Portland State University
Maseeh College of Engineering and Computer Science
Electrical and Computer Engineering Department
EE372 -Microprocessor Interfacing And Embedded Systems
Summer 2023

ARM Processor C Language Programming

BeagleBone Black USR LEDs Control Program

Complete Version

By signing this statement, I affirm that I did not give any help to any other person, did not receive any help from any other person, except TA and Instructor and did not obtain any information from the Internet or other sources. Signature: *WAIL .S .S AL KALBANI*

Designed by: **Wail (Wa'el) S. AL KALBANI**
Instructor: Tyler Hull
TA: Robert Wilcox
Date: July 6th, 2023

Table of Content

Introduction

This report presents the development of a C program for the BeagleBone Black microprocessor, which controls the lighting sequence of four LEDs. The program utilizes a provided button and resistor, with GPIO1_14 designated for button input. Figure 1 illustrates the circuit configuration for the project.

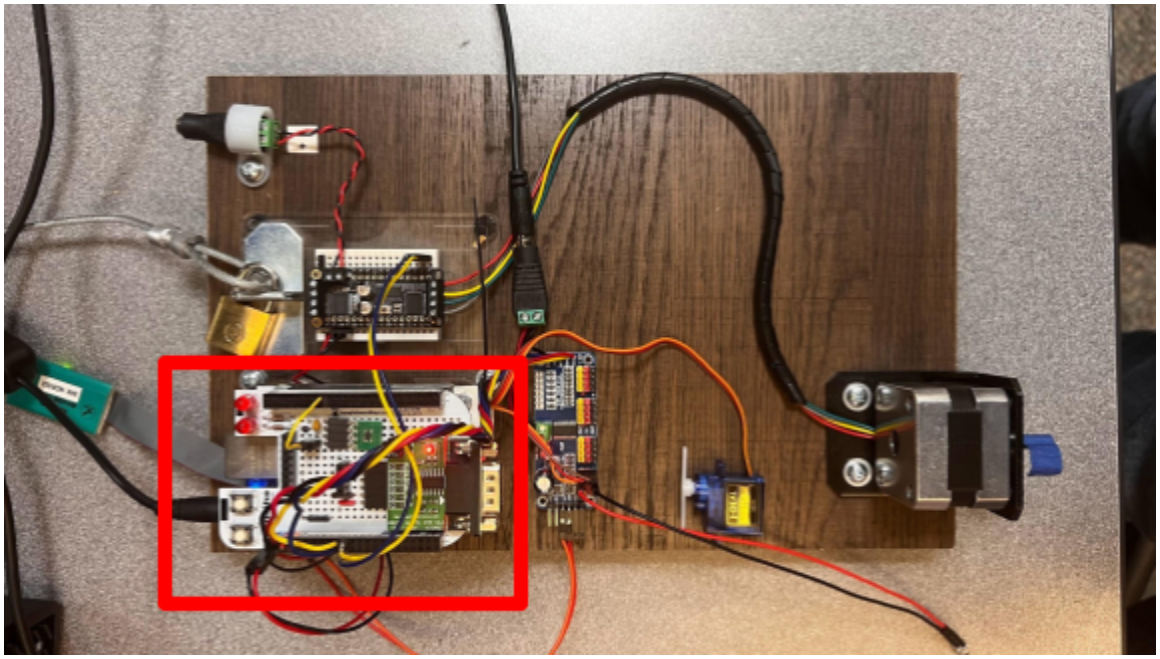


Figure 1: Circuit configuration for LED control with a button

The program follows a predefined pattern where LED0 and LED1 are illuminated for 750 milliseconds, followed by LED1 and LED2 for 750 milliseconds, and LED2 and LED3 for 750 milliseconds. This sequence repeats continuously unless interrupted by a button press. Implemented in the C programming language, the program configures GPIO pins, timers, and the interrupt controller, allowing interaction with the BeagleBone Black's hardware components. The report provides concise guidance on the programming concepts and techniques employed in C, including LED control, interrupt handling, precise timing using timers, and integration with external components. By the end of this report, readers will have a comprehensive understanding of developing C programs for embedded systems, specifically for LED control and interaction with buttons. Figure 1 serves as a visual reference for the circuit configuration, enhancing the comprehension of the project's hardware implementation.

A. Design Log

The table below provides a detailed breakdown design log of the tasks required to complete the project parts.

Task No.	Description	Completion Date
	Lighting LEDs	
1	Study the BeagleBone Black System Manual to determine which GPIO pins are connected to the 4 USR LEDs and the logic level required to turn on one of the LEDs. In the manual, looked under Hardware Files, Latest Production files for C Version, and System Reference Manual. Skimed through the User Manual until you find the GPIO connection for the 4 User LEDs and the logic level required to turn on an LED.	July 4th, 2023
2	Practiced the tutorial file, then Developed a high level algorithm for a program that lights LED0 and LED 1 for 750ms, then switches off LED 0 and Led 1 and lights LED 1 and LED 2 for750ms. Finally, it switches off LED 1 and LED 2 and turns on LED 2 and LED 3 for 750ms. Set this pattern to repeat.	July 5th, 2023
3	Carefully work through the section of Hall Chapter 4 that describes the AM3358 GPIO pins and the memory mapped registers that control them. As part of this, determine the registers that control the GPIO pins connected to the LEDs. Also study the text section that shows how to set up bit templates for working with the GPIO registers.	July 5th, 2023
4	Set up the templates needed for the GPIO pins you are using (GPIO21_24). and study table 4-11 for ARM Cortex A-8 GPIO Control register functions and offsets from base address: *#define GPIO1BA 0x4804C000*	July 5th, 2023

	#define GPIO_SET_DATA_OUT 0x194 *#define GPIO_CLEAR_DATA_OUT 0x190* *#define GPIO_FALLING_DETECT 0x14C* *#define GPIO_IRQSTATUS_0 0x2C* *#define GPIO_IRQSTATUS_SET_0 0x34* *#define GPIO_IRQSTATUS_SET_1 0x38* *#define GPIO_IRQWAKEN_1 0x48*	
5	Determined values and addresses you need to output a high or output a low on a GPIO pin.	July 5th, 2023
6	Determined the RMW sequence of instructions and addresses required to program the GPIO pins for the LEDs as outputs and initialize the LEDs in the off state.	July 5th, 2023
7	Developed the low level algorithm, as shown in chapter 4, for your program, including the delay loop	July 5th, 2023
8	Edited the tutorial C code according to algorithm	July 5th, 2023
9	Build, Load, Run, and Debug the program. Note that to single step through the program easily you can initially use a very small delay constant, so you don't have to single step forever to get from one LED to the next.	July 5th, 2023
10	When the program worked, an electronic copy was made that you will be used for developing the next part of the project	July 5th, 2023
Creating an Interrupt Procedure for Servicing a Button Push		
11	Read Hall Chapter 5 and button service program developed following the steps described in the text	July 5th, 2023
12	Initialization list developed with values needed for the project, including GPIO1_14 for the pushbutton	July 5th, 2023
13	Required steps of the assembly language program in Figure 5-14 modified according to the detailed initialization list	July 5th, 2023
14	The revised program tested using the CODE COMPOSER and the required screenshots taken	July 5th, 2023

15	Developed a report on the results of the testing and the details of the modifications made to the program Develop a report on the results of the testing and the details of the modifications made to the program	July 5th, 2023
16	Set a time with the instructor for review and feedback	July 5th, 2023
17	Write a test plan for the interrupt procedure, including test cases for various possible scenarios.	July 5th, 2023
18	Develop and debug the interrupt procedure using the initialization list and button service program.	July 5th, 2023
19	Perform testing of the interrupt procedure using the test plan and document the results.	July 5th, 2023
Adding a flag to keep track of button		
20	Review the Timer 2 program discussed in text chapter 5 to understand how to set up a timer to produce interrupts at desired time intervals.	July 5th, 2023
21	Study the timer section of Chapter 5 very carefully to learn how to add timer capability to your button program.	July 5th, 2023
22	Edited the int_hander function so it will determine if a push button us pressed then call button_svc function: *void int_handler(void)* *{* *if (HWREG(0x482000D8) == 0x20000000) {* *timer5_int();* *}* *else if (HWREG(GPIO1BA +* *GPIO_IRQSTATUS_0) == 0x4000) {* *button_svc();* *}* *asm volatile(* *"LDMFD SP!, {LR}\n\t"* *"LDMFD SP!, {LR}\n\t"* *"SUBS PC, LR, #0x4"* *);*	July 5th, 2023
23	Create an interrupt service routine for Timer 4 that will	July 5th, 2023

	switch between the 4 LEDs.	
27	Write a summary report of the project, including a description of the interrupt procedure and its function, test results, and any issues encountered.	July 5th, 2023

B. Algorithms

In order to complete each part of the project outlined above, we will be following a set of algorithms to ensure the efficient and accurate development of the program. The algorithms will be as follows:

High-level Algorithm

1. Initialize the necessary variables, stacks, and registers for program execution.
2. Set up the GPIO pins for LED control and button input.
3. Configure the falling edge detection for the button.
4. Initialize and configure Timer 5 for precise timing.
5. Set up the Interrupt Controller (INTC) and enable interrupts.
6. Set the initial state and timing for the LED sequence.
7. Enter a wait loop, waiting for interrupts to occur.
8. If the interrupt is from Timer 5, execute the timer5_int function.
9. Toggle the LED state and control the corresponding LEDs.
10. Update the LED sequence state and set the timer based on the LED state.
11. If the interrupt is from the button, execute the button_svc function.
12. Toggle the button state and handle the LED sequence and timer accordingly.
13. Implement the turn_off_leds function to turn off all LEDs.
14. Enable interrupts using the IntMasterIRQEnable function.
15. Return to the wait loop and repeat the LED sequence until interrupted.

Low-level Algorithm

1. Initialization:
    a. Set up the User (USR) stack by loading the base address of the USR_STACK variable into register R13.
    b. Add the stack size (0x100) to R13 to allocate space for the stack.
    c. Set the Processor State (CPS) to System mode (0x12) to access the User mode stack.

   d. Set up the Interrupt (IRQ) stack by loading the base address of the INT_STACK variable into register R13.
   e. Add the stack size (0x100) to R13 to allocate space for the stack.
   f. Set the Processor State (CPS) to IRQ mode (0x13) to access the IRQ mode stack.
2. GPIO Initialization:
   a. Configure the CLKWKUPS register to initialize GPIO1.
   b. Set the GPIO_CLEAR_DATA_OUT register to turn off all LEDs initially.
   c. Configure the GPIO output enable register to enable output for the LEDs.
   d. Button Configuration:
   e. Set the GPIO_FALLING_DETECT register to enable falling edge detection for the button.
   f. Set the GPIO_IRQSTATUS_SET_0 register to clear any pending interrupts.
   g. Enable the specific GPIO interrupt for the button by setting the corresponding register bits.
3. Timer 5 Initialization:
   a. Set the CLKWKUPS register to enable Timer 5.
   b. Set the CLKWKUPS register to set the desired clock speed for Timer 5.
   c. Perform a software reset by setting the software reset bit in the TIMER5_BA register.
   d. Clear any pending interrupts by setting the interrupt status register (TIMER5_BA + 0x28) to 0x7.
   e. Enable the overflow interrupt by setting the overflow IRQ bit in the TIMER5_BA register.
4. INTC Initialization:
   a. Reset the INTC by setting the reset bit in the INTCBA register.
   b. Unmask the desired interrupt source (INTC_TINT5) by setting the corresponding register bits.
   c. Clear any pending interrupts in the INTC by setting the appropriate register bits.
   d. Interrupt Enable:
   e. Call the IntMasterIRQEnable function to enable interrupts.
   f. Retrieve the CPSR register value, clear the IRQ disable bit (0x80), and store the modified value back to CPSR.
5. LED Sequence Initialization:
   a. Set the timer duration for the LED sequence by configuring the TIMER5_BA + 0x3C register to the desired value.
   b. Start the timer by setting the start bit in the TIMER5_BA + 0x38 register.
6. Wait Loop:
   a. Enter an infinite loop (wait_loop) to wait for interrupts.
   b. Interrupt Handler (int_handler):

   c. Check if the interrupt is from Timer 5 by comparing the value in register 0x482000D8 to the Timer 5 interrupt value.
   d. If it is a Timer 5 interrupt, execute the timer5_int function.
   e. Otherwise, check if the interrupt is from the button by comparing the value in GPIO1BA + GPIO_IRQSTATUS_0 to the button interrupt value.
   f. If it is a button interrupt, execute the button_svc function.
7. Timer 5 Interrupt Handler (timer5_int):
   a. Clear the Timer 5 interrupts by setting the appropriate bits in the TIMER5_BA + 0x28 register.
   b. Clear the NEWIRQ bit in the INTC by setting the corresponding bit in the INTCBA + 0x48 register.
   c. Toggle the LED state by negating the led_state variable.
   d. If the LED state is true:
   e. Check the current_state variable and turn on the corresponding LEDs using the appropriate GPIO_SET_DATA_OUT value.
   f. If the LED state is false, turn off all LEDs using the turn_off_leds function.
   g. Update the current_state variable by incrementing it and resetting it to 1 if it exceeds 3.
   h. Set the timer duration based on the LED state by configuring the TIMER5_BA + 0x3C register.
   i. Start the timer by setting the start bit in the TIMER5_BA + 0x38 register.
8. Button Interrupt Handler (button_svc):
   a. Clear the button interrupt status by setting the GPIO1BA + GPIO_IRQSTATUS_0 register to the button interrupt value.
   b. Clear the IRQWAKEN bit in the INTC by setting the corresponding bit in the INTCBA + GPIO_IRQWAKEN_1 register.
   c. Toggle the button state by inverting the button_state variable.
   d. If the button state is true, set the appropriate bit in the TIMER5_BA + GPIO_IRQSTATUS_SET_1 register.
   e. If the button state is false, turn off all LEDs using the turn_off_leds function and clear the corresponding bit in the TIMER5_BA + GPIO_IRQSTATUS_SET_1 register.
9. LED Turn Off Function (turn_off_leds):
   a. Clear the GPIO pins for all LEDs by setting the GPIO1BA + GPIO_CLEAR_DATA_OUT register to the LIGHT_BITS value.
   b. Interrupt Master Enable Function (IntMasterIRQEnable):
   c. Retrieve the CPSR register value.
   d. Clear the IRQ disable bit by bitwise ANDing the register value with the inverse of the IRQ disable bit mask (0x80).
   e. Store the modified CPSR value back to the CPSR register.
10. Return to Wait Loop:

a. Return to the wait_loop function and repeat the LED sequence until interrupted.

