```c
/**
* Mummy.c
* This file contains the state machine and helper functions for the
* for the Mummy DDM.
*/

#include "ES_Configure.h"
#include "ES_Framework.h"
#include "ADCSWTrigger.h"
#include "PWMTiva.h"
#include "EnablePA25_PB23_PD7_PF0.h"
#include "Mummy.h"
#include "VibrateMotor.h"
#include "GearMotor.h"
#include "ButtonLED.h"

#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "inc/hw_sysctl.h"

#define DDM_TIME 60000 // DDM disARMing time limit [ms]
#define JOY_TIME 30000 // Joy time after disARMing [ms]
#define PWM_FREQ 50 // T = 20 ms (for servos)
#define SERVO_HOPE 2100 // face servo pulse width for Hope
#define SERVO_FEAR 2720 // face servo pulse width for Fear 2700
#define SERVO_JOY 1600 // face servo pulse width for Joy

static uint8_t MyPriority; // Module-level variable for the Mummy module's priority
static MummyState_t CurrentState; // Module-level variable for Mummy module's current state

/* Prototypes */
bool InitializeMummy(uint8_t Priority);
ES_Event RunMummySM(ES_Event CurrentEvent);
bool PostMummySM(ES_Event ThisEvent);
MummyState_t QueryMummySM(void);
static void initMummyPorts(void);
static void initMummy(void);
static void vibrateMotor(uint8_t motorNum);
//static void pinTest(void); // Uncomment when testing pins
static void wireGameLED(uint8_t location);
static void faceServo(uint8_t state);
static void doorMotor(uint8_t state);
static void initMotors(void);
static void initGearMotor(void);
static void rgbEyes(uint8_t choice);
static void armState (uint8_t choice);

/* Module variables */
static uint8_t firstButton[] = {3, 2, 1}; // array to store (arbitrary) sequence for first button (1 <= x <= 3)
static uint8_t secondButton[] = {5, 4, 6}; // array to store (arbitrary) sequence for second button (4 <= x <= 6)
static uint8_t buttonCount = 0; // stores how many button stages have been passed

/**
* Initializes the Mummy service.
* @param Priority: priority of the Mummy service
*/
bool InitializeMummy(uint8_t Priority) {
    ES_Event ThisEvent;
    MyPriority = Priority; // set the priority
    initMummyPorts(); // initialize Mummy ports
    CurrentState = InitMummy; // initial state => InitMummy

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

/**
* Runs the Mummy state machine.
* @param CurrentEvent: the current event that has occurred
*/
ES_Event RunMummySM(ES_Event CurrentEvent) {
    ES_Event ReturnEvent;
    ReturnEvent.EventType = ES_NO_EVENT; // assume no problems will occur
    switch (CurrentState) {
        case InitMummy: {
            if (CurrentEvent.EventType == ES_INIT) {
                initMummy();
                armState(1) ; // Reflect ARMed state
                CurrentState = ARMed; // set pseudo-state to be ARMed
                printf("ARMed\r\n");
            }
        }
        break;
        case Standby: {
            if (CurrentEvent.EventType == ButtonPressed) { // any button pressed
```

```
        if (CurrentEvent.EventParam == 1) {
            doorMotor(0);
        } else {
            CurrentState = ARMed; // next state = ARMed
            printf("ARMed\r\n");
            initMummy();
            armState(1) ; // Reflect ARMed state
        }
    }
}
break;
case ARMed: {
    if (CurrentEvent.EventType == ButtonPressed) {
        CurrentState = WaitForFirstButtonSuccess; // any button pressed => next state = WaitForFirstButtonSuccess
        printf("WaitForFirstButtonSuccess\r\n");
        buttonInput(firstButton[buttonCount]); // turn on button 1 LED
        ES_Timer_InitTimer(DDM_TIMER, DDM_TIME); // start 60-s timer
        ES_Timer_InitTimer(CLOCK_TIMER, TICK); // start clock ticking timer
    }
}
break;
case WaitForFirstButtonSuccess: {
    switch (CurrentEvent.EventType) {
        case ButtonPressed: {
            if (CurrentEvent.EventParam == firstButton[buttonCount]) { // first button pressed
                CurrentState = WaitForSecondButtonSuccess; // next state = WaitForSecondButtonSuccess
                printf("WaitForSecondButtonSuccess\r\n");
                vibrateMotor(firstButton[buttonCount]); // vibrate button 1
                buttonInput(secondButton[buttonCount]); // turn on button 2 LED
            }
        }
        break;
        case ES_TIMEOUT: { // 60-s timer expired
            if (CurrentEvent.EventParam == DDM_TIMER) {
                CurrentState = Standby; // next state = Standby
