ECE371 Project 2
December 13, 2019

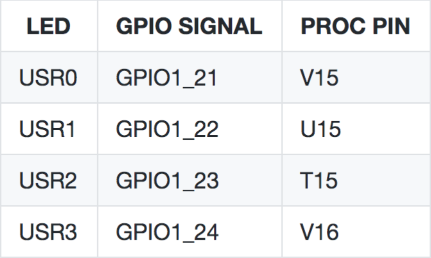
Project 2 Full report and design log

Nick Porter

Part 1:
The first part of this project required developing a program which turns on two pairs of LEDs built into the BeagleBone Black board and alternating pulses between them.

I started the design process by looking through the relevant sections of the textbook and picking out the necessary addresses and values I would need to set up the GPIO and turn the LEDs on and off. Then I developed a high and low level algorithm to implement. These steps are all documented in my design log scans, included later in this report. I have organized the relevant addresses, values and algorithms in the following section.

The BeagleBone Black system reference manual gives the locations of the 4 user LEDs on the board:



LED	GPIO SIGNAL	PROC PIN
USR0	GPIO1_21	V15
USR1	GPIO1_22	U15
USR2	GPIO1_23	T15
USR3	GPIO1_24	V16

The textbook lists the base addresses of each GPIO module and GPIO1 has a base address of 0x4048C000. Using bit tables (shown in the scans of my design logs) I found the values to write to set the appropriate addresses to set outputs and set high or low.

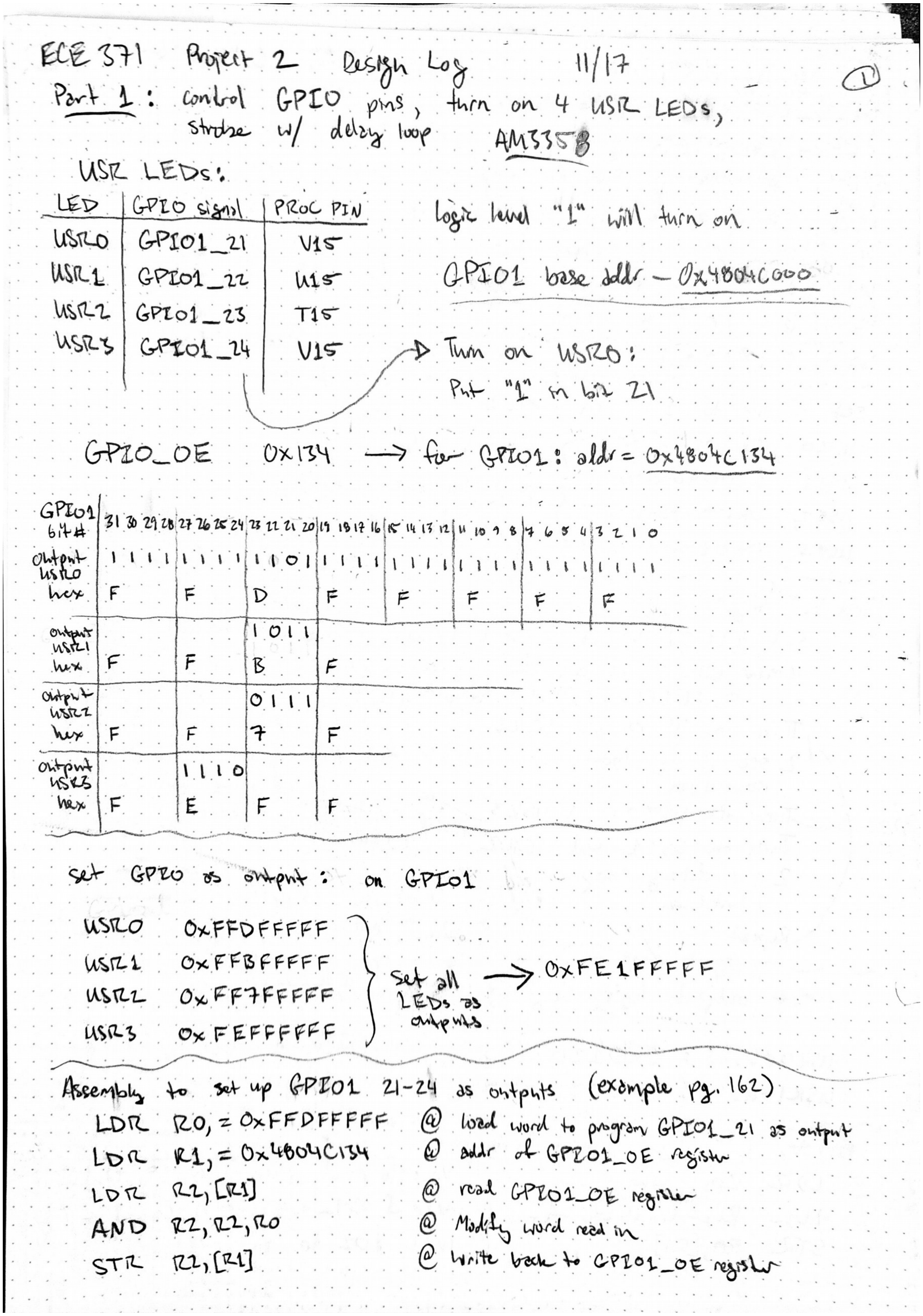
High-level algorithm:

```
Set GPIO outputs
Enable outputs
Repeat
     Turn on LEDs 0 and 3
     Repeat
          countdown for 2 seconds
     Turn off LEDs 0 and 3
     Turn on LEDs 1 and 2
     Repeat
          countdown for 2 seconds
until count=0, or forever
```