C. Discussion

As shown in the figures below, the program C code was initializes and configures the microcontroller's GPIO (General Purpose Input/Output) and timer registers. The code uses the timer to create a cycle of lighting up four LEDs in sequence. LED0 and LED 1 for 750ms, then switches off LED 0 and Led 1 and lights LED 1 and LED 2 for750ms. Finally, it switches off LED 1 and LED 2 and turns on LED 2 and LED 3 for 750ms. Set. The cycle repeats continuously. The code initializes the GPIO registers to output the signal to the LED pins. It also detects the falling edge on GPIO1_14 to enable interrupts. It then sets up and initializes the interrupt control module to enable IRQ interrupts. The timer is configured to generate periodic interrupts. The loop at the end of the code waits for an interrupt to occur. When the timer generates an interrupt, the code executes an interrupt service routine (ISR) that lights up the appropriate LED and updates the cycle. The ISR then returns to the main code, which continues waiting for the next interrupt.

```
void int_handler(void)
{
   if (HWREG(0x482000D8) == 0x20000000) {
      timer5_int();
   }
   else if (HWREG(GPIO1BA + GPIO_IRQSTATUS_0) == 0x4000) {
      button_svc();
   }

   asm volatile(
      "LDMFD SP!, {LR}\n\t"
      "LDMFD SP!, {LR}\n\t"
      "SUBS PC, LR, #0x4"
   );
}

void timer5_int(void)
{
   HWREG(TIMER5_BA + 0x28) = 0x7;   // Clear timer5 interrupts
   HWREG(INTCBA + 0x48) = 0x1;      // Clear NEWIRQ bit in INTC
```

```
    // Toggle the LED state
    led_state = !led_state;

    if (led_state) {
        // Turn on the corresponding LEDs based on the LED state
        if (current_state == 1) {
            HWREG(GPIO1BA + GPIO_SET_DATA_OUT) = LIGHT_LED0_LED1;
//turn on led0 and led1
        } else if (current_state == 2) {
            HWREG(GPIO1BA + GPIO_SET_DATA_OUT) = LIGHT_LED1_LED2;
//turn on led1 and led2
        } else if (current_state == 3) {
            HWREG(GPIO1BA + GPIO_SET_DATA_OUT) = LIGHT_LED2_LED3;
//turn on led2 and led3
        }
    } else {
        // Turn off all LEDs
        turn_off_leds();
    }

    // Update the current state for the next iteration
    current_state++;
    if (current_state > 3) {
        current_state = 1;
    }

    // Set the timer based on the LED state
    if (led_state) {
        HWREG(TIMER5_BA + 0x3C) = 0xFFFFA002;   // Set timer for 750 ms
    } else {
        HWREG(TIMER5_BA + 0x3C) = 0xFFFFE000;   // Set timer for 250 ms
    }

    HWREG(TIMER5_BA + 0x38) = 0x1;    // Start timer
}

void button_svc(void)
{
    HWREG(GPIO1BA + GPIO_IRQSTATUS_0) = 0x4000;
    HWREG(INTCBA + GPIO_IRQWAKEN_1) = 0x1;

    button_state = 1 - button_state; // Toggle the button state

    if (button_state) {
        HWREG(TIMER5_BA + GPIO_IRQSTATUS_SET_1) = 0x1;
    }
    else {
        turn_off_leds();
        HWREG(TIMER5_BA + GPIO_IRQSTATUS_SET_1) = 0x0;
    }
}
```

D. Assembly Code Printouts

See next page

main.c

```
 1
 2
 3
 4
 5
 6
 7 //Defines Section
 8 #define HWREG(x) (*((volatile unsigned int *)(x)))
 9
10 //GPIO defines
11 #define GPIO1BA 0x4804C000
12 #define GPIO_SET_DATA_OUT 0x194
13 #define GPIO_CLEAR_DATA_OUT 0x190
14 #define GPIO_FALLING_DETECT 0x14C
15 #define GPIO_IRQSTATUS_0 0x2C
16 #define GPIO_IRQSTATUS_SET_0 0x34
17 #define GPIO_IRQSTATUS_SET_1 0x38
18 #define GPIO_IRQWAKEN_1 0x48
19
20
21 //INTC defines
22 #define INTCBA 0x48200000
23 #define INTC_MIR_CLEAR3 0xE8
24
25 //Timer 5 defines
26 #define TIMER5_BA 0x48046000
27
28 //other defines
29 #define CLKWKUPS 0x44E00000
30 #define LIGHT_BITS 0x01E00000
31 #define LIGHT_LED0_LED1 0x00600000
32 #define LIGHT_LED1_LED2 0x00C00000
33 #define LIGHT_LED2_LED3 0x01800000
34
35
36 //Function Declarations
37
38 void IntMasterIRQEnable();
39 void int_handler();
40 void turn_off_leds();
41 void timer5_int();
42 void wait_loop();
43 void return_from_int();
44 void button_svc();
45
46 //global variables
47 int led_state = 0;
48 int current_state = 1;
49 int button_state =1;
50 int x;
51 volatile unsigned int USR_STACK[100];
52 volatile unsigned int INT_STACK[100];
53
54 int main(void) {
55     //**** SET UP STACKS ****
56     // asm command allows for inline assembly.
57
```

Page 1

main.c

```
58      //**** init USR stack ****
59      asm("LDR R13, =USR_STACK");         // Set register 13 = to USR_STACK variables
60      asm("ADD R13, R13, #0x100");        // Put 0x100 into R13 for our stack
61      //**** init IRQ stack ****
62      asm("CPS #0x12");
63      asm("LDR R13, =INT_STACK");         // Set register 13 = to USR_STACK variables
64      asm("ADD R13, R13, #0x100");      // Put 0x100 into R13 for our stack
65      asm("CPS #0x13");                               // Write 0x13 to CPS
66
67      //**** LED INIT ****
68      HWREG(CLKWKUPS + 0xAC) = 0x2;               //GPIO1 initialization code
69      HWREG(GPIO1BA + GPIO_CLEAR_DATA_OUT) = LIGHT_BITS;  //Set initial GPIO values
70      HWREG(GPIO1BA + 0x134) &= 0xFE1FFFFF;                               // set output enable
71
72      //**** FALLING DETECT ***//
73      HWREG(GPIO1BA + GPIO_FALLING_DETECT) |= 0x4000;
74      HWREG(GPIO1BA + GPIO_IRQSTATUS_SET_0) = 0x4000;
75
76      //**** TIMER 5 INIT ****
77      HWREG(CLKWKUPS + 0xEC) = 0x2;                   //wakeup timer 5
78      HWREG(CLKWKUPS + 0x518) = 0x2;              //set clock speed
79      HWREG(TIMER5_BA + 0x10) = 0x1;              //software reset
80      HWREG(TIMER5_BA + 0x28) = 0x7;              //clear irqs
81      HWREG(TIMER5_BA + 0x2C) = 0x2;              //enable overflow IRQ
82
83      //**** INTC INIT ****
84      HWREG(INTCBA + 0x10) = 0x2;                     //reset INTC
85      HWREG(INTCBA + 0xC8) = 0x20000000;      //unmast INTC_TINT5
86      HWREG(INTCBA + INTC_MIR_CLEAR3) = 0x04;
87
88      //**** ENABLE IRQ ****
89      IntMasterIRQEnable();
90
91      //**** INIT INTERNAL STATE ****
92      HWREG(TIMER5_BA + 0x3C) = 0xFFFFF000;   //set timer for 250 ms
93      HWREG(TIMER5_BA + 0x38) = 0x1;              //start timer
94
95      wait_loop();
96      return 0;
97  }
98
99  //-------------------------------------------------------------------------------
100 // FUNCTIONS - Below are the funcitons used in this program
101 //-------------------------------------------------------------------------------
102 void wait_loop(void)
103 // The wait_loop function is a while loop with nothing in it.
104 // It is used for the system to wait for an interrupt to take action on.
105 {
106     while(1)
107     {
108         //do nothing loop
109     }
110 }
111
112 void int_handler(void)
113 {
114     if (HWREG(0x482000D8) == 0x20000000) {
```

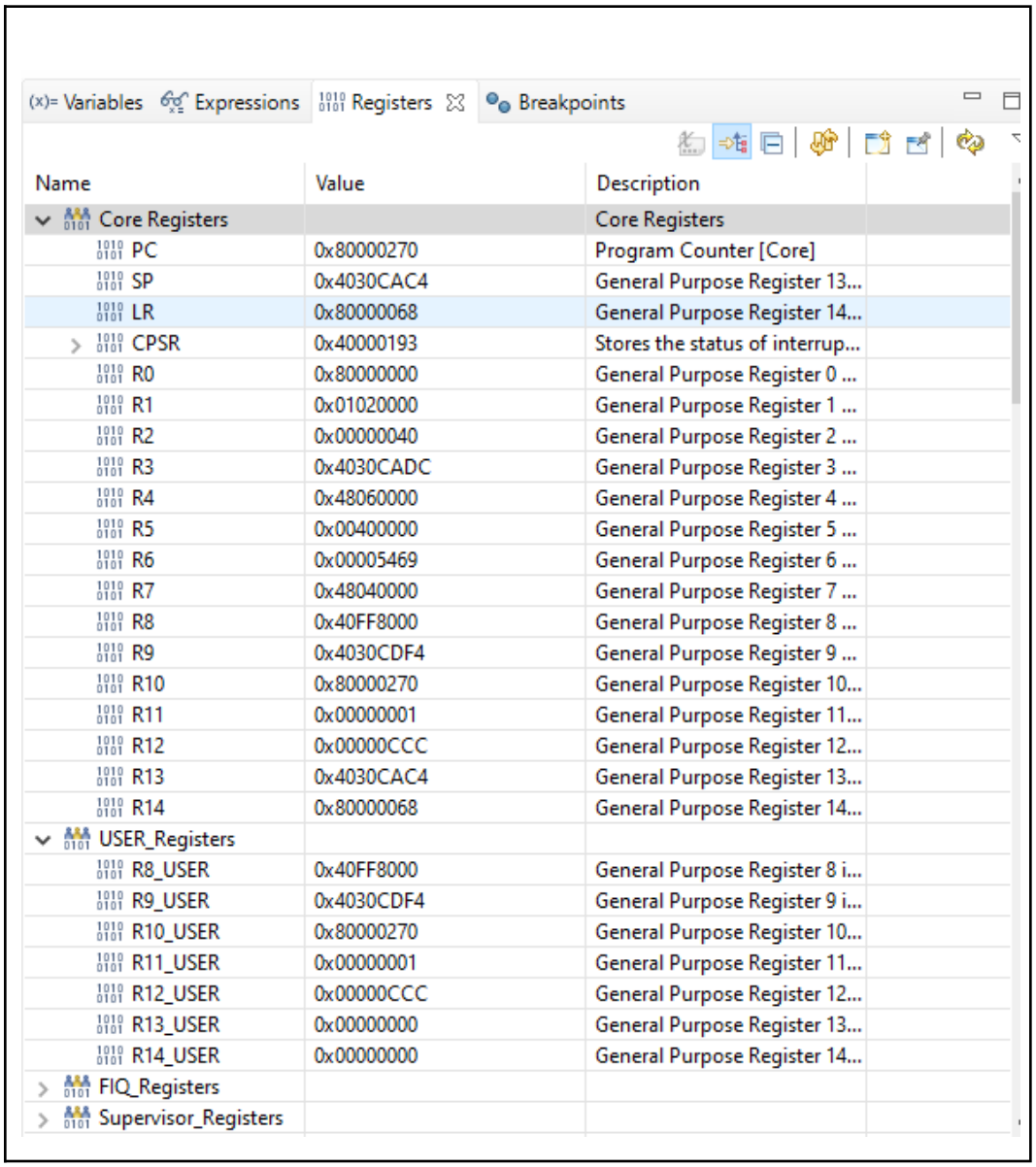
main.c

```
115         timer5_int();
116     }
117     else if (HWREG(GPIO1BA + GPIO_IRQSTATUS_0) == 0x4000) {
118         button_svc();
119     }
120 
121     asm volatile(
122         "LDMFD SP!, {LR}\n\t"
123         "LDMFD SP!, {LR}\n\t"
124         "SUBS PC, LR, #0x4"
125     );
126 }
127 
128 void timer5_int(void)
129 {
130     HWREG(TIMER5_BA + 0x28) = 0x7;   // Clear timer5 interrupts
131     HWREG(INTCBA + 0x48) = 0x1;      // Clear NEWIRQ bit in INTC
132 
133     // Toggle the LED state
134     led_state = !led_state;
135 
136     if (led_state) {
137         // Turn on the corresponding LEDs based on the LED state
138         if (current_state == 1) {
139             HWREG(GPIO1BA + GPIO_SET_DATA_OUT) = LIGHT_LED0_LED1;    //turn on led0 and led1
140         } else if (current_state == 2) {
141             HWREG(GPIO1BA + GPIO_SET_DATA_OUT) = LIGHT_LED1_LED2;    //turn on led1 and led2
142         } else if (current_state == 3) {
143             HWREG(GPIO1BA + GPIO_SET_DATA_OUT) = LIGHT_LED2_LED3;    //turn on led2 and led3
144         }
145     } else {
146         // Turn off all LEDs
147         turn_off_leds();
148     }
149 
150     // Update the current state for the next iteration
151     current_state++;
152     if (current_state > 3) {
153         current_state = 1;
154     }
155 
156     // Set the timer based on the LED state
157     if (led_state) {
158         HWREG(TIMER5_BA + 0x3C) = 0xFFFFA002;   // Set timer for 750 ms
159     } else {
160         HWREG(TIMER5_BA + 0x3C) = 0xFFFFE000;   // Set timer for 250 ms
161     }
162 
163     HWREG(TIMER5_BA + 0x38) = 0x1;    // Start timer
164 }
165 
166 void button_svc(void)
167 {
168     HWREG(GPIO1BA + GPIO_IRQSTATUS_0) = 0x4000;
169     HWREG(INTCBA + GPIO_IRQWAKEN_1) = 0x1;
170 
171     button_state = 1 - button_state; // Toggle the button state
```