                printf("Standby\r\n");
                buttonInput(10); // turn off button LEDs
                armState(1) ; // Reflect ARMed state
                rgbEyes(1); // red
                faceServo(1);
            }
        }
        break;
    }
}
break;
case WaitForSecondButtonSuccess: {
    switch (CurrentEvent.EventType) {
        case ButtonPressed: {
            if (CurrentEvent.EventParam == secondButton[buttonCount]) { // second button pressed
                CurrentState = WaitForBeardSuccess; // next state = WaitForBeardSuccess
                printf("WaitForBeardSuccess\r\n");
                vibrateMotor(secondButton[buttonCount]); // vibrate button 2
                buttonInput(7); // turn on beard LED (and turn off button 2 LED)
            }
        }
        break;
        case ButtonReleased: {
            if (CurrentEvent.EventParam == firstButton[buttonCount]) { // first button released
                CurrentState = WaitForFirstButtonSuccess; // next state = WaitForFirstButtonSuccess
                printf("WaitForFirstButtonSuccess\r\n");
                buttonInput(firstButton[buttonCount]); // turn on button 1 LED
            }
        }
        break;
        case ES_TIMEOUT: {
            if (CurrentEvent.EventParam == DDM_TIMER) { // 60-s timer expired
                CurrentState = Standby; // next state = Standby
                printf("Standby\r\n");
                buttonInput(10); // turn off button LEDs
                armState(1) ; // Reflect ARMed state
                rgbEyes(1); // red
                faceServo(1);
            }
        }
        break;
    }
}
break;
case WaitForBeardSuccess: {
    switch (CurrentEvent.EventType) {
        case BeardRotated: { // beard rotated
            if (CurrentEvent.EventParam == buttonCount) { // event param = buttonCount
                if (buttonCount < 2) { // button game continues
                    buttonCount++;
                    CurrentState = WaitForFirstButtonSuccess;
                    printf("WaitForFirstButtonSuccess\r\n");
                    buttonInput(firstButton[buttonCount]); // turn on button 1 LED
                } else { // button game over
                    CurrentState = WaitForLoopMiddle;
                    printf("WaitForLoopMiddle\r\n");
```

```
                    doorMotor(1); // open door/lid
                    wireGameLED(1); // turn on middle LED (and turn off start LEDs)
                    buttonInput(10); // turn off button LEDs

                    /* Display Hope */
                    rgbEyes(0); // eyes
                    faceServo(0); // face
                }
            }
        }
        break;
        case ButtonReleased: {
            if (CurrentEvent.EventParam == secondButton[buttonCount]) { // second button released
                CurrentState = WaitForSecondButtonSuccess; // next state = WaitForSecondButtonSuccess
                printf("WaitForSecondButtonSuccess\r\n");
                buttonInput(secondButton[buttonCount]); // turn on button 2 LED
            } else if (CurrentEvent.EventParam == firstButton[buttonCount]) { // first button released
                CurrentState = WaitForFirstButtonSuccess; // next state = WaitForFirstButtonSuccess
                printf("WaitForFirstButtonSuccess\r\n");
                buttonInput(firstButton[buttonCount]); // turn on button 1 LED
            }
        }
        break;
        case ES_TIMEOUT: {
            if (CurrentEvent.EventParam == DDM_TIMER) { // 60-s timer expired
                CurrentState = Standby; // next state = Standby
                printf("Standby\r\n");
                buttonInput(10);// turn off beard LED
                armState(1) ; // Reflect ARMed state
                rgbEyes(1); // red
                faceServo(1);
            }
        }
        break;
    }
}
break;
case WaitForLoopMiddle: {
    switch (CurrentEvent.EventType) {
        case MiddleHallsFall: { // 2A & 2B event
            //CurrentState = WaitForLoopEnd; // next state = WaitForLoopEnd
            //printf("WaitForLoopEnd\r\n");
            //wireGameLED(0); // turn on start LEDs (and turn off middle LEDs)

            /** Bypass the second stage of the wire loop game and simply disARM **/
            /** This significantly reduces the time required to disARM the DDM **/
