```
/*******************************************************************************
 Module
   TargetControl.c

 Revision
   1.0.1

 Description
   This is an event checker for potentiometer sensing and vibration motor
   control

 Notes

 History
 When           Who     What/Why
 -------------- ---     --------
 01/16/12 09:58 jec      began conversion from TemplateFSM.c
 11/13/13       whg      converted from template to vibration control
*******************************************************************************/
/*----------------------------- 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 "TargetControl.h"
#include "PWMS12.h"
#include "GameFSM.h"
#include "BalanceFSM.h"
#include "CommFSM.h"
#include "TargetFSM.h"
#include "ES_Timers.h"

/*----------------------------- Module Defines ----------------------------*/
//#define PRINTS // comment this out to removes prints

#define SEL_A_IND            0
#define SEL_B_IND            1
#define NUM_TARGETS        4
#define TARG_PATTERN_LENGTH     10

// PWM definitions
#define OFF         100 // enable is inverted
#define ON          0
#define PULSING   50

#define BlinkOn    0
#define BlinkOff   1
#define UNCOVERED   0
#define COVERED     1

/*---------------------------- Module Functions ---------------------------*/
/* prototypes for private functions for this service.They should be functions
```

```
   relevant to the behavior of this service
*/
void SwitchToTarget(const unsigned char newTarget);

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

static const unsigned char practiceTarget = 1; // target "1" will be the practice/calibration
target
static const unsigned char TargetSelect[4][2] = {{0, 0}, {0, 1}, {1, 0}, {1, 1}}; // mux select
patterns for each target
static const unsigned char targetPattern[TARG_PATTERN_LENGTH] = {0, 1, 2, 3, 2, 1, 0, 3, 2, 0};
static const unsigned char showPattern[NUM_TARGETS] = {0, 1, 2, 3};

static signed char currentTargetInd;
static unsigned char currentShowState;


/*------------------------------ Module Code ------------------------------*/
/****************************************************************************
 Function
     InitTargetControl

 Parameters
     uint8_t : the priority 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 InitTargetControl ( uint8_t Priority )
{
  ES_Event ThisEvent;

  MyPriority = Priority;
  /********************************************
   in here you write your initialization code */
    // initialize pins for mux's and sensor
    TARGET_SENSOR_DIR = TARGET_SENSOR_DIR & (~TARGET_SENSOR_PIN); // set as input
    TARGET_SEL_A_DIR = TARGET_SEL_A_DIR | TARGET_SEL_A_PIN; // output
    TARGET_SEL_B_DIR = TARGET_SEL_B_DIR | TARGET_SEL_B_PIN; // output
    // init PWM pins
    // Already init'ed in VibrationControl
```

```
    // leave default period of 2 ms (f = 500Hz)
    PWMS12_SetPeriod(0x401E, PWMS12_GRP0);

    currentShowState = BlinkOn;

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

/****************************************************************************
 Function
     PostTargetControl

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

/****************************************************************************
 Function
    RunTargetControl

 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 RunTargetControl( ES_Event ThisEvent )
{
  ES_Event ReturnEvent;
  ReturnEvent.EventType = ES_NO_EVENT; // assume no errors



    // check if target pulse timeout
    if ( (ThisEvent.EventType == ES_TIMEOUT) &&
         (ThisEvent.EventParam == TARGET_PULSE_TIMER) ) {
        // Target must have become uncovered
        ES_Event NewEvent;
        NewEvent.EventType = ES_AUncovered;
        PostTargetFSM(NewEvent);
        //puts("AH THE HAND IS GONE!!!\n\r");
        lastTargetState = UNCOVERED;
    }

    // check if target show timeout
    if ( (ThisEvent.EventType == ES_TIMEOUT) &&
         (ThisEvent.EventParam == WAITING_SHOW_TIMER) ) {
        // Continue to next target in show
        RunWaitingForStartShow();

        //puts("Run Start Show \r");
    }

        // check if end show timeout
    if ( (ThisEvent.EventType == ES_TIMEOUT) &&
         (ThisEvent.EventParam == END_SHOW_TIMER) ) {
        // Continue to next target in show
        RunEndOfGameShow();

        //puts("Run Start Show \r");
    }