Low-level algorithm:

```
Base address for GPIO1 = 0x4804C000
Enable clock for GPIO1 (#0x02 to address 0x44E000AC)
Set GPIO1 bits 21-24 to low by writing 0x01E00000 to
     GPIO_CLEARDATAOUT at (base address + 0x190)
Set GPIO1 bits 21-24 as outputs by Read/modify/write
     value 0xFE1FFFFF to GPIO_OE at (base address + 0x134)
Repeat PULSE:
     Load value to set USR0 and USR3 high: 0x01200000
     Load GPIO1_SETDATAOUT: base address + 0x194
     Store value at address
     Load delay loop constant
     Repeat DELAY:
          countdown delay loop counter
          branch out when zero
     CLEAR: Load value to clear all LEDs
          value 0x01E00000 to base address + 0x190
     Load value to set USR1 and USR2 high: 0x00C00000
     Load GPIO1_SETDATAOUT base address + 0x194
     Store value at address
     Branch to DELAY and CLEAR
Until arbitrary time
```

Project Design Log:



ECE 371 Project 2 Design Log 11/17

1

Part 1: control GPIO pins, turn on 4 USR LEDs, strobe w/ delay loop AM3358

USR LEDs:

LED	GPIO signal	PROC PIN
USR0	GPIO1_21	V15
USR1	GPIO1_22	U15
USR2	GPIO1_23	T15
USR3	GPIO1_24	V15

logic level "1" will turn on

GPIO1 base addr – 0x4804C000

Turn on USR0:
Put "1" in bit 21

GPIO_OE 0x134 → for GPIO1: addr = 0x4804C134

GPIO1 bit #	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
output USR0	1 1 1 1	1 1 1 1	1 1 0 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1
hex	F	F	D	F	F	F	F	F
output USR1			1 0 1 1					
hex	F	F	B	F				
output USR2			0 1 1 1					
hex	F	F	7	F				
output USR3		1 1 1 0						
hex	F	E	F	F				

set GPIO as output: on GPIO1

USR0	0xFFDFFFFF	Set all LEDs as outputs → 0xFE1FFFFF
USR1	0xFFBFFFFF	
USR2	0xFF7FFFFF	
USR3	0xFEFFFFFF	

Assembly to set up GPIO1 21-24 as outputs (example pg. 162)

```
LDR R0, = 0xFFDFFFFF   @ load word to program GPIO1_21 as output
LDR R1, = 0x4804C134   @ addr of GPIO1_OE register
LDR R2, [R1]           @ read GPIO1_OE register
AND R2, R2, R0         @ Modify word read in
STR R2, [R1]           @ Write back to GPIO1_OE register
```

Setting up data out: GPIO1 base addr. = 0x4804C000 ②

GPIO_CLEARDATAOUT = 0x190

GPIO_SETDATAOUT = 0x194

11/21

GPIO1 #bit	31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
USR0	0000	0000	0010	0000	0000	0000	0000	0000
hex	0	0	2	0	0	0	0	0
USR1	0	0	4	0				
USR2	0	0	8	0				
USR3	0	1	0					

set output to 1 on GPIO1

USR0 0x00200000

USR1 0x00400000

USR2 0x00800000

USR3 0x01000000

} send to addr GPIO1

set data out: 0x4804C194

High level algorithm:

```
Set bits 21-24 on GPIO1 to low to clear   (set up)
RMW bits 21-24 set low to set as outputs   (enable)
    0xFE1FFFFF RMW to GPIO1_OE
Turn on USR0 and USR3
2 sec delay loop
  countdown
Turn off USR0 and USR3
Turn on USR1 and USR2
2 sec delay loop
  countdown
Repeat
```

Set output high in pairs:

USR0+USR3 : 0x01200000

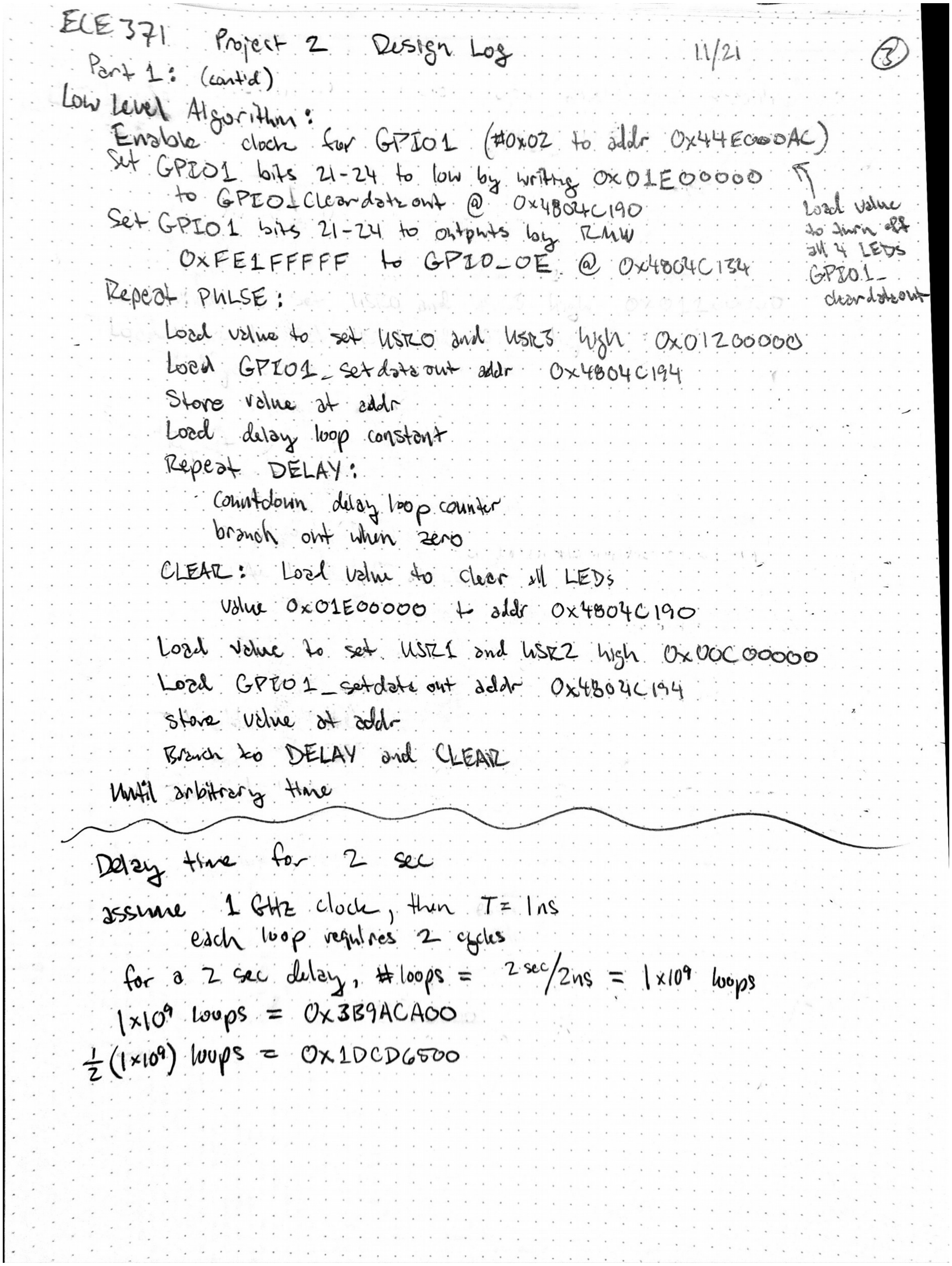
USR1+USR2 : 0x00C00000

All LEDs:

0x01E00000

Enable **GPIO1**; first thing to do

```
LDR  R0, #0x02            @ value to enable clock for GPIO module
LDR  R1, =0x44E000AC      @ addr of CM_PER_GPIO1_CLKCTRL reg
STR  R0, [R1]             @ write #02 to reg
```



ECE 371 Project 2 Design Log 11/21 ③

Part 1: (cont'd)

Low Level Algorithm:

```
  Enable clock for GPIO1 (#0x02 to addr 0x44E000AC)
  Set GPIO1 bits 21-24 to low by writing 0x01E00000
      to GPIO1_Cleardataout @ 0x4804C190
  Set GPIO1 bits 21-24 to outputs by RMW
      0xFE1FFFFF to GPIO_OE @ 0x4804C134
  Repeat PULSE:
      Load value to set USR0 and USR3 high 0x01200000
      Load GPIO1_setdataout addr 0x4804C194
      Store value at addr
      Load delay loop constant
      Repeat DELAY:
          Countdown delay loop counter
          branch out when zero
      CLEAR: Load value to clear all LEDs
          value 0x01E00000 to addr 0x4804C190
      Load value to set USR1 and USR2 high 0x00C00000
      Load GPIO1_setdata out addr 0x4804C194
      store value at addr
      Branch to DELAY and CLEAR
  Until arbitrary time
```

