12/8/22
Dmitrii Fotin

Part 1

The initial idea of how the program should work is below.

Load pointer to LED 3
WHILE true DO
    set LED 3 high
    delay 1 s
    set LED 4 low
    delay 1 s

The board LEDs have the following GPIO connections per BBB SRM

LED	GPIO SIGNAL	PROC PIN
USR0	GPIO1_21	V15
USR1	GPIO2_22	U15
USR2	GPIO2_23	T15
USR3	GPIO2_24	V16

- Found out all LEDs are actually under GPIO1_xx, GPIO2 is a mistake. This means only GPIO1 needs to be initialized.
- Had some issues with initializing the clock. Found out I should use MOV instead of LDR for small numbers.
- Unable to figure out the right delay loop counter, as the actual number calculated based on the 1GHz processor speed was not resulting in a 1 second delay.
- In the end, came up with an arbitrary number that delays the loop by about a second.
- Got 1 LED to work

After getting LED 0 to work, I proceeded to write an algorithm to cycle back and forth through all four LEDs.

```
Initialize clock at R0
Set GPIO1 LEDs to output at R0 (clock doesn't need it anymore)
      Offset GPIO1 address 0x4804C000 with 0x134 for data out (R1/R9)
      Write 0xFE1FFFFF that denotes the four LEDs as zeroes to 0x4804C134
Assign clock counter to R8
Assign edge addresses for LED 0 and LED 3 to registers (R6/R7, to check
when to change cycle direction)
Assign bool direction value 1 to R4
Assign current LED address to R5 (R5 changes to turn on/off LEDs while R6 &
R7 remain the same to check edge addresses)
Set LED 0 HIGH (0x4804C194)
```

```
WHILE always DO (no condition to end the loop)
      Set the LED at the current address in R5 HIGH
      Go to delay loop

REPEAT
      Decrement clock counter by 1
UNTIL 0x00FFFFFF empty cycles passed, i.e. clock counter is 0

Set the LED currently on to clear data (0x4804C190)
Set the address back to 0x4804C194 for the next LED
Reset the clock counter to 0x00FFFFFF
Check if direction (R4) is forward (1) or backward (0)

IF R4==1 THEN
      Divide current LED address by two (move right by one bit)
      Check if new address is LED3

      IF address is LED3 THEN
            Change R4 (bool for direction) to 0
            Branch back to the main loop to set the next LED high
IF R4 != 1 THEN
      Multiply current LED address by two (move left by one bit)
      Check if new address is LED0

      IF address is LED0 THEN
            Change R4 (bool for direction) to 1
            Branch back to the main loop to set the next LED high
```

- The final program works as expected. The full code for Project 2 part 1 is below.

@ BBB built-in LED cycle-through program
@ Turns on on-board LEDs one after the other for 1 s

@ Uses R0-R9

@ Dmitrii Fotin December 2022

```
.text
.global _start
_start:

      @initialize clock
      MOV R0,#0x02
```

```
      LDR R1,=0x44E000AC
      STR R0,[R1]

      @initialize GPIO1 LEDs as data out
      LDR R0,=0x4804C000
      ADD R1,R0,#0x134
      LDR R9,[R1]
      MOV R2,#0xFE1FFFFF
      AND R9,R2,R9
      STR R9,[R1]

      LDR R8,=0x00FFFFFF      @1s clock counter for 1GHz processor
      MOV R7,#0x01000000      @adress for LED3, R7
      MOV R6,#0x00200000      @adress for LED0, R6
      MOV R4,#1

      ADD R3,R0,#0x194        @set GPIO to output
      MOV R5,R7               @copy address to R5, so R5 can be changed
while R7
                              @stays the same for edge address checking

      @turn on LED loop
LED_LOOP:
      STR R5,[R3]             @set current LED HIGH
      B WAIT_1S               @go to delay loop

      @delay loop of 1s
WAIT_1S:
      SUBS R8,R8,#1           @decrement clock counter
      BNE WAIT_1S       @until it's 0
      SUB R3,R3,#0x4          @set address to clear LED (set low)
      STR   R5,[R3]           @clear LED
      ADD R3,R3,#0x4          @set address back to set LED (set high)
      LDR R8,=0x000FFFFF      @reset clock counter
      TEQ R4,#1               @check if direction is forward/1 or
backward/0
      BEQ STEP_FOR_LED  @go to step forward branch if 1
      BNE STEP_BACK_LED @go to step backward branch if 0

      @step to next LED (forward direction)
STEP_FOR_LED:
      MOVS R5, R5, LSR #1     @go to next LED address (divide by 2)
      TEQ R5,#0x00200000      @check if new address is LED3
```

```
      BEQ BACKWARD            @if yes, go to branch to change direction
      B LED_LOOP              @if no, go to main loop to set new LED high

      @step to next LED (backward direction)
STEP_BACK_LED:
      MOVS R5, R5, LSL #1     @go to next LED address (multiply by 2)
      TEQ R5,#0x01000000      @check if new address is LED0
      BEQ FORWARD       @if yes, go to branch to change direction
      B LED_LOOP              @if no, go to main loop to set new LED high

      @set direction for cycling through LEDs
FORWARD:
      ADD R4,R4,#1            @change R4 to 1 to cycle from LED3 to LED0
      B LED_LOOP              @go to main loop to set new LED high
BACKWARD:
      SUB R4,R4,#1            @change R4 to 0 to cycle from LED0 to LED3
      B LED_LOOP              @go to main loop to set new LED high

.END
```

