```
/*******************************************************************************
 Module
   ServoService.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 "ServoService.h"

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

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

/*---------------------------- Module Variables ---------------------------*/
// with the introduction of Gen2, we need a module level Priority variable
static uint8_t MyPriority;

static uint16_t Servo_Pulse_Widths[5];

static uint16_t Servo_Period;

  #ifdef SERVO_TEST_HARNESS

        static short last_pot_input;

  #endif



/*------------------------------ Module Code ------------------------------*/
/*******************************************************************************
 Function
     InitServoService

 Parameters
     uint8_t : the priorty of this service
```

```
 Returns
     boolean, 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 InitServoService ( uint8_t Priority )
{
    ES_Event ThisEvent;

    MyPriority = Priority;

  #ifdef SERVO_TEST_HARNESS

    //Initialize Port AD0 to be an analog input port
    ADS12_Init("AAAAAAAA");

         //Initialize last_Pot_Value to present value of Port AD0
    last_pot_input = ADS12_ReadADPin(0);

   #endif

//Set default servo pulse widths
Servo_Pulse_Widths[0] =
(uint16_t)(((float)SERVO_0_DEFAULT)*TICKS_PER_MICROSECOND);
Servo_Pulse_Widths[1] =
(uint16_t)(((float)SERVO_1_DEFAULT)*TICKS_PER_MICROSECOND);
Servo_Pulse_Widths[2] =
(uint16_t)(((float)SERVO_2_DEFAULT)*TICKS_PER_MICROSECOND);
Servo_Pulse_Widths[3] =
(uint16_t)(((float)SERVO_3_DEFAULT)*TICKS_PER_MICROSECOND);
Servo_Pulse_Widths[4] =
(uint16_t)(((float)SERVO_4_DEFAULT)*TICKS_PER_MICROSECOND);

//Configure servo period
Servo_Period = (uint16_t)(((float)(20000 -
1000*NUMBER_OF_SERVOS))*TICKS_PER_MICROSECOND);

//Configure servo channel data direction registers
//and initialize output as low
if(NUMBER_OF_SERVOS > 0)
{
        SERVO_0_DDR |= SERVO_0_BITHI;
        SERVO_0_PT &= ~SERVO_0_BITHI;

//      Servo_Period = (uint16_t)(((float)(SERVO_PERIOD_MICROSECONDS -
SERVO_0_DEFAULT))*TICKS_PER_MICROSECOND);
}

if(NUMBER_OF_SERVOS > 1)
{
```

```
        SERVO_1_DDR |= SERVO_1_BITHI;
        SERVO_1_PT &= ~SERVO_1_BITHI;

//      Servo_Period = (uint16_t)(((float)Servo_Period) -
SERVO_1_DEFAULT*TICKS_PER_MICROSECOND);
}

if(NUMBER_OF_SERVOS > 2)
{
        SERVO_2_DDR |= SERVO_2_BITHI;
        SERVO_2_PT &= ~SERVO_2_BITHI;

        //Servo_Period = (uint16_t)(((float)Servo_Period) -
SERVO_2_DEFAULT*TICKS_PER_MICROSECOND);
}

if(NUMBER_OF_SERVOS > 3)
{
        SERVO_3_DDR |= SERVO_3_BITHI;
        SERVO_3_PT &= ~SERVO_3_BITHI;

//      Servo_Period = (uint16_t)(((float)Servo_Period) -
SERVO_3_DEFAULT*TICKS_PER_MICROSECOND);
}

if(NUMBER_OF_SERVOS > 4)
{
        SERVO_4_DDR |= SERVO_4_BITHI;
        SERVO_4_PT &= ~SERVO_4_BITHI;

//      Servo_Period = (uint16_t)(((float)Servo_Period) -
SERVO_4_DEFAULT*TICKS_PER_MICROSECOND);
}

// Configure Timer 2, Channel 5 to time up to 5 servo channels
// Configure clock scaler to divide system clock by 16
TIM2_TSCR2 |= (_S12_PR2);
TIM2_TSCR2 &= (~_S12_PR1)&(~_S12_PR0);

// Configure Channel 5 as output compare
TIM2_TIOS |= _S12_IOS5;

// Configure Channel 5 to leave pin disconnected
TIM2_TCTL1 &= (~_S12_OM5)&(~_S12_OL5);

// Configure Channel 5 MODRR to leave control of pin to PWM subsystem
MODRR |= _S12_MODRR1;

// Initialize output compare register
TIM2_TC5 = TIM2_TCNT + Servo_Period;

// Clear interrupt flags
TIM2_TFLG1 = _S12_C5F;

// Enable interrupts for channel 5
TIM2_TIE |= _S12_C5I;
```

```
#ifdef SERVO_TEST_HARNESS

        // Configure Timer 1, Channel 5 to provide a polling interval to check
for
        //new input commands during servo calibration
        TIM1_TSCR2 |= (_S12_PR2)|(_S12_PR1)|(_S12_PR0);

        // Configure Channel 5 as output compare
        TIM1_TIOS |= _S12_IOS5;

        // Configure Channel 5 to leave pin disconnected
        TIM1_TCTL1 &= (~_S12_OM5)&(~_S12_OL5);

        // Initialize output compare register
        TIM1_TC5 = TIM1_TCNT + UPDATE_INTERVAL;

        // Clear interrupt flags
        TIM1_TFLG1 = _S12_C5F;

        // Enable interrupts for channel 5
        TIM1_TIE |= _S12_C5I;

        // Enable Timer
        TIM1_TSCR1 |= _S12_TEN;

#endif

// Enable Timer
TIM2_TSCR1 |= _S12_TEN;

// Enable Interrupts
EnableInterrupts;

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

/****************************************************************************
 Function
     PostServoService

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

/*****************************************************************************
 Function
    RunServoService

 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
*****************************************************************************/
ES_Event RunServoService( ES_Event ThisEvent )
{
  ES_Event ReturnEvent;

  #ifdef SERVO_TEST_HARNESS

        short current_pot_input;

        uint16_t pulse_width;