Load value to turn off all 4 LEDs GPIO1_cleardataout

Delay time for 2 sec

assume 1 GHz clock, then $T = 1\text{ns}$

each loop requires 2 cycles

for a 2 sec delay, # loops = $2\text{ sec}/2\text{ns} = 1\times10^9$ loops

1×10^9 loops = 0x3B9ACA00

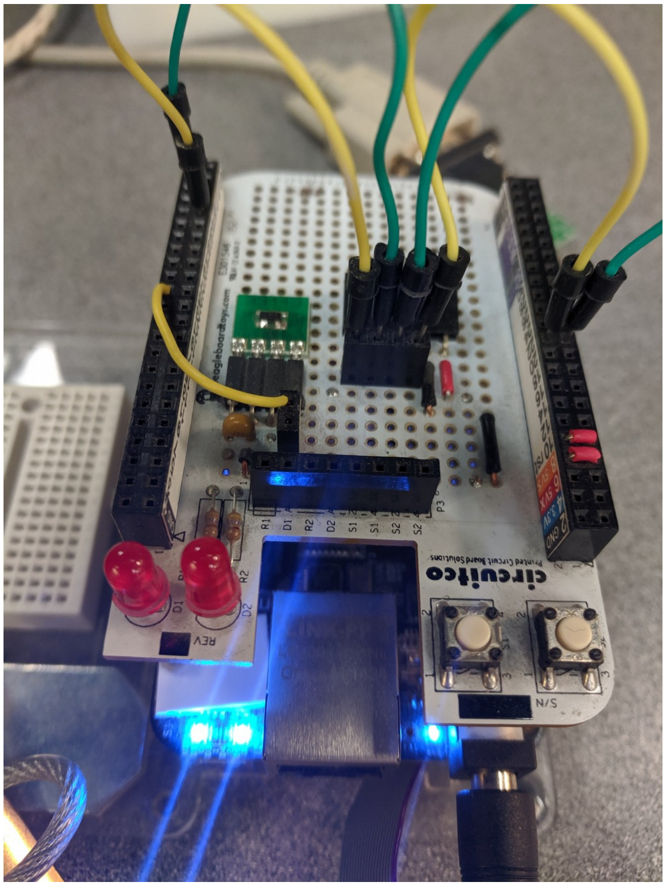
$\frac{1}{2}(1\times10^9)$ loops = 0x1DCD6500

The final delay value used in my code below, for a 2 second delay, comes from the Complete Button Service program in the book, page 237.

