"
#include "ShootSM.h"
#include "SupplySM.h"
#include "Location.h"
#include "OwnPWM.h"
#include "LOCMaster.h"
#include "Servo_Control.h"
#include "Ultrasonic.h"

/* Team Definition */
#define RED 0
#define GREEN 1

/* Time Definition*/
#define ONE_SEC 976
#define HALF_SEC (ONE_SEC /2)
#define TWO_SEC (ONE_SEC *2)
#define FIVE_SEC (ONE_SEC *5)
#define TEN_SEC (ONE_SEC *10)
#define TWENTY_SEC (ONE_SEC * 20)
#define FORTY_FIVE_DEGREE_ROTATE_TIME 600
#define TWO_MIN (ONE_SEC * 120)

/* PWM */
#define PWM_FREQ 1000

/* Status related masks*/
#define GREEN_SB1_MASK (BIT0HI|BIT1HI|BIT2HI|BIT3HI) << 20
#define RED_SB1_MASK (BIT0HI|BIT1HI|BIT2HI|BIT3HI) << 16
#define GREEN_SCORE_MASK (BIT0HI|BIT1HI|BIT2HI|BIT3HI|BIT4HI|BIT5HI) << 8
#define RED_SCORE_MASK (BIT0HI|BIT1HI|BIT2HI|BIT3HI|BIT4HI|BIT5HI)
#define GAME_STATUS_MASK (BIT7HI)

#define NUM_BALL_START 4


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

/*---------------------------- Module Functions ---------------------------*/
static ES_Event DuringInitState( ES_Event Event);
static ES_Event DuringIdleState( ES_Event Event);
static ES_Event DuringStagingState( ES_Event Event);
static ES_Event DuringShootingState( ES_Event Event);
static ES_Event DuringSupplyingState( ES_Event Event);
static void Init_LED_Port(void);
static void Team_Selection(void);
static void Set_Construction_LED(void);
```

```c
static void Clear_Construction_LED(void);
static void update_Status(void);
uint32_t get_destination_y(void);
uint32_t get_destination_x(void);

/*---------------------------- 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 MasterVehicleState_t CurrentState;
static uint8_t MyPriority;
static int team;
static uint32_t status = 0;
static uint32_t destination_x = 0;
static uint32_t destination_y = 0;
uint32_t frequency = 0;
static const uint32_t PERIOD_TABLE[16] = {1333,1277,1222,1166,1111,1055,1000
,944,889,833,778,722,667,611,556,500};
static const int period_table_length = 16;
static int NUM_BALL_AVAILABLE;
static uint8_t green_score = 0;
static uint8_t red_score = 0;

/* Coordinate Variables for the locations on the stage (ME218B Winter 2017) */

// Green Coordinates
static const uint32_t StagingXCoordinateEncoderGreen[3]={58,85,59};
static const uint32_t StagingYCoordinateEncoderGreen[3]={23,147,197};

static const uint32_t ShootingXCoordinateEncoderGreen[3]={62,62,62};
static const uint32_t ShootingYCoordinateEncoderGreen[3]={17,59,191};

static const uint32_t green_supply_x = 24;
static const uint32_t green_supply_y = 2;

// Red Coordinates
static const uint32_t StagingXCoordinateEncoderRed[3]={37,12,37};
static const uint32_t StagingYCoordinateEncoderRed[3]={23,52,197};

static const uint32_t ShootingXCoordinateEncoderRed[3]={37,37,37};
static const uint32_t ShootingYCoordinateEncoderRed[3]={17,128,191};

static const uint32_t red_supply_x = 69;
static const uint32_t red_supply_y = 2;

/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
     InitMasterSM

 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 InitMasterVehicleSM ( uint8_t Priority )
{
    ES_Event ThisEvent;
    MyPriority = Priority;  // save our priority
    ThisEvent.EventType = ES_ENTRY;

    // Initialize port and pin needed
    Init_LED_Port();

    // Initialize location module
    Init_Location();
```

```c

    // Initialize PWM for Servo
    InitServo();

    // Start the Master State machine
    StartMasterVehicleSM( ThisEvent );

