Program 1

Source Code

Assign6Prog1MAIN.s

```
.equ RED,    0   // Red    LED on PB6 = LED #0
.equ BLUE,   1   // Blue   LED on PB7 = LED #1
.equ ORANGE, 2   // Orange LED on PB8 = LED #2
.equ GREEN,  3   // Green  LED on PB9 = LED #3

.syntax unified
.section .text.ButtonLED
.global main

// Delay - do nothing for N half-seconds
//   r0 = # half seconds
//   r1 modified
Delay:
    ldr   r1, =0x0200000 // delay count for .5 seconds
Dloop1:
    subs  r1, #1         // decrement delay count
    bne   Dloop1         // repeat
    subs  r0, #1         // # half seconds
    bne   Delay          // repeat for each half second
    bx    lr             // return to main

// Phase0 - All LEDs OFF
Phase0:
    bl    AllLEDsOff
P0Loop:
    ldr   r0, =PHASE
    ldr   r1, [r0]
    cmp   r1, #0
    beq   P0Loop
    bx    lr

// Phase1 - Red-Blue-Orange-Green ON, Red-Blue-Orange ON, Red-Blue ON, Red ON, All
OFF
Phase1:
    mov   r2, #0
P1Loop:
    mov   r0, r2
    bl    LED_OffOn
    add   r2, #1
    cmp   r2, #4
    blt   P1Loop
    mov   r0, #1
    bl    Delay
    ldr   r0, =PHASE
```

```
    ldr   r1, [r0]
    cmp   r1, #1
    beq   Phase1
    bx    lr

// Phase2 - Green-Orange-Blue-Red ON, Green-Orange-Blue ON, Green-Orange ON, Green
ON, All OFF
Phase2:
    mov   r2, #3
P2Loop:
    mov   r0, r2
    bl    LED_OffOn
    sub   r2, #1
    cmp   r2, #-1
    bgt   P2Loop
    mov   r0, #2
    bl    Delay
    ldr   r0, =PHASE
    ldr   r1, [r0]
    cmp   r1, #2
    beq   Phase2
    bx    lr

// Main program
main:
    bl    InitLEDs   // Initialize PB9-6 as outputs to LEDs
    bl    InitButton  // Initialize PA0 as input from button
    bl    Init_EXTI0  // Initialize EXTI0 for button
    ldr   r0, =PHASE
    mov   r1, #0
    str   r1, [r0]    // initialize PHASE

// Main loop
MainLoop:
    bl    Phase0
    bl    Phase1
    bl    Phase2
    b     MainLoop   // Repeat the main loop
```

Button_Drivers.s

```
// Functions for LEDs on PB9-6 and input button on PA0

.include "Equates.s"           // peripheral addresses

// Functions in this file
.global InitButton             // initialize PA0
.global Init_EXTI0             // init button as EXTI0
.global CheckButton            // return button state
.global EXTI0_IRQHandler       // EXTI0 interrupt handler

.syntax unified
```

```
.section .text.ButtonDriver

// GPIO initialization for button
InitButton:
    ldr   r0, =RCC              // RCC register block
    ldr   r1, [r0,#AHBENR]      // read RCC_AHB1ENR
    orr   r1, #GPIOAEN          // enable GPIOA clock
    str   r1, [r0, #AHBENR]     // update AHB1ENR
    ldr   r0, =GPIOA            // GPIOA register block
    ldr   r1, [r0, #MODER]      // current mode register
    bic   r1, #0x03
    str   r1, [r0, #MODER]      // update mode register
    bx    lr

// Initialize EXTI0 for button
Init_EXTI0:
    // Enable clock to SYSCFG
    ldr r0, =RCC
    add r0, r0, #APB2ENR
    ldr r1, [r0]
    orr r1, #1<<0   // enable SYSCFG clock
    str r1, [r0]

    // Configure EXTI0 to trigger on falling edge of PA0
    ldr r0, =SYSCFG
    add r0, r0, #EXTICR1
    ldr r1, [r0]
    bic r1, #0x000F
    str r1, [r0]
    ldr r0, =EXTI
    add r0, r0, #IMR
    ldr r1, [r0]
    orr r1, #1<<0
    str r1, [r0]
    add r0, r0, #(FTSR-IMR)
    ldr r1, [r0]
    orr r1, #1<<0
    str r1, [r0]

    // Enable EXTI0 in NVIC
    ldr r0, =NVIC_ISER
    ldr r1, [r0]
    orr r1, #1<<6
    str r1, [r0]

    bx lr

// CheckButton - return state of push button
// r0 = return value of 0 or 1
CheckButton:
    ldr   r0, =GPIOA            // GPIO port A
    ldrh  r0, [r0, #IDR]        // set bit
    and   r0, #0x01             // mask all but bit 6
    bx    r14                   // return
```

```
    // EXTI0 interrupt handler
.section .text.EXTI0_IRQHandler
.global EXTI0_IRQHandler
EXTI0_IRQHandler:
    push  {r0-r1, lr}
    ldr   r0, =PHASE            // point to PHASE
    ldr   r1, [r0]              // read current value
    add   r1, #1                // increment PHASE
    cmp   r1, #3
    it    eq
    moveq r1, #0                // if PHASE = 3, reset to 0
    str   r1, [r0]              // update PHASE

    mov   r0, #4                // 10ms delay
    bl    Delay                 // call delay

    ldr   r0, =EXTI             // EXTI module
    mov   r1, #1
    str   r1, [r0, #PR]         // reset pending bit in EXTI

    ldr   r0, =NVIC_ICPR0       // NVIC module
    str   r1, [r0]              // reset pending bit in NVIC

    pop   {r0-r1, lr}
    bx    lr
```