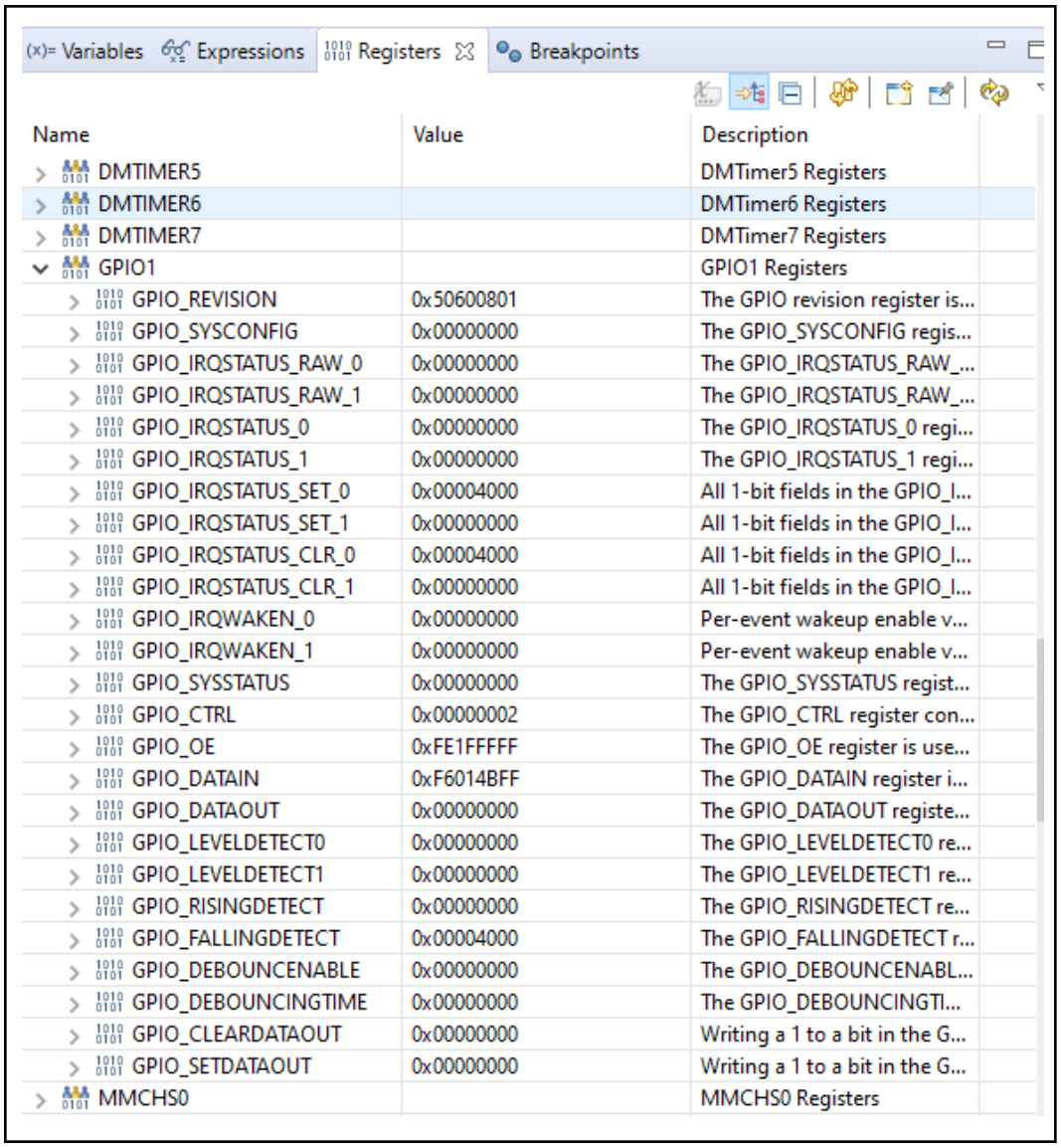
main.c

```
172
173     if (button_state) {
174         HWREG(TIMER5_BA + GPIO_IRQSTATUS_SET_1) = 0x1;
175     }
176     else {
177         turn_off_leds();
178         HWREG(TIMER5_BA + GPIO_IRQSTATUS_SET_1) = 0x0;
179     }
180 }
181
182
183 void turn_off_leds(void)
184 {
185     HWREG(GPIO1BA + GPIO_CLEAR_DATA_OUT) = LIGHT_BITS;  //turn off leds
186
187 }
188
189
190 void IntMasterIRQEnable(void)
191 {
192     asm("   mrs     r0, CPSR\n\t"
193         "   bic     r0, r0, #0x80\n\t"
194         "   msr     CPSR_c, R0");
195 }
196
```

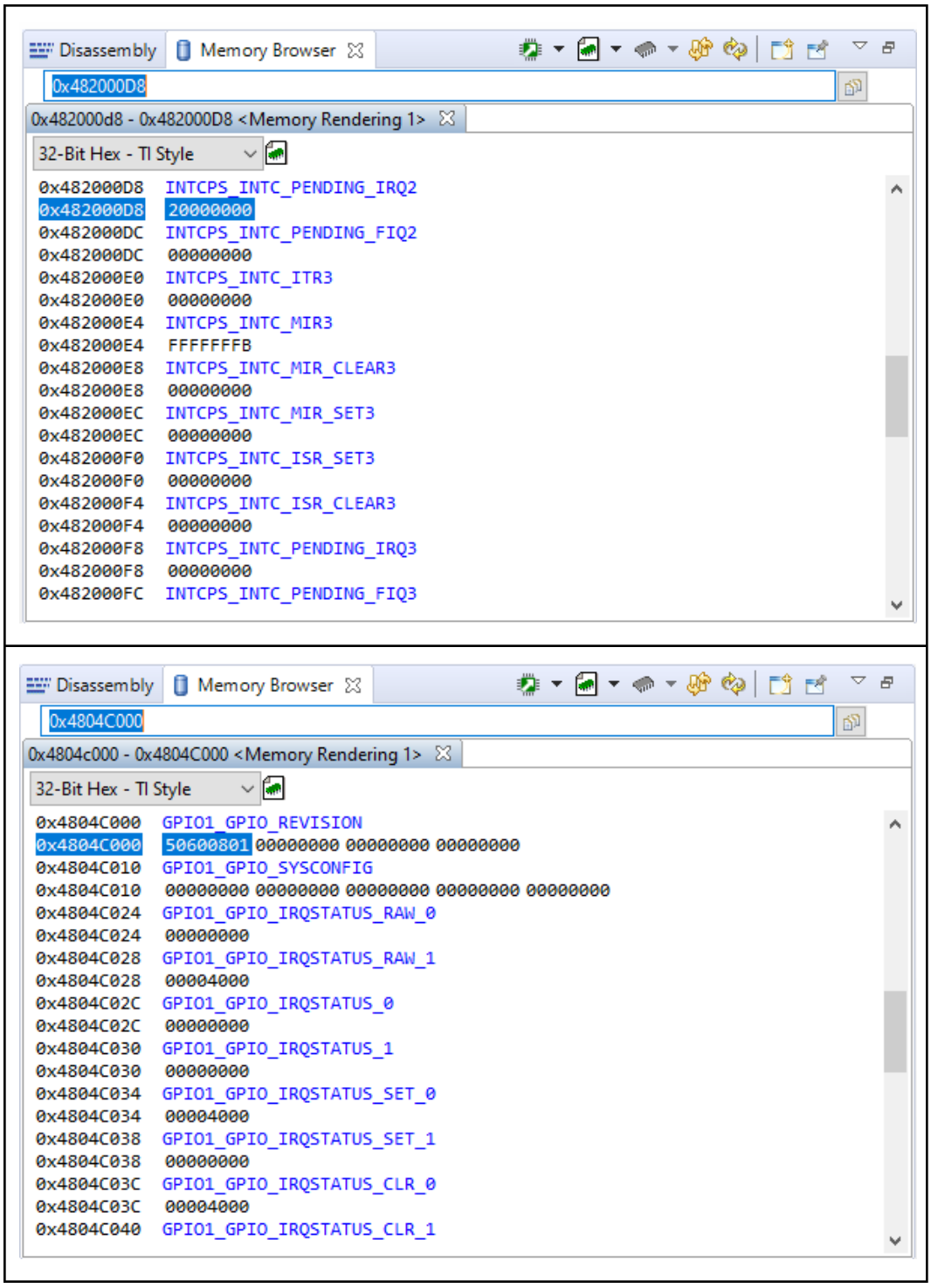
E. Debugger Memory and Registers



Name	Value	Description
Core Registers		Core Registers
PC	0x80000270	Program Counter [Core]
SP	0x4030CAC4	General Purpose Register 13...
LR	0x80000068	General Purpose Register 14...
CPSR	0x40000193	Stores the status of interrup...
R0	0x80000000	General Purpose Register 0 ...
R1	0x01020000	General Purpose Register 1 ...
R2	0x00000040	General Purpose Register 2 ...
R3	0x4030CADC	General Purpose Register 3 ...
R4	0x48060000	General Purpose Register 4 ...
R5	0x00400000	General Purpose Register 5 ...
R6	0x00005469	General Purpose Register 6 ...
R7	0x48040000	General Purpose Register 7 ...
R8	0x40FF8000	General Purpose Register 8 ...
R9	0x4030CDF4	General Purpose Register 9 ...
R10	0x80000270	General Purpose Register 10...
R11	0x00000001	General Purpose Register 11...
R12	0x00000CCC	General Purpose Register 12...
R13	0x4030CAC4	General Purpose Register 13...
R14	0x80000068	General Purpose Register 14...
USER_Registers		
R8_USER	0x40FF8000	General Purpose Register 8 i...
R9_USER	0x4030CDF4	General Purpose Register 9 i...
R10_USER	0x80000270	General Purpose Register 10...
R11_USER	0x00000001	General Purpose Register 11...
R12_USER	0x00000CCC	General Purpose Register 12...
R13_USER	0x00000000	General Purpose Register 13...
R14_USER	0x00000000	General Purpose Register 14...
FIQ_Registers		
Supervisor_Registers		



Name	Value	Description
DMTIMER5		DMTimer5 Registers
DMTIMER6		DMTimer6 Registers
DMTIMER7		DMTimer7 Registers
GPIO1		GPIO1 Registers
GPIO_REVISION	0x50600801	The GPIO revision register is...
GPIO_SYSCONFIG	0x00000000	The GPIO_SYSCONFIG regis...
GPIO_IRQSTATUS_RAW_0	0x00000000	The GPIO_IRQSTATUS_RAW_...
GPIO_IRQSTATUS_RAW_1	0x00000000	The GPIO_IRQSTATUS_RAW_...
GPIO_IRQSTATUS_0	0x00000000	The GPIO_IRQSTATUS_0 regi...
GPIO_IRQSTATUS_1	0x00000000	The GPIO_IRQSTATUS_1 regi...
GPIO_IRQSTATUS_SET_0	0x00004000	All 1-bit fields in the GPIO_I...
GPIO_IRQSTATUS_SET_1	0x00000000	All 1-bit fields in the GPIO_I...
GPIO_IRQSTATUS_CLR_0	0x00004000	All 1-bit fields in the GPIO_I...
GPIO_IRQSTATUS_CLR_1	0x00000000	All 1-bit fields in the GPIO_I...
GPIO_IRQWAKEN_0	0x00000000	Per-event wakeup enable v...
GPIO_IRQWAKEN_1	0x00000000	Per-event wakeup enable v...
GPIO_SYSSTATUS	0x00000000	The GPIO_SYSSTATUS regist...
GPIO_CTRL	0x00000002	The GPIO_CTRL register con...
GPIO_OE	0xFE1FFFFF	The GPIO_OE register is use...
GPIO_DATAIN	0xF6014BFF	The GPIO_DATAIN register i...
GPIO_DATAOUT	0x00000000	The GPIO_DATAOUT registe...
GPIO_LEVELDETECT0	0x00000000	The GPIO_LEVELDETECT0 re...
GPIO_LEVELDETECT1	0x00000000	The GPIO_LEVELDETECT1 re...
GPIO_RISINGDETECT	0x00000000	The GPIO_RISINGDETECT re...
GPIO_FALLINGDETECT	0x00004000	The GPIO_FALLINGDETECT r...
GPIO_DEBOUNCENABLE	0x00000000	The GPIO_DEBOUNCENABL...
GPIO_DEBOUNCINGTIME	0x00000000	The GPIO_DEBOUNCINGTI...
GPIO_CLEARDATAOUT	0x00000000	Writing a 1 to a bit in the G...
GPIO_SETDATAOUT	0x00000000	Writing a 1 to a bit in the G...
MMCHS0		MMCHS0 Registers


Disassembly
Memory Browser
0x482000D8
0x482000d8 - 0x482000D8 <Memory Rendering 1>
32-Bit Hex - TI Style
0x482000D8 INTCPS_INTC_PENDING_IRQ2
0x482000D8 20000000
0x482000DC INTCPS_INTC_PENDING_FIQ2
0x482000DC 00000000
0x482000E0 INTCPS_INTC_ITR3
0x482000E0 00000000
0x482000E4 INTCPS_INTC_MIR3
0x482000E4 FFFFFFFB
0x482000E8 INTCPS_INTC_MIR_CLEAR3
0x482000E8 00000000
0x482000EC INTCPS_INTC_MIR_SET3
0x482000EC 00000000
0x482000F0 INTCPS_INTC_ISR_SET3
0x482000F0 00000000
0x482000F4 INTCPS_INTC_ISR_CLEAR3
0x482000F4 00000000
0x482000F8 INTCPS_INTC_PENDING_IRQ3
0x482000F8 00000000
0x482000FC INTCPS_INTC_PENDING_FIQ3
Disassembly
Memory Browser
0x4804C000
0x4804c000 - 0x4804C000 <Memory Rendering 1>
32-Bit Hex - TI Style
0x4804C000 GPIO1_GPIO_REVISION
0x4804C000 50600801 00000000 00000000 00000000
0x4804C010 GPIO1_GPIO_SYSCONFIG
0x4804C010 00000000 00000000 00000000 00000000 00000000
0x4804C024 GPIO1_GPIO_IRQSTATUS_RAW_0
0x4804C024 00000000
0x4804C028 GPIO1_GPIO_IRQSTATUS_RAW_1
0x4804C028 00004000
0x4804C02C GPIO1_GPIO_IRQSTATUS_0
0x4804C02C 00000000
0x4804C030 GPIO1_GPIO_IRQSTATUS_1
0x4804C030 00000000
0x4804C034 GPIO1_GPIO_IRQSTATUS_SET_0
0x4804C034 00004000
0x4804C038 GPIO1_GPIO_IRQSTATUS_SET_1
0x4804C038 00000000
0x4804C03C GPIO1_GPIO_IRQSTATUS_CLR_0
0x4804C03C 00004000
0x4804C040 GPIO1_GPIO_IRQSTATUS_CLR_1

Conclusion

In conclusion, this report has detailed the successful development of a program that controls the lighting of four USR LEDs on the BeagleBone Black microprocessor. The project was divided into three parts, each building on the previous one, with the final outcome being a practical demonstration of the application of C language programming and embedded systems concepts. The step-by-step guide provided in this report, along with the explanations of programming concepts and techniques used, serve as a useful resource for those interested in developing similar projects.

