```
/*******************************************************************************
 Module
   Motors.c

 Revision
   1.0.1

 Description
   This is the service to send the reload signal during reload state

 Notes

 History
 When           Who     What/Why
 -------------- ---     --------
 01/15/12 11:12 jec      revisions for Gen2 framework
 11/07/11 11:26 jec      made the queue static
 10/30/11 17:59 jec      fixed references to CurrentEvent in RunTemplateSM()
 10/23/11 18:20 jec      began conversion from SMTemplate.c (02/20/07 rev)
 02/17/13 10:00 YtZ              began modification
*******************************************************************************/
/*----------------------------- 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 "SignalService.h"
#include <stdio.h>
#include <mc9s12e128.h>
#include "ME218_E128.h"
#include "ServoLib_E128.h"
#include <stdio.h>

#include <Bin_Const.h>
#include <termio.h>
#include <hidef.h>
#include "S12eVec.h"


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

#define Signal_On_Time 480     //480 ms to send more than 10 pulses
#define Wait_Ball_Time 3000 //3 sec interval for each ball
#define Signal_Pin (BIT4HI)
#define PERIOD 7500         //10ms

/*********************************Original
Definitions*****************************************/
```

```
/*---------------------------- Module Functions ----------------------------*/
/* prototypes for private functions for this machine.They should be functions
   relevant to the behavior of this state machine
*/

void Init_SignalGene(void);
void SignalGene (unsigned char flag);


/*---------------------------- Module Variables ----------------------------*/
// everybody needs a state variable, you may need others as well.
// type of state variable should match htat of enum in header file
SignalServiceState_t CurrentSignalState;

// Motor Timer Interrupt Counter. Multiply by 0.025 to get time in seconds.
//static int MotorTimeCount = 0;

// with the introduction of Gen2, we need a module level Priority var as well
static uint8_t MyPriority;
unsigned char SignalCounter = 0;


/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
     InitMotors

 Parameters
     uint8_t : the priorty of this service

 Returns
     boolean, False if error in initialization, True otherwise

 Description
     Saves away the priority, sets up the initial transition and does any
     other required initialization for this state machine
 Notes

 Author
     J. Edward Carryer, 10/23/11, 18:55
****************************************************************************/
boolean InitSignalService ( uint8_t Priority )
{
  ES_Event ThisEvent;
  puts("Initialize Reload Module \r\n");

  MyPriority = Priority;

  ES_Timer_Init(ES_Timer_RATE_1MS);             // Start Timer
  Init_SignalGene();                                    //Init the signal, since E128 only
have one PWM timer, the setting is same as other PWM used


  ThisEvent.EventType = ES_INIT;
```

```
  CurrentSignalState= WaitForSignalCommand;

  DDRT |=(Signal_Pin);                          //PT4 for the IR emitter
  PTT &=~(Signal_Pin);
  SignalGene(0);                                        //Make sure the signal deactivated


  if (ES_PostToService( MyPriority, ThisEvent) == True)
  {
      return True;
  }else
  {
      return False;
  }
}

/****************************************************************************
 Function
     PostMotors

 Parameters
     EF_Event ThisEvent , the event to post to the queue

 Returns
     boolean False if the Enqueue operation failed, True otherwise

 Description
     Posts an event to this state machine's queue
 Notes

 Author
     J. Edward Carryer, 10/23/11, 19:25
****************************************************************************/
boolean PostSignalService( ES_Event ThisEvent )
{
  return ES_PostToService( MyPriority, ThisEvent);
}

/****************************************************************************
 Function
    RunMotors

 Parameters
   ES_Event : the event to process

 Returns
   ES_Event, ES_NO_EVENT if no error ES_ERROR otherwise

 Description
   add your description here
 Notes
   uses nested switch/case to implement the machine.
