```c
/****************************************************************************
SupplySM.c Pseudocode

****************************************************************************/
/*----------------------------- 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

#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;

/****************************************************************************
  Function
    RunSupplySM
 ****************************************************************************/
ES_Event RunSupplySM( ES_Event CurrentEvent )
{
   initialize marktransition variable to false
   set nextstate to CurrentState

   switch ( CurrentState )
   {
      case CurrentState is "Waiting State":
        set CurrentEvent to the result from running duringwaiting function
        if CurrentEvent is not ES_NO_EVENT
          if the eventtype of CurrentEvent is timeout and it is from STAGE_TIMER
            consume this event
          break
        if CurrentEvent is NO_BALL
          set nextstate to SUPPLY_MOVE_X
          set marktransition to true
          consume this event
          break

      case currentState is "MOVE IN X"
        Execute During function for "MOVE IN X" state. ES_ENTRY & ES_EXIT are
        processed here allow the lower level state machines to re-map
        or consume the event, we have entry function: start motor in x

            if CurrentEvent is not ES_NO_EVENT
        {
          switch the event type
          {
            if the eventtype of CurrentEvent is timeout and it is from STAGE_TIMER
              consume this event
              break
            if event is X_REACHED
              next state will be "MOVE IN Y"
              set marktransition to true
              break
            if event is CONSTRUCTION_END
              set next state to "Waiting State"
              set marktransition to true
              set return event to CONSTRUCTION_END event
              break
          }
        }
        break
      // repeat state pattern as required for other states
```

```
case  current state is "MOVE IN Y"
  Execute During function for "MOVE IN Y" state. ES_ENTRY & ES_EXIT are
  processed here allow the lower level state machines to re-map
  or consume the event. We have an entry functon that starts the motor in y direction.

      if CurrentEvent is not ES_NO_EVENT
   {
      switch the event type
      {
        if the event type is TIMEOUT and it is from STAGE_TIMER
          consume this event
          break

        if event type is TIMEOUT and it is from SUPPLY_RAMMING_TIMER
          stop the motor
          set location checker flag to true
          set next state to "PULSE SUPPLY"
          set mark transition to true
          consume event
          break

        if event type is "Y_REACHED"
          check if the location x is still correct
          if location x is correct
            set duty cycle of the motor to full speed
            set location checker flag to false
            run the motor in the Northward direction
            start SUPPLY_RAMMING_TIMER with time equals to two seconds

          else if location x is incorrect
            set duty cycle of the motor to correction speed
            set next state to "MOVE IN X"
            set mark transition to true
          break

        if event is CONSTRUCTION_END
          set next state to "Waiting State"
          set marktransition to true
          set return event to CONSTRUCTION_END event
          break
      }
   }
   break;

case  current state is PULSE_SUPPLY
  Execute During function for PULSE_SUPPLY. ES_ENTRY & ES_EXIT are
  processed here allow the lower level state machines to re-map
  or consume the event. We have entry function: start a 10ms timer,
  and write pulsing line high

      if CurrentEvent is not ES_NO_EVENT
  {
    switch (CurrentEvent.EventType)
      {
        case event type is SCORE_CHANGED
          consume this event
          break

        case the event type is TIMEOUT and it is from STAGE_TIMER
          consume this event
          break
```

```
              case event type is TIMEOUT and it is from SUPPLY_LED_TIMER
                if we have not finish loading
                  if supply led is currently on
                    set the supply led low

                  else
                    set the supply led high

                start SUPPLY_LED_TIMER with half a second timeout
                consume this event
                break

              case event type is TIMEOUT and it is from SUPPLY_TIMER
                if the current number of balls is more than maximum number of balls
                  set loaded complete flag to true
                  set next state to "Waiting State"
                  set mark transition to true
                  post LOADED_COMPLETE event to MasterVehicle
                  consume this event

                else
                  set pulse counter to 0