Appendix. A: Design specifications and requirements

see next page

Intro Project (2023)

Introduction

The purpose of this exercise is to give you all a chance to get your Code Composer Studio environment setup with some working C code before we dig into the final project. In addition, it will give you some practice writing in C for the Beaglebone Black and let you explore the Sitara AM3358 technical reference manual.

In order to complete the project successfully, you'll have to use the R3T3SD process listed in the syllabus to work your way through the problems of a new design project. The lesson here is that if you carefully work through each step, rather than rushing to the end, you won't find this project difficult. The same will be true for our final project in the class, so this should be good practice for you.

NOTE: You must do all the work on this project by yourself, with no help from anyone but the instructor. Everything you need is in this project guide, the Sitara Technical Reference manual, the BeagleBone Black system reference manual, and the ECE 371 text. You will be doing yourself a favor by using only those sources, as you are getting to the point in your classes where things become hard to find accurate solutions through google.

NOTE: You must keep an AS-YOU-GO log of the steps you took, including your research findings, your thinking, your high and low level algorithms, your designs, problems encountered, how you solved these problems, how you tested your program, etc.

References

HALL chapters 4 and 5 (ECE 371)
BeagleBone Black System Reference Manual
TI Sitara AM335x Technical Reference Manual

Using C in Code Composer Studio

In this project, you'll use the C program tutorial that was posted to the project folder on Canvas to set up your CCS project for using the C programming language instead of ARM assembly. Then, using the provided example in the tutorial as a base, you will add a button to start and stop an LED pattern of your choosing. You can use any pattern you want here, just change it to something more interesting than the default from the example program.

Suggested Procedure

1. Read over the entire C Program Tutorial file to get an idea of how C and Assembly can be used together in a project.
2. Read and re-read the program example until you understand how to write to and read from registers in the BeagleBone Black (BBB) memory.
3. Review the DMTIMER register offsets in the Sitara Technical Reference Manual to remind yourself how to start and stop the timers, and load them with a value.
4. Review the INTC register offsets to remind yourself how to turn on interrupts and how to determine the bit position of your interrupt in a given register offset.
5. Use the tutorial to set up your CCS project for using the C programming language.
6. If you need a refresher on using CCS, the CCS tutorial is also posted to D2L.
7. Type in the program at the end of the C Program Tutorial. (Remember that copy and paste from a PDF can add ascii formatting characters that will cause errors when you try to run your program. It's good practice to just type this in.)
8. Debug the example program until it is running properly.
9. Write a High and Low level algorithm for the program with an added button that starts and stops a custom LED sequence of your design.
10. Reference your code from your 371 projects and add a button to your system.
    a. If you have your own setup, you can use any GPIO you choose. (Beware that some may have different defaults or pullup/pulldown resistors enabled that may cause issues with your circuit)
        i. Use a 47k resistor as a pullup resistor to 3.3v on one side of the button. Connect your GPIO to this side. The other side of the button should be connected to ground.
    b. If using a setup in the Lab, use the provided button and resistor, and your program should use GPIO1_14.
11. Revise your program to allow the button to start and stop the LED sequence.
12. Now change the lighting sequence to do something more interesting by implementing your custom LED sequence you detailed in your algorithm.
13. Once you have this working, demo for the TA or instructor your button starting and stopping your lighting sequence.

Deliverables

- **High and Low level algorithms.**
- **A detailed design log that would allow another engineer to repeat your process.**
- **Signoff from TA or instructor on Demo showing your button starting/stopping your LED sequence.**

Grading

100 pts Total - Working Program: 50, Complete High and Low Algorithms - 25, Detailed Design Log - 25

Appendix. B: LED Control Refrence Main Project

see next page

Portland State University
Maseeh College of Engineering and Computer Science
Electrical and Computer Engineering Department
EE371 - Microprocessors, Winter 2023

ARM Processor Assembly Language Programming

BeagleBone Black USR LEDs Control Program

Final *Design Project, Complete Version*

This Project is a final version that consists of two major parts:
Part 1: Lighting LEDs

Part 2: Creating an Interrupt Procedure for Servicing a Button Push

Part 3: Using a Programmable Timer to control LED switching times

By signing this statement, I affirm that I did not give any help to any other person, did not receive any help from any other person, except TA and Instructor and did not obtain any information from the Internet or other sources. Signature: WAIL .S .S AL KALBANI

Designed by: **Wail (Wa'el) S. AL KALBANI**
Instructor: Jonathan Anchell
TA: Louis Brennan
Date: March 16th, 2023

Introduction

The purpose of this report is to detail the development of a program that controls the lighting of four USR LEDs on the BeagleBone Black microposssseor. This project is divided into three parts, with each part building on the previous one. In Part 1, the focus is on developing a high-level algorithm for the LED pattern and implementing the low-level code to control the GPIO pins. Part 2 involves creating an interrupt procedure that services a button push and modifies the LED strobing pattern. Finally, Part 3 uses a programmable timer to control the LED switching times instead of a delay loop. The report will provide a step-by-step guide to the development process for each part, including detailed explanations of the programming concepts and techniques used. The goal of this project is to provide a practical demonstration of the application of assembly language programming and embedded systems concepts.

A. Design Log

The table below provides a detailed breakdown design log of the tasks required to complete the project parts.

Task No.	Description	Completion Date
	Part 1: Lighting LEDs	
1	Study the BeagleBone Black System Manual to determine which GPIO pins are connected to the 4 USR LEDs and the logic level required to turn on one of the LEDs. In the manual, looked under Hardware Files, Latest Production files for C Version, and System Reference Manual. Skimed through the User Manual until you find the GPIO connection for the 4 User LEDs and the logic level required to turn on an LED.	February 23, 2023
2	Developed a high level algorithm for a program that lights LED0 and LED 1 for 1 second, then switches off LED 0 and Led 1 and lights LED 1 and LED 2 for one second. Finally, it switches off LED 1 and LED 2 and turns on LED 2 and LED 3 for one second. Set this pattern to repeat.	February 28, 2023
	Carefully work through the section of Hall Chapter 4 that describes the AM3358 GPIO pins and the memory	February 28, 2023

3	mapped registers that control them. As part of this, determine the registers that control the GPIO pins connected to the LEDs. Also study the text section that shows how to set up bit templates for working with the GPIO registers.	
4	Set up the templates needed for the GPIO pins you are using.	February 28, 2023
5	Determined values and addresses you need to output a high or output a low on a GPIO pin.	February 28, 2023
6	Determined the RMW sequence of instructions and addresses required to program the GPIO pins for the LEDs as outputs and initialize the LEDs in the off state.	February 28, 2023
7	Developed the low level algorithm, as shown in chapter 4, for your program, including the delay loop	February 28, 2023
8	Write and carefully check the assembly language program.	February 28, 2023
9	Build, Load, Run, and Debug the program. Note that to single step through the program easily you can initially use a very small delay constant, so you don't have to single step forever to get from one LED to the next.	February 28, 2023
10	When the program worked, an electronic copy was made that you will be used for developing the next part of the project	February 28, 2023
Part 2: Creating an Interrupt Procedure for Servicing a Button Push		
11	Hall Chapter 5 read and button service program developed following the steps described in the text	March 10, 2023
12	Initialization list developed with values needed for the project, including GPIO2_1 for the pushbutton	March 10, 2023
13	Required steps of the assembly language program in Figure 5-14 modified according to the detailed initialization list	March 10, 2023
14	The revised program tested using the CODE COMPOSER and the required screenshots taken	March 10, 2023

15	Developed a report on the results of the testing and the details of the modifications made to the program Develop a report on the results of the testing and the details of the modifications made to the program	March 10, 2023
16	Set a time with the instructor for review and feedback	March 12, 2023
17	Write a test plan for the interrupt procedure, including test cases for various possible scenarios.	March 12, 2023
18	Develop and debug the interrupt procedure using the initialization list and button service program.	March 12, 2023
19	Perform testing of the interrupt procedure using the test plan and document the results.	March 12, 2023
Part 3 Using a Programmable Timer to control LED switching times		
20	Review the Timer 2 program discussed in text chapter 5 to understand how to set up a timer to produce interrupts at desired time intervals.	March 20, 2023
21	Study the timer section of Chapter 5 very carefully to learn how to add timer capability to your button program.	March 20, 2023
22	Modify the Timer 2 program to use Timer 4 instead of Timer 2 for this project.	March 20, 2023
23	Create an interrupt service routine for Timer 4 that will switch between the 4 LEDs.	March 20, 2023
24	Develop a way to keep track of which of the 4 LEDs is on when a timer interrupt occurs and take appropriate action.	March 20, 2023
25	Integrated rotating LED actions into the program.	March 20, 2023
26	Test the program to ensure that the LEDs switch at the desired time intervals.	March 20, 2023
27	Write a summary report of the project, including a description of the interrupt procedure and its function, test results, and any issues encountered.	March 21, 2023

B. Algorithms

In order to complete each part of the project outlined above, we will be following a set of algorithms to ensure the efficient and accurate development of the program. The algorithms will be as follows:

Part 1: Lighting LEDs

High-level Algorithm

1. Load a value into register R0.
2. Load a memory address into register R1.
3. Write the value from R0 to the memory address pointed to by R1.
4. Load the base address of GPIO1 into register R0.
5. Load a value to turn off all LEDs into register R2.
6. Add an offset to the GPIO1 base address to obtain the address of the GPIO1_OE register.
7. Read the GPIO1 enable register into register R4.
8. Load a value to enable GPIO1_21-24 into register R2.
9. Use a bitwise AND operation to clear bits 21-24 of the GPIO1 enable register (R4) and enable GPIO1_21-24.
10. Write the modified GPIO1 enable register (R2) to the GPIO1 enable register.
11. Enter a loop to blink the LEDs.
12. Add an offset to the GPIO1 base address to obtain the address of the register controlling the USR LED_0 and LED_1.
13. Load a value to turn on the USR LED_0 and LED_1 into register R2.
14. Write the value in R2 to the memory address pointed to by R1.
15. Call a subroutine named WAIT.
16. Add an offset to the GPIO1 base address to obtain the address of the register controlling the USR LED_0 and LED_1.
17. Load a value to turn off the USR LED_0 and LED_1 into register R2.
18. Write the value in R2 to the memory address pointed to by R5.
19. Add an offset to the GPIO1 base address to obtain the address of the register controlling the USR LED_1 and LED_2.
20. Load a value to turn on the USR LED_1 and LED_2 into register R2.
21. Write the value in R2 to the memory address pointed to by R1.
22. Call the WAIT subroutine.
23. Add an offset to the GPIO1 base address to obtain the address of the register controlling the USR LED_1 and LED_2.
24. Load a value to turn off the USR LED_1 and LED_2 into register R2.
25. Write the value in R2 to the memory address pointed to by R5.

26. Add an offset to the GPIO1 base address to obtain the address of the register controlling the USR LED_2 and LED_3.
27. Load a value to turn on the USR LED_2 and LED_3 into register R2.
28. Write the value in R2 to the memory address pointed to by R1.
29. Call the WAIT subroutine.
30. Add an offset to the GPIO1 base address to obtain the address of the register controlling the USR LED_2 and LED_3.
31. Load a value to turn off the USR LED_2 and LED_3 into register R2.
32. Write the value in R2 to the memory address pointed to by R5.
33. Call the WAIT subroutine.
34. Branch to the start of the BLINK_LEDS loop.
35. Enter a subroutine named WAIT.
36. Load a value to delay for one second into register R2.
37. Enter a loop to count down from R2.
38. If the counter has not reached zero, branch back to the COUNTER loop.
39. When the counter reaches zero, branch back to the instruction that called the WAIT subroutine.

Low-level Algorithm

1. Load the value 0x02 into register R0.
2. Load the address of the GPIO1 control registers into register R1.
3. Store the value in R0 to the address in R1 to enable GPIO1.
4. Load the base address of the GPIO1 control register into R0.
5. Load the value 0x01E00000 into R2 to turn off all LEDs.
6. Add 0x190 to R0 to get the address of the GPIO1 output set register.
7. Store the value in R2 to the address in R1 to turn off all LEDs.
8. Add 0x134 to R0 to get the address of the GPIO1 output enable register.
9. Load the value of the GPIO1 output to enable register into R4.
10. Load the value 0xFE1FFFFF into R2 to enable GPIO1 pins 21-24.
11. Clear bits 21-24 in R4 and store the result in R2 to enable GPIO1 pins 21-24.
12. Store the modified value of the GPIO1 output to enable register back into memory.
13. Define a label called "BLINK_LEDS".

REAPET: BLINK_LEDS

14. Add 0x194 to R0 to get the address of the GPIO1 output set register.
15. Load the value 0x00600000 into R2 to turn on LED_0 & LED_1.
16. Store the value in R2 to the address in R1 to turn on LED_0 & LED_1.
17. Branch and link to the "WAIT" label.
18. Add 0x190 to R0 to get the address of the GPIO1 output clear register.
19. Load the value 0x00600000 into R2 to turn off LED_0 & LED_1.
20. Store the value in R2 to the address in R5 to turn off LED_0 & LED_1.
21. Add 0x194 to R0 to get the address of the GPIO1 output set register.
22. Load the value 0x00C00000 into R2 to turn on LED_1 & LED_2.
23. Store the value in R2 to the address in R1 to turn on LED_1 & LED_2.

