```
/****************************************************************************
Module
SupplySM.c

Description
This is based on 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 "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/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 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 "SupplySM.h"
#include "Location.h"
#include "MasterVehicle.h"
#include "LOCMaster.h"
#include "Location.h"
/*----------------------------- Module Defines ----------------------------*/
// define constants for the states for this machine
// and any other local defines

#define NORMAL_OPERATION
//#define TESTING_SUPPLY // for debugging
#define ENTRY_STATE SUPPLY_WAITING
#define TWO_SEC 2000
#define TEN_MS 10
#define THIRTY_MS 30
#define THREE_SEC 3000
#define HALF_SEC 500
#define MAX_BALL_RECEIVED 4
#define COMPETITION_FULL_DUTY 75
#define CORRECTION_FULL_DUTY 55
#define NF 0x08
/*---------------------------- 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 DuringWaiting( ES_Event Event);
static ES_Event DuringMoveX( ES_Event Event);
static ES_Event DuringMoveY( ES_Event Event);
static ES_Event DuringPulseSupply( ES_Event Event);

/*---------------------------- Module Variables ---------------------------*/
// everybody needs a state variable, you may need others as well
static SupplyingState_t CurrentState;
static bool flag_10ms_timer = false;
static bool flag_30ms_timer = false;
static bool flag_3000ms_timer = false;
static int pulse_count = 0;
static bool supply_led_on = false;
static bool loaded_complete = false;
static uint32_t OneShotTimeout_10ms = 40000000*10/1000;
static uint32_t OneShotTimeout_30ms = 40000000*30/1000;
static int counter = 0;
static bool count_valid = true;
/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
Function
```

```
RunSupplySM

Parameters
ES_Event: the event to process

Returns
ES_Event: an event to return

****************************************************************************/
ES_Event RunSupplySM( ES_Event CurrentEvent )
{
  bool MakeTransition = false;/* are we making a state transition? */
  SupplyingState_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

  switch ( CurrentState )
  {
    case SUPPLY_WAITING :       // If current state is waiting
    // 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 = DuringWaiting(CurrentEvent);
    //process any events
    if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
    {
      switch (CurrentEvent.EventType)
      {
        case ES_TIMEOUT: // if we get stage timeout, consume this event
        if(CurrentEvent.EventParam == STAGE_TIMER){
          ReturnEvent.EventType = ES_NO_EVENT;
        }
        break;
        case NO_BALL: //If event is no_ball, go to move x state
        // Execute action function for state one : event one
        NextState = SUPPLY_MOVE_X;
        MakeTransition = true;
        ReturnEvent.EventType = ES_NO_EVENT;
        break;
        default:
        break;
      }
    }
    break;

    case SUPPLY_MOVE_X :       // If current state is move x 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 = DuringMoveX(CurrentEvent);
    //process any events
    if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
    {
      switch (CurrentEvent.EventType)
      {
        case ES_TIMEOUT:
        if(CurrentEvent.EventParam == STAGE_TIMER){
          ReturnEvent.EventType = ES_NO_EVENT;
        }
        break;

        case X_REACHED : //If event is reaching destination x
        // Execute action function for state one : event one
        NextState = SUPPLY_MOVE_Y; // set next state to moving in y
        // for internal transitions, skip changing MakeTransition
        MakeTransition = true; //mark that we are taking a transition
        ReturnEvent.EventType = ES_NO_EVENT;
        break;
        case CONSTRUCTION_END: //If event is construction end
        NextState = SUPPLY_WAITING; //set next state to waiting
        MakeTransition = true; //mark that we are taking a transition
        ReturnEvent.EventType = CONSTRUCTION_END; // return this event to upper SM
```

```
          break;
          default:
          break;
        }
      }
      break;

      case SUPPLY_MOVE_Y: // if current state is moving in y
      // 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 = DuringMoveY(CurrentEvent);
      //process any events
      if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
      {
        switch (CurrentEvent.EventType)
        {
          case ES_TIMEOUT:
          //consume stage timer time out event
          if(CurrentEvent.EventParam == STAGE_TIMER){
            ReturnEvent.EventType = ES_NO_EVENT;
          }
          //if we finished ramming the supply, stop the motor
          if(CurrentEvent.EventParam == SUPPLY_RAMMING_TIMER){
            stopMotor();
            set_location_checker_flag(true); //reenable location checker
            NextState = PULSE_SUPPLY; //set next state to pulse supply
            MakeTransition = true;
            ReturnEvent.EventType = ES_NO_EVENT;
          }
          break;

          case Y_REACHED: //If event is reaching y destination
          #ifdef NORMAL_OPERATION
          //during normal operation, we reverify if x location is correct
          if(verify_x_location()){
            //if x location is correct, set the speed to high speed
            set_PWM_Full_Duty(COMPETITION_FULL_DUTY);
            set_location_checker_flag(false);
            //this function run the motor in the northward direction
            runMotor(NF);
            //start the ramming timer
            ES_Timer_InitTimer(SUPPLY_RAMMING_TIMER, TWO_SEC);
          }
          else{
            //if x location is incorrect, we set speed to correction speed (low speed)
            set_PWM_Full_Duty(CORRECTION_FULL_DUTY);
            //set next state to moving in x
            NextState = SUPPLY_MOVE_X;
            MakeTransition = true;
          }
          #endif

          //during testing mode, we do not have location verification loop
          #ifdef TESTING_SUPPLY
          //set next state to pulse supply
          NextState = PULSE_SUPPLY;
          MakeTransition = true;
          #endif
          ReturnEvent.EventType = ES_NO_EVENT;
          break;

          case CONSTRUCTION_END: //If event is construction end
          NextState = SUPPLY_WAITING; //set next state to waiting
          MakeTransition = true; //mark that we are taking a transition
          ReturnEvent.EventType = CONSTRUCTION_END; //return this event
          break;
          default:
          break;
        }
      }
      break;
```

```c

      case PULSE_SUPPLY :       // If current state is pulse supply
      // 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 = DuringPulseSupply(CurrentEvent);
      //process any events
      if ( CurrentEvent.EventType != ES_NO_EVENT ) //If an event is active
      {
        switch (CurrentEvent.EventType)
        {
          // consume score changed event
          case SCORE_CHANGED:
          ReturnEvent.EventType = ES_NO_EVENT;
          break;

          case ES_TIMEOUT: //If event is event one
          //ignore stage timer timeout event
          if(CurrentEvent.EventParam == STAGE_TIMER){
            ReturnEvent.EventType = ES_NO_EVENT;
          }

          // This timeout is for indicator LED
          else if(CurrentEvent.EventParam == SUPPLY_LED_TIMER){
            if (!loaded_complete){
              if(supply_led_on){
                printf("LED ON\r\n");
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, BIT3LO); //Light Supplying LED
                supply_led_on = false;
              }
              else{
                printf("LED OFF\r\n");
                GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, BIT3HI); //Light Supplying LED
                supply_led_on = true;
              }
              //restart led timer
              ES_Timer_InitTimer(SUPPLY_LED_TIMER, HALF_SEC);
              ReturnEvent.EventType = ES_NO_EVENT; //Consume event
            }
          }
          else if(CurrentEvent.EventParam == SUPPLY_TIMER){
            //  if we have maximum number of balls or more
            if(get_num_ball() >= MAX_BALL_RECEIVED){
              loaded_complete = true; //set loaded complete flag to true
              printf("now have max_num_ball balls, post LOADED_COMPLETE\r\n");
              //post event to master state machine to return to idle
              NextState = SUPPLY_WAITING;
              MakeTransition = true;
              ES_Event topost;
              topost.EventType = LOADED_COMPLETE;
              PostMasterVehicleSM(topost);
              ReturnEvent.EventType = ES_NO_EVENT;
            }
            //if we don't have 5 balls, continue requesting for ball
            else if(get_num_ball() < MAX_BALL_RECEIVED){
              pulse_count = 0;
              counter = 0;
              //write the pulsing line high
              GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, BIT0HI);