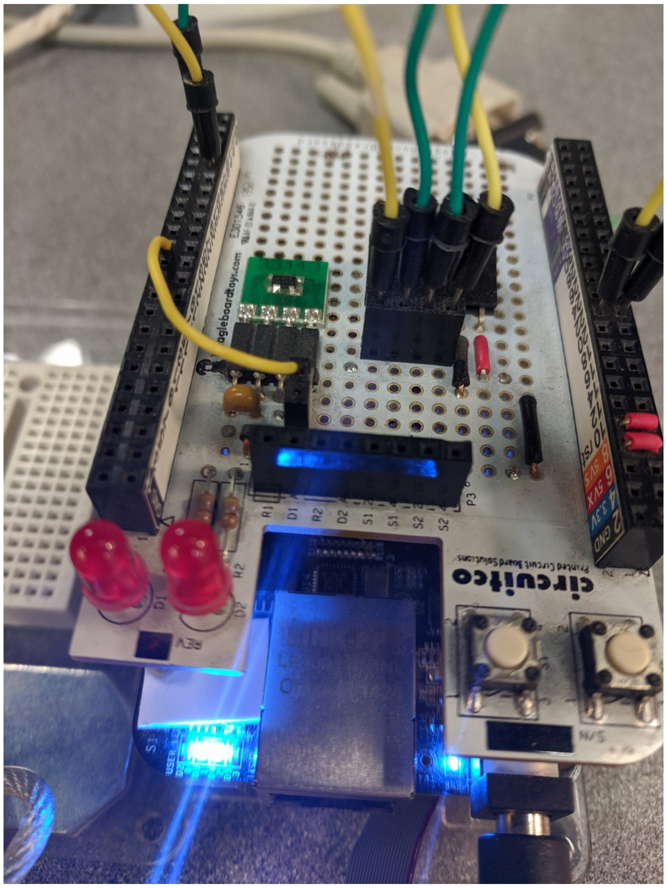
The complete program code for part 1 is below, and included as a text file. A few syntax errors and count values were changed during debugging to end up with the following program:

```
@ Part 1 of Project 2
@ ECE371
@ pulses 4 USR LEDs
@ Nick Porter Nov 22, 2019

.text
.global _start
_start:
.equ DEL, 0x00400000
@ enable clock for GPIO1
             MOV    R0,#0x02            @ value to enable clocks for a GPIO module
             LDR    R1,=0x44E000AC      @ addr of CM_PER_GPIO1_CLKCTRL register
             STR    R0,[R1]             @ write to register
             LDR    R0,=0x4804C000      @ base addr for GPIO registers
@ load value to turn off all 4 USR LEDs
             MOV    R7,#0x01E00000      @ GPIO 21-24
             ADD    R4,R0,#0x190        @ make GPIO_CLEARDATAOUT register addr
             STR    R7,[R4]             @ write to GPIO_CLEARDATAOUT register
@ set GPIO1 bits 24-21 as outputs
             ADD    R1,R0,#0x134        @ make GPIO1_OE register addr
             LDR    R6,[R1]             @ read current GPIO1_OE register
             MOV    R7,#0xFE1FFFFF      @ word to enable bits 21-24
             AND    R6,R7,R6            @ clear bits 21-24
             STR    R6,[R1]             @ write to GPIO1_OE register
             MOV    R5,#0x05            @ set loop counter
LOOP:
@ light LEDs 0 and 3
PULSE1:      MOV    R2,#0x01200000      @ load value to light USR0 and USR3
             ADD    R3,R0,#0x194        @ load addr of GPIO1_SETDATAOUT register
             STR    R2,[R3]             @ GPIO1_SETDATAOUT register to light LEDs 0 and 3
             LDR    R7,=DEL             @ load delay loop constant
@ delay loop
DELAY1:      SUBS   R7,R7,#1            @ decrement loop counter
             BNE          DELAY1

@ turn off all LEDs
CLEAR1:      MOV    R7,#0x01E00000      @ GPIO 21-24
             ADD    R4,R0,#0x190        @ make GPIO_CLEARDATAOUT register addr
             STR    R7,[R4]             @ write to GPIO_CLEARDATAOUT register

@ light LEDs 1 and 2
PULSE2:      MOV    R2,#0x00C00000      @ load value to light USR1 and USR2
             ADD    R3,R0,#0x194        @ load addr of GPIO1_SETDATAOUT register
             STR    R2,[R3]             @ GPIO1_SETDATAOUT register to light LEDs 1 and 2
             LDR    R7,=DEL             @ load delay loop constant
@ delay loop
DELAY2:      SUBS   R7,R7,#1            @ decrement loop counter
             BNE    DELAY2

@ turn off all LEDs
CLEAR2:      MOV    R7,#0x01E00000      @ GPIO 21-24
             ADD    R4,R0,#0x190        @ make GPIO_CLEARDATAOUT register addr
             STR    R7,[R4]             @ write to GPIO_CLEARDATAOUT register

             SUBS   R5,#1               @ decrement loop counter
             BNE    LOOP                @ pulse lights for number of loop counts
             NOP
.end
```

Pictures:



LEDs USR0 and USR3



LEDs USR1 and USR2

Part 2:
This part introduced interrupt procedures for using a button to toggle the LEDs blinking on and off. When the button is pressed, the two sets of LEDs begin blinking alternately, as in part 1 of the project. When the button is pressed again, they stop flashing.

For this part I followed the example from the textbook closely, substituting the appropriate GPIO and IRQ addresses and values.

High Level Algorithm:

```
Hook/chain interrupt procedure (startup file)
Set up stack for SVC and IRQ, point to top
Initialize procedures for GPIO1 and LED output, button interrupt
Enable IRQ (set checkvalue=0)
Idle loop, no pulse
Int Director procedure
     push used registers, LR
     check button status
     If not button
          return to idle loop, no pulse
     Else, if button
          turn off IRQ request
          If checkvalue=0,
               set checkvalue=1
               pulse LEDs loop, wait for button
          Else if checkvalue=1,
               set checkvalue=0
               turn off LEDs, wait for button
     Return to idle loop
```

Low-level procedure algorithms:

```
Mainline:
Turn on GPIO1 clock
     initialize GPIO1 bits 21-24 off (from part1)
          RMW as outputs
Initialize GPIO1_29 as interrupt source for falling edge
     1 to bit 29 of GPIO1_FALLINGEDGEDETECT register
          write 0x20000000 to addr 0x4804C14C
     1 to bit 29 of GPIO_IRQSTATUS_SET_0
          write 0x20000000 to addr 0x4804C034
INTC initialize
     unmask bit 2 of INTC_MIR_SET3
          write 0x04 to addr 0x482000E8
Enable IRQ input
     clear bit 7 of CPSR
Set global register to track LED status, initialize to 0
Wait loop…
```

```
INT_DIRECTOR:
push registers to stack
check source of IRQ
     verbatim example code pg. 237
          replace 0x00004000 with 0x20000000
If button push
     branch to button SVC
Else
     restore registers
     return addr of mainline wait loop

Button_SVC:
Turn off interrupt request
     bit 29 to GPIO_IRQSTATUS_0
          store 0x20000000 to addr 0x4804C02C
Turn off NEWIRQ bit in IRQ_CONTROL
If LEDstatus=0
     alternate LEDs on
     delay
IF LEDstatus=1
     turn off LEDs
     restore registers
     return to wait loop in mainline
```

Part 3:
The last part of this project replaced the delay loops from the previous part with another interrupt based on a timer overflow. This part was deceptively hard, and I was only able to get it working after developing a full understanding of the example program from the textbook, figure 5-18.