        // check if blink timeout
    if ( (ThisEvent.EventType == ES_TIMEOUT) &&
         (ThisEvent.EventParam == BLINK_TIMER) ) {
        // Continue to next target in show
        BlinkTarget();

        //puts("Run Start Show \r");
    }


  return ReturnEvent;
}

/****************************************************************************
```

```
 Event Checkers
 ***************************************************************************/

 bool CheckForTargetCovered(void) {
    /* ------ General approach ------
    Signal is a 0-5V pulse at ~100Hz (period = 10 ms)
    check if pin has changed
    if it is a falling edge
        start timer with length > period (ex: 25 ms)
        post event TargetCovered
    end if
    -> when timer expires, target was uncovered
    */

    unsigned char ReturnVal = False;
    unsigned char currentPinState = TARGET_SENSOR_PORT & TARGET_SENSOR_PIN;

    // check if falling edge
    if ( (currentPinState != lastPinState) &&
        (currentPinState != TARGET_SENSOR_PIN) ) {
        // Target must have become covered
        ES_Timer_InitTimer(TARGET_PULSE_TIMER, TARGET_PULSE_TIME);  // restart timer every pulse

        // only post event on first rise
        if (lastTargetState == UNCOVERED) {
            ES_Event NewEvent;
            NewEvent.EventType = ES_ACovered;
            PostTargetFSM(NewEvent);
            // post ES_ACovered
            #ifdef PRINTS
                if (QueryGameFSM() != Lockdown && QueryGameFSM() != WaitForReady)
                puts("OMIGOSH A HAND!!");
            #endif

            lastTargetState = COVERED;
            ReturnVal = True;
        }
    }
    lastPinState = currentPinState;


return ReturnVal;
}

/***************************************************************************
 Public Interface Functions
 ***************************************************************************/
 /* StartPracticeRound will turn on the calibration target
INPUTS:     ---
OUTPUTS:    ---
Comments: State machine should turn off all targets once this target is "hit"
*/
void StartPracticeRound( void ){
```

```
    SwitchToTarget(practiceTarget);
    #ifdef PRINTS
        puts("Start Target Practice \n\r");
    #endif
}

/* StartSynchroRound will turn on the first target in the synchro target list
INPUTS:     ---
OUTPUTS:    ---
Comments: The target order is preset in arrays
          A target timer is started when the target turns on
*/
void StartSynchroRound( void ){
    currentTargetInd = 0;
    if ( AdvanceTarget() == AdvanceSuccess ) {
        #ifdef PRINTS
            puts("Beginning Synchro Round ---------- GO! \n\r");
        #endif
    }
}
/* AdvanceTarget will turn off all targets and turn on the next target in the list
INPUTS:     ---
OUTPUTS:    Flag denoting status of transition
Comments: The target order is preset in arrays
          A target timer is started when the target turns on
*/
TargetFlag_t AdvanceTarget( void ){
    unsigned char nextTarget;
    TargetFlag_t returnFlag = AdvanceSuccess; // assume no errors

    // read next target from array
    currentTargetInd++;
    // do any neccesary index checking
    if ( (currentTargetInd >= TARG_PATTERN_LENGTH) || (currentTargetInd < 0) ) {
        currentTargetInd = 0; // loop to beginning of pattern
        #ifdef PRINTS
            puts("pattern complete - returning to beginning \n\r");
        #endif
    }
    nextTarget = targetPattern[currentTargetInd];
    #ifdef PRINTS
        printf("Next target is #%i \n\r", nextTarget);
    #endif

    // switch targets
    SwitchToTarget(nextTarget);

    // start target timer
    ES_Timer_InitTimer(TARGET_TIMER, TARGET_TIME);

    return returnFlag;
}
```

```
/* RunWaitingForStartShow will slowly cycle through the targets
INPUTS:     ---
OUTPUTS:    ---
Comments: only ONE target can be on at a time
*/
void RunWaitingForStartShow( void ) {
    // turn on next target in show pattern
    unsigned char nextTarget;

    // read next target from array
    currentTargetInd++;
    // do any neccesary index checking
    if ( currentTargetInd >= NUM_TARGETS || currentTargetInd < 0 ) {
        currentTargetInd = 0; // loop to beginning of pattern
    }
    nextTarget = targetPattern[currentTargetInd];

    // switch targets
    SwitchToTarget(nextTarget);

    // restart show timer
    ES_Timer_InitTimer(WAITING_SHOW_TIMER, WAITING_SHOW_TIME);