Part 2

For part two, I reused the LED cycle code and built additional functionality around it.

- Had some issues with setting up the interrupt vector, didn't realize the startup_ARMCA8.s also needed to be changed. Once I carefully followed the instructions in the book, the interrupt was set up correctly
- Didn't fully understand the importance of exiting the interrupt and had issues with the button not working as a result, since the program was stuck in the interrupt and couldn't initiate a new one. Managed to get full responsiveness from the button after putting commands to load 'next to last' instruction address to PC
- The fact that the program was returning to where it was interrupted made the algorithm a lot more complex, as now I had to set up conditional statements in both loops to branch out to correct instructions per button press

The final algorithm is shown below.

```
Initialize the stacks
Initialize clock at R0
Set GPIO1 LEDs to output at R0 (clock doesn't need it anymore)
      Offset GPIO1 address 0x4804C000 with 0x134 for data out (R1/R9)
      Write 0xFE1FFFFF that denotes the four LEDs as zeroes to 0x4804C134
Set GPIO1 button pin to input
```

```
      Offset GPIO1 address 0x4804C000 with 0x14C for falling edge detection
      Write 0x00000008 that denotes pin 3 for the button to 0x4804C14C
      Offset the resulting GPIO1 address to set IRQ status
Initialize INTC
RESET branch (in part 1, all this was the initial set up, in part 2 this
needs to be reset every time the program switches from LED loop to empty
loop, it executes at the very beginning as well, as it's the first set of
instructions after clock, interrupt and some non-changing
registers/inputs/outputs are initialized)
Assign clock counter to R8
Assign edge addresses for LED 0 and LED 3 to registers (R6/R7, to check
when to change cycle direction)
Assign bool direction value 1 to R4 (we start with LED3 and go forward each
time we enter the LED loop with button push)
Clear all LEDs
Assign LED 0 address to R5
(EMPTY LOOP)
WHILE always DO
      (Check if R12 is 0 or 1, i.e. empty loop or LED loop)
      IF R12 == 0 THEN
            do nothing (repeat the loop)
      ELSE
            go to LED loop
(When interrupted, the program goes to the custom interrupt director)

(INTERRUPT DIRECTOR)
      Store registers in stack
      Check bit 2 at INTC-PENDING-IRQ3 (0x482000F8)

      IF not button push THEN
            Go to empty loop
      ELSE
            (check bit 3 (button pin) of GPIO1_IRQSTATUS_0)
            IF bit 3 is HIGH THEN
                  Go to branch to set/clear R12 (bool that dictates what
                  loop to go to after interrupt is exited)
            ELSE
                  Go to empty loop

(ON/OFF branch)
      Turn off GPIO_3 interrupt in GPIO1_IRQSTATUS_0
      Clear bit 0 in INTC_CONTROL
      Check if R12 is 0 or 1
```

```
      IF R12==0 THEN
            Go to OFF branch
      ELSE
            Go to ON branch

(ON branch)
      Change R12 to 0
      Restore registers
      Load prior instruction address to PC (go to LED loop, where it checks
R12 and redirects to empty loop cause R12==0)
(OFF branch)
      Change R12 to 1
      Restore registers
      Load prior instruction address to PC (go to empty loop, where it
checks R12 and redirects to LED loop cause R12==1)

(LED LOOP)
WHILE always DO (no condition to end the loop)
      Set the LED at the current address in R5 HIGH
      Go to delay loop
(DELAY LOOP)
REPEAT
      IF R12==0 THEN
            Go to empty loop
      ELSE
            Decrement clock counter by 1
UNTIL 0x00FFFFFF empty cycles passed, i.e. clock counter is 0

Set the LED currently on to clear data (0x4804C190)
Set the address back to 0x4804C194 for the next LED
Reset the clock counter to 0x00FFFFFF
Check if direction (R4) is forward (1) or backward (0)

IF R4==1 THEN
      Divide current LED address by two (move right by one bit)
      Check if new address is LED3

      IF address is LED3 THEN
            Change R4 (bool for direction) to 0
            Branch back to the main loop to set the next LED high
IF R4 != 1 THEN
      Multiply current LED address by two (move left by one bit)
      Check if new address is LED0
```

```
	IF address is LED0 THEN
		Change R4 (bool for direction) to 1
		Branch back to the main loop to set the next LED high
```

