```
/*******************************************************************************
 Module
   TemplateFSM.c

 Revision
   1.0.1

 Description
   This is a template file for implementing flat state machines under the
   Gen2 Events and Services Framework.

 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)
*******************************************************************************/
/*----------------------------- 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 "CommFSM.h"
#include "mc9s12c32.h"
#include <stdio.h>
#include "EventCheckers.h"
#include "ES_Types.h"
#include "TemplateService.h"
#include "ES_Events.h"
#include "ES_PostList.h"
#include "GameFSM.h"
#include "TargetFSM.h"
#include "BalanceFSM.h"
#include "ES_Timers.h"

/*----------------------------- Module Defines ----------------------------*/
static unsigned int D0_received; //hold recieved data
static unsigned int D1_received; //hold received data
static unsigned int D2_received; //hold received data

static ES_Event SavedEvent;

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

```
/*---------------------------- 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
static CommState_t CurrentState;

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


/*---------------------------- Module Code ----------------------------------*/
/****************************************************************************
 Function
     InitCommFSM

 Parameters
     uint8_t : the priorty of this service

 Returns
     bool, 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
****************************************************************************/
bool InitCommFSM ( uint8_t Priority )
{
  ES_Event ThisEvent;

  MyPriority = Priority;
  // put us into the Initial PseudoState
  CurrentState = InitPCommState;

  //Set up hardware

 D0_DIR = INPUT;
 D1_DIR = INPUT;
 D2_DIR = INPUT;
 ACK_DIR = INPUT;
 STROBE_DIR = INPUT;

 D0_received = HI;
 D1_received = HI;
 D2_received = HI;

 LastStrobeState = STROBE_BIT;

 SavedEvent.EventType = ES_NO_EVENT;

  // post the initial transition event
```

```
  ThisEvent.EventType = ES_INIT;
  if (ES_PostToService( MyPriority, ThisEvent) == True)
  {
      return True;
  }else
  {
      return False;
  }
}

/****************************************************************************
 Function
     PostCommFSM

 Parameters
     EF_Event ThisEvent , the event to post to the queue

 Returns
     bool 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
****************************************************************************/
bool PostCommFSM( ES_Event ThisEvent )
{
  return ES_PostToService( MyPriority, ThisEvent);
}

/****************************************************************************
 Function
    RunCommFSM

 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 RunCommFSM( ES_Event ThisEvent )
{
  ES_Event ReturnEvent;
  //Decalre data to prevent stall of two data send
```

```
  static unsigned char tempMessage = 0xFF;

  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

  switch ( CurrentState )
  {
      case InitPCommState :       // If current state is initial Psedudo State
          if ( ThisEvent.EventType == ES_INIT )// only respond to EF_Init
          {
              CurrentState = Idle;
              //puts("in init state\n\r");


           }
          break;



              // now put the machine into the actual initial state  */


      case Idle :       // If current state is state one
        //puts("in idle state\n\r");
          switch ( ThisEvent.EventType )
          {
              case ES_SEND_REQUEST : //If event is event one
              {
                 //puts("event send request\n\r");
                 if(STROBE_BIT == HI)
                 {
                   //prevent strobe from already being high
                  //store the data in tempMessage until you get the current signal
                  tempMessage = (unsigned char)ThisEvent.EventParam;
                  printf("message aready sending so i am waiting");
                  CurrentState = WaitToReceive;
                  }
                   else if(STROBE_BIT == LO) //strove not assered at start
                   {
                        tempMessage = (unsigned char)ThisEvent.EventParam;
                        D0_DIR = OUTPUT;
                        D1_DIR = OUTPUT;
                        D2_DIR = OUTPUT;
                        STROBE_DIR = OUTPUT;

                        //PUT DATA ON LINE AND STROBE TO HI
                        D0_BIT = ((ThisEvent.EventParam & BIT0HI)==BIT0HI); //if the bit is HI,
                        assigned HI
                        D1_BIT = ((ThisEvent.EventParam & BIT1HI)==BIT1HI);
                        D2_BIT = ((ThisEvent.EventParam & BIT2HI)==BIT2HI);

                        D0_received = D0_BIT;
                         D1_received = D1_BIT;
                         D2_received = D2_BIT;
```

```
                    STROBE_BIT = HI;

                    //initiate the timeout timer
                    ES_Timer_InitTimer(COMM_TIMER, COMM_TIME);

                    //set current state to WaitToAck
                    CurrentState = WaitToAck;
        //          puts("go to waittoack\n\n\r");
                }
        }

           break;

        case ES_STROBE_RISE :
            {
        //  puts("STROBE RISE\n\r");
            //read data and asert the acknowledgement
            D0_received = D0_BIT;
    //      printf("Bit 0: %u \n\r", D0_BIT);
            D1_received = D1_BIT;
    //      printf("Bit 1: %u \n\r", D1_BIT);
            D2_received = D2_BIT;
    //      printf("Bit 2: %u \n\r", D2_BIT);
            DecodeThenPost();