              //increment pulse count counter
              //start a 10ms timer
              StartOneShot_Supply_10ms();
              ReturnEvent.EventType = ES_NO_EVENT;
            }
            else{
              // print for debugging purpose
              printf("We should not have more than 5 balls\r\n");
            }
          }
          else{
            printf("Some timer in Supply we don't deal with\r\n");
            CurrentEvent.EventType = ES_EXIT;
```

```
            RunSupplySM(CurrentEvent);
            ReturnEvent.EventType = ES_NO_EVENT;
          }
          break;

          // if we get loaded complete, return this event to upper SM
          case LOADED_COMPLETE:
          NextState = SUPPLY_WAITING;//Decide what the next state will be
          MakeTransition = true; //mark that we are taking a transition
          ReturnEvent.EventType = LOADED_COMPLETE;
          break;

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

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

      CurrentState = NextState; //Modify state variable

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

  Parameters
  None

  Returns
  None

  Description
  Does any required initialization for this state machine

  ****************************************************************************/
  void StartSupplySM ( ES_Event CurrentEvent )
  {
    // 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 = ENTRY_STATE;
    }
    // call the entry function (if any) for the ENTRY_STATE
    RunSupplySM(CurrentEvent);
  }

  /****************************************************************************
  Function
  QuerySupplySM
```

```
361
362   Parameters
363   None
364
365   Returns
366   SupplhyingState_t The current state of the Supply state machine
367
368   Description
369   returns the current state of the Supply state machine
370
371   ****************************************************************************/
372   SupplyingState_t QuerySupplySM ( void )
373   {
374     return(CurrentState);
375   }
376
377   /****************************************************************************
378   private functions
379   ****************************************************************************/
380   static ES_Event DuringWaiting( ES_Event Event)
381   {
382     ES_Event ReturnEvent = Event; // assme no re-mapping or comsumption
383     return(ReturnEvent);
384   }
385
386   static ES_Event DuringMoveX( ES_Event Event)
387   {
388     ES_Event ReturnEvent = Event; // assme no re-mapping or comsumption
389
390     // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
391     if ( (Event.EventType == ES_ENTRY) ||
392       (Event.EventType == ES_ENTRY_HISTORY) )
393     {
394       // implement any entry actions required for this state machine
395       // call move in x funtion
396       move_X(get_Supply_location_x());
397     }
398     else if ( Event.EventType == ES_EXIT )
399     {
400       // on exit, give the lower levels a chance to clean up first
401       // make sure the motor is not running when exiting this state
402       stopMotor();
403
404     }else
405       // do the 'during' function for this state
406     {
407     }
408     // return either Event, if you don't want to allow the lower level machine
409     // to remap the current event, or ReturnEvent if you do want to allow it.
410     return(ReturnEvent);
411   }
412
413   static ES_Event DuringMoveY( ES_Event Event)
414   {
415     ES_Event ReturnEvent = Event; // assme no re-mapping or comsumption
416
417     // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
418     if ( (Event.EventType == ES_ENTRY) ||
419       (Event.EventType == ES_ENTRY_HISTORY) )
420     {
421       // implement any entry actions required for this state machine
422       move_Y(get_Supply_location_y());
423       // set this flag to trigger speed control in location.c
424       // that is, when the robot get close to the supply destination,
425       // it will automatically reduces it speed so we don't ram into the wall
426       set_going_to_supply_flag();
427     }
428     else if ( Event.EventType == ES_EXIT )
429     {
430       // on exit, give the lower levels a chance to clean up first
431       // make sure the motor is not running when exiting this state
432       stopMotor();
```

```
      clear_going_to_supply_flag(); //reset this flag

    }else
      // do the 'during' function for 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 DuringPulseSupply( ES_Event Event)
  {