The final code for parts 1 and 2 is below.

```
@ BBB built-in LED cycle-through program with button input
@ Turns on on-board LEDs one after the other for 1s each if button is pressed
@ Turns off all LEDs if button pressed again
@ Continues to cycle through LEDs or do nothing with each button press

@ Uses R0-R10, R12

@ Dmitrii Fotin December 2022

.text
.global _start
.global INT_DIRECTOR
_start:
	@stack initialization
	LDR R13,=STACK1
	ADD R13,R13,#0x1000
	CPS #0x12
	LDR R13,=STACK2
	ADD R13,R13,#0x1000
	CPS #0x13

	@initialize clock
	MOV R0,#0x02
	LDR R1,=0x44E000AC
	STR R0,[R1]

	@initialize GPIO1 LEDs as data out
	LDR R0,=0x4804C000	@GPIO1 address
	ADD R1,R0,#0x134	@set address to data out
	LDR R9,[R1]
	MOV R2,#0xFE1FFFFF	@LED addresses
	AND R9,R2,R9		@logical AND to make sure the final
				@address is correct
	STR R9,[R1]		@set LEDs to data out
```

```
      MOV R12,#0                @boolean register that determines if program
                                @runs the LED loop or the empty loop

@set up the button input GPIO
      ADD R6,R0,#0x14C          @GPIO1 base address + falling edge offset
                                @assignment to R6
      MOV R7,#0x00000008        @GPIO_3 address assignment to R7
      LDR R10,[R6]              @load GPIO1 base address + falling edge
                                @offset to R10
      ORR R10,R10,R7            @zero out the bit for GPIO_3 and store
                                @the final word in R10
      STR R10,[R6]              @store the address of the final word in R6
      ADD R6,R0,#0x34           @GPIO1 base address + OE assignment to R6
      STR R7,[R6]               @enable GPIO_3 request to PONTRPEND1

@initialize INTC
      LDR R6,=0x482000E8        @address of INTC_MIR_CLEAR3
      MOV R7,#0x04              @value to unmask INTC INT 98, GPIOINT1A
      STR R7,[R6]               @write to INTC_MIR_CLEAR3

      MRS R10, CPSR             @copy CPR to R10
      BIC R10,#0x80             @clear bit 7
      MSR CPSR_c,R10            @write back to CPSR

@default values for program start & when everything needs to be reset
@before going back into empty loop after LED loop
RESET:
      LDR R8,=0x000FFFFF        @1s clock counter for 1GHz processor
      ADD R3,R0,#0x190          @default GPIO1 address + clear values offset
      MOV R5,#0x01E00000        @adress for all on-board LEDs
      STR R5,[R3]               @clear all LEDs
      ADD R3,R3,#0x4            @offset to set LEDs
      MOV R4,#1                 @boolean value that determines
                                @if LEDs cycle forward or backward
      MOV R5,#0x01000000        @address for on-board LED3
```

```
@empty loop that waits for button interrupt
WAIT_FOR_BUTTON:
      NOP
      TEQ R12,#1              @as the interrupt returns to the empty loop
                              @after even button press due to PC value,
      BEQ LED_LOOP            @R12 is changed during interrupt to allow to
                              @branch to LED loop
      BNE WAIT_FOR_BUTTON     @otherwise, continue executing empty loop

@chaining the interrupt vector with custom instructions
INT_DIRECTOR:
      STMFD SP!,{R0,R6,R7,R10,LR}   @push registers on stack
      LDR R0,=0x482000F8            @address of INTC-PENDING-IRQ3
      LDR R6,[R0]                   @read INTC-PENDING-IRQ3
      TEQ R6,#0x00000004            @check bit 2
      BNE WAIT_FOR_BUTTON           @if not from GPIOINT1A, go to empty
loop
      LDR R0,=0x4804C02C            @if yes, load GPIO1_IRQSTATUS_0 address
      LDR R6,[R0]                   @read STATUS
      TEQ R6,#0x00000008            @check if bit 3 == 1