LED_Drivers.s

```
// Functions for LEDs on PB9-6

    .include "Equates.s"           // peripheral addresses

// Functions in this file
    .global InitLEDs              // init GPIOB9-6 for LEDs
    .global LED_OffOn             // individual LED OFF/ON
    .global DisplayNum            // display 4-bit # on LEDs

// Global variables defined in main file

    .syntax unified
    .section .text.LEDdrivers

// GPIOB initialization for LEDs: PB9-8-7-6
InitLEDs:
    // enable clock to GPIOB
    ldr     r0, =RCC
    ldr     r1, [r0, #AHBENR]
    orr     r1, #GPIOBEN
    str     r1, [r0, #AHBENR]
    // configure PB9-6 as output pins
    ldr     r0, =GPIOB
    ldr     r1, [r0, #MODER]
```

```
    bic     r1, #0x000FF000
    orr     r1, #0x00055000
    str     r1, [r0, #MODER]
    // set initial output values to 0
    ldr     r1, [r0, #ODR]
    bic     r1, #0x03C0
    str     r1, [r0, #ODR]
    bx      lr

// r0 = bit for LED# 3-0, corresponds to PB9-6
// r1 = 0 for off, 1 for on
LED_OffOn:
    push    {r0-r4}
    add     r0, #6              // change 3:0 to 9:6 for PB9-6
    mov     r4, #1              // on value
    lsl     r4, r4, r0          // shift 1 to position in 9:6
    ldr     r2, =GPIOB          // GPIO port B
    ldrh    r3, [r2, #ODR]      // read current ODR value
    bic     r3, r4              // clear bit for PBx
    cmp     r1, #1              // ON?
    it      ne
    orrne   r3, r4              // set bit for PBx if ON
    strh    r3, [r2, #ODR]      // write new ODR value
    pop     {r0-r4}
    bx      lr                  // return
```

Program 2

Source Code

Main Program of Project

```
       .include "Equates.s"

       .global main
       .global PHASE
       .global PATTERN
       .global COUNT


       .syntax unified
       .section .data
       .align 4
PHASE: .word 0
PATTERN: .word 0
COUNT: .word 0
```

```
       .section .text

main:
    // Initialize User Button (GPIOA0) with external interrupts
    bl InitButton

    // Initialize LEDs (GPIOB9-6)
    bl InitLEDs

    // Initialize TIM6
    bl TIM6_Init

    // Enable TIM6 interrupts
    ldr r0, =NVIC_ISER0
    mov r1, #1
    lsl r1, r1, #TIM6_BIT
    str r1, [r0, #TIM6_OFF]

    // Initialize global variables
    movs r0, #0
    ldr r1, =PHASE
    str r0, [r1]
    ldr r1, =PATTERN
    str r0, [r1]
    ldr r1, =COUNT
    str r0, [r1]

loop:
    // Call the Delay subroutine for a 1-second delay
    mov r0, #1000
    bl Delay

    // Increment COUNT
    ldr r0, =COUNT
    ldr r1, [r0]
    adds r1, r1, #1
    str r1, [r0]

    b loop
```

User_button_drivers.s

```
       .include "Equates.s"

       .global InitButton
       .global EXTI0_IRQHandler

       .syntax unified
       .section .text.UserButton

// Initialize the User Button (GPIOA0) with external interrupts
InitButton:
```

```
    // Enable clock to GPIOA
    ldr r0, =RCC
    ldr r1, [r0, #AHBENR]
    orr r1, #GPIOAEN
    str r1, [r0, #AHBENR]

    // Configure PA0 as input
    ldr r0, =GPIOA
    ldr r1, [r0, #MODER]
    bic r1, #0x00000003
    str r1, [r0, #MODER]

    // Enable EXTI0 interrupt
    ldr r0, =NVIC_ISER0
    mov r1, #1
    str r1, [r0]

    // Set EXTI0 to trigger on the rising edge
    ldr r0, =EXTI
    ldr r1, [r0, #RTSR]
    orr r1, #1
    str r1, [r0, #RTSR]

    // Unmask EXTI0 interrupt
    ldr r1, [r0, #IMR]
    orr r1, #1
    str r1, [r0, #IMR]

    bx lr

// EXTI0 Interrupt Handler
EXTI0_IRQHandler:
    push {r4, lr}

    // Toggle global variable PHASE
    ldr r4, =PHASE
    ldr r1, [r4]
    eor r1, #1
    str r1, [r4]

    // Clear EXTI0 pending interrupt
    ldr r0, =EXTI
    mov r1, #1
    str r1, [r0, #PR]

    pop {r4, lr}
    bx lr
```