            //set ack to output and assert
            ACK_DIR = OUTPUT;
            ACK_BIT = HI;
        //  puts("acknowledgement sent\n\r");

            //next: wait for strobe fall
            CurrentState = WaitForStrobeFall;
            }
              // repeat cases as required for relevant events
          }                           // end switch on CurrentEvent
          break;

case WaitToReceive :
     //puts("In waiting to recieve\n\r");
     switch ( ThisEvent.EventType )
     {
        case ES_STROBE_RISE :
     {
   //  puts("STROBE RISE\n\r");
        //read data and asert the acknowledgement
        D0_received = D0_BIT;
        D1_received = D1_BIT;
        D2_received = D2_BIT;

        DecodeThenPost();

        //set ack to output and assert
        ACK_DIR = OUTPUT;
```

```
            ACK_BIT = HI;
            //puts("acknowledgement sent\n\r");

            //next: wait for strobe fall
            CurrentState = WaitToSend;
            }
            break;
        }
    break;

    case WaitToAck :
    //  puts("In state wait to ack\n\r");
        switch (ThisEvent.EventType)
        {
            case ES_TIMEOUT : //if a timeout occurs with both C32's asking for ACK
            {
            //re-post the send request
            ES_Event ThisEvent;
            ThisEvent.EventType = ES_SEND_REQUEST;
            ThisEvent.EventParam = tempMessage ;
            PostCommFSM(ThisEvent);
            printf("retry send request\n\r");

            //set all lines back to input
            D0_DIR = INPUT;
            D1_DIR = INPUT;
            D2_DIR = INPUT;
            ACK_DIR = INPUT;
            STROBE_DIR = INPUT;

            //set the current state to idle to restart the process
            CurrentState = Idle;
            //printf("----------------------RETRY MESSAGE: D0 D1 D2 : %d %d %d\n",
            D0_received, D1_received, D2_received);
            }
        break;

    case ES_ACK_RISE : //acknowledgement rise
    {
    //  puts("HAVE AN ACK RISE\n\n\r");
        ES_Timer_StopTimer(COMM_TIMER);
        //deassert data and strobe
            D0_DIR = INPUT;
            D1_DIR = INPUT;
            D2_DIR = INPUT;
            ACK_DIR = INPUT;
            STROBE_DIR = INPUT;

        //SET CURRENTSTATE TO IDEL
        CurrentState = Idle;
    //printf("----------------------MESSAGE SENT: D0 D1 D2 : %d %d %d\n", D0_received,
    D1_received, D2_received);
    }
```

```
    break;
}
break;

    case WaitToSend :
    //puts("in State Waiting to Send\n\n\r");
        switch (ThisEvent.EventType )
        {
            case ES_STROBE_FALL :
            {
                //Desassert ack, assert tempMessage and strobe, start the timer
                ACK_DIR = INPUT;

                //chance the direction of data pins and strobe to output
                D0_DIR = OUTPUT;
                D1_DIR = OUTPUT;
                D2_DIR = OUTPUT;
                STROBE_DIR = OUTPUT;

                //put tempMessage on the data line
                D0_BIT = (( tempMessage & BIT0HI ) == BIT0HI);
                D1_BIT = (( tempMessage & BIT1HI ) == BIT1HI);
                D2_BIT = (( tempMessage & BIT2HI ) == BIT2HI);

                //strobe HI
                STROBE_BIT = HI;

                //start the ack timer
                ES_Timer_InitTimer(COMM_TIMER, COMM_TIME);

                //set currentstate to WaitToAck
                CurrentState = WaitToAck;
                }
                break;
            }
            break;

            case WaitForStrobeFall :
            //  puts("In State waitforstrobefall\n\n\r");
                switch( ThisEvent.EventType )
                {
                    case ES_STROBE_FALL : //Event that strobe signal falls
                    {
                        //Deassert Ack
                        ACK_DIR = INPUT;

                        //Set currentstate to IDLE
                        CurrentState = Idle;

                        if(SavedEvent.EventType!= ES_NO_EVENT)
                        {
                        PostCommFSM(SavedEvent);
                        printf("In WF StrobFall: retry send request\n\r");
```

```
                        SavedEvent.EventType = ES_NO_EVENT;
                    }
                    //printf("----------------------MESSAGE COMPLETE: D0 D1 D2 : %d %d
                    %d\n", D0_received, D1_received, D2_received);
                }
                break;

                //If you receive a send event request -> wait to post it after strobe fall
               case ES_SEND_REQUEST:
                  SavedEvent= ThisEvent;
                  break;

            }
           break;


        //read the data and assert an

    // repeat state pattern as required for other states

  }                                   // end switch on Current State
  return ReturnEvent;
}