            CurrentState = disARMed; // next state = disARMed
            printf("disARMed\r\n");
            ES_Timer_InitTimer(JOY_TIMER, JOY_TIME); // Initialize 30-s timer
            armState(0); // Reflect disARMed state

            /* Display Joy */
            rgbEyes(2); // eyes
            faceServo(2); // face
        }
        break;
        case WireTouch: { // wire touch event
            CurrentState = WaitForLoopResetStage1; // next state = WaitForLoopResetStage1
            printf("WaitForLoopResetStage1\r\n");
            wireGameLED(0); // turn on start LEDs (and turn off middle LEDs)
        }
        break;
        case ES_TIMEOUT: {
            if (CurrentEvent.EventParam == DDM_TIMER) { // 60-s timer expired
                CurrentState = Standby; // next state = Standby
                printf("Standby\r\n");
                armState(1) ; // Reflect ARMed state
                rgbEyes(1); // red
                faceServo(1);
            }
        }
        break;
    }
}
break;
case WaitForLoopResetStage1: {
    switch (CurrentEvent.EventType) {
        case StartHallsFall: { // 1A & 1B event
            CurrentState = WaitForLoopMiddle; // next state = WaitForLoopMiddle
            printf("WaitForLoopMiddle\r\n");
            wireGameLED(1); // turn on middle LEDs (and turn off start LEDs)
        }
        break;
        case ES_TIMEOUT: {
            if (CurrentEvent.EventParam == DDM_TIMER) { // 60-s timer expired
                CurrentState = Standby; // next state = Standby
                printf("Standby\r\n");
                armState(1) ; // Reflect ARMed state
                rgbEyes(1); // red
                faceServo(1);
            }
```

```
                }
                break;
            }
        }
        break;
        case WaitForLoopEnd: {
            switch (CurrentEvent.EventType) {
                case StartHallsFall: { // 1A & 1B event
                    CurrentState = disARMed; // next state = disARMed
                    printf("disARMed\r\n");
                    ES_Timer_InitTimer(JOY_TIMER, JOY_TIME); // Initialize 30-s timer

                    /* Display Joy */
                    rgbEyes(2); // eyes
                    faceServo(2); // face

                }
                break;
                case WireTouch: { // wire touch event
                    CurrentState = WaitForLoopResetStage2; // next state = WaitForLoopResetStage2
                    printf("WaitForLoopResetStage2\r\n");
                    wireGameLED(1); // turn on middle LEDs (and turn off start LEDs)
                }
                break;
                case ES_TIMEOUT: {
                    if (CurrentEvent.EventParam == DDM_TIMER) { // 60-s timer expired
                        CurrentState = Standby; // next state = Standby
                        printf("Standby\r\n");
                        armState(1) ; // Reflect ARMed state
                        rgbEyes(1); // red
                        faceServo(1);
                    }
                }
                break;
            }
        }
        break;
        case WaitForLoopResetStage2: {
            switch (CurrentEvent.EventType) {
                case MiddleHallsFall: {
                    CurrentState = WaitForLoopEnd; // next state = WaitForLoopEnd
                    printf("WaitForLoopEnd\r\n");
                    wireGameLED(0); // turn on start LEDs (and turn off middle LED)
                }
                break;
                case ES_TIMEOUT: {
                    CurrentState = Standby; // next state = Standby
                    printf("Standby\r\n");
                    armState(1) ; // Reflect ARMed state
                    rgbEyes(1); // red
                    faceServo(1);
                }
                break;
            }
        }
        break;
        case disARMed: {
            if (CurrentEvent.EventType == ES_TIMEOUT && CurrentEvent.EventParam == JOY_TIMER) { // 30-s timer expired
                CurrentState = Standby; // next state = Standby
                printf("Standby\r\n");
                armState(1) ; // Reflect ARMed state
                rgbEyes(1); // red
                faceServo(1);
            }
        }
        break;
    }
    return ReturnEvent;
}

/**
* Posts events to the Mummy state machine.