    //Initially start with 4 balls
    NUM_BALL_AVAILABLE = NUM_BALL_START;

    return true;
}

/****************************************************************************
 Function
     PostMasterSM

 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 PostMasterVehicleSM( ES_Event ThisEvent )
{
    return ES_PostToService( MyPriority, ThisEvent);
}

/****************************************************************************
 Function
    RunMasterVehicleSM

 Parameters
   ES_Event: the event to process

 Returns
   ES_Event: an event to return

 Description
   the run function for the top level state machine

****************************************************************************/
ES_Event RunMasterVehicleSM( ES_Event CurrentEvent )
{
   bool MakeTransition = false;/* are we making a state transition? */
   MasterVehicleState_t NextState = CurrentState;
   ES_Event EntryEventKind = { ES_ENTRY, 0 };// default to normal entry to new state
   ES_Event ReturnEvent = { ES_NO_EVENT, 0 }; // assume no error

   switch ( CurrentState )
   {
      case INIT_STATE :  // If current state is Init State
         // Execute During function for state one. ES_ENTRY & ES_EXIT are
         // processed here allow the lowere level state machines to re-map
         // or consume the event
         CurrentEvent = DuringInitState(CurrentEvent);
         //process any events
         if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
         {
            switch (CurrentEvent.EventType)
            {
               case CONSTRUCTION_START: //If event is event one
                  // Execute action function for state one : event one
                  NextState = IDLE_STATE;//Decide what the next state will be
                  // for internal transitions, skip changing MakeTransition
                  MakeTransition = true; //mark that we are taking a transition
                  // if transitioning to a state with history change kind of entry
                  // EntryEventKind.EventType = ES_ENTRY_HISTORY;
```

```
                    // optionally, consume or re-map this event for the upper
                    // level state machine
                    ReturnEvent.EventType = ES_NO_EVENT;
                    break;

                 default:
                    break;
              }
           }
           break;

        case IDLE_STATE :  // If current state is idle state
           // Execute During function for state one. ES_ENTRY & ES_EXIT are
           // processed here allow the lower level state machines to re-map
           // or consume the event
           CurrentEvent = DuringIdleState(CurrentEvent);
           //process any events

           if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
           {

              switch (CurrentEvent.EventType)
              {
                 case STAGE_ACTIVE: //If event is stage active
                    update_Status();
                    // update game status to reflect current active staging area
                    NextState = STAGING_STATE;//set next state to staging
                    MakeTransition = true; //mark that we are taking a transition
                    ReturnEvent.EventType = ES_NO_EVENT;
                    break;

                 case SHOOT_ACTIVE: //If event is shoot active
                    update_Status();
                    // update game status to reflect current active shooting area
                    NextState = SHOOTING_STATE;//set next state to shooting
                    MakeTransition = true; //mark that we are taking a transition
                    ReturnEvent.EventType = ES_NO_EVENT;
                    break;

                 case SHOOT_ACTIVE_4: //If event is shoot active (for last 18 seconds)
                    update_Status();
                    // update game status to reflect current active staging area
                    NextState = SHOOTING_STATE;//set next state to shooting
                    MakeTransition = true; //mark that we are taking a transition
                    ReturnEvent.EventType = ES_NO_EVENT;
                    break;

                 case NO_BALL: //If event is event one
                    update_Status();
                    // update game status to reflect current active staging area
                    NextState = SUPPLYING_STATE;//set next state to supplying
                    MakeTransition = true; //mark that we are taking a transition
                    ReturnEvent.EventType = ES_NO_EVENT;
                    break;

                 case CONSTRUCTION_END: //If event is construction end
                    NextState = INIT_STATE;
                    //set next state to init state (we want to stop everything)
                    MakeTransition = true; //mark that we are taking a transition
                    ReturnEvent.EventType = ES_NO_EVENT;
                    break;

                 default:
                    break;
              }
           }
           break;

        case STAGING_STATE : // If current state is staging state
           // Execute During function for state one. ES_ENTRY & ES_EXIT are
           // processed here allow the lowere level state machines to re-map
           // or consume the event
```

