아래는 csd_asm.S 코드입니다.

```
#define csd_LED_ADDR 0x41200000

#include "uart_init.s"
#include "uart_print.s"
#include "uart_echo.s"


.extern csd_main

.align 8

// Our interrupt vector table
csd_entry:
        b csd_reset
        b .
        b .
        b .
        b .
        b csd_irq
        b .

.global main
csd_reset:
main:

        uart_init // initialize UART
        ldr r5, =csd_LED_ADDR // save the address status of LED in r5

uart_print:

        uart_print // print LED duration menu
        uart_echo // user input is stored in r3

        // match user input with the LED duration
        cmp r3, #49
        ldreq r3, =#0x68fb0 // iterate 430000 times
        cmp r3, #50
        ldreq r3, =#0xD1F60 // iterate 860000 times
        cmp r3, #51
        ldreq r3, =#0x13AF10 // iterate 1290000 times
        cmp r3, #52
        ldreq r3, =#0x1A3EC0 // iterate 1720000 times
        cmp r3, #53
        ldreq r3, =#0x20CE70 // iterate 2150000 times
        cmp r3, #54
        ldreq r3, =#0x275E20 // iterate 2580000 times
        cmp r3, #55
        ldreq r3, =#0x2D3DD0 // iterate 3010000 times
        cmp r3, #56
        ldreq r3, =#0x419CE0 // iterate 4300000 times

        mov r7, r3 // store the number of iteration (which is already stored in r3)
also in r7

        first_loop:

                mov r6, #1 // LD0 will be turned on (based on GPIO 8-bit register)
                str r6, [r5] // LD0 is turned on
                sub r7, r7, #1 // count down the remaining number of times
                cmp r7, #0 // check if LED should be turned off
                bne first_loop // if times are left, then go to first_loop
```

```
mov r7, r3 // store the number of iteration also in r7

second_loop:

        mov r6, #2 // LD1 will be turned on
        str r6, [r5] // LD1 is turned on
        sub r7, r7, #1 // count down the remaining number of times
        cmp r7, #0 // check if LED should be turned off
        bne second_loop // if times are left, then go to second_loop

mov r7, r3 // store the number of iteration also in r7

third_loop:

        mov r6, #4 // LD2 will be turned on
        str r6, [r5] // LD2 is turned on
        sub r7, r7, #1 // count down the remaining number of times
        cmp r7, #0 // check if LED should be turned off
        bne third_loop // if times are left, then go to third_loop

mov r7, r3 // store the number of iteration also in r7

fourth_loop:

        mov r6, #8 // LD3 will be turned on
        str r6, [r5] // LD3 is turned on
        sub r7, r7, #1 // count down the remaining number of times
        cmp r7, #0 // check if LED should be turned off
        bne fourth_loop // if times are left, then go to fourth_loop

mov r7, r3 // store the number of iteration also in r7

fifth_loop:

        mov r6, #16 // LD4 will be turned on
        str r6, [r5] // LD4 is turned on
        sub r7, r7, #1 // count down the remaining number of times
        cmp r7, #0 // check if LED should be turned off
        bne fifth_loop // if times are left, then go to fifth_loop

mov r7, r3 // store the number of iteration also in r7

sixth_loop:

        mov r6, #32 // LD5 will be turned on
        str r6, [r5] // LD5 is turned on
        sub r7, r7, #1 // count down the remaining number of times
        cmp r7, #0 // check if LED should be turned off
        bne sixth_loop // if times are left, then go to sixth_loop

mov r7, r3 // store the number of iteration also in r7

seventh_loop:

        mov r6, #64 // LD6 will be turned on
        str r6, [r5] // LD6 is turned on
        sub r7, r7, #1 // count down the remaining number of times
        cmp r7, #0 // check if LED should be turned off
        bne seventh_loop // if times are left, then go to seventh_loop

mov r7, r3 // store the number of iteration also in r7

eighth_loop:
```

```
                mov r6, #128 // LD7 will be turned on
                str r6, [r5] // LD7 is turned on
                sub r7, r7, #1 // count down the remaining number of times
                cmp r7, #0 // check if LED should be turned off
                bne eighth_loop // if times are left, then go to eighth_loop

        mov r6, #0
        str r6, [r5] // turn off the LED

        b uart_print // continue receiving user input and keep iterations

forever:
        nop
        b forever

.data
.align 4

// Normal Interrupt Service Routine
csd_irq:
        b .
```

아래는 uart_regs.h 코드입니다.