* @param ThisEvent: event to post to Mummy state machine
* @return true if successfully posted; false otherwise
*/
bool PostMummySM(ES_Event ThisEvent) {
  return ES_PostToService(MyPriority, ThisEvent);
}

/**
* Queries the Mummy state machine to determine its state.
* @return state of Mummy state machine
*/
MummyState_t QueryMummySM(void) {
   return CurrentState;
}

/**
* Initializes the port pins for the Mummy DDM.
*/
static void initMummyPorts(void) {
```

```
    ADC0_InitSWTriggerSeq3(BEARD_PIN); // beard potentiometer
    PWM_TIVA_Init();
    PWM_TIVA_SetFreq(PWM_FREQ, FACE_GROUP); // face servo
    PWM_TIVA_SetFreq(PWM_FREQ, CLOCK_GROUP); // clock servo
    PortFunctionInit(); // unlock locked pins

    /* Port A */
    HWREG(SYSCTL_RCGCGPIO) |= SYSCTL_RCGCGPIO_R0; // Port is enabled (1)
    uint8_t dummy = HWREG(SYSCTL_RCGCGPIO); // Allow Port to initialize
    HWREG(GPIO_PORTA_BASE + GPIO_O_DEN) |= (SREG_SCLK_PIN | EYE_RED_PIN | EYE_GREEN_PIN | EYE_BLUE_PIN | MOTOR_1_PIN | MOTOR_2_PIN); // di
    HWREG(GPIO_PORTA_BASE + GPIO_O_DIR) |= (SREG_SCLK_PIN | EYE_RED_PIN | EYE_GREEN_PIN | EYE_BLUE_PIN | MOTOR_1_PIN | MOTOR_2_PIN); // ou

    /* Port B */
    HWREG(SYSCTL_RCGCGPIO) |= SYSCTL_RCGCGPIO_R1; // Port is enabled (1)
    dummy = HWREG(SYSCTL_RCGCGPIO); // Allow Port to initialize
    HWREG(GPIO_PORTB_BASE + GPIO_O_DEN) |= (GMOTOR_IN_1_PIN | GMOTOR_IN_2_PIN | SREG_SER_PIN); // digital I/O pin (1)
    HWREG(GPIO_PORTB_BASE + GPIO_O_DIR) |= (GMOTOR_IN_1_PIN | GMOTOR_IN_2_PIN | SREG_SER_PIN); // output (1), without disturbing other bit

    /* Port C */
    HWREG(SYSCTL_RCGCGPIO) |= SYSCTL_RCGCGPIO_R2; // Port is enabled (1)
    dummy = HWREG(SYSCTL_RCGCGPIO); // Allow Port to initialize
    HWREG(GPIO_PORTC_BASE + GPIO_O_DEN) |= (MOTOR_3_PIN | MOTOR_4_PIN | MOTOR_5_PIN | MOTOR_6_PIN); // digital I/O pin (1)
    HWREG(GPIO_PORTC_BASE + GPIO_O_DIR) |= (MOTOR_3_PIN | MOTOR_4_PIN | MOTOR_5_PIN | MOTOR_6_PIN); // output (1), without disturbing othe

    /* Port D */
    HWREG(SYSCTL_RCGCGPIO) |= SYSCTL_RCGCGPIO_R3; // Port is enabled (1)
    dummy = HWREG(SYSCTL_RCGCGPIO); // Allow Port to initialize
    HWREG(GPIO_PORTD_BASE + GPIO_O_DEN) |= (HALL_2B_PIN | WIRE_PIN | SREG_RCLK_PIN | WIRE_GAME_LED_PIN | BUTTON_3_PIN ); // digital I/O pi
    HWREG(GPIO_PORTD_BASE + GPIO_O_DIR) &= ~(HALL_2B_PIN | WIRE_PIN | BUTTON_3_PIN); // input (0), without disturbing other bits
    HWREG(GPIO_PORTD_BASE + GPIO_O_DIR) |= (SREG_RCLK_PIN | WIRE_GAME_LED_PIN); // output (1), without disturbing other bits

    /* Port E */
    HWREG(SYSCTL_RCGCGPIO) |= SYSCTL_RCGCGPIO_R4; // Port is enabled (1)
    dummy = HWREG(SYSCTL_RCGCGPIO); // Allow Port to initialize
    HWREG(GPIO_PORTE_BASE + GPIO_O_DEN) |= (BUTTON_2_PIN | BUTTON_4_PIN | BUTTON_5_PIN | ARM_STATE_PIN); // digital I/O pin (1)