```
            CurrentEvent = DuringStagingState(CurrentEvent);
            //process any events
            if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
            {
               switch (CurrentEvent.EventType)
               {

                  case ES_TIMEOUT: //if event is timeout, we go back to idle state
                     NextState = IDLE_STATE;//set next state to idle state
                     MakeTransition = true; //mark that we are taking a transition
                     ReturnEvent.EventType = ES_NO_EVENT;
                     check_active_event(); //check if there is any active event on our way out
                     break;

                  case SHOOT_ACTIVE_4: //if event is shoot active (during the last 18 seconds)
                     NextState = IDLE_STATE; //set next state to idle state
                     MakeTransition = true; //mark that we are taking a transition
                     ReturnEvent.EventType = ES_NO_EVENT;
                     check_active_event(); //check if there is any active event on our way out
                     break;

                  case FINISHED_STAGING: //if we successfully verify frequency
                     NextState = IDLE_STATE; //set next state to idle state
                     MakeTransition = true; //mark that we are taking a transition
                     ReturnEvent.EventType = ES_NO_EVENT;
                     break;

                  case CONSTRUCTION_END: //If event is construction end
                     NextState = INIT_STATE; //set next state to init state
                     MakeTransition = true; //mark that we are taking a transition
                     ReturnEvent.EventType = ES_NO_EVENT;
                     break;

                  default:
                     break;
               }
            }
            break;

         case SHOOTING_STATE : // If current state is shooting state
            // Execute During function for state one. ES_ENTRY & ES_EXIT are
            // processed here allow the lowere level state machines to re-map
            // or consume the event
            CurrentEvent = DuringShootingState(CurrentEvent);
            //process any events
            if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
            {
               switch (CurrentEvent.EventType)
               {
                  case ES_TIMEOUT: //If event is 20 SECOND SHOOTING TIMEOUT
                     printf("ES TIMEOUT\r\n");
                     NextState = IDLE_STATE;//set next state to idle state
                     MakeTransition = true; //mark that we are taking a transition
                     ReturnEvent.EventType = ES_NO_EVENT;
                     check_active_event(); //check if there is any active event on our way out
                     break;

                  case SCORE_CHANGED: //If event is SCORE_CHANGED
                     printf("Score Changed\r\n");
                     NextState = IDLE_STATE;//set next state to idle state
                     MakeTransition = true; //mark that we are taking a transition
                     ReturnEvent.EventType = ES_NO_EVENT;
                     check_active_event(); //check if there is any active event on our way out
                     break;

                  case NO_BALL: //If event is no ball
                     printf("No Ballr\n");
                     NextState = SUPPLYING_STATE;//set next state to supply state
                     MakeTransition = true; //mark that we are taking a transition
                     ReturnEvent.EventType = ES_NO_EVENT;
                     break;

```

```
                 case CONSTRUCTION_END: //If event is construction end
                    printf("Construction Ends\r\n");
                    NextState = INIT_STATE;//set next state to init state
                    MakeTransition = true; //mark that we are taking a transition
                    ReturnEvent.EventType = ES_NO_EVENT;
                    break;

                 default:
                    break;
              }
           }
           break;

        case SUPPLYING_STATE : // If current state is supplying state
           // Execute During function for state one. ES_ENTRY & ES_EXIT are
           // processed here allow the lowere level state machines to re-map
           // or consume the event
           CurrentEvent = DuringSupplyingState(CurrentEvent);
           //process any events
           if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
           {
              switch (CurrentEvent.EventType)
              {
                 case LOADED_COMPLETE: //If event is loaded complete
                    NextState = IDLE_STATE;//set next state to idle state
                    MakeTransition = true; //mark that we are taking a transition
                    ReturnEvent.EventType = ES_NO_EVENT;
                    break;

                 case CONSTRUCTION_END: //If event is construction end
                    NextState = INIT_STATE;//set next state to init state
                    MakeTransition = true; //mark that we are taking a transition
                    ReturnEvent.EventType = ES_NO_EVENT;
                    break;

                 default:
                    break;
              }
           }
           break;

        default:
           break;
         // repeat state pattern as required for other states
      }

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

         CurrentState = NextState; //Modify state variable

         // Execute entry function for new state
         // this defaults to ES_ENTRY
         RunMasterVehicleSM(EntryEventKind);
       }