24. Branch and link to the "WAIT" label.
25. Add 0x190 to R0 to get the address of the GPIO1 output clear register.
26. Load the value 0x00C00000 into R2 to turn off LED_1 & LED_2.
27. Store the value in R2 to the address in R5 to turn off LED_1 & LED_2.
28. Add 0x194 to R0 to get the address of the GPIO1 output set register.
29. Load the value 0x01800000 into R2 to turn on LED_2 & LED_3.
30. Store the value in R2 to the address in R1 to turn on LED_2 & LED_3.
31. Branch and link to the "WAIT" label.
32. Add 0x190 to R0 to get the address of the GPIO1 output clear register.
33. Load the value 0x01800000 into R2 to turn off LED_2 & LED_3.
34. Store the value in R2 to the address in R5 to turn off LED_2 & LED_3.
35. Branch and link to the "WAIT" label.

UNTIL

36. Branch to the "BLINK_LEDS" label to start the cycle over.
37. Define a brack called WAIT
38. REAPET: WAIT
    a. load value to delay for one-second
    b. REAPET: COUNTER
        i. Set a down counter to consume time
    c. REPEAT
39. Branch back to the same line within BLINK_LEDS loop

Part 2: Creating an Interrupt Procedure for Servicing a Button Push

High-level Algorithm

1. Load the base address of STACK1 into R13 for SVC mode.
2. Add 0x1000 to R13 to point to the top of STACK1.
3. Switch to IRQ mode.
4. Load the base address of STACK2 into R13 for IRQ mode.
5. Add 0x1000 to R13 to point to the top of STACK2.
6. Switch back to SVC mode.
7. Load the value 0x02 into R0 to enable the clock for GPIO module.
8. Load the address of the CM_PER_GPI02_CLKCTRL register into R1.
9. Write the value in R0 to the address in R1 to turn on GPIO1 CLK.
10. Repeat steps 7-9 to turn on GPIO2 CLK.
11. Load the base address for GPI01 registers into R0.
12. Load the value 0x01E00000 into R2 to turn off all USR LEDs.
13. Add 0x190 to R0 to get the address for the GPIO_CLEARDATAOUT register.
14. Write the value in R2 to the address in R1 to turn off all USR LEDs.
15. Add 0x134 to R0 to get the address for the GPIO_OE register.
16. Load the value in the address pointed by R1 into R4.
17. Load the value 0xFE1FFFFF into R2.

18. Perform a bitwise AND operation between R4 and R2 to preserve the current settings of GP101_21-24 and clear the other bits.
19. Write the updated value in R2 to the address pointed by R1 to program GP101_21-24 as outputs.
20. Load the base address for GPI02 registers into R0.
21. Add 0x14C to R0 to get the address for the GPIO1_FALLINGDETECT register.
22. Load the value 0x00000002 into R2 to detect falling edge on GPIO2_1 and enable it.
23. Load the value in the address pointed by R1 into R3.
24. Perform a bitwise OR operation between R3 and R2 to set the bit corresponding to GP102 LED.
25. Write the updated value in R3 to the address pointed by R1 to turn on GP102 LED.
26. Add 0x34 to R0 to get the address for the GPIO OE register.
27. Write the value in R2 to the address pointed by R1 to set GP102 pin as an output.
28. Load the base address for interrupt control module registers into R1.
29. Load the value 0x01 into R4.
30. Write the value in R4 to the address pointed by R1 to enable generation of IRQ interrupt signals.
31. Move the current program status register (CPSR) to R3.
32. Clear the I bit in CPSR to enable IRQ interrupts.
33. Move the value in R3 back to CPSR.
34. Load the address of SWITCH into R3.
35. Load the value of the memory address pointed to by R3 into R2.
36. Test if the first bit of R2 is 1.
37. If the first bit of R2 is 0, branch to step 38.
38. If the first bit of R2 is not 0, branch to step 41.
39. Load the base address for GPIO1 into R1.
40. Add 0x194 to R0 to get the address for turning on USR LED_0 & USER LED_1.
41. Load the value 0x00600000 into R2 to turn on LED_0
42. Write the value in R2 to the address pointed to by R1 to turn on LED_0.
43. Load the value 0x00C00000 into R2 to turn on LED_1.
44. Write the value in R2 to the address pointed to by R1 to turn on LED_1.
45. Wait for 1 second
46. Load the base address for GPIO1 into R1.
47. Add 0x190 to R0 to get the address for turning off USR LED_0 & USER LED_1.
48. Load the value 0x00600000 into R2 to turn off LED_0.
49. Write the value in R2 to the address pointed to by R1 to turn off LED_0.
50. Load the value 0x00C00000 into R2 to turn off LED_1.
51. Write the value in R2 to the address pointed to by R1 to turn off LED_1.
52. Wait for 1 second

53. Turn on USR LED_2 and USER LED_3 by adding the value of R0 and 0x194 and storing the result in R1. Load a value of 0x01800000 to turn on the LEDs and write it to the address stored in R1.
54. Wait for 1 second using the WAIT subroutine.
55. Turn off USR LED_2 and USER LED_3 by adding the value of R0 and 0x190 and storing the result in R5. Load a value of 0x01800000 to turn off the LEDs and write it to the address stored in R5.
56. Branch to the LOOP label.
57. If the program reaches the OFF_LEDS label, load the base address for GPIO1_CLEARDATAOUT register into R0. Load a value of 0x01E00000 to turn off pins 21-24. Add the value of R0 and 0x190 and store the result in R1. Store the value in R4 to the address stored in R1 to turn off the LEDs. Branch to the LOOP label.
58. REPEAT: WAIT
    a. load the address of SWITCH into R10
    b. Load the value stored at the address in R10 into R11.
    c. Compare the value in R11 to 1.
    d. If R11 equals 1
        i. branch to the WAIT label.
    e. If not
        i. subtract the value in R9 by 1 and set the condition flags.
        ii. If the result is not zero, branch to the WAIT label.
    f. load a value of 0x00200000 into R9
    g. Return to the instruction that is called the WAIT subroutine.
59. UNTIL
60. Initialize the system and set up the stack for interrupts.
61. Check if there is a pending interrupt on INTC_PENDING_IRQ1.
62. If there is, check if it's a GPIO interrupt on GPIO_IQRSTATUS_0 bit 1.
63. If it is, turn off the GPIO interrupt request, turn off the INTC interrupt request, turn on an LED on GPIO1_12, and return to the wait loop.
64. If it's not a GPIO interrupt, simply return to the wait loop.
65. If there is no pending interrupt, continue waiting for an interruption.

Low-level Algorithm

1. Load the base address of STACK1 into R13 for SVC mode.
2. Add 0x1000 to R13 to point to the top of STACK1.
3. Switch to IRQ mode.
4. Load the base address of STACK2 into R13 for IRQ mode.
5. Add 0x1000 to R13 to point to the top of STACK2.
6. Switch back to SVC mode.
7. Load the value 0x02 into R0 to enable the clock for GPIO module.
8. Load the address of the CM_PER_GPI02_CLKCTRL register into R1.
9. Write the value in R0 to the address in R1 to turn on GPIO1 CLK.
10. Repeat steps 7-9 to turn on GPIO2 CLK.
11. Load the base address for GPI01 registers into R0.

12. Load the value 0x01E00000 into R2 to turn off all USR LEDs.
13. Add 0x190 to R0 to get the address for the GPIO_CLEARDATAOUT register.
14. Write the value in R2 to the address in R1 to turn off all USR LEDs.
15. Add 0x134 to R0 to get the address for the GPIO_OE register.
16. Load the value in the address pointed by R1 into R4.
17. Load the value 0xFE1FFFFF into R2.
18. Perform a bitwise AND operation between R4 and R2 to preserve the current settings of GP101_21-24 and clear the other bits.
19. Write the updated value in R2 to the address pointed by R1 to program GP101_21-24 as outputs.
20. Load the base address for GPI02 registers into R0.
21. Add 0x14C to R0 to get the address for the GPIO1_FALLINGDETECT register.
22. Load the value 0x00000002 into R2 to detect falling edge on GPIO2_1 and enable it.
23. Load the value in the address pointed by R1 into R3.
24. Perform a bitwise OR operation between R3 and R2 to set the bit corresponding to GP102 LED.
25. Write the updated value in R3 to the address pointed by R1 to turn on GP102 LED.
26. Add 0x34 to R0 to get the address for the GPIO OE register.
27. Write the value in R2 to the address pointed by R1 to set GP102 pin as an output.
28. Load the base address for interrupt control module registers into R1.
29. Load the value 0x01 into R4.
30. Write the value in R4 to the address pointed by R1 to enable generation of IRQ interrupt signals.
31. Move the current program status register (CPSR) to R3.
32. Clear the I bit in CPSR to enable IRQ interrupts.
33. Move the value in R3 back to CPSR.
34. Load the address of SWITCH into R3.
35. Load the value of the memory address pointed to by R3 into R2.
36. Test if the first bit of R2 is 1.
37. If the first bit of R2 is 0, branch to step 38.
38. If the first bit of R2 is not 0, branch to step 41.
39. Load the base address for GPIO1 into R1.
40. Add 0x194 to R0 to get the address for turning on USR LED_0 & USER LED_1.
41. Load the value 0x00600000 into R2 to turn on LED_0
42. Write the value in R2 to the address pointed to by R1 to turn on LED_0.
43. Load the value 0x00C00000 into R2 to turn on LED_1.
44. Write the value in R2 to the address pointed to by R1 to turn on LED_1.
45. Wait for 1 second
46. Load the base address for GPIO1 into R1.

47. Add 0x190 to R0 to get the address for turning off USR LED_0 & USER LED_1.
48. Load the value 0x00600000 into R2 to turn off LED_0.
49. Write the value in R2 to the address pointed to by R1 to turn off LED_0.
50. Load the value 0x00C00000 into R2 to turn off LED_1.
51. Write the value in R2 to the address pointed to by R1 to turn off LED_1.
52. Wait for 1 second
53. Turn on USR LED_2 and USER LED_3 by adding the value of R0 and 0x194 and storing the result in R1. Load a value of 0x01800000 to turn on the LEDs and write it to the address stored in R1.
54. Wait for 1 second using the WAIT subroutine.
55. Turn off USR LED_2 and USER LED_3 by adding the value of R0 and 0x190 and storing the result in R5. Load a value of 0x01800000 to turn off the LEDs and write it to the address stored in R5.
56. Branch to the LOOP label.
57. If the program reaches the OFF_LEDS label, load the base address for GPIO1_CLEARDATAOUT register into R0. Load a value of 0x01E00000 to turn off pins 21-24. Add the value of R0 and 0x190 and store the result in R1. Store the value in R4 to the address stored in R1 to turn off the LEDs. Branch to the LOOP label.
58. REPEAT: WAIT
59. load the address of SWITCH into R10
60. Load the value stored at the address in R10 into R11.
61. Compare the value in R11 to 1.
62. If R11 equals 1
63. branch to the WAIT label.
64. If not
65. subtract the value in R9 by 1 and set the condition flags.
66. If the result is not zero, branch to the WAIT label.
67. load a value of 0x00200000 into R9
68. Return to the instruction that is called the WAIT subroutine.
69. UNTIL
70. The algorithm begins by saving the current state of the processor on the stack. This is done by using the STMFD instruction to push the contents of registers R0 through R3 and R9, as well as the Link Register (LR), onto the stack.
71. The algorithm then loads the address of the INTC_PENDING_IRQ1 register into register R0 and loads the contents of that register into register R1.
72. Tests whether Bit 0 (Bit 32) of the content of R1 is set or not by performing a bitwise AND operation with the value 0x00000001.
    a. If the zero flag is set, it means that Bit 0 is clear, and the algorithm branches to the PASS_ON label.
    b. If Bit 0 is set, the algorithm loads the address of the GPIO_IQRSTATUS_0 register into register R0 and loads the contents of that register into register R1.

73. Tests whether Bit 1 (Bit 32) of the content of R1 is clear or not by performing a bitwise AND operation with the value 0x00000002.
    a. If the zero flag is clear, it means that Bit 1 is set, and the algorithm branches to the BUTTON_SVC label.
    b. If Bit 1 is clear, the algorithm branches to the PASS_ON label.
74. The PASS_ON label pops the saved state of the processor from the stack using the LDMFD instruction and subtracts 4 from the content of the LR register to return to the interrupted program.
75. The BUTTON_SVC label turns off the GPIO01_14 interrupt request by writing the value 0x00000002 to the GPIO1_IRQSTATUS_0 register. It also clears Bit 0 of the INTC__CONTROL register to allow the processor to respond to new IRQs.
76. The label then turns on an LED connected to the GPIO1_12 pin by loading the address of the SWITCH label into register R3 and loading the contents of that label into register R9.
77. Compares the content of R9 with the value 0x0 and sets the content of R9 to 0x1 if the zero flag is set and to 0x0 if it is clear.
78. Writes the contents of R9 to the SWITCH label.
79. Restores the saved state of the processor from the stack using the LDMFD instruction and subtracts 4 from the content of the LR register to return to the interrupted program.
80. Include s SYS_IRQ label, which is used to store the address of the system's IRQ handler.
81. Also includes two stacks, STACK1 and STACK2, each with 1024 words of memory.
82. Includes a SWITCH label, which is used to control the LED connected to the GPIO1 pin. The label is initialized to 0x01, which turns the LED on by default.

Part 3: Using a Programmable Timer to control LED switching times

High-level Algorithm

1. **Define the timer interval**
2. Load the base address of STACK1 into R13 for SVC mode.
3. Add 0x1000 to R13 to point to the top of STACK1.
4. Switch to IRQ mode.
5. Load the base address of STACK2 into R13 for IRQ mode.
6. Add 0x1000 to R13 to point to the top of STACK2.
7. Switch back to SVC mode.
8. Load the value 0x02 into R0 to enable the clock for GPIO module.
9. Load the address of the CM_PER_GPI02_CLKCTRL register into R1.
10. Write the value in R0 to the address in R1 to turn on GPIO1 CLK.