    HWREG(GPIO_PORTE_BASE + GPIO_O_DIR) &= ~(BUTTON_2_PIN | BUTTON_4_PIN | BUTTON_5_PIN); // input (0), without disturbing other bits
    HWREG(GPIO_PORTE_BASE + GPIO_O_DIR) |= ARM_STATE_PIN; // output (1), without disturbing other bits

    /* Port F */
    HWREG(SYSCTL_RCGCGPIO) |= SYSCTL_RCGCGPIO_R5; // Port is enabled (1)
    dummy = HWREG(SYSCTL_RCGCGPIO); // Allow Port to initialize
    HWREG(GPIO_PORTF_BASE + GPIO_O_DEN) |= (BUTTON_1_PIN | BUTTON_6_PIN | HALL_1A_PIN | HALL_1B_PIN | HALL_2A_PIN); // digital I/O pin (1)
    HWREG(GPIO_PORTF_BASE + GPIO_O_DIR) &= ~(BUTTON_1_PIN | BUTTON_6_PIN | HALL_1A_PIN | HALL_1B_PIN | HALL_2A_PIN); // input (0), without

    //pinTest(); // DEBUG
}

/**
* Tests pins for debugging purposes. Uncomment when testing pins.
*/
/*
static void pinTest(void) {
    //SREG_SCLK_PORT |= SREG_SCLK_PIN;
    //EYE_RED_PORT |= EYE_RED_PIN;
    //EYE_GREEN_PORT |= EYE_GREEN_PIN;
    //EYE_BLUE_PORT |= EYE_BLUE_PIN;
    //MOTOR_1_PORT |= MOTOR_1_PIN;
    //MOTOR_2_PORT |= MOTOR_2_PIN;

    //GMOTOR_IN_1_PORT |= GMOTOR_IN_1_PIN;
    //GMOTOR_IN_2_PORT |= GMOTOR_IN_2_PIN;
    //SREG_SER_PORT |= SREG_SER_PIN;

    //MOTOR_3_PORT |= MOTOR_3_PIN;
    //MOTOR_4_PORT |= MOTOR_4_PIN;
    //MOTOR_5_PORT |= MOTOR_5_PIN;
    //MOTOR_6_PORT |= MOTOR_6_PIN;

    //printf("%i", (HALL_2B_PORT & HALL_2B_PIN));
    //printf("%i", (WIRE_PORT & WIRE_PIN));
    //SREG_RCLK_PORT |= SREG_RCLK_PIN;
    //WIRE_GAME_LED_PORT |= WIRE_GAME_LED_PIN;

    //printf("%i", (BUTTON_1_PORT & BUTTON_1_PIN));
    //printf("%i", (BUTTON_2_PORT & BUTTON_2_PIN));
    //printf("%i", (BUTTON_3_PORT & BUTTON_3_PIN));
    //printf("%i", (BUTTON_4_PORT & BUTTON_4_PIN));
    //printf("%i", (BUTTON_5_PORT & BUTTON_5_PIN));
    //ARM_STATE_PORT |= ARM_STATE_PIN;

    //printf("%i", (BUTTON_6_PORT & BUTTON_6_PIN));
    //printf("%i", (HALL_1A_PORT & HALL_1A_PIN));
    //printf("%i", (HALL_1B_PORT & HALL_1B_PIN));
    //printf("%i", (HALL_2A_PORT & HALL_2A_PIN));
    //printf("%i", (LIM_SWITCH_PORT & LIM_SWITCH_PIN));

    //PWM_TIVA_SetPulseWidth(3200, CLOCK_PIN);
    //PWM_TIVA_SetPulseWidth(3200, FACE_PIN);

    //printf("%u\r\n", ADC0_InSeq3());
}
```

```
*/

/**
* Initializes the Mummy into the ARMed state
*/
static void initMummy(void) {
    /* Display Fear */
    rgbEyes(1); // red
    faceServo(1);

    buttonCount = 0; // reset the button game counter
    buttonLightsOn(); // clear all the output pins on shift register
    buttonLightsOn(); // turn on all button LEDs
    initMotors(); // makes certain vibration motors are not active initially
    initGearMotor(); // makes certain gear motor is not moving initially
}

/**
* Initializes all vibration motors to be off
*/
static void initMotors(void) {
    MOTOR_1_PORT &= ~MOTOR_1_PIN;
    MOTOR_2_PORT &= ~MOTOR_2_PIN;
    MOTOR_3_PORT &= ~MOTOR_3_PIN;
    MOTOR_4_PORT &= ~MOTOR_4_PIN;
    MOTOR_5_PORT &= ~MOTOR_5_PIN;
    MOTOR_6_PORT &= ~MOTOR_6_PIN;
}