/****************************************************************************
 Function
     QueryCommFSM

 Parameters
     None

 Returns
     TemplateState_t The current state of the Template state machine

 Description
     returns the current state of the Template state machine
 Notes

 Author
     J. Edward Carryer, 10/23/11, 19:21
****************************************************************************/
CommState_t QueryCommFSM ( void )
{
   return(CurrentState);
}

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

void DecodeThenPost( void )
{
    ES_Event ThisEvent;
```

```
 ThisEvent.EventParam = 1;
//If Bits correspond to message 0 : 000
if((D2_received == LO) && (D1_received == LO) && (D0_received == LO))
{
     ThisEvent.EventType =  ES_AccessCardDetected;
     ES_PostList00(ThisEvent);
     puts("----------------------MESSAGE RECEIVED: Access Card Detected \n\r");
}
//001
if((D2_received == LO) && (D1_received == LO) && (D0_received == HI))
{
     ThisEvent.EventType = ES_StageEnd;
     ES_PostList02(ThisEvent);
     puts("----------------------MESSAGE RECEIVED: Stage End \n\r");
}
//010
if((D2_received == LO) && (D1_received == HI) && (D0_received == LO))
{
     ThisEvent.EventType = ES_INIT;
     ES_PostList00(ThisEvent);
     puts("----------------------MESSAGE RECEIVED: ES INIT \n\r");
}
//011
if((D2_received == LO) && (D1_received == HI) && (D0_received == HI))
{
     ThisEvent.EventType = ES_AdvanceTarget;
     PostGameFSM(ThisEvent);
     puts("----------------------MESSAGE RECEIVED: Advance Target \n\r");
}
//100
if((D2_received == HI) && (D1_received == LO) && (D0_received == LO))
{
     ThisEvent.EventType = ES_GameEnd;
     ES_PostList00(ThisEvent);
     puts("----------------------MESSAGE RECEIVED: Game End \n\r");
}
//101
if((D2_received == HI) && (D1_received == LO) && (D0_received == HI))
{
     ThisEvent.EventType = ES_AccessCardRemoved;
     ES_PostList00(ThisEvent);
     puts("----------------------MESSAGE RECEIVED: Access Card Removed \n\r");
}
//110
if((D2_received == HI) && (D1_received == HI) && (D0_received == LO))
{
    //ThisEvent.EventType =
    puts("----------------------MESSAGE RECEIVED: 1,1,0 \n\r");
}
//111
if((D2_received == HI) && (D1_received == HI) && (D0_received == HI))
{
    //ThisEvent.EventType =
```

```
       puts("----------------------MESSAGE RECEIVED: 1,1,1 \n\r");
   }

    //Now Post the Event to The GamePlay State machine

}

/****************** EVENT CHECKERS FOR COM TEST ***********************************************/

bool Check4Strobe(void)
{


  //Obtain current Strobe State
  unsigned char CurrentStrobeState;
  bool ReturnVal = False;

  //Get current state of Strobe
  CurrentStrobeState = STROBE_BIT;

  // check for Strobe high and different from last time
  if ( (CurrentStrobeState != LastStrobeState) &&
       (CurrentStrobeState == HI) )
  { // Rising event detected, so post Rising detected event
    ES_Event ThisEvent;
    ThisEvent.EventType = ES_STROBE_RISE;
    ThisEvent.EventParam = 1;
    PostCommFSM(ThisEvent); // this could be any SM post function or EF_PostAll
    ReturnVal = True;

  }
  else if( (CurrentStrobeState != LastStrobeState) &&
       (CurrentStrobeState == LO) )
  { // Falling event detected, so post Falling detected event
    ES_Event ThisEvent;
    ThisEvent.EventType = ES_STROBE_FALL;
    ThisEvent.EventParam = 1;
    PostCommFSM(ThisEvent); // this could be any SM post function or EF_PostAll
    ReturnVal = True;



  }
  LastStrobeState = CurrentStrobeState;

  return ReturnVal;

}



bool Check4Ack(void){
```

```
  //Initialize Last Strobe State to Low
  static unsigned char LastAckState = 0;

  //Obtain current Strobe State
  unsigned char CurrentAckState;
  bool ReturnVal = False;

  //Get current state of Strobe
  CurrentAckState = ACK_BIT;

  // check for Strobe high and different from last time
  if ( (CurrentAckState != LastAckState) &&
       (CurrentAckState == HI) )
  { // Rising event detected, so post Rising detected event
    ES_Event ThisEvent;
    ThisEvent.EventType = ES_ACK_RISE;
    ThisEvent.EventParam = 1;
    PostCommFSM(ThisEvent); // this could be any SM post function or EF_PostAll
    ReturnVal = True;

  }

  LastAckState = CurrentAckState;

  return ReturnVal;

}
```