}

/* RunEndOfGameShow will blink all of the targets on and off
INPUTS:     ---
OUTPUTS:    ---
Comments: cycle targets at 30+ Hz to look like they're all on
*/
void RunEndOfGameShow( void ){
    // implement small state machine for end of game show
    static unsigned int counter;
    unsigned char nextTarget;


    // increment counter
    counter++;

    // if counter reaches limit, change state (essentially a delayed timer)

    switch( currentShowState ) {
        case BlinkOn:
                // cycle through targets

            // read next target from array
            currentTargetInd++;
            // do any neccesary index checking
            if ( currentTargetInd >= NUM_TARGETS || currentTargetInd < 0 ) {
                currentTargetInd = 0; // loop to beginning of pattern
            }
            nextTarget = targetPattern[currentTargetInd];
            //puts("switch target show");
```

```
            // switch targets
            SwitchToTarget(nextTarget);

            if (counter > (10) ) {
                currentShowState = BlinkOff; // stay in this state for now
                counter = 0;
                TurnOffAllTargets();
                puts("end game show - blink off");
            }
            break;

        case BlinkOff:
            // wait
            if (counter > (10)) {
                currentShowState = BlinkOn;
                counter = 0;
                puts("end game show - blink on");
            }
            break;

        // can add servo stuff here
    }
    ES_Timer_InitTimer(END_SHOW_TIMER, END_SHOW_TIME);
}

/* TurnOffAllTargets turns off all of the targets
INPUTS:     ---
OUTPUTS:    ---
Comments:
*/
void TurnOffAllTargets( void ){
    // set enables to off
    PWMS12_SetDuty(OFF, TARGET_CONTROL_CHNL);
    PWMS12_SetDuty(OFF, BRIGHTEN_CONTROL_CHNL);
    StopShow();
}

/* BrightenTarget turns on the other half of target LEDs
INPUTS:     ---
OUTPUTS:    ---
Comments:
*/
void BrightenTarget(void){
    // set brighten duty to 100
    PWMS12_SetDuty(ON, BRIGHTEN_CONTROL_CHNL);
}

/* DimTarget turns on the other half of target LEDs
INPUTS:     ---
OUTPUTS:    ---
Comments:
*/
```

```
void DimTarget(void){
    // set brighten duty to 0
    PWMS12_SetDuty(OFF, BRIGHTEN_CONTROL_CHNL);
}

void StopShow(void){
      ES_Timer_StopTimer(WAITING_SHOW_TIMER);
      ES_Timer_StopTimer(END_SHOW_TIMER);
      ES_Timer_StopTimer(BLINK_TIMER);
      currentTargetInd = 0;
      currentShowState = BlinkOn;
}

void BlinkTarget(void){

    // implement small state machine for blink
    static unsigned int counter;
    // increment counter
    counter++;

    // if counter reaches limit, change state (essentially a delayed timer)

    switch( currentShowState ) {
        case BlinkOn:

            if (counter > (2) ) {
                currentShowState = BlinkOff; // stay in this state for now
                counter = 0;
                BrightenTarget();
                puts("end game show - blink off");
            }
            break;

        case BlinkOff:
            // wait
            if (counter > (2)) {
                currentShowState = BlinkOn;
                counter = 0;
                DimTarget();
            }
            break;

        // can add servo stuff here
    }
    ES_Timer_InitTimer(BLINK_TIMER, BLINK_TIME);

}
/*****************************************************************************
 Private Helper Functions
 *****************************************************************************/

 /* DimTarget turns on the other half of target LEDs
INPUTS:     ---
```

```
OUTPUTS:    ---
Comments:
*/
void SwitchToTarget(const unsigned char newTarget) {
    // turn off current target
    TurnOffAllTargets();

    // change selects to new target
    if ( TargetSelect[newTarget][SEL_A_IND] == 0 ) {
        // set select A low
        TARGET_SEL_A_PORT &= ~TARGET_SEL_A_PIN;
    } else {
        // set select A high
        TARGET_SEL_A_PORT |= TARGET_SEL_A_PIN;
    }

    if ( TargetSelect[newTarget][SEL_B_IND] == 0 ) {
        // set select B low
        TARGET_SEL_B_PORT &= ~TARGET_SEL_B_PIN;
    } else {
        // set select B high
        TARGET_SEL_B_PORT |= TARGET_SEL_B_PIN;
    }

    // turn target on
    PWMS12_SetDuty(PULSING, TARGET_CONTROL_CHNL);
}


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