    ES_Event ReturnEvent = Event; // assme no re-mapping or comsumption

    // process ES_ENTRY, ES_ENTRY_HISTORY & ES_EXIT events
    if ( (Event.EventType == ES_ENTRY) ||
      (Event.EventType == ES_ENTRY_HISTORY) )
    {
      // implement any entry actions required for this state machine
      //start one shot timer for ir led pulsing
      InitOneShotInt_Supply();
      if(get_num_ball() < MAX_BALL_RECEIVED){
        loaded_complete = false;
      }
      counter = 0;
      //start a 10ms timer
      StartOneShot_Supply_10ms();
      //start supply LED timer
      ES_Timer_InitTimer(SUPPLY_LED_TIMER, HALF_SEC);
    }
    else if ( Event.EventType == ES_EXIT )
    {
      // on exit, give the lower levels a chance to clean up first
      //lower IR and raise indicator
      printf("Set led low\r\n");
      GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, BIT0LO); //IR led low
      GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, BIT3HI); //Construction led high
    }else
      // do the 'during' function for 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);
  }


  /* Private Function */

  void InitOneShotInt_Supply( void ){
    // start by enabling the clock to the timer (Wide Timer 4)
    HWREG(SYSCTL_RCGCWTIMER) |= SYSCTL_RCGCWTIMER_R4;
    // kill a few cycles to let the clock get going
    while((HWREG(SYSCTL_PRWTIMER) & SYSCTL_PRWTIMER_R4) != SYSCTL_PRWTIMER_R4)
    {
    }
    // make sure that timer (Timer B) is disabled before configuring
    HWREG(WTIMER4_BASE+TIMER_O_CTL) &= ~TIMER_CTL_TBEN; //TBEN = Bit8
    // set it up in 32bit wide (individual, not concatenated) mode
    // the constant name derives from the 16/32 bit timer, but this is a 32/64
    // bit timer so we are setting the 32bit mode
    HWREG(WTIMER4_BASE+TIMER_O_CFG) = TIMER_CFG_16_BIT; //bits 0-2 = 0x04
    // set up timer B in 1-shot mode so that it disables timer on timeouts
    // first mask off the TAMR field (bits 0:1) then set the value for
    // 1-shot mode = 0x01
    HWREG(WTIMER4_BASE+TIMER_O_TBMR) =
    (HWREG(WTIMER4_BASE+TIMER_O_TBMR)& ~TIMER_TBMR_TBMR_M)|
    TIMER_TBMR_TBMR_1_SHOT;
    // set timeout
    HWREG(WTIMER4_BASE+TIMER_O_TBILR) = OneShotTimeout_10ms;
    // enable a local timeout interrupt. TBTOIM = bit 8
```

```
  HWREG(WTIMER4_BASE+TIMER_O_IMR) |= TIMER_IMR_TBTOIM; // 8
  // enable the Timer B in Wide Timer 0 interrupt in the NVIC
  // it is interrupt number 103 so appears in EN3 at bit 1
  HWREG(NVIC_EN3) |= BIT7HI;
  // make sure interrupts are enabled globally
  __enable_irq();
  //StartTime = ES_Timer_GetTime();
  // now kick the timer off by enabling it and enabling the timer to
  // stall while stopped by the debugger. TAEN = Bit0, TASTALL = bit1
  HWREG(WTIMER4_BASE+TIMER_O_CTL) |= (TIMER_CTL_TBEN | TIMER_CTL_TBSTALL);
}

void StartOneShot_Supply_10ms( void ){
  // start by grabbing the start time
  //StartTime = ES_Timer_GetTime();
  // now kick the timer off by enabling it and enabling the timer to
  // stall while stopped by the debugger
  HWREG(WTIMER4_BASE+TIMER_O_CTL) &= ~TIMER_CTL_TBEN;
  HWREG(WTIMER4_BASE+TIMER_O_TBILR) = OneShotTimeout_10ms;
  __enable_irq();
  HWREG(WTIMER4_BASE+TIMER_O_CTL) |= (TIMER_CTL_TBEN | TIMER_CTL_TBSTALL);
}

void StartOneShot_Supply_30ms( void ){
  // start by grabbing the start time
  //StartTime = ES_Timer_GetTime();
  // now kick the timer off by enabling it and enabling the timer to
  // stall while stopped by the debugger
  HWREG(WTIMER4_BASE+TIMER_O_CTL) &= ~TIMER_CTL_TBEN;
  HWREG(WTIMER4_BASE+TIMER_O_TBILR) = OneShotTimeout_30ms;
  __enable_irq();
  HWREG(WTIMER4_BASE+TIMER_O_CTL) |= (TIMER_CTL_TBEN | TIMER_CTL_TBSTALL);
}

void OneShotIntResponse_Supply( void ){
  // start by clearing the source of the interrupt
  HWREG(WTIMER4_BASE+TIMER_O_ICR) = TIMER_ICR_TBTOCINT;
  if(counter < 20){
    if(counter % 2 == 0){
      GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, BIT0HI); //set IR LED High
      counter++; // increment supply ir led counter
      StartOneShot_Supply_10ms();
    }
    else{
      GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0, BIT0LO); //set IR LED Low
      StartOneShot_Supply_30ms();
      counter++; // increment supply ir led counter
    }
  }
  // if counter is 20, we increment ball( here we include fix for increasing ball twice)
  if(counter == 20){
    if(count_valid){
      ES_Timer_InitTimer(SUPPLY_TIMER, THREE_SEC);
      increment_num_ball();
      count_valid = false;
    }
    else{
      count_valid = true;
    }
  }
}
```