11. Repeat steps 7-9 to turn on GPIO2 CLK.
12. Load the base address for GPI01 registers into R0.
13. Load the value 0x01E00000 into R2 to turn off all USR LEDs.
14. Add 0x190 to R0 to get the address for the GPIO_CLEARDATAOUT register.
15. Write the value in R2 to the address in R1 to turn off all USR LEDs.
16. Add 0x134 to R0 to get the address for the GPIO_OE register.
17. Load the value in the address pointed by R1 into R4.
18. Load the value 0xFE1FFFFF into R2.
19. Perform a bitwise AND operation between R4 and R2 to preserve the current settings of GP101_21-24 and clear the other bits.
20. **Configure the timer**
21. **Initialize and enable the timer interrupt**
22. Write the updated value in R2 to the address pointed by R1 to program GP101_21-24 as outputs.
23. Load the base address for GPI02 registers into R0.
24. Add 0x14C to R0 to get the address for the GPIO1_FALLINGDETECT register.
25. Load the value 0x00000002 into R2 to detect falling edge on GPIO2_1 and enable it.
26. Load the value in the address pointed by R1 into R3.
27. Perform a bitwise OR operation between R3 and R2 to set the bit corresponding to GP102 LED.
28. Write the updated value in R3 to the address pointed by R1 to turn on GP102 LED.
29. Add 0x34 to R0 to get the address for the GPIO OE register.
30. Write the value in R2 to the address pointed by R1 to set GP102 pin as an output.
31. Load the base address for interrupt control module registers into R1.
32. Load the value 0x01 into R4.
33. Write the value in R4 to the address pointed by R1 to enable generation of IRQ interrupt signals.
34. Move the current program status register (CPSR) to R3.
35. Clear the I bit in CPSR to enable IRQ interrupts.
36. Move the value in R3 back to CPSR.
37. Load the address of SWITCH into R3.
38. Load the value of the memory address pointed to by R3 into R2.
39. Test if the first bit of R2 is 1.
40. If the first bit of R2 is 0, branch to step 38.
41. If the first bit of R2 is not 0, branch to step 41.
42. Load the base address for GPIO1 into R1.
43. Add 0x194 to R0 to get the address for turning on USR LED_0 & USER LED_1.
44. Load the value 0x00600000 into R2 to turn on LED_0
45. Write the value in R2 to the address pointed to by R1 to turn on LED_0.
46. Load the value 0x00C00000 into R2 to turn on LED_1.
47. Write the value in R2 to the address pointed to by R1 to turn on LED_1.
48. Wait for 1 second
49. Load the base address for GPIO1 into R1.
50. Add 0x190 to R0 to get the address for turning off USR LED_0 & USER LED_1.

51. Load the value 0x00600000 into R2 to turn off LED_0.
52. Write the value in R2 to the address pointed to by R1 to turn off LED_0.
53. Load the value 0x00C00000 into R2 to turn off LED_1.
54. Write the value in R2 to the address pointed to by R1 to turn off LED_1.
55. Wait for 1 second
56. Turn on USR LED_2 and USER LED_3 by adding the value of R0 and 0x194 and storing the result in R1. Load a value of 0x01800000 to turn on the LEDs and write it to the address stored in R1.
57. Wait for 1 second using the WAIT subroutine.
58. Turn off USR LED_2 and USER LED_3 by adding the value of R0 and 0x190 and storing the result in R5. Load a value of 0x01800000 to turn off the LEDs and write it to the address stored in R5.
59. Branch to the LOOP label.
60. If the program reaches the OFF_LEDS label, load the base address for GPIO1_CLEARDATAOUT register into R0. Load a value of 0x01E00000 to turn off pins 21-24. Add the value of R0 and 0x190 and store the result in R1. Store the value in R4 to the address stored in R1 to turn off the LEDs. Branch to the LOOP label.
61. REPEAT: WAIT
    a. load the address of SWITCH into R10
    b. Load the value stored at the address in R10 into R11.
    c. Compare the value in R11 to 1.
    d. If R11 equals 1
        i. branch to the WAIT label.
    e. If not
        i. subtract the value in R9 by 1 and set the condition flags.
    f. If the result is not zero, branch to the WAIT label.
    g. load a value of 0x00200000 into R9
    h. Return to the instruction that is called the WAIT subroutine.
62. UNTIL
63. **Initialize DMTimer register**
64. **Reset timer at CFG configuration register**
65. **Enable overflow interrupt at IRQENABLE_SET**
66. **Count for 1 second**
67. **Write to TCRR**
68. **Enable IRQ in CPSR**
69. Initialize the system and set up the stack for interrupts.
70. Check if there is a pending interrupt on INTC_PENDING_IRQ1.
71. If there is, check if it's a GPIO interrupt on GPIO_IQRSTATUS_0 bit 1.
72. If it is, turn off the GPIO interrupt request, turn off the INTC interrupt request, turn on an LED on GPIO1_12, and return to the wait loop.
73. If it's not a GPIO interrupt, simply return to the wait loop.
74. If there is no pending interruption, continue waiting for an interruption.
75. **REPEAT: TIMER_INTC**
    a. **Reading INTC_PENDING_IRQ2 Timer overflow**
    b. **Test pin of INTC_PENDING_IRQ2**

c. If INTC_PENDING_IRQ2 is 0 flag Z
i. Branch to PASS_ON
d. If INTC_PENDING_IRQ2 is 1 flag Z is clear
i. Loopback
76. UNTIL
77. Turn off timer request

Low-level Algorithm

1. Define the timer interval
    a. TIMER_INTERVAL EQU 0x100000
2. Load the base address of STACK1 into R13 for SVC mode.
3. Add 0x1000 to R13 to point to the top of STACK1.
4. Switch to IRQ mode.
5. Load the base address of STACK2 into R13 for IRQ mode.
6. Add 0x1000 to R13 to point to the top of STACK2.
7. Switch back to SVC mode.
8. Load the value 0x02 into R0 to enable the clock for GPIO module.
9. Load the address of the CM_PER_GPI02_CLKCTRL register into R1.
10. Write the value in R0 to the address in R1 to turn on GPIO1 CLK.
11. Repeat steps 7-9 to turn on GPIO2 CLK.
12. Load the base address for GPI01 registers into R0.
13. Load the value 0x01E00000 into R2 to turn off all USR LEDs.
14. Add 0x190 to R0 to get the address for the GPIO_CLEARDATAOUT register.
15. Write the value in R2 to the address in R1 to turn off all USR LEDs.
16. Add 0x134 to R0 to get the address for the GPIO_OE register.
17. Load the value in the address pointed by R1 into R4.
18. Load the value 0xFE1FFFFF into R2.
19. Perform a bitwise AND operation between R4 and R2 to preserve the current settings of GP101_21-24 and clear the other bits.
20. Configure the timer
    a. Load timer interval value into R0
    b. Load the address of the Timer 1 load register
    c. Write timer interval value to Timer 1 load register
    d. Set Timer 1 control register to periodic mode
    e. Load the address of the Timer 1 control register
    f. Write Timer 1 control register value
21. Initialize and enable the timer interrupt
    a. Load the address of the interrupt enable set register
    b. Set the bit for Timer 1 interrupt
    c. Write interrupt enable set register value
22. Write the updated value in R2 to the address pointed by R1 to program GP101_21-24 as outputs.
23. Load the base address for GPI02 registers into R0.
24. Add 0x14C to R0 to get the address for the GPIO1_FALLINGDETECT register.

25. Load the value 0x00000002 into R2 to detect falling edge on GPIO2_1 and enable it.
26. Load the value in the address pointed by R1 into R3.
27. Perform a bitwise OR operation between R3 and R2 to set the bit corresponding to GP102 LED.
28. Write the updated value in R3 to the address pointed by R1 to turn on GP102 LED.
29. Add 0x34 to R0 to get the address for the GPIO OE register.
30. Write the value in R2 to the address pointed by R1 to set GP102 pin as an output.
31. Load the base address for interrupt control module registers into R1.
32. Load the value 0x01 into R4.
33. Write the value in R4 to the address pointed by R1 to enable generation of IRQ interrupt signals.
34. Move the current program status register (CPSR) to R3.
35. Clear the I bit in CPSR to enable IRQ interrupts.
36. Move the value in R3 back to CPSR.
37. Load the address of SWITCH into R3.
38. Load the value of the memory address pointed to by R3 into R2.
39. Test if the first bit of R2 is 1.
40. If the first bit of R2 is 0, branch to step 38.
41. If the first bit of R2 is not 0, branch to step 41.
42. Load the base address for GPIO1 into R1.
43. Add 0x194 to R0 to get the address for turning on USR LED_0 & USER LED_1.
44. Load the value 0x00600000 into R2 to turn on LED_0
45. Write the value in R2 to the address pointed to by R1 to turn on LED_0.
46. Load the value 0x00C00000 into R2 to turn on LED_1.
47. Write the value in R2 to the address pointed to by R1 to turn on LED_1.
48. Wait for 1 second
49. Load the base address for GPIO1 into R1.
50. Add 0x190 to R0 to get the address for turning off USR LED_0 & USER LED_1.
51. Load the value 0x00600000 into R2 to turn off LED_0.
52. Write the value in R2 to the address pointed to by R1 to turn off LED_0.
53. Load the value 0x00C00000 into R2 to turn off LED_1.
54. Write the value in R2 to the address pointed to by R1 to turn off LED_1.
55. Wait for 1 second
56. Turn on USR LED_2 and USER LED_3 by adding the value of R0 and 0x194 and storing the result in R1. Load a value of 0x01800000 to turn on the LEDs and write it to the address stored in R1.
57. Wait for 1 second using the WAIT subroutine.
58. Turn off USR LED_2 and USER LED_3 by adding the value of R0 and 0x190 and storing the result in R5. Load a value of 0x01800000 to turn off the LEDs and write it to the address stored in R5.
59. Branch to the LOOP label.
60. If the program reaches the OFF_LEDS label, load the base address for GPIO1_CLEARDATAOUT register into R0. Load a value of 0x01E00000 to turn off pins 21-24. Add the value of R0 and 0x190 and store the result in R1. Store

the value in R4 to the address stored in R1 to turn off the LEDs. Branch to the LOOP label.
61. REPEAT: WAIT
62. load the address of SWITCH into R10
63. Load the value stored at the address in R10 into R11.
64. Compare the value in R11 to 1.
65. If R11 equals 1
66. branch to the WAIT label.
67. If not
68. subtract the value in R9 by 1 and set the condition flags.
69. If the result is not zero, branch to the WAIT label.
70. load a value of 0x00200000 into R9
71. Return to the instruction that is called the WAIT subroutine.
72. UNTIL
73. Initialize DMTimer register
74. Reset timer at CFG configuration register
75. Enable overflow interrupt at IRQENABLE_SET
76. Count for 1 second
77. Write to TCRR
78. Enable IRQ in CPSR
79. Initialize the system and set up the stack for interrupts.
80. Check if there is a pending interrupt on INTC_PENDING_IRQ1.
81. If there is, check if it's a GPIO interrupt on GPIO_IQRSTATUS_0 bit 1.
82. If it is, turn off the GPIO interrupt request, turn off the INTC interrupt request, turn on an LED on GPIO1_12, and return to the wait loop.
83. If it's not a GPIO interrupt, simply return to the wait loop.
84. If there is no pending interruption, continue waiting for an interruption.
85. REPEAT: TIMER_INTC
86. Reading INTC_PENDING_IRQ2 Timer overflow
87. Test pin of INTC_PENDING_IRQ2
88. If INTC_PENDING_IRQ2 is 0 flag Z
89. Branch to PASS_ON
90. If INTC_PENDING_IRQ2 is 1 flag Z is clear
91. Loopback
92. UNTIL
93. Turn off timer request

C. Discussion

As shown in the figures below, the program assembly code was initializes and configures the microcontroller's GPIO (General Purpose Input/Output) and timer registers. The code uses the timer to create a cycle of lighting up four LEDs in sequence. LED 3 is lit for one second, then LED 2 is lit for one second, followed by LED 1 and then LED 0. The cycle repeats continuously. The code initializes the GPIO registers to output the signal to the LED pins. It also detects the falling edge on GPIO2_1 to enable interrupts. It then sets up and initializes the interrupt control module to enable IRQ interrupts. The timer is configured to generate periodic interrupts. The loop at the end of the code waits for an interrupt to occur. When the timer generates an interrupt, the code executes an interrupt service routine (ISR) that lights up the appropriate LED and updates the cycle. The ISR then returns to the main code, which continues waiting for the next interrupt.