     // in the absence of an error the top level state machine should
     // always return ES_NO_EVENT, which we initialized at the top of function
     return(ReturnEvent);
  }


  /****************************************************************************
   Function
       StartMasterVehicleSM

   Parameters
       ES_Event CurrentEvent
```

```

 Returns
     nothing

 Description
     Does any required initialization for this state machine

****************************************************************************/
void StartMasterVehicleSM ( ES_Event CurrentEvent )
{
  // if there is more than 1 state to the top level machine you will need
  // to initialize the state variable
  CurrentState = INIT_STATE; //start currentstate with init state
  // 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
  RunMasterVehicleSM(CurrentEvent);
  return;
}


/* This function return the current state of this state machine */
MasterVehicleState_t QueryMasterVehicleSM ( void )
{
   return(CurrentState);
}

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

static ES_Event DuringInitState( ES_Event Event)
{
   ES_Event ReturnEvent = Event;
   if ( (Event.EventType == ES_ENTRY) || (Event.EventType == ES_ENTRY_HISTORY) ){
      Team_Selection(); //read team input and set team collor
      Clear_Construction_LED(); //make sure construction led is off
      stopMotor(); //make sure motor is not running when we start the game
   }
   else if ( Event.EventType == ES_EXIT ){
      //when we exit init state, that means game is on. So turn on the led
      Set_Construction_LED();
   }
   else{
      // No lower state machine
   }
   return ReturnEvent;
}


static ES_Event DuringIdleState( ES_Event Event)
{
   ES_Event ReturnEvent = Event;
   return ReturnEvent;
}

static ES_Event DuringStagingState( ES_Event Event)
{
   ES_Event ReturnEvent = Event;
   if ( (Event.EventType == ES_ENTRY) || (Event.EventType == ES_ENTRY_HISTORY) ){
      //when we enter this state, post STAGE_ACTIVE event to StageSM
      Event.EventType = STAGE_ACTIVE;
      StartStageSM(Event);
   }
   else if ( Event.EventType == ES_EXIT ){
      RunStageSM(Event);
      //We have 2 sec limit in staging SM, we want to stop this timer
      //when we are in shooting or supplying state
      ES_Timer_StopTimer(STAGE_TIMER);
   }
   else{
      ReturnEvent = RunStageSM(Event);
   }
```

```c
   return ReturnEvent;
}

static ES_Event DuringShootingState( ES_Event Event)
{
   ES_Event ReturnEvent = Event;
   if ( (Event.EventType == ES_ENTRY) || (Event.EventType == ES_ENTRY_HISTORY) ){
      //when we enter this state, post SHOOT_ACTIVE event to ShootSM
      Event.EventType = SHOOT_ACTIVE;
      StartShootSM(Event);
      // start 20 seconds timer limit for shooting
      ES_Timer_InitTimer(SHOOT_20S_TIMER, TWENTY_SEC);
   }
   else if ( Event.EventType == ES_EXIT ){
      RunShootSM(Event);
   }
   else{
      ReturnEvent = RunShootSM(Event);
   }
   return ReturnEvent;
}

static ES_Event DuringSupplyingState( ES_Event Event)
{
   ES_Event ReturnEvent = Event;
   if ( (Event.EventType == ES_ENTRY) || (Event.EventType == ES_ENTRY_HISTORY) ){
      //when we enter this state, post NO_BALL event to SupplySM
      Event.EventType= NO_BALL;
      StartSupplySM(Event);
   }
   else if ( Event.EventType == ES_EXIT ){
      RunSupplySM(Event);
      //we we exit, we want to check which event is currently active
      check_active_event();
   }
   else{
      ReturnEvent = RunSupplySM(Event);
   }
   return ReturnEvent;
}

static void Init_LED_Port(){
   //Initializing Port B, E, F for led and team selection switch
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
   SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
   GPIOPinTypeGPIOInput(GPIO_PORTB_BASE, GPIO_PIN_0); //TEAM SELECTION SWITCH
   GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_1); //TEAM RED LED
   GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_2); //RESUPPLYING LED
   GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_3); //CONSTRUCTION LED
   GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_4); //TEAM GREEN LED

