```
/*******************************************************************************
 Module
   TemplateService.c

 Revision
   1.0.1

 Description
   This is a template file for implementing a simple service under the
   Gen2 Events and Services Framework.

 Notes

 History
 When           Who     What/Why
 -------------- ---     --------
 01/16/12 09:58 jec      began conversion from TemplateFSM.c
*******************************************************************************/
/*----------------------------- 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 "JSRBrainService.h"

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

/*---------------------------- Module Functions ---------------------------*/
/* prototypes for private functions for this service.They should be functions
   relevant to the behavior of this service
*/
void InitSPI(void);
void InitTimer(void);
void fillMatchArray(void);


/*---------------------------- Module Variables ---------------------------*/
// with the introduction of Gen2, we need a module level Priority variable
static uint8_t MyPriority;
static int counter = 0;
static int value;
static int messageFlag = 0;
static uint16_t carrier;
static uint16_t command;
static uint16_t redKnight;
static uint16_t darkKnight;
static int matchStatus[8]= {1,1,1,1,1,1,1,1};
static int Masks[8] = {BIT0HI, BIT1HI, BIT2HI, BIT3HI, BIT4HI, BIT5HI,
BIT6HI, BIT7HI};


/*------------------------------ Module Code ------------------------------*/
/*******************************************************************************
 Function
     InitTemplateService
```

```
 Parameters
     uint8_t : the priorty of this service

 Returns
     bool, false if error in initialization, true otherwise

 Description
     Saves away the priority, and does any
     other required initialization for this service
 Notes

 Author
     J. Edward Carryer, 01/16/12, 10:00
*****************************************************************************/
bool InitJSRBrainService ( uint8_t Priority )
{
  ES_Event ThisEvent;

  MyPriority = Priority;
  /********************************************
   in here you write your initialization code
   *******************************************/
        //"just in case" explicit configuring of ports for SPI
        DDRS &= BIT4LO;
        DDRS |= BIT5HI | BIT6HI | BIT7HI;

        InitSPI();
        InitTimer();
  // post the initial transition event
  ThisEvent.EventType = ES_INIT;
  if (ES_PostToService( MyPriority, ThisEvent) == true)
  {
      return true;
  }else
  {
      return false;
  }
}

/*****************************************************************************
 Function
     PostTemplateService

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

/****************************************************************************
 Function
    RunTemplateService

 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

 Author
   J. Edward Carryer, 01/15/12, 15:23
****************************************************************************/

void InitSPI()
{
        //Enable SPI and set E128 to Master
        SPICR1 |= _S12_SPE | _S12_MSTR;

        //Set Baud Rate to divide 24MHz clock by 1792, which is (6+1) *
2^(7+1)
        SPIBR = _S12_SPPR2 | _S12_SPPR1;        //gets the 6 in upper equation
        SPIBR |= _S12_SPR2 | _S12_SPR1 | _S12_SPR0;   //gets the 7 in the
exponent

        //Set clock to idle high and sample even edges
        SPICR1 |= _S12_CPOL | _S12_CPHA;

        //Disable slave select ouput
        SPICR1 &= ~_S12_SSOE;

        //MSB First
        SPICR1 &= ~_S12_LSFBE;

        // Enable SPI Interrupt
        SPICR1 |= _S12_SPIE;

        //Disable MODFEN to give SPI control of SS line
        SPICR2 &= ~_S12_MODFEN;
}

void InitTimer()
{
        /************Using Timer 0, Channel 7 and Timer 1, Channel
5*************/
        //This is the setup for Timer 0

        // turn timer system on
```

```
        TIM0_TSCR1 = _S12_TEN;

        // set prescale clock to /64 = 375kHz
        TIM0_TSCR2 = _S12_PR1 | _S12_PR2;

        /*  Set up Output Compare 7 as interbyte timer */

        TIM0_TIOS |= _S12_IOS7;                                         //set up
comp 7 to oc, leave the rest alone
        TIM0_TCTL1 &= ~(_S12_OL7 | _S12_OM7);        //no pin connected for
OC7
        TIM0_TC7 = TIM0_TCNT + INTER_BYTE_PERIOD;
        //sched first rise
        TIM0_TFLG1 = _S12_C7F;
        TIM0_TIE |= _S12_C7I;                                                                   //enab

        /*
        //This is the setup for Timer 1

        TIM1_TSCR1 = _S12_TEN;                                          //turn