D. Assembly Code Printouts

See next page

```
.text
.global _start
_start:

        LDR R0, =0x02
        LDR R1, =0x44E000AC
        STR R0, [R1]

        @ GPIO1 Clock enable & pins setup
        LDR R0, =0x4804C000         @ Load base GPIO1 address GPIO1 (CM_PER_GPIO1_CLKCTRL)

        @ trun off all LEDs
        MOV R2, #0x01E00000         @ load a value to turn off all LEDs
        ADD R1, R0, #0x190
        STR R2, [R1]                @ Write value to register

        @ Program GPIO1_21-24
        ADD R1, R0, #0x134          @ make GPIO1_OE register address
        LDR R4, [R1]                @ READ GPIO1 enable register
        MOV R2, #0xFE1FFFFF         @ load values to enable GPIO1_21-24
        AND R2, R4, R2              @ clear bits 21-24 (MODIFY)
        STR R2, [R1]                @ Write to GPIO1 enable register

        BLINK_LEDS:
        @TURN ON USR LED_0 & LED_1
        ADD R1, R0, #0x194          @ Add 0x194 to the value in R0 and store the value in R1
        MOV R2, #0x00600000         @ Load the value 0x00600000 into R2
        STR R2, [R1]                @ Store the value in R2 into the memory address pointed to by R1

            BL WAIT                 @ Branch and link to the "WAIT"

        @TURN OFF USR LED_0 & LED_1
        ADD R5, R0, #0x190          @ Add the value of R0 and 0x190 and store the result in R2
        MOV R2, #0x00600000         @ Load a value to turn off  LED_0 & USER LED_1
        STR R2, [R5]                @ Write  to turn off LED_0 & USER LED_1

        @TURN ON USR LED_1 & LED_2
        ADD R1, R0, #0x194          @ Add the value of R0 and 0x194 and store the result in R1
        MOV R2, #0x00C00000         @ Load a value to turn on LED_1 & USER LED_2
        STR R2, [R1]                @ Write to turn on LED_1 & USER LED_2

            BL WAIT

        @TURN OFF USR LED_1 & LED_2
        ADD R5, R0, #0x190          @ Add the value of R0 and 0x190 and store the result in R2
        MOV R2, #0x00C00000         @ Load a value to turn off LED_1 & USER LED_2
        STR R2, [R5]                @ Write  to turn off LED_1 & USER LED_2

        @TURN ON USR LED_2 & LED_3
        ADD R1, R0, #0x194          @ Add the value of R0 and 0x190 and store the result in R2
        MOV R2, #0x01800000         @ Load a value to turn on LED_2 & USER LED_3
        STR R2, [R1]                @ Write to turn on LED_2 & USER LED_3

            BL WAIT

        @TURN OFF USR LED_2 & LED_3
        ADD R5, R0, #0x190          @ Add the value of R0 and 0x194 and store the result in R1
        MOV R2, #0x01800000         @ Load a value to turn off LED_2 & USER LED_3
        STR R2, [R5]                @ Write  to turn off LED_2 & USER LED_3

            BL WAIT

        B BLINK_LEDS

        WAIT:
        MOV R2,#0x00200000          @ load value to delay for one second
            COUNTER:
            SUBS R2,#1                  @ set a down counter to consume time
            BNE COUNTER                 @ loop back
            MOV PC, LR                  @ beanch back to same ON line
.END
```

Figure 1: Part1, LED Program Code

```
.text
.global _start
.global INT_DIRECTOR                @ Declares the INT_DIRECTOR label as a global symbol
_start:

        LDR R13, =STACK1            @ Point to base of STACK1 for SVC mode
        ADD R13, R13, #0x1000       @ Point to top of STACK
        CPS #0x12                   @ Switch to IRQ mode
        LDR R13, =STACK2            @ Point to STACK2 for IRQ mode
        ADD R13, R13, #0x1000       @ Point to top of STACK
        CPS #0x13                   @ Back to SVC mode

    @ Turn on GPIO CLK
            @ Turn on GPIO1 CLK
            LDR R0, =0x02               @ Value to enable clock for GPIO module
            LDR R1, =0x44E000B0         @ Address of CM_PER_GPI02_CLKCTRL Register
            STR R0, [R1]                @ Write to register, this is done to turn on GPI02

            @ Turn on GPIO2 CLK
            LDR R0, =0x02               @ Value to enable clock for GPIO module
            LDR R1, =0x44E000AC         @ Address of CM_PER_GPI02_CLKCTRL Register
            STR R0, [R1]                @ Write to register, this is done to turn on GPI02

            @ Base address initilization
            LDR R0, =0x4804C000     @ Base address for GPI01 registers

            @ Turn off all USR LEDs
            MOV R2, #0x01E00000     @ Load value to CLEARDATAOUT to turn off LEDs
            ADD R1, R0, #0x190      @ R1 = address for GPIO_CLEARDATAOUT register
            STR R2, [R1]            @ Write to GPIO_CLEARDATAOUT register

    @ Program GPIO1_21-24 as output
        ADD R1, R0, #0x134          @ Program GP101_21-24 as outputs
        LDR R4, [R1]
        MOV R2, #0xFE1FFFFF
        AND R2, R4, R2
        STR R2, [R1]

    @ Detect falling edge on GPIO2_1 and enable
        LDR R0, =0x481AC000         @ Base address for GPI02 registers
        ADD R1, R0, #0x14C          @ R1 = address for GPIO1_FALLINGDETECT register
        MOV R2, #0x00000002
        LDR R3, [R1]                @ Read value from SETDATAOUT register
        ORR R3, R3, R2              @ Turn on GP102 LED by setting its bit in the register
        STR R3, [R1]                @ Write the updated value back to the SETDATAOUT register
        ADD R1, R0, #0x34           @ Make GPIO OE register address
        STR R2, [R1]                @ Set GP102 pin as an output

    @ Initialize INTC
        LDR R1, =0x482000A8         @ Base address for interrupt control module registers
        MOV R4, #0x01
        STR R4, [R1]                @ Enable generation of IRQ interrupt signals

    @ Make sure processor IQR enable in CPSR
        MRS R3, CPSR                @ Move current program status register (CPSR) to R3
        BIC R3, #0x80               @ Clear the I bit in CPSR to enable IRQ interrupts
        MSR CPSR_c, R3              @ Move R3 back to CPSR

    @ Wait for interupt
    LOOP:
        LDR R3, =SWITCH           @ Load the address of SWITCH into R3
        LDR R2, [R3]              @ Load the value of the memory address pointed to by R3 into R2
        TST R2, #0x1              @ Test if the first bit of R2 is 1
        BEQ ON_LEDS               @ If the first bit of R2 is 0, branch to ON_LEDS
        BNE OFF_LEDS              @ If the first bit of R2 is not 0, branch to OFF_LEDS

    ON_LEDS:
        LDR R1, =0x4804C000       @ Load base address for GPIO1

    @ Turn on USR LED_0 & USER LED_1
        ADD R1, R0, #0x194        @ Add the value of R0 and 0x194 and store the result in R1
        MOV R2, #0x00600000       @ Load a value to turn on LED_0 & USER LED_1
        STR R2, [R1]              @ Write to turn on LED_0 & USER LED_1
```

```
82    @ Wait for 1 second
83        BL WAIT
84
85    @ Turn off USR LED_0 & USER LED_1
86        ADD R5, R0, #0x190        @ Add the value of R0 and 0x190 and store the result in R2
87        MOV R2, #0x00600000       @ Load a value to turn off  LED_0 & USER LED_1
88        STR R2, [R5]              @ Write  to turn off LED_0 & USER LED_1
89
90    @ Turn on USR LED_1 & USER LED_2
91        ADD R1, R0, #0x194        @ Add the value of R0 and 0x194 and store the result in R1
92        MOV R2, #0x00C00000       @ Load a value to turn on LED_1 & USER LED_2
93        STR R2, [R1]              @ Write to turn on LED_1 & USER LED_2
94
95    @ Wait for 1 second
96        BL WAIT
97
98    @ Turn off USR LED_1 & USER LED_2
99        ADD R5, R0, #0x190        @ Add the value of R0 and 0x190 and store the result in R2
100       MOV R2, #0x00C00000       @ Load a value to turn off LED_1 & USER LED_2
101       STR R2, [R5]              @ Write  to turn off LED_1 & USER LED_2
102
103   @ Turn on USR LED_2 & USER LED_3
104       ADD R1, R0, #0x194        @ Add the value of R0 and 0x190 and store the result in R2
105       MOV R2, #0x01800000       @ Load a value to turn on LED_2 & USER LED_3
106       STR R2, [R1]              @ Write to turn on LED_2 & USER LED_3
107
108   @ Wait for 1 second
109       BL WAIT
110
111   @ Turn off USR LED_2 & USER LED_3
112       ADD R5, R0, #0x190        @ Add the value of R0 and 0x194 and store the result in R1
113       MOV R2, #0x01800000       @ Load a value to turn off LED_2 & USER LED_3
114       STR R2, [R5]              @ Write  to turn off LED_2 & USER LED_3
115
116       B LOOP                    @ Branch to LOOP
117
118   OFF_LEDS:
119       LDR R0, =0x4804C000       @Base address for GPIO1_CLEARDATAOUT register
120
121           @ Turn off all USR LEDs
122           MOV R4, #0x01E00000   @ Load a value to turn off pin 21-24
123           ADD R1, R0, #0x190    @ Add the value of R0 and 0x190 and store the result in R1
124           STR R4, [R1]          @ Store the value to GPIO1_CLEARDATAOUT register
125
126           B LOOP                @ Branch to LOOP
127
128   WAIT:
129       LDR R10, =SWITCH          @ Load word to program from SWITCH
130       LDR R11,[R10]             @ Load the value into register R11
131       CMP R11, #0x1             @ Compare the value in R11 to the constant value 1
132           BEQ WAIT              @ Branch to the label WAIT if the previous comparison was
133                                 @ true (i.e., if R11 equals 1)
134
135       SUBS R9,R9,#1             @ Subtract the constant value 1 from the value in register
136                                 @ R9 and set the condition flags based on the result
137           BNE WAIT              @ Branch to the label WAIT if the previous subtraction did
138                                 @ not result in R9 being zero
139       MOV R9,#0x00200000
140       MOV PC,LR
141
142   INT_DIRECTOR:
143       STMFD SP!, {R0-R3, R9, LR}     @ Push the contents of registers R0 through R3 and LR onto the stack
144       LDR R0, =0x482000B8       @ Address for INTC_PENDING_IRQ1
145       LDR R1, [R0]              @ Load the content of INTC_PENDING_IRQ1
146       TST R1, #0x00000001       @ Test if the bitwise AND of the content of R1 (Bit 0 = Bit 32)
147       BEQ PASS_ON               @ Branch to the instruction located at the label PASS_ON if the zero flag is set
148       LDR R0, =0x481AC02C       @ Load GPIO_IQRSTATUS_0
149       LDR R1, [R0]              @ Load the content of the memory location pointed to by R0 into R1
150       TST R1, #0x00000002       @ Test if the bitwise AND of the content of R1 (Bit 1 = Bit 32)
151       BNE BUTTON_SVC            @ Branch to the instruction located at the label BUTTON_SVC if the zero flag is clear
152       BEQ PASS_ON               @ Branch to the instruction located at the label PASS_ON if the zero flag is set
153
```

```
153
154     PASS_ON:
155         LDMFD SP!, {R0-R3, R9, LR}     @ Pop the contents of registers R0 through R3 and LR from the stack
156         SUBS PC, LR, #4           @ Subtract 4 from the content of LR and store the result in PC
157
158     BUTTON_SVC:
159         MOV R1, #0x00000002       @ Value turns off GPI01_14 Interrupt request
160                                   @ Also turns off INTC interrupt request
161         STR R1, [R0]              @ Write to GPIO1_IRQSTATUS_0 register
162
163     @ Turn off NEWIRQA bit in INT_CONTROL, so processor can respond to new IRQ
164         LDR R0, =0x48200048       @ Address of INTC__CONTROL register
165         MOV R1, #01               @ Value to clear bit 0
166         STR R1, [R0]              @ Write to INTC__CONTROL register
167
168     @ Turn on LED ON GPIO1_12
169         LDR R3, =SWITCH           @ Load the address of the label SWITCH into R3
170         LDR R9, [R3]              @ Load value of SWITCH register
171         CMP R9, #0x0              @ Compare the content of R9 with the hexadecimal value 0x0
172         MOVEQ R9, #0x1            @ If the zero flag is set, move the hexadecimal value 0x1 into R9
173         MOVNE R9, #0x0
174         STR R9, [R3]
175
176     @ Return to wait loop
177         LDMFD SP!, {R0-R3, R9, LR}   @ Restore registers
178         SUBS PC, LR, #4           @ Return from IRQ interrupt procedure
179
180     .align 2
181     SYS_IRQ:        .word 0       @ Location to store systems IRQ address
182     .data
183     .align 2
184     STACK1:         .rept 1024
185                     .word 0x00
186                     .endr
187     STACK2:         .rept 1024
188                     .word 0x00
189                     .endr
190     SWITCH:         .word 0x01
191 .END
```

Figure 2: Part2, Creating an Interrupt Procedure for Servicing a Button Push

```
.text
.global _start
.global INT_DIRECTOR                @ Declares the INT_DIRECTOR label as a global symbol
_start:
        EQU TIMER_INTERVAL, 0x100000   @ Define timer interval as 1 second (in cycles)

        LDR R13, =STACK1            @ Point to base of STACK1 for SVC mode
        ADD R13, R13, #0x1000       @ Point to top of STACK
        CPS #0x12                   @ Switch to IRQ mode
        LDR R13, =STACK2            @ Point to STACK2 for IRQ mode
        ADD R13, R13, #0x1000       @ Point to top of STACK
        CPS #0x13                   @ Back to SVC mode

    @ Turn on GPIO CLK
            @ Turn on GPIO1 CLK
            LDR R0, =0x02               @ Value to enable clock for GPIO module
            LDR R1, =0x44E000B0         @ Address of CM_PER_GPI02_CLKCTRL Register
            STR R0, [R1]                @ Write to register, this is done to turn on GPI02

            @ Turn on GPIO2 CLK
            LDR R0, =0x02               @ Value to enable clock for GPIO module
            LDR R1, =0x44E000AC         @ Address of CM_PER_GPI02_CLKCTRL Register
            STR R0, [R1]                @ Write to register, this is done to turn on GPI02

            @ Base address initilization
            LDR R0, =0x4804C000     @ Base address for GPI01 registers

            @ Turn off all USR LEDs
            MOV R2, #0x01E00000     @ Load value to CLEARDATAOUT to turn off LEDs
            ADD R1, R0, #0x190      @ R1 = address for GPIO_CLEARDATAOUT register
            STR R2, [R1]            @ Write to GPIO_CLEARDATAOUT register

    @ Program GPIO1_21-24 as output
        ADD R1, R0, #0x134          @ Program GP101_21-24 as outputs
        LDR R4, [R1]
        MOV R2, #0xFE1FFFFF
        AND R2, R4, R2
        STR R2, [R1]

    @ Detect falling edge on GPIO2_1 and enable
        LDR R0, =0x481AC000         @ Base address for GPI02 registers
        ADD R1, R0, #0x14C          @ R1 = address for GPIO1_FALLINGDETECT register
        MOV R2, #0x00000002
        LDR R3, [R1]                @ Read value from SETDATAOUT register
        ORR R3, R3, R2              @ Turn on GP102 LED by setting its bit in the register
        STR R3, [R1]                @ Write the updated value back to the SETDATAOUT register
        ADD R1, R0, #0x34           @ Make GPIO OE register address
        STR R2, [R1]                @ Set GP102 pin as an output

    @ Initialize INTC
        LDR R1, =0x482000A8         @ Base address for interrupt control module registers
        MOV R4, #0x01
        STR R4, [R1]                @ Enable generation of IRQ interrupt signals

    @ Make sure processor IQR enable in CPSR
        MRS R3, CPSR                @ Move current program status register (CPSR) to R3
        BIC R3, #0x80               @ Clear the I bit in CPSR to enable IRQ interrupts
        MSR CPSR_c, R3              @ Move R3 back to CPSR

    @ Initialize and configure timer
        LDR R0, =TIMER_INTERVAL      @ Load timer interval value into R0
        LDR R1, =0x48200040          @ Load the address of the Timer 1 load register
        STR R0, [R1]                 @ Write timer interval value to Timer 1 load register
        MOV R0, #0x2                 @ Set Timer 1 control register to periodic mode
        LDR R1, =0x48200044          @ Load the address of the Timer 1 control register
        STR R0, [R1]                 @ Write Timer 1 control register value
```