   //Initialize PF0
   HWREG(GPIO_PORTF_BASE + GPIO_O_PUR) = HWREG(GPIO_PORTF_BASE + GPIO_O_PUR) | (1<<0);
   HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
   HWREG(GPIO_PORTF_BASE + GPIO_O_CR) = (1<<0);
   GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_0); //IR LED (FOR RESUPPLY)

   //Initialize PF1
   HWREG(GPIO_PORTF_BASE + GPIO_O_PUR) = HWREG(GPIO_PORTF_BASE + GPIO_O_PUR) | (1<<1);
   HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
   HWREG(GPIO_PORTF_BASE + GPIO_O_CR) = (1<<0);
   GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_1); //BUMPER SWITCH
}

/* This function read value from PB0 and set team accordinly */
static void Team_Selection(){
   //if PB0 is low, set team to RED
   if((GPIOPinRead(GPIO_PORTB_BASE, GPIO_PIN_0) & BIT0HI) == 0){
      team = RED;
   }
   //if PB0 is high, set team to GREEN
```

```c
    else{
       team = GREEN;
    }
    if(team == RED){
       //light up PB1 which is connected to red led
       GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, BIT1HI);
       GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_4, BIT4LO);
    }
    else if(team == GREEN){
       //light up PF4 which is connected to green led
       GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_4, BIT4HI);
       GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_1, BIT1LO);
    }
}

static void Set_Construction_LED(){
   //Construction In Progress. set LED (set PF3)
   GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, BIT3HI);
}

static void Clear_Construction_LED(){
   //"Clear Construction LED (clear PB4)
   GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, BIT3LO);
}

static void update_Status(){
   //Get status update from LOC master module
   status = queryStatusBytes();
}

/* This function return the current team */
int get_Team(void){
   return team;
}

/* This function return the current supply location (x)*/
uint32_t get_Supply_location_x(){
   if (team == RED){
      return red_supply_x;
   }
   else{
      return green_supply_x;
   }
}

/* This function return the current supply location (y)*/
uint32_t get_Supply_location_y(){
   if (team == RED){
      return red_supply_y;
   }
   else{
      return green_supply_y;
   }
}

/* The folowing functions return the loation of stage and shoot area for red and green */
uint32_t get_Stage_Green_X(uint8_t index){
   //we have to subtract one from index as the input is
   //from 1 to 4 but our array index start at 0
   return StagingXCoordinateEncoderGreen[index-1];
}
uint32_t get_Stage_Green_Y(uint8_t index){
   return StagingYCoordinateEncoderGreen[index-1];
}
uint32_t get_Stage_Red_X(uint8_t index){
   return StagingXCoordinateEncoderRed[index-1];
}
uint32_t get_Stage_Red_Y(uint8_t index){
   return StagingYCoordinateEncoderRed[index-1];
}
uint32_t get_Shoot_Green_X(uint8_t index){
   return ShootingXCoordinateEncoderGreen[index-1];
```

```
}
uint32_t get_Shoot_Green_Y(uint8_t index){
   return ShootingYCoordinateEncoderGreen[index-1];
}
uint32_t get_Shoot_Red_X(uint8_t index){
   return ShootingXCoordinateEncoderRed[index-1];
}
uint32_t get_Shoot_Red_Y(uint8_t index){
   return ShootingYCoordinateEncoderRed[index-1];
}

/* This function return the current number of ball */
int get_num_ball(void){
   return NUM_BALL_AVAILABLE;
}

/* This function increases the current number of ball by 1*/
void increment_num_ball(void){
   NUM_BALL_AVAILABLE++;
}

/* This function decreases the current number of ball by 1*/
void decrement_num_ball(void){
   NUM_BALL_AVAILABLE--;
}

/* This function updates the score variable using the inputs */
void update_score(uint8_t red_input, uint8_t green_input){
   red_score = red_input;
   green_score = green_input;
}

/* This function returns the current red score */
uint8_t get_red_score(void){
   return red_score;
}

/* This function returns the current green score */
uint8_t get_green_score(void){
   return green_score;
}
```