Timer 1 on
        TIM1_TSCR2 = _S12_PR1 | _S12_PR2;            //set prescale clock to
/64 = 375kHz

        // Set up Output Compare 5 as refresh timer

        TIM1_TIOS |= _S12_IOS5;                                         //set up
comp 5 to oc, leave the rest alone
        TIM1_TCTL1 &= ~(_S12_OL5 | _S12_OM5);        //no pin connected for
OC5
        TIM1_TC5 = TIM1_TCNT + REFRESH_PERIOD;               //sched first
rise
        TIM1_TFLG1 = _S12_C5F;                                          //clear
OC5 Flag
        TIM1_TIE |= _S12_C5I;                                           //enable
OC5 interrupt
        */
        EnableInterrupts;

}

void fillMatchArray(void)
{
int i;
for(i=0; i <8; i++)
{
        value = carrier & Masks[i];
        if (value == Masks[i])
        {
                matchStatus[i] = 1;
        }
        else
        {
                matchStatus[i] = 0;
        }
}
```

```
}

void interrupt _Vec_tim0ch7 TimeoutSystem(void)
{
        ES_Event ThisEvent;

        TIM0_TFLG1 = _S12_C7F;
        EnableInterrupts;
        PTS &= BIT7LO;

        counter++;                                              //increment
module level counter


        if (counter < 6)
        {
                        switch (counter)
        {
                case 1:
                        if (messageFlag == 0)
                        {

                                if ((SPISR & _S12_SPTEF) == _S12_SPTEF)                 //read
                                {
                                        SPIDR = 0x3F;
                                }
                                messageFlag = 1;

                        }
                        else if (messageFlag == 1)
                        {
                                if ((SPISR & _S12_SPTEF) == _S12_SPTEF)                 //read
                                {
                                        SPIDR = 0xC3;
                                }
                                messageFlag = 0;

                        }


                break;


                case 2:

                        if ((SPISR & _S12_SPTEF) == _S12_SPTEF)
        //read SPISR and send command if transmit empty flag is set
                        {
                                SPIDR = 0x00;
                        }

                break;
```

```
                case 3:

                        if ((SPISR & _S12_SPTEF) == _S12_SPTEF)
        //read SPISR and send command if transmit empty flag is set
                        {
                                SPIDR = 0x00;
                        }

                break;



                case 4:

                        if ((SPISR & _S12_SPTEF) == _S12_SPTEF)
        //read SPISR and send command if transmit empty flag is set
                        {
                                SPIDR = 0x00;
                        }

                break;


                case 5:


                        ThisEvent.EventType = Status_Update;
                        fillMatchArray();
                        PostJSRBrainService(ThisEvent);


                break;



        }
                TIM0_TC7 += INTER_BYTE_PERIOD;

        }
        else
        {
                counter = 0;
                TIM0_TC7 += REFRESH_PERIOD;
                //PTS |= BIT7HI;
        }

}

void interrupt _Vec_spi ReadInterrupt(void)
{

   //EnableInterrupts;

        if (counter < 5)
        {
                switch(counter)
                {
```

```
                        case 1 :

                        if ((SPISR & _S12_SPIF) == _S12_SPIF)          //read
data if receive flag is set
                        {
                                command = SPIDR;

                        }

                        #ifdef INTERRUPT_DEBUG_PRINT

                        printf("counter: %i \n\r", counter);
                        //puts("sending 0x3F");
                        printf("command value: %i \n\r", command);

                        #endif

                        break;

                        case 2:

                        if ((SPISR & _S12_SPIF) == _S12_SPIF)          //read
data if receive flag is set
                        {
                                command = SPIDR;
                        }

                        #ifdef INTERRUPT_DEBUG_PRINT

                        printf("counter: %i \n\r", counter);
                        //puts("sending 0x00-first");
                        printf("command value: %i \n\r", command);

                        #endif

                        break;

                        case 3:

                        /************  if messageFlag is set to enable Match
Status Query ******************/
                        if (messageFlag == 1)
                        {

                                if ((SPISR & _S12_SPIF) == _S12_SPIF)
        //read data if receive flag is set
                                {
                                command = SPIDR;
                                }

                                #ifdef INTERRUPT_DEBUG_PRINT

                                printf("counter: %i \n\r", counter);
                                //puts("sending 0x00-second");
                                printf("command value: %i \n\r", command);

                                #endif
```

```
                        }

                        /*************** if messageFlag is set to enable Score
Query **********************/
                        else if (messageFlag == 0)
                        {
                                if ((SPISR & _S12_SPIF) == _S12_SPIF)
        //read data if receive flag is set
                                {
                                redKnight = SPIDR;
                                }

                                #ifdef INTERRUPT_DEBUG_PRINT

                                printf("counter: %i \n\r", counter);
                                //puts("sending 0x00-second");
                                printf("redKnight: %i \n\r", redKnight);

                                #endif

                        }

                        break;

                        case 4:

                        /*************** if messageFlag is set to enable Match
Status Query *****************/
                        if (messageFlag == 1)
                        {
                                if ((SPISR & _S12_SPIF) == _S12_SPIF)
        //read data if receive flag is set
                                {

                                carrier = SPIDR;
                                PTS |= BIT7HI;

                                }

                                #ifdef INTERRUPT_DEBUG_PRINT

                                printf("counter: %i \n\r", counter);
                                //puts("sending 0x00-third");
                                printf("command value: %i \n\r", carrier);
                                #endif
                        }

                        /*************** if messageFlag is set to enable Score
Status Query ******************/
                        else if (messageFlag == 0)
                        {
                                        if ((SPISR & _S12_SPIF) == _S12_SPIF)          //read
                                        {
                                                darkKnight = SPIDR;
                                                PTS |= BIT7HI;
                                        }
```

```
                                        #ifdef INTERRUPT_DEBUG_PRINT

                                        printf("counter: %i \n\r", counter);
                                        //puts("sending 0x00-third");
                                        printf("darkKnight: %i \n\r",
darkKnight);
                                        printf("MESSAGEFLAG: %i \n\r",
messageFlag);

                                        #endif


                        }


                        break;
                }
        }


}



ES_Event RunJSRBrainService( ES_Event ThisEvent )
{
  ES_Event ReturnEvent;
  ES_Event ThatEvent;


  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

  if (ThisEvent.EventType == Status_Update)
  {

        /************** Waiting *****************/

        if (matchStatus[0] == 0 && matchStatus[1] == 0 && matchStatus[2] == 0
&& matchStatus[3] == 0)
        {
                //post waiting to HSM and STOP to motor
                #ifdef ENABLE_JSR

                        ThatEvent.EventType = Waiting;
                        PostMasterHSM(ThatEvent);

                        //ThatEvent.EventType = STOP;
                        //PostMotorService (ThatEvent);

                #endif

                #ifdef READ_ARRAY_DEBUG

                puts("Waiting\n\r");
```

```
                #endif

        }

        /**************** Pas d'Armes ****************/

        if  (matchStatus[0] == 1 && matchStatus[1] == 0 && matchStatus[2] == 0
&& matchStatus[3] == 0)
        {
                //post Pas d'Armes to HSM and charging field to motor
                #ifdef ENABLE_JSR

                        ThatEvent.EventType = pas_d_armes;
                        PostMasterHSM(ThatEvent);

                        //ThatEvent.EventType = ChargingFieldFwd;