I wrote out some preliminary high-level algorithms for the procedures I would need, after reading over the relevant section of the textbook, which are included in my handwritten design log. These would be updated drastically as I built my program, so my final high-level algorithm is included later in this report.

The most difficult part about setting up the algorithms for this part was getting the needed addresses and values for initializing the timer and interrupts, especially for setting the clock on the timer. From appendix D, I got the bit to select timer 7 (bit 31), which is in the same module as timer2. I started by looking up all the base addresses that I would need:

Module:	Base Address:
GPIO1	0x4804C000
CM_PER	0x44E00000 (with offset 0x500 for CM_DPLL base)
INTC	0x48200000
timer7	0x4804A000

After finding these, I started writing out the low-level startup process that I would need, by following the textbook example exactly and looking up and replacing the addresses and values I would need for using a different timer and GPIO.

I also calculated the hex value I would need to load into the timer to give a 2 second delay interrupt. From the textbook, the Desired Time = (FFFF FFFFh – TLDR + 1) * (timer clock period). 0x00008000 pulses for 1 second means that 2 seconds = 0x00010000 pulses.

0x1 0000 0000 – 0x0001 0000 = 0xFFFF 0000

So I can write 0xFFFF 0000 to TCRR (0x4804A03C) and TLDR (0x4804A040).

Low-level mainline interrupt setup:
```
Enable timer7 INTC
     Interrupt associated with timer7: #95 - bit 31 to
     INTC_MIR_CLEAR2
          write 0x80000000 to 0x482000C8
Turn on clock to timer7
     bit 1 to enable at offset for CM_PER_TIMER7_CLKCTRL register
          write 0x02 to 0x44E0007C
Set clock frequency
     CM_DPLL base at 0x44E00500
     bit 1 to enable at offset 4 for 32.768kHz clock
          write 0x02 to 0x44E00504
Initialize timer
     reset timer7
          bit 0 to timer7 config register at offset 0x10
               write 0x1 to 0x4804A010
     enable timer
```

```
          bit 1 to timer7 IRQ_ENABLE_SET register
               write 0x2 to 0x4804A02C
Write count time to timer7 load and count registers
     write 0xFFFF0000 to 0x4804A03C and 0x4804A040
```

High-level algorithm:

```
MAINLINE:
Set up stacks for SVC and IRQ
Enable clock for GPIO1
Initialize LEDs and set as outputs
Set button as interrupt
Set timer7 overflow as interrupt
Turn on timer7 clock
Initialize timer7 with count and overflow values
Enable IRQ in CPSR
Set led status registers
     LEDstatus = 00 means leds are off, 01 means on
     pulsestatus = 01 means outside leds are on
     pulsestatus = 10 means inside leds are on
Wait loop

INT_DIRECTOR:
push registers to stack
Check if interrupt from GPIO1
IF not
     go to TCHK to check timer interrupt
ELSE
     check button for interrupt
     IF true
          go to BUTTON_SVC
     ELSE
          clear INTC control
          restore registers and return to wait loop

TCHK:
check timer7 for interrupt
     IF not
          clear INTC control
          restore registers and return to wait loop
     ELSE
          check timer7 overflow interrupt
          IF not
               restore registers and return to wait loop
          ELSE
               go to LED

BUTTON_SVC:
```

```
check LEDstatus
     IF LEDstatus=01
          turn off all LEDs
          set LEDstatus=00
          reset timer7
          clear INTC control
          return to wait loop
     ELSE
          turn LEDs on (pulsestatus=01 for outside leds)
          reset timer7
          clear INTC control
          return to wait loop
LED:
reset timer7 overflow request
IF pulsestatus=01
     turn off LEDs
     turn on inside LEDs
     set pulsestatus=10
ELSE
     turn off LEDs
     turn on outside LEDs
     set pulsestatus=01
reset INTC control
restore registers and return to wait loop
```