                  set the IR LED high (PF0)
                  start one shot timer (10ms)
                  consume this event

                break

              case event is LOADED_COMPLETE
                set next state to "Waiting State"
                set marktransition to true
                set return event to LOADED_COMPLETE event
                break

              case event type is CONSTRUCTION_END
                set next state to "Waiting State"
                set marktransition to true
                set return event to CONSTRUCTION_END event
                break
            }
        }
        break

    }

    if we are making a state transition
      Execute exit function for current state
        Modify state variable
        Execute entry function for new state, using regular ES_ENTRY with no history

    return ReturnEvent
}
```

```
/****************************************************************************
  Function
    StartSupplySM
 ****************************************************************************/
void StartSupplySM ( ES_Event CurrentEvent )
{
  if the entry event is a normal ES_ENTRY
      set the current state to entry state

  call the entry function (if any) for the ENTRY_STATE
}

TemplateState_t QueryTemplateSM ( void )
{
   return CurrentState;
}

/****************************************************************************
  Function
    QuerySupplySM
 ****************************************************************************/
SupplyingState_t QuerySupplySM (void){
    return the current state
}

/****************************************************************************
  Private functions
 ****************************************************************************/
static ES_Event DuringWaiting( ES_Event Event)
{
    set ES_Event ReturnEvent to Event // assme no re-mapping or comsumption
    return ReturnEvent
}


static ES_Event DuringMoveInX( ES_Event Event)
{
    set ReturnEvent to Event, assuming no re-mapping or comsumption

    if the event is ES_ENTRY
      call move_X function with destination from get_Supply_location_x()
    else if the event is EXIT
      stop the motor

    return ReturnEvent
}

static ES_Event DuringMoveInY( ES_Event Event)
{
    set ReturnEvent to Event, assuming no re-mapping or comsumption

    if the event is ES_ENTRY
      call move_Y function with destination from get_Supply_location_y()
      set going to supply flag
    else if the event is EXIT
      stop the motor
      clear going to supply flag

    return ReturnEvent
}
```

```
static ES_Event DuringPulseSupply( ES_Event Event)
{
    set ReturnEvent to Event, assuming no re-mapping or comsumption

    if the event is ES_ENTRY
      init one shot interrupt response for SupplySM
      if current number of balls is less than maximum number of balls
        set loaded complete flag to false
      set pulse counter to 0
      start one shot 10ms timer
      start SUPPLY_LED_TIMER with timeout of half a second

    else if the event is EXIT
      set IR LED low (PF0)
      set construction LED high (PF3)

    return ReturnEvent
}

void InitOneShotInt_Supply( void ){
    start by enabling the clock to the timer (Wide Timer 4)
    kill a few cycles to let the clock get going
    make sure that timer (Timer B) is disabled before configuring

    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

    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

    set timeout to 10ms
    enable a local timeout interrupt. TBTOIM = bit 8
    enable the Timer B in Wide Timer 0 interrupt in the NVIC EN3 at bit 7
    make sure interrupts are enabled globally

    now kick the timer off by enabling it and enabling the timer to
    stall while stopped by the debugger. TAEN = Bit0, TASTALL = bit1
}

void StartOneShot_Supply_10ms( void ){
    make sure that timer (Timer B) is disabled before configuring
    set timeout to 10ms
    make sure interrupts are enabled globally
    now kick the timer off by enabling it and enabling the timer to
    stall while stopped by the debugger. TAEN = Bit0, TASTALL = bit1
}

void StartOneShot_Supply_30ms( void ){
    make sure that timer (Timer B) is disabled before configuring
    set timeout to 30ms
    make sure interrupts are enabled globally
    now kick the timer off by enabling it and enabling the timer to
    stall while stopped by the debugger. TAEN = Bit0, TASTALL = bit1
}

void OneShotIntResponse_Supply( void ){
    start by clearing the source of the interrupt

    if pulse counter is less than 20
      if pulse counter is an even number
```

```
        set IR LED high
        increment pulse counter by 1
        start 10ms one shot timer

      else
        set IR LED low
        start 30ms one shot timer
        increment pulse counter by 1

    if pulse counter is 20
      start SUPPLY_TIMER with timeout of 3 seconds
      increment number of ball by 1

}
```