      BEQ ON_OR_OFF                 @if yes, go to check/set bool
                                    @for LED loop vs. empty loop
      LDMFD SP!,{R0,R6,R7,R10,LR}   @if not, then not button interrupt,
                                    @restore registers
      B WAIT_FOR_BUTTON             @and go back to empty loop

ON_OR_OFF:
      MOV R6,#0x00000008      @turn off GPIO_3 interrupt
      STR R6,[R0]             @write to GPIO1_IRQSTATUS_0
      LDR R0,=0x48200048      @address of INTC_CONTROL
      MOV R6,#0x01            @clear bit 0
      STR R6,[R0]             @write to INTC_CONTROL

      TEQ R12,#1              @check if R12 (bool for LED vs. empty) is 1/0
      BEQ ON                  @go to ON branch to run the LED loop
      BNE OFF                 @go to OFF branch to run the empty loop
```

```
@if the last state was LED loop on, change R12 to cycle through
@empty loop next, restore registers and go back to PC address
@(in the LED loop)
ON:
      SUB R12,R12,#1                  @set R12 to 0, so empty loop is run
      LDMFD SP!,{R0,R6,R7,R10,LR}     @restore registers
      SUBS PC,LR,#4                   @set PC to prior instruction address
(in LED loop)

@if the last state was empty loop on, change R12 to cycle through
@LED loop next, restore registers and go back to PC address
@(in the empty loop)

OFF:
      ADD R12,R12,#1                  @set R12 to 1, so LED loop is run
      LDMFD SP!,{R0,R6,R7,R10,LR}     @restore registers
      SUBS PC,LR,#4                   @set PC to prior instruction address
                                      @(in empty loop)

@turn on LED loop, turns on the LED at the current address in R5
@R3 contains the based GPIO1 address + set value offset
LED_LOOP:
      STR R5,[R3]               @set current LED HIGH
      B WAIT_1S                 @go to delay loop

@delay loop that also includes steps to turn off the LED currently on
@after a delay
WAIT_1S:
      TEQ R12,#1                @PC returns to this point after interrups
      BNE RESET                 @if R12 is set to 0, go to empty loop
      SUBS R8,R8,#1             @decrement time counter,
                                @i.e. delay until counter is 0
      BNE WAIT_1S
      SUB R3,R3,#0x4            @offset to clear LED
      STR R5,[R3]               @clear LED
      ADD R3,R3,#0x4            @offset back to set LED
      LDR R8,=0x000FFFFF        @reset counter
      TEQ R4,#1                 @check cycle direction (1 forward, 0 back)
      BEQ STEP_FOR_LED          @go to forward branch if R4 is 1
      BNE STEP_BACK_LED         @go to backward branch if R4 is 0
```

```
@step to next LED (forward direction)
STEP_FOR_LED:
      MOVS R5, R5, LSR #1     @divide LED address by 2 (step to lower bit)
      TEQ R5,#0x00200000      @check if LED0/lowest bit reached
      BEQ BACKWARD            @branch to change direction if yes
      B LED_LOOP              @restart LED loop if no

@step to next LED (backward direction)
STEP_BACK_LED:
      MOVS R5, R5, LSL #1     @multiply LED address by 2 (to to higher bit)
      TEQ R5,#0x01000000      @check if LED3/highest bit is reached
      BEQ FORWARD             @branch to change direction if yes
      B LED_LOOP              @restart LED loop if no

@set direction for cycling through LEDs
FORWARD:
      ADD R4,R4,#1            @set R4 to 1 to go in forward to direction
      B LED_LOOP              @restart LED loop
BACKWARD:
      SUB R4,R4,#1            @set R4 to 0 to go in backward to direction
      B LED_LOOP              @restart LED loop

.data
.align 2
STACK1:     .rept 1024
            .word 0x0000
            .endr
STACK2:     .rept 1024
            .word 0x0000
            .endr

.END
```