```
//////////////////////////////////////////////////

#define slcr_UART_RST_CTRL                              0xF8000228
#define slcr_UART_CLK_CTRL                              0xF8000154

#define UART1_BASE                                              0xE0001000
#define UART_CONTROL_REG0_OFFSET            0x0
#define UART_MODE_REG0_OFFSET                       0x4
#define UART_INTRPT_EN_REG0_OFFSET                  0x8
#define UART_INTRPT_DIS_REG0_OFFSET                 0xC
#define UART_INTRPT_MASK_REG0_OFFSET        0x10
#define UART_BAUD_RATE_GEN_REG0_OFFSET      0x18
#define UART_RCVR_TIMEOUT_REG0_OFFSET       0x1C
#define UART_RCVR_FIFO_TRG_LV0_OFFSET       0x20
#define UART_CHANNEL_STS_REG0_OFFSET        0x2C
#define UART_TX_RX_FIFO0_OFFSET                     0x30
#define UART_BAUD_RATE_DIV_REG0_OFFSET      0x34

//////////////////////////////////////////////////////////

#define uart_base                                               0xE0001000
#define uart_Control_reg0                               uart_base
#define uart_mode_reg0                                  uart_base + 0x00000004
#define uart_Baud_rate_gen_reg0                 uart_base + 0x00000018
#define uart_Baud_rate_divider_reg0         uart_base + 0x00000034
#define uart_Modem_ctrl_reg0                uart_base + 0x00000024
#define uart_Modem_sts_reg0                 uart_base + 0x00000028
#define uart_TX_RX_FIFO0                                uart_base + 0x00000030
#define uart_Channel_sts_reg0               uart_base + 0x0000002C
```

아래는 uart_init.s 코드입니다.

```
#include "uart_regs.h"

.macro uart_init

/*
# 1.Reset controller
        ldr             r0, =slcr_UART_RST_CTRL
        ldr             r1, [r0, #0]                    @ read
slcr.UART_RST_CTRL
        orr             r1, r1, #0x0000000A             @ set both
slcr.UART_RST_CTRL[UART1_REF_RST, UART1_CPU1X_RST] (bit[3][1])
        str             r1, [r0, #0]                    @ update

# 2.Configure I/O signal routing
# 3.Configure UART_Ref_Clk
        ldr             r0, =slcr_UART_CLK_CTRL
        ldr             r1, =0x00001402                 @ write 0x00001402 to
slcr.UART_CLK_CTRL @ mov    r1, #0x00001402(*** ERROR ***)
        str             r1, [r0, #0]                    @ update
*/

# 4.Configure controller functions
        ldr             r0, =UART1_BASE

#       4-1. Configure UART character frame
        mov             r1, #0x00000020
        str             r1, [r0, #UART_MODE_REG0_OFFSET]

#       4-2. Configure the Baud Rate
        # a-b. Disable Rx Path and Tx Path
        ldr             r1, [r0, #UART_CONTROL_REG0_OFFSET]       @ read
uart.Control_reg0
        bic             r1, r1, #0x0000003C                                 @
clear TXDIS, TXEN, RXDIS, RXEN (bit[5][4][3][2])
        orr             r1, r1, #0x00000028                                 @
TXDIS = 1 TXEN = 0 RXDIS = 1 RXEN = 0 (bit[5][4][3][2])
        str             r1, [r0, #UART_CONTROL_REG0_OFFSET]       @ update
        # c-d. Write the calculated CD value and BDIV
        mov             r1, #0x0000003E
                        @ CD = 62 (Baud rate 115200)
        str             r1, [r0, #UART_BAUD_RATE_GEN_REG0_OFFSET]       @
update uart.Baud_rate_gen_reg0
        mov             r1, #0x00000006
                @ BDIV = 6 (Baud rate 115200)
        str             r1, [r0, #UART_BAUD_RATE_DIV_REG0_OFFSET]       @
update uart.Baud_rate_divider_reg0 @ strb      r1, [r0,
#UART_BAUD_RATE_DIV_REG0_OFFSET]
        # e. Reset Tx and Px Path
        ldr             r1, [r0, #UART_CONTROL_REG0_OFFSET]       @ read
uart.Control_reg0
        orr             r1, r1, #0x00000003                                 @
set TXRST, RXRST (bit[1][0]:self-clearing) - this resets Tx and Rx paths
        str             r1, [r0, #UART_CONTROL_REG0_OFFSET]       @ update
        # f-g. Enable Rx Path and Tx Path
        ldr             r1, [r0, #UART_CONTROL_REG0_OFFSET]       @ read
uart.Control_reg0
        bic             r1, r1, #0x0000003C                                 @
clear TXDIS, TXEN, RXDIS, RXEN (bit[5][4][3][2])
        orr             r1, r1, #0x00000014                                 @
TXDIS = 0 TXEN = 1 RXDIS = 0 RXEN = 1 (bit[5][4][3][2])
        str             r1, [r0, #UART_CONTROL_REG0_OFFSET]       @ update
```

```
#       4-5. Enable the Controller
        ldr             r1, =0x00000117                                 @
write 0x00000117 to uart.Control_reg0
        str             r1, [r0, #UART_CONTROL_REG0_OFFSET]       @ update

.endm
```

아래는 uart_print.s 코드입니다.