/**
* Initializes the gear motor to be off
*/
static void initGearMotor(void) {
    GMOTOR_IN_1_PORT |= GMOTOR_IN_1_PIN;
    GMOTOR_IN_2_PORT |= GMOTOR_IN_2_PIN;
}

/**
* Vibrates the button motors by posting to the VibrateMotor service
* @param motorNum: number of the motor to vibrate
*/
static void vibrateMotor(uint8_t motorNum) {
    ES_Event ThisEvent;
    ThisEvent.EventType = MotorOn;
    if (motorNum > 6) return; // motorNum cannot be greater than 6
    ThisEvent.EventParam = motorNum;
    PostVibrateMotor(ThisEvent);
}

/**
* Controls the wire game LEDs
* @param location: 0 for start LEDs; 1 for middle LED
*/
static void wireGameLED(uint8_t location) {
    if (location == 0) { // start LEDs
        WIRE_GAME_LED_PORT |= WIRE_GAME_LED_PIN;
    } else if (location == 1) { // middle LED
        WIRE_GAME_LED_PORT &= ~WIRE_GAME_LED_PIN;
    }
}

/**
* Makes the face servo show Hope/Fear/Joy
* @param state: the desired state for the face
*/
static void faceServo(uint8_t state) {
    switch (state) {
        case 0: { // Hope
            PWM_TIVA_SetPulseWidth(SERVO_HOPE, FACE_PIN);
        }
        break;
        case 1: { // Fear
            PWM_TIVA_SetPulseWidth(SERVO_FEAR, FACE_PIN);
        }
        break;
        case 2: { // Joy
            PWM_TIVA_SetPulseWidth(SERVO_JOY, FACE_PIN);
        }
        break;
    }
}

/**
* Opens/closes the mummy lid/door by posting to GearMotor.
* @param state: 0 for close; 1 for open
*/
static void doorMotor(uint8_t state) {
    if (state == 0) { // close door
        ES_Event ThisEvent;
        ThisEvent.EventType = DoorClose;
        PostGearMotor(ThisEvent);
```

```
    } else if (state == 1) { // open door
        ES_Event ThisEvent;
        ThisEvent.EventType = DoorOpen;
        PostGearMotor(ThisEvent);
    }
}

/**
* Controls the color of the Mummy eyes (RGB LEDs).
* @param choice: 0 for Hope; 1 for Fear; 2 for Joy
*/
static void rgbEyes(uint8_t choice) {
    switch (choice) {
        case 0: { // blue => Hope
            EYE_BLUE_PORT |= EYE_BLUE_PIN;
            EYE_GREEN_PORT &= ~EYE_GREEN_PIN;
            EYE_RED_PORT &= ~EYE_RED_PIN;
        }
        break;
        case 1: { // red => Fear
            EYE_RED_PORT |= EYE_RED_PIN;
            EYE_GREEN_PORT &= ~EYE_GREEN_PIN;
            EYE_BLUE_PORT &= ~EYE_BLUE_PIN;
        }
        break;
        case 2: { // green => Joy
            EYE_GREEN_PORT |= EYE_GREEN_PIN;
            EYE_BLUE_PORT &= ~EYE_BLUE_PIN;
            EYE_RED_PORT &= ~EYE_RED_PIN;
        }
        break;
    }
}

/**
* Reflects the ARMed state of the DDM by toggling the state of a pin.
* @param choice: 1 if ARMed; 0 if disARMed
*/
static void armState(uint8_t choice) {
    if (choice == 1) {
        ARM_STATE_PORT |= ARM_STATE_PIN; // Set output pin high
    } else {
        ARM_STATE_PORT &= ~ARM_STATE_PIN; // Set output pin low
    }
}
```