LED_controls.s

```
        .include "Equates.s"

        .global InitLEDs
        .global LED_OffOn
        .global DisplayCount
        .global TIM6_DAC_IRQHandler
        .global update_pattern

        .syntax unified
        .section .text.LEDdrivers

InitLEDs:
    // Initialize GPIOB9-6 for LEDs
    // Enable clock to GPIOB
    ldr r0, =RCC
    ldr r1, [r0, #AHBENR]
    orr r1, #GPIOBEN
    str r1, [r0, #AHBENR]
    // Configure PB9-6 as output pins
    ldr r0, =GPIOB
    ldr r1, [r0, #MODER]
    bic r1, #0x000FF000
    orr r1, #0x00055000
    str r1, [r0, #MODER]
    // Set initial output values to 0
    ldr r1, [r0, #ODR]
    bic r1, #0x03C0
    str r1, [r0, #ODR]
    bx lr

// r0 = bit for LED# 3-0, corresponds to PB9-6
// r1 = 0 for off, 1 for on
LED_OffOn:
    push {r0-r4}
    add r0, #6        // Change 3:0 to 9:6 for PB9-6
    mov r4, #1        // On value
    lsl r4, r4, r0    // Shift 1 to position in 9:6
    ldr r2, =GPIOB    // GPIO port B
    ldrh r3, [r2, #ODR]  // Read current ODR value
    bic r3, r4        // Clear bit for PBx
    cmp r1, #1        // ON?
    bne L1            // Skip if ON
    orr r3, r4        // Set bit for PBx
L1: strh r3, [r2, #ODR]  // Write new ODR value
    pop {r0-r4}
    bx lr

// TIM6 interrupt handler
.section .text.TIM6_DAC_IRQHandler, "ax", %progbits
.type TIM6_DAC_IRQHandler, %function
TIM6_DAC_IRQHandler:
```

```
    push {lr}

    // Clear UIF flag
    ldr r0, =TIM6
    ldr r1, [r0, #SR]
    bic r1, r1, #1
    str r1, [r0, #SR]

    // Perform action depending on PHASE value
    ldr r1, =PHASE
    ldr r1, [r1]
    cmp r1, #0
    beq Phase0

Phase1:
    // Perform LED control actions for Phase1
    bl update_pattern

    pop {lr}
    bx lr

Phase0:
    // Turn off all LEDs
    movs r0, #0
    movs r1, #1
    bl LED_OffOn
    movs r0, #1
    bl LED_OffOn
    movs r0, #2
    bl LED_OffOn
    movs r0, #3
    bl LED_OffOn

    pop {lr}
    bx lr

    // Update LED pattern for Phase1
update_pattern:
    push {r4-r5}

    // Load PATTERN value
    ldr r4, =PATTERN
    ldr r4, [r4]

    // Set new LED pattern based on the PATTERN value
    cmp r4, #0
    beq all_on
    cmp r4, #1
    beq three_on
    cmp r4, #2
    beq two_on
    cmp r4, #3
    beq one_on
    b all_off
```

```
all_on:
    // All LEDs on
    movs r0, #0
    movs r1, #1
    bl LED_OffOn
    movs r0, #1
    bl LED_OffOn
    movs r0, #2
    bl LED_OffOn
    movs r0, #3
    bl LED_OffOn
    b update_done

three_on:
    // Three LEDs on
    movs r0, #0
    movs r1, #1
    bl LED_OffOn
    movs r0, #1
    bl LED_OffOn
    movs r0, #2
    bl LED_OffOn
    movs r0, #3
    movs r1, #0
    bl LED_OffOn
    b update_done

two_on:
    // Two LEDs on
    movs r0, #0
    movs r1, #1
    bl LED_OffOn
    movs r0, #1
    bl LED_OffOn
    movs r0, #2
    movs r1, #0
    bl LED_OffOn
    movs r0, #3
    bl LED_OffOn
    b update_done

one_on:
    // One LED on
    movs r0, #0
    movs r1, #1
    bl LED_OffOn
    movs r0, #1
    movs r1, #0
    bl LED_OffOn
    movs r0, #2
    bl LED_OffOn
    movs r0, #3
    bl LED_OffOn
    b update_done
```

```
all_off:
    // All LEDs off
    movs r0, #0
    movs r1, #0
    bl LED_OffOn
    movs r0, #1
    bl LED_OffOn
    movs r0, #2
    bl LED_OffOn
    movs r0, #3
    bl LED_OffOn

update_done:
    // Increment PATTERN value
    ldr r5, =PATTERN
    ldr r5, [r5]
    add r5, r5, #1
    cmp r5, #5
    bne store_pattern
    movs r5, #0
store_pattern:
    ldr r4, =PATTERN
    str r5, [r4]

    pop {r4-r5}
    bx lr
```