        current_pot_input = ADS12_ReadADPin(SERVO_TEST_POT_PIN);

        if(abs(current_pot_input - last_pot_input) > 5)
        {

                pulse_width = (uint16_t)(MIN_PULSE_WIDTH +
((float)(current_pot_input))*
                (MAX_PULSE_WIDTH - MIN_PULSE_WIDTH)/1024);

                if(pulse_width > MAX_PULSE_WIDTH)
                {
                        pulse_width = MAX_PULSE_WIDTH;
                }
                else if(pulse_width < MIN_PULSE_WIDTH)
                {
                        pulse_width = MIN_PULSE_WIDTH;
                }

                printf("Servo Pulse Width: %d microseconds\n\r", pulse_width);

                if(NUMBER_OF_SERVOS > 0)
```

```
                {
                        UpdateServo(0, pulse_width);
                }

                if(NUMBER_OF_SERVOS > 1)
                {
                        UpdateServo(1, pulse_width);
                }

                if(NUMBER_OF_SERVOS > 2)
                {
                        UpdateServo(2, pulse_width);
                }

                if(NUMBER_OF_SERVOS > 3)
                {
                        UpdateServo(3, pulse_width);
                }

                if(NUMBER_OF_SERVOS > 4)
                {
                        UpdateServo(4, pulse_width);
                }

        }

        last_pot_input = current_pot_input;

  #endif

  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors

  return ReturnEvent;

}

void UpdateServo(unsigned char ServoChannel, uint16_t PulseWidthMicroseconds)
{

        float delta_pulse_width;

        float new_period_microseconds;

        //check for valid values
        if(PulseWidthMicroseconds < MIN_PULSE_WIDTH)
        {
                PulseWidthMicroseconds = MIN_PULSE_WIDTH;

                printf("Invalid servo pulse width, min value is %d us\n\r",
MIN_PULSE_WIDTH);
        }
        else if(PulseWidthMicroseconds > MAX_PULSE_WIDTH)
        {
                PulseWidthMicroseconds = MAX_PULSE_WIDTH;

                printf("Invalid servo pulse width, max value is %d us\n\r",
MAX_PULSE_WIDTH);
```

```
        }

        //update servo channel pulse with
        Servo_Pulse_Widths[ServoChannel] =
(uint16_t)(((float)PulseWidthMicroseconds)*TICKS_PER_MICROSECOND);




}


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

//loop through output compare ISR to fire pulses for five separate servo
channels
void interrupt  _Vec_tim2ch5 ActuateServos(void)
{
   //track current servo channel
   static unsigned char i = 0;

   // clear flag
   TIM2_TFLG1 = _S12_C5F;

   EnableInterrupts;

   //sequentially pulse servo channels
   switch(i)
   {
                case 0:
                {
                        SERVO_0_PT |= SERVO_0_BITHI;
                }
                break;

                case 1:
                {
                        SERVO_0_PT &= ~SERVO_0_BITHI;

                        if(NUMBER_OF_SERVOS > 1)
                        {
                                SERVO_1_PT = SERVO_1_BITHI;
                        }
                }
                break;

                case 2:
                {
                        SERVO_1_PT &= ~SERVO_1_BITHI;

                        if(NUMBER_OF_SERVOS > 2)
```

```
                        {
                                SERVO_2_PT = SERVO_2_BITHI;
                        }
                }
                break;

                case 3:
                {
                        SERVO_2_PT &= ~SERVO_2_BITHI;

                        if(NUMBER_OF_SERVOS > 3)
                        {
                                SERVO_3_PT = SERVO_3_BITHI;
                        }
                }
                break;

                case 4:
                {
                        SERVO_3_PT &= ~SERVO_3_BITHI;

                        if(NUMBER_OF_SERVOS > 4)
                        {
                                SERVO_4_PT = SERVO_4_BITHI;
                        }
                }
                break;

                case 5:
                {
                        SERVO_4_PT &= ~SERVO_4_BITHI;
                }
                break;

                default:
                {
                        printf("ERROR: Default case in servo library ISR\n\r");
                }

        }

        if(i < NUMBER_OF_SERVOS)
        {
        TIM2_TC5 = TIM2_TCNT + Servo_Pulse_Widths[i];

        i++;
        }
        else
        {
                TIM2_TC5 = TIM2_TCNT + Servo_Period;

                i = 0;
        }

}

#ifdef SERVO_TEST_HARNESS
```

```
        //poll potentiometer input and update servo position for calibratio
purposes
        void interrupt  _Vec_tim1ch5 PollPot(void)
        {

                ES_Event ThisEvent;

                // clear flag
        TIM1_TFLG1 = _S12_C5F;

        EnableInterrupts;

                PostServoService(ThisEvent);

                TIM1_TC5 = TIM1_TCNT + UPDATE_INTERVAL;

        }

#endif


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