Final Program:

```
@ Part 3 of Project 2
@ ECE371
@ full LED button interrupt procedure with timers
@ Nick Porter Dec 13, 2019

.text
.global _start
.global INT_DIRECTOR
_start:
@ set up stacks
     LDR        R13,=STACK1           @ point to base of STACK for SVC mode
     ADD        R13,R13,#0x1000       @ point to top of stack
     CPS        #0x12                 @ switch to IRQ mode
     LDR        R13,=STACK2           @ point to IRQ stack
     ADD        R13,R13,#0x1000       @ point to top of stack
     CPS        #0x13                 @ back to SVC mode
@ enable clock for GPIO1
     MOV        R0,#0x02              @ value to enable clocks for a GPIO
module
     LDR        R1,=0x44E000AC        @ addr of CM_PER_GPIO1_CLKCTRL
register
     STR        R0,[R1]               @ write to register
     LDR        R0,=0x4804C000        @ base addr for GPIO registers
@ load value to turn off all 4 USR LEDs
     MOV        R7,#0x01E00000        @ GPIO 21-24
     ADD        R4,R0,#0x190          @ make GPIO_CLEARDATAOUT register
                                      @ addr
     STR        R7,[R4]               @ write to GPIO_CLEARDATAOUT register
@ set GPIO1 bits 24-21 as outputs
     ADD        R1,R0,#0x134          @ make GPIO1_OE register addr
     MOV        R7,#0xFE1FFFFF        @ word to enable bits 21-24
     STR        R7,[R1]               @ write to GPIO1_OE register
@ Detect falling edge on GPIO1_29 and enable to assert POINTRPEND1
     ADD        R1,R0,#0x14C          @ R1=addr of GPIO1_FALLINGDETECT
                                      @ register
     MOV        R2,#0x20000000        @ load value for bit 29
     STR        R2,[R1]               @ write back
     ADD        R1,R0,#0x34           @ addr of GPIO_IRQSTATUS_SET_0
                                      @ register
     STR        R2,[R1]               @ enable GIO1_29 request on
                                      @ POINTRPEND1
@ Init INTC
     LDR        R1,=0x48200000        @ base addr for INTC
     MOV        R2,#0x2               @ value to reset INTC
     STR        R2,[R1,#0x10]         @ write to INTC config register
     MOV        R2,#0x80000000        @ unmask INTC INT 95 timer7 interrupt
     STR        R2,[R1,#0xC8]         @ write to INTC_MIR_CLEAR2 register
     MOV        R2,#0x04              @ value to unmask INTC INT 98,
                                      @ GPIOINT1A
     STR        R2,[R1,#0xE8]         @ write to INTC_MIR_CLEAR3 register
@ Turn on timer7 CLK
```

```
     MOV        R2,#0x2               @ value to enable timer7 clk
     LDR        R1,=0x44E0007C        @ addr of CM_PER_TIMER7_CLKCTRL
     STR        R2,[R1]               @ turn on
     LDR        R1,=0x44E00504        @ addr of PRCMCLKSEL_timer7 register
     STR        R2,[R1]               @ select 32kHz clk for timer7
@ init timer 7 registers with count/overflow interrupt generation
     LDR        R1,=0x4804A000        @ base addr for timer7 registers
     MOV        R2,#0x1               @ value to reset timer7
     STR        R2,[R1,#0x10]         @ write to timer7 CFG register
     MOV        R2,#0x2               @ value to enable overflow interrupt
     STR        R2,[R1,#0x2C]         @ write to timer7 IRQENABLE_SET
     LDR        R2,=0xFFFF0000        @ count value for 2 seconds
     STR        R2,[R1,#0x40]         @ timer7 TLDR load register
     STR        R2,[R1,#0x3C]         @ timer7 TCRR count register
@ enable IRQ in CPSR
     MRS        R3,CPSR               @ copy CPSR to R3
     BIC        R3,#0x80              @ clear bit 7
     MSR        CPSR_c,R3             @ write back to CPSR
@ set ledstatus
     MOV        R9,#0x01              @ set blinkstatus 01 means blink on
     MOV        R10,#0x00             @ set ledstatus to off, 01 is leds on
     MOV        R11,#0x01             @ set pulsestatus for which leds on
                                      @ 01=pulse1 = outside leds
                                      @ 10=pulse2 = inside leds

@ idle when no exceptions:
WAITLOOP:
     NOP
     B          WAITLOOP

@ direct interrupts:
INT_DIRECTOR:
     STMFD SP!,{R0-R3,LR}             @ push registers on stack
     LDR        R1,=0x482000F8        @ addr of INTC-PENDING_IRQ3 register
     LDR        R2,[R1]               @ read INTC-PENDING_IRQ3 register
     TST        R2,#0x00000004        @ test bit 2
     BEQ        TCHK                  @ not from GPIOINT1A, check timer7,
                                      @ else
     LDR        R0,=0x4804C02C        @ load GPIO1_IRQSTATUS_0 register
                                      @ addr
     LDR        R1,[R0]               @ read status register to see if
                                      @ button
     TST        R1,#0x20000000        @ check if bit 29=1
     BNE        BUTTON_SVC            @ if bit 29=1, go to button pushed
     LDR        R0,=0x48200048        @ else, go back. INTC_CONTROL
                                      @ register
     MOV        R1,#01                @ value to clear bit 0
     STR        R1,[R0]               @ write to INTC_CONTROL register
     LDMFD SP!,{R0-R3,LR}             @ restore registers
     SUBS       PC,LR,#4              @ pass execution to wait loop for now