 Author
   J. Edward Carryer, 01/15/12, 15:23
```

```
*****************************************************************************/
ES_Event RunSignalService( ES_Event ThisEvent )
{
  ES_Event ReturnEvent;
  ES_Event RefreshEvent;
  SignalServiceState_t NextSignalState;

  static char Pulse_Counter =0;                //Variable to count the number of pulses
  static char Ball_Counter =0;                 //Variable to count the number of balls
collected
  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors
  NextSignalState = CurrentSignalState;

 RefreshEvent.EventType =ES_NO_EVENT;


  switch(CurrentSignalState)
  {

      case WaitForSignalCommand:                        //Idle state, waiting for command
to trigger the signal
      {
          SignalGene(0);                                        //Keep the signal
deactivated
         if(ThisEvent.EventType == ES_RELOAD)   //If command received
         {
          NextSignalState = SigOn;                     //Set the next state to signal on
          Pulse_Counter = 1;                           //Initialize the pulse counter
          Ball_Counter = 0;                            //Initialize the ball counter
          SignalGene(1);                                       //Start signal generating
          ES_Timer_InitTimer(SignalTimer,Signal_On_Time);              //Start the timer
to generate 10 pulses
          puts("C_R;S_ON \r\n");
         }

       } break;

      case SigOn:                       //State when signal is generated
      {
           if (ThisEvent.EventType == ES_TIMEOUT)       //480ms time out
          {
              NextSignalState = WaitForNextBall;        //NExt state to wait for the ball
              SignalGene(0);                                        //Deactiviate the
signal
              ES_Timer_InitTimer (SignalTimer,Wait_Ball_Time);          //Start the 3000
ms timer for ball collection
              Ball_Counter ++;                                  //Add a ball
              puts("S_Off \r\n");
          }

      }break;


      case WaitForNextBall:
```

```
      {
          //SignalGene(0);
          if (Ball_Counter <5)               //if the ball collected is less than 5
          {
            printf("Ball: %d",Ball_Counter );
            if (ThisEvent.EventType == ES_TIMEOUT)
            {
               ES_Timer_InitTimer(SignalTimer,Signal_On_Time);
               PostSignalService(RefreshEvent);
               NextSignalState = SigOn;   //set the signal on to trigger next ball
               SignalGene(1);                           //Activate the signal

            }
          }
          else
          {
               PostSignalService(RefreshEvent);
               SignalGene(0);
               NextSignalState = WaitForSignalCommand;          //Go back to idle state
               puts("Done!\n");
          }

       }break;

  }
     CurrentSignalState= NextSignalState;
     return ReturnEvent;

   }

   SignalServiceState_t QuerySignalService ( void )
{
   return (CurrentSignalState);
}

/*****************************Functions***************************************/

void Init_SignalGene(void)

{
   //Enable timer system 1 and prescale values
   TIM1_TSCR1 = _S12_TEN;     //enable timer system
   TIM1_TSCR2 = (_S12_PR0|_S12_PR2);    //prescale to /32, 1 tick = 1.33 uS

   TIM1_TIOS |= _S12_IOS4; //set output compare
   TIM1_TC4 = TIM1_TCNT + PERIOD; // schedule first rise
   TIM1_TFLG1 = _S12_C4F;     //clear any existing flag coming out of reset
   TIM1_TIE |= _S12_C4I;      //enable local interrupt
   EnableInterrupts;          //globally enable interrupts


}
```

```
void interrupt _Vec_tim1ch4 SignalResponse(void)
{

   TIM1_TFLG1 = _S12_C4F;   // clear flag
   if (SignalCounter %2 == 1)  //if it is off
   {
      TIM1_TCTL1 |=( _S12_OL4|_S12_OM4);
      TIM1_TC4 = TIM1_TC4 + 3 * PERIOD;  //3 periods
   }
   else
   {
      TIM1_TCTL1 |= _S12_OM4;
      TIM1_TCTL1 &= ~(_S12_OL4);

      TIM1_TC4 = TIM1_TC4 + PERIOD;
   }


   SignalCounter ++;

}


void SignalGene (unsigned char flag)
{
   TIM1_TCTL1 |= _S12_OM4;
   TIM1_TCTL1 &= ~(_S12_OL4);
   if (flag == 0)
   {

      TIM1_TIE &= (~_S12_C4I); //disable local interrupt
   }
   else
   {
      TIM1_TIE |= _S12_C4I;  //enable local interrupt
   }
}
```