```
.data
string:

        .ascii "---------- LED On Duration ---------"
        .byte 0x0D
        .byte 0x0A
        .ascii "1. 100ms 2. 200ms 3. 300ms 4. 400 ms"
        .byte 0x0D
        .byte 0x0A
        .ascii "5. 500ms 6. 600ms 7. 700ms 8. 1 sec"
        .byte 0x0D
        .byte 0x0A
        .ascii "-----------------------------------"
        .byte 0x0D
        .byte 0x0A
        .ascii "Select:  "
        .byte 0x00

.text

.macro uart_print

        bl uart_trans

.endm


#include "uart_regs.h"

uart_trans:

//////////////////////////////////////////////
//       아래에 오류 발생 시 clean project  //
//////////////////////////////////////////////

        ldr     r0, =0xE0001000
        ldr     r1, =string

TRANSMIT_loop:

        // ---------  Check to see if the Tx FIFO is empty
------------------------------
        ldr     r2, [r0, #0x2C] @ get Channel Status Register
        and     r2, r2, #0x8            @ get Transmit Buffer Empty bit(bit[3:3])
        cmp     r2, #0x8                                @ check if TxFIFO is empty
and ready to receive new data
        bne     TRANSMIT_loop           @ if TxFIFO is NOT empty, keep checking
until it is empty

//--------------------------------------------------------------------------
---

        ldrb    r3, [r1], #1
        strb            r3, [r0, #0x30] @ fill the TxFIFO with 0x48
        cmp     r3, #0x00
        bne             TRANSMIT_loop

        mov             pc, lr                          @       return to the caller
```

아래는 uart_echo.s 코드입니다.

```
.macro uart_echo

        bl uart_configuration

.endm

#include "uart_regs.h"

uart_configuration:

        ldr r0,  =slcr_UART_RST_CTRL
        ldr r1, =0x0     //      reset UART
        str r1, [r0]

        ldr r0, =slcr_UART_CLK_CTRL
        ldr r1, =0x1402 // divisor = 0x14 (ref clk = 50MHz), srcsel = 0, CLKACT1 =
true, CLKACT0 = false
        str r1, [r0]

        ldr r0, =uart_mode_reg0
        ldr r1, =0x20
        str r1, [r0]

        ldr r0, =uart_Control_reg0
        ldr r1, =0x28    //uart off
        str r1, [r0]

        ldr r0, =uart_Baud_rate_gen_reg0
        ldr r1, =0x3e
        str r1, [r0]

        ldr r0, =uart_Baud_rate_divider_reg0
        ldr r1, =0x6
        str r1, [r0]

        ldr r0, =uart_Control_reg0
        ldr r1, =0x00000117     //uart start
        str r1, [r0]

        ldr r0, =uart_Modem_ctrl_reg0
        ldr r1, [r0]

        ldr r0, =uart_Modem_sts_reg0
        ldr r1, [r0]

        ldr r0, =uart_TX_RX_FIFO0
        ldr r1, =uart_Channel_sts_reg0

recvloop:

        // check empty: bit#1 is Rx FIFO empty or not (1: empty, 0: not empty)
        ldr r2, [r1]
        and r2, r2, #1<<1
        cmp r2, #0

        bne recvloop // if nothing received yet then keep checking

        // read data and transmit it back to sender
        ldreq r3, [r0]
        streq r3, [r0]

        // since user doesn't press enter, program should add LF next to CR
```

```
mov r4, #13 // 0xD (13) is carriage return in ASCII
streq r4, [r0]
moveq r4, #10 // 0xA (10) is linefeed in ASCII
streq r4, [r0]
streq r4, [r0] // UART transmit two linefeeds and echo operation finishes

mov             pc, lr                          @     return to the caller
```