                        //PostMotorService (ThatEvent);

                #endif


                #ifdef READ_ARRAY_DEBUG

                puts("Pas d'Armes\n\r");


                #endif
        }



        /***************** Recess ******************/

        if (matchStatus[0] == 1 && matchStatus[1] == 1 && matchStatus[2] == 0
&& matchStatus[3] == 0)
        {
                //post Recess to HSM and STOP to motor
                #ifdef ENABLE_JSR

                        ThatEvent.EventType = recess;
                        PostMasterHSM(ThatEvent);

                        //ThatEvent.EventType = STOP;
                        //PostMotorService (ThatEvent);

                #endif

                #ifdef READ_ARRAY_DEBUG

                puts("Recess\n\r");

                #endif
        }


        /****************** Sudden Death ******************/
```

```
        if (matchStatus[0] == 0 && matchStatus[1] == 0 && matchStatus[2] == 1
&& matchStatus[3] == 0)
        {
                //Sudden Death HSM and charging field to motor
                #ifdef ENABLE_JSR

                        ThatEvent.EventType = sudden_death;
                        PostMasterHSM(ThatEvent);

                #endif

                #ifdef READ_ARRAY_DEBUG

                puts("Sudden Death\n\r");

                #endif
        }


        /****************** Match End **********************/

        if (matchStatus[0] == 1 && matchStatus[1] == 0 && matchStatus[2] == 1
&& matchStatus[3] == 0)
        {
                //post MatchEnd to HSM and STOP to motor
                #ifdef ENABLE_JSR

                        ThatEvent.EventType = end_of_match;
                        PostMasterHSM(ThatEvent);

                        //ThatEvent.EventType = STOP;
                        //PostMotorService (ThatEvent);

                #endif


                //ThatEvent.EventType = STOP;
                //PostMotorService (ThatEvent);
                #ifdef READ_ARRAY_DEBUG

                puts("Match End\n\r");

                #endif
        }


        /****************** Red Status: Reload and Unhorsed ***************/


        /****************** Red Reload Allowed **********************/

        if (matchStatus[4] == 1)
        {
                //post Red Reload to HSM

                #ifdef ENABLE_JSR
```

```
                        ThatEvent.EventType = red_reload_allowed;
                        PostMasterHSM(ThatEvent);

                #endif

                #ifdef READ_ARRAY_DEBUG

                puts("Red Reload Allowed\n\r");

                #endif

        }


        /***************** Red Unhorsed ****************************/

        if (matchStatus[3] == 1)
        {
                //post Red unhorsed to HSM

                #ifdef ENABLE_JSR

                        ThatEvent.EventType = red_unhorsed;
                        PostMasterHSM(ThatEvent);

                #endif

                #ifdef READ_ARRAY_DEBUG

                puts("Red Unhorsed\n\r");

                #endif

        }


        /****************** Black Knight Status: Reload and Unhorsed
****************/


        /****************** Black Reload Allowed **********************/

        if (matchStatus[6] == 1)
        {
                //post Black Reload to HSM

                #ifdef ENABLE_JSR

                        ThatEvent.EventType = black_reload_allowed;
                        PostMasterHSM(ThatEvent);

                #endif

                #ifdef READ_ARRAY_DEBUG

                puts("Black Reload Allowed\n\r");
```

```
                #endif

        }


        /***************** Black Knight Unhorsed
*****************************/

        if (matchStatus[7] == 1)
        {
                //post Black unhorsed to HSM

                #ifdef ENABLE_JSR

                        ThatEvent.EventType = black_unhorsed;
                        PostMasterHSM(ThatEvent);

                #endif

                #ifdef READ_ARRAY_DEBUG

                puts("Black Unhorsed\n\r");

                #endif

        }


        /********************  Display Knight Scores
*******************************/

                #ifdef READ_ARRAY_DEBUG

                printf("dark Knight: %i \n\r", darkKnight);
                printf("red Knight: %i \n\r", redKnight);

                #endif




  }

  return ReturnEvent;
}

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

/*------------------------------- Footnotes -------------------------------*/
/*------------------------------ End of file ------------------------------*/
```