TCHK:
     LDR        R1,=0x482000D8        @ addr of INTC PENDING_IRQ2 register
     LDR        R0,[R1]               @ read value
     TST        R0,#0x80000000        @ check if interrupt from timer7
     BEQ        PASS_ON               @ No, return, yes, check overflow
     LDR        R1,=0x4804A028        @ addr timer7 IRQSTATUS register
     LDR        R0,[R1]               @ read value
     TST        R0,#0x2               @ check bit 1
     BNE        LED                   @ if overflow, go toggle led
PASS_ON:                              @ else go back to wait loop
     LDR        R0,=0x48200048        @ addr of INTC_CONTROL register
     MOV        R1,#01                @ value to clear bit 0
     STR        R1,[R0]               @ write to INTC_CONTROL register
     LDMFD      SP!,{R0-R3,LR}        @ restore registers
     SUBS       PC,LR,#4              @ pass execution to wait loop for now

@ if button is pushed...
BUTTON_SVC:
     MOV        R1,#0x20000000        @ value turns off GPIO1_29 interrupt
                                      @ request
                                      @ also turns off INTC interrupt
                                      @ request
     STR        R1,[R0]               @ write to GPIO_IRQSTATUS_0 register
     TST        R10,#0x01             @ test if ledstatus on
     BNE        LEDOFF                @ turn off leds if on
@ handle LED status
LEDON:
     LDR        R0,=0x4804C194        @ load addr of GPIO1_SETDATAOUT
                                      @ register
     MOV        R1,#0x01200000        @ load value to light USR0 and USR3
     STR        R1,[R0]               @ write to GPIO1_SETDATAOUT register
     MOV        R10,#0x01             @ set led status to on
     MOV        R11,#0x01             @ set pulsestatus to first set of
                                      @ leds
     MOV        R2,#0x03              @ load value to auto reload timer and
                                      @ start
     LDR        R1,=0x4804A038        @ addr of timer7 TCLR register
     STR        R2,[R1]               @ write to TCLR register
     B          RESETINT
LEDOFF:
     LDR        R0,=0x4804C000        @ load GPIO1 base addr
     MOV        R7,#0x01E00000        @ GPIO 21-24
     ADD        R4,R0,#0x190          @ make GPIO_CLEARDATAOUT register
                                      @ addr
     STR        R7,[R4]               @ write to GPIO_CLEARDATAOUT register
     MOV        R10,#0x00             @ set led status to off
     MOV        R2,#0x00              @ load value to reset timer
     LDR        R1,=0x4804A038        @ addr of timer7 TCLR register
     STR        R2,[R1]               @ write to TCLR register
@ turn off NEWIRQA bit in INTC_CONTROL, so processor can respond to IRQ
RESETINT:
     LDR        R0,=0x48200048        @ addr of INTC_CONTROL register
     MOV        R1,#01                @ value to clear bit 0
     STR        R1,[R0]               @ write to INTC_CONTROL register
     LDMFD      SP!,{R0-R3,LR}        @ restore registers
```

```
      SUBS      PC,LR,#4              @ pass execution to wait loop for now

@ switch between sets of leds on a timer
LED:
@ turn off timer7 interrupt request and enable INTC for next IRQ
      LDR       R1,=0x4804A028        @ load addr of timer7 IRQSTATUS
                                      @ register
      MOV       R2,#0x2               @ value to reset timer7 overflow IRQ
                                      @ request
      STR       R2,[R1]               @ write to register

@ toggle LED
      MOV       R3,#0x01E00000        @ load value for all leds
      LDR       R0,=0x4804C000        @ load base addr of GPIO1
      STR       R3,[R0,#0x190]        @ LED off, turn on with
GPIO1_SETDATAOUT
      TST       R11,#0x01             @ test pulsestatus
      BNE       PULSE2                @ if true, load values for pulse2
      BEQ       PULSE1                @ else load values for pulse1

PULSE1:
      MOV       R2,#0x01200000        @ load value to light USR0 and USR3
      ADD       R3,R0,#0x194          @ load addr of GPIO1_SETDATAOUT
                                      @ register
      MOV       R11,#0x01             @ set pulsestatus to pulse1
      B         BACK
PULSE2:
      MOV       R2,#0x00C00000        @ load value to light USR1 and USR2
      ADD       R3,R0,#0x194          @ load addr of GPIO1_SETDATAOUT
                                      @ register
      MOV       R11,#0x10             @ set pulsestatus to pulse2
BACK:
      STR       R2,[R3]               @ write to GPIO1_SETDATAOUT register
      LDR       R1,=0x48200048        @ addr of INTC_CONTROL register
      MOV       R2,#0x01              @ value to enable new IRQ response in
INTC
      STR       R2,[R1]               @ write
      LDMFD     SP!,{R0-R3,LR}        @ restore registers
      SUBS      PC,LR,#4              @ return from IRQ interrupt procedure

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

“I developed and wrote this program by myself with NO help from anyone except the instructor and/or the T.A. and I did not give any assistance to anyone else.”

– Nicholas Porter