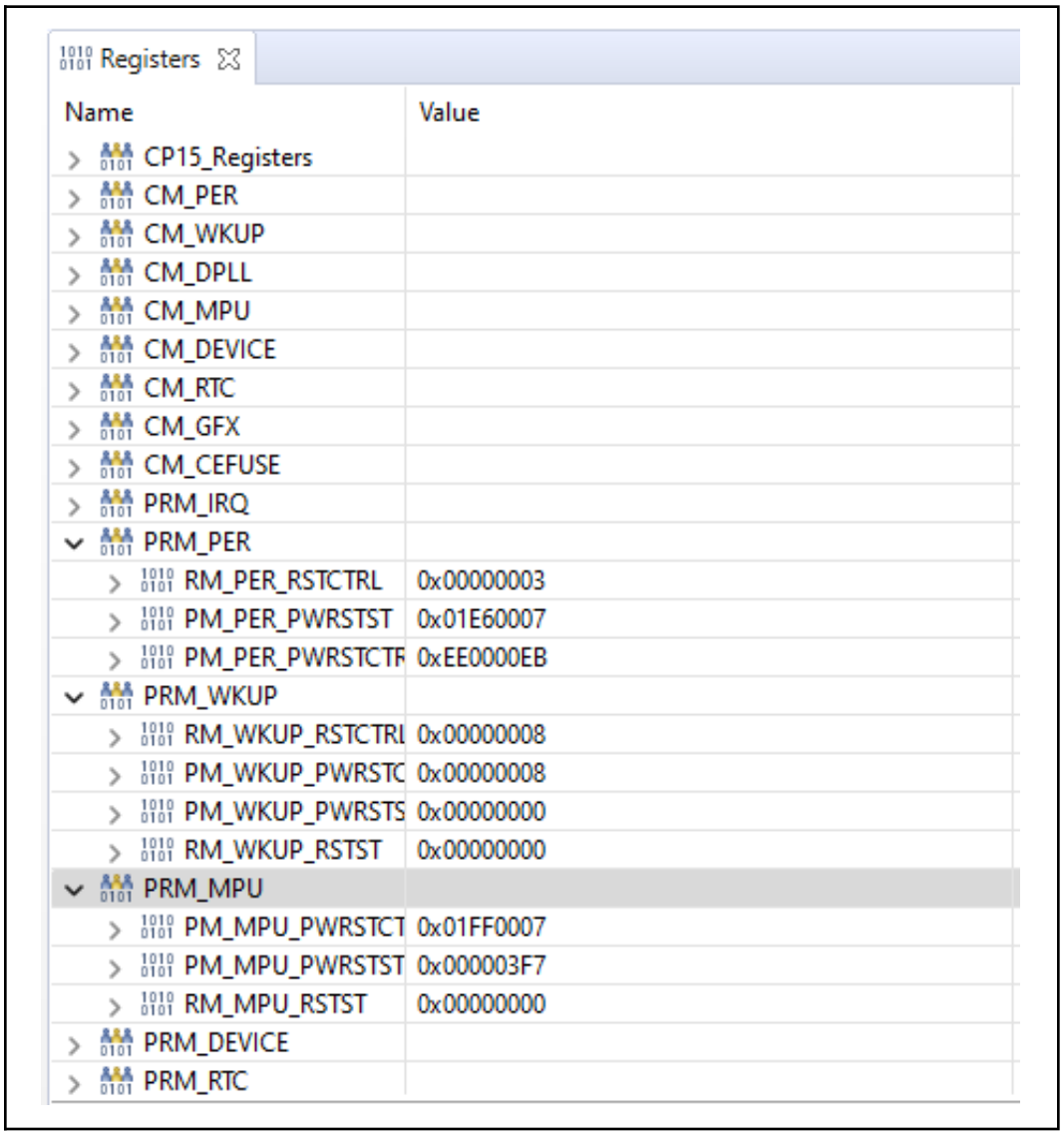
```
@ Wait for interupt
LOOP:
    LDR R3, =SWITCH           @ Load the address of SWITCH into R3
    LDR R2, [R3]              @ Load the value of the memory address pointed to by R3 into R2
    TST R2, #0x1              @ Test if the first bit of R2 is 1
    BEQ ON_LEDS               @ If the first bit of R2 is 0, branch to ON_LEDS
    BNE OFF_LEDS              @ If the first bit of R2 is not 0, branch to OFF_LEDS

TIMER_SVC:
@turn off IRQ request for TIMER
    LDR R0,=0x48042028      @ Load IRQSTATUS_TIMER3 register
    MOV R2,#0x02            @ value to reset Timer Overflow IRQ request
    STR R2,[R0]             @ Write value to turn off IRQ signal

ON_LEDS:
    LDR R1, =0x4804C000       @ Load base address for GPIO1

@ Turn on USR LED_0 & USER LED_1
    ADD R1, R0, #0x194        @ Add the value of R0 and 0x194 and store the result in R1
    MOV R2, #0x00600000       @ Load a value to turn on LED_0 & USER LED_1
    STR R2, [R1]              @ Write to turn on LED_0 & USER LED_1

@ Wait for 1 second
    BL WAIT

@ Turn off USR LED_0 & USER LED_1
    ADD R5, R0, #0x190        @ Add the value of R0 and 0x190 and store the result in R2
    MOV R2, #0x00600000       @ Load a value to turn off  LED_0 & USER LED_1
    STR R2, [R5]              @ Write  to turn off LED_0 & USER LED_1

@ Turn on USR LED_1 & USER LED_2
    ADD R1, R0, #0x194        @ Add the value of R0 and 0x194 and store the result in R1
    MOV R2, #0x00C00000       @ Load a value to turn on LED_1 & USER LED_2
    STR R2, [R1]              @ Write to turn on LED_1 & USER LED_2

@ Wait for 1 second
    BL WAIT

@ Turn off USR LED_1 & USER LED_2
    ADD R5, R0, #0x190        @ Add the value of R0 and 0x190 and store the result in R2
    MOV R2, #0x00C00000       @ Load a value to turn off LED_1 & USER LED_2
    STR R2, [R5]              @ Write  to turn off LED_1 & USER LED_2

@ Turn on USR LED_2 & USER LED_3
    ADD R1, R0, #0x194        @ Add the value of R0 and 0x190 and store the result in R2
    MOV R2, #0x01800000       @ Load a value to turn on LED_2 & USER LED_3
    STR R2, [R1]              @ Write to turn on LED_2 & USER LED_3

@ Wait for 1 second
    BL WAIT

@ Turn off USR LED_2 & USER LED_3
    ADD R5, R0, #0x190        @ Add the value of R0 and 0x194 and store the result in R1
    MOV R2, #0x01800000       @ Load a value to turn off LED_2 & USER LED_3
    STR R2, [R5]              @ Write  to turn off LED_2 & USER LED_3

    B LOOP                    @ Branch to LOOP

OFF_LEDS:
    LDR R0, =0x4804C000       @Base address for GPIO1_CLEARDATAOUT register

        @ Turn off all USR LEDs
        MOV R4, #0x01E00000   @ Load a value to turn off pin 21-24
        ADD R1, R0, #0x190    @ Add the value of R0 and 0x190 and store the result in R1
        STR R4, [R1]          @ Store the value to GPIO1_CLEARDATAOUT register
```

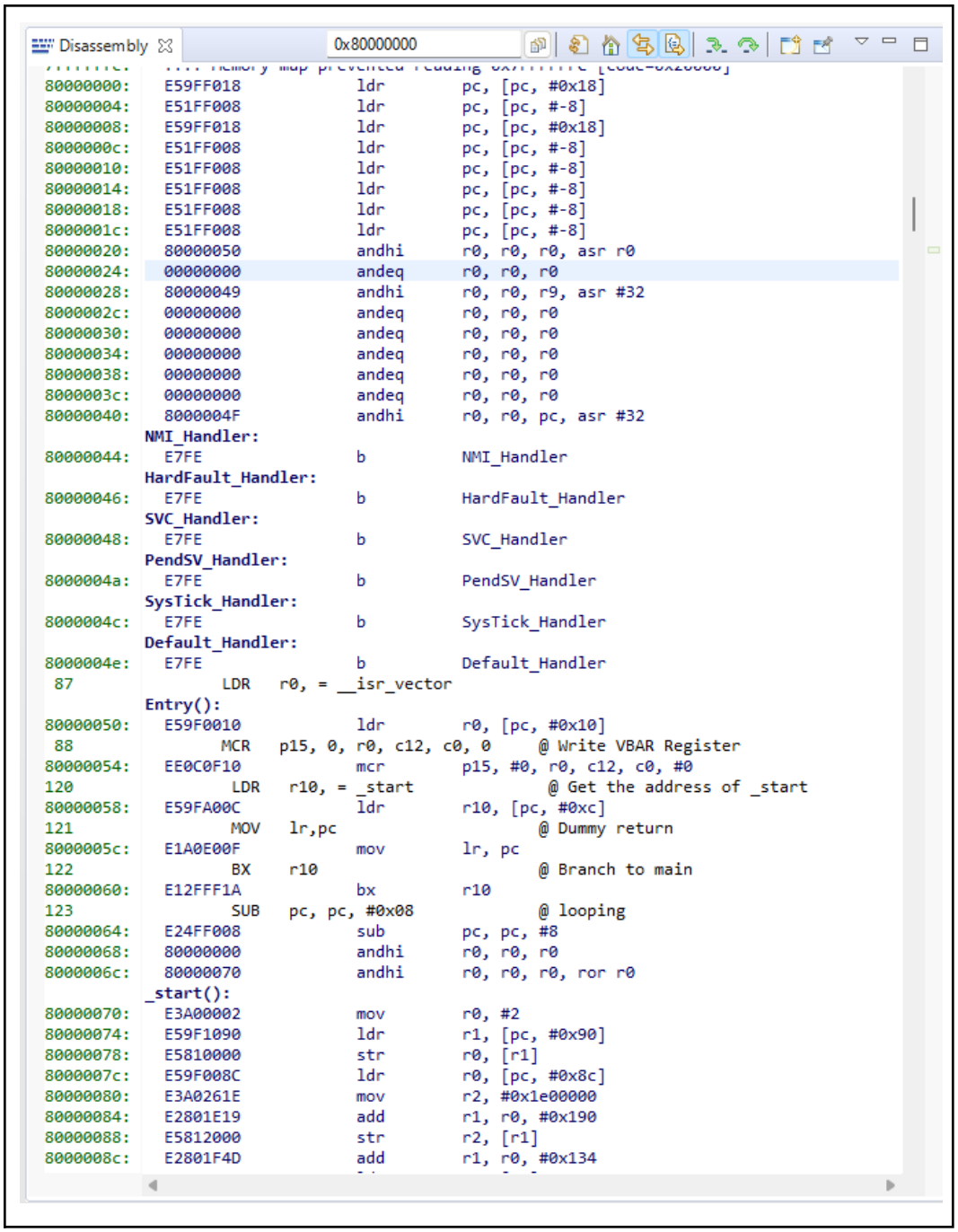
```
97    @ Wait for 1 second
98        BL WAIT
99
100   @ Turn off USR LED_0 & USER LED_1
101       ADD R5, R0, #0x190        @ Add the value of R0 and 0x190 and store the result in R2
102       MOV R2, #0x00600000       @ Load a value to turn off  LED_0 & USER LED_1
103       STR R2, [R5]              @ Write  to turn off LED_0 & USER LED_1
104
105   @ Turn on USR LED_1 & USER LED_2
106       ADD R1, R0, #0x194        @ Add the value of R0 and 0x194 and store the result in R1
107       MOV R2, #0x00C00000       @ Load a value to turn on LED_1 & USER LED_2
108       STR R2, [R1]              @ Write to turn on LED_1 & USER LED_2
109
110   @ Wait for 1 second
111       BL WAIT
112
113   @ Turn off USR LED_1 & USER LED_2
114       ADD R5, R0, #0x190        @ Add the value of R0 and 0x190 and store the result in R2
115       MOV R2, #0x00C00000       @ Load a value to turn off LED_1 & USER LED_2
116       STR R2, [R5]              @ Write  to turn off LED_1 & USER LED_2
117
118   @ Turn on USR LED_2 & USER LED_3
119       ADD R1, R0, #0x194        @ Add the value of R0 and 0x190 and store the result in R2
120       MOV R2, #0x01800000       @ Load a value to turn on LED_2 & USER LED_3
121       STR R2, [R1]              @ Write to turn on LED_2 & USER LED_3
122
123   @ Wait for 1 second
124       BL WAIT
125
126   @ Turn off USR LED_2 & USER LED_3
127       ADD R5, R0, #0x190        @ Add the value of R0 and 0x194 and store the result in R1
128       MOV R2, #0x01800000       @ Load a value to turn off LED_2 & USER LED_3
129       STR R2, [R5]              @ Write  to turn off LED_2 & USER LED_3
130
131       B LOOP                    @ Branch to LOOP
132
133   OFF_LEDS:
134       LDR R0, =0x4804C000       @Base address for GPIO1_CLEARDATAOUT register
135
136           @ Turn off all USR LEDs
137           MOV R4, #0x01E00000   @ Load a value to turn off pin 21-24
138           ADD R1, R0, #0x190    @ Add the value of R0 and 0x190 and store the result in R1
139           STR R4, [R1]          @ Store the value to GPIO1_CLEARDATAOUT register
140
141           B LOOP                @ Branch to LOOP
142
143   WAIT:
144       LDR R10, =SWITCH          @ Load word to program from SWITCH
145       LDR R11,[R10]             @ Load the value into register R11
146       CMP R11, #0x1             @ Compare the value in R11 to the constant value 1
147           BEQ WAIT              @ Branch to the label WAIT if the previous comparison was
148                                 @ true (i.e., if R11 equals 1)
149
150       SUBS R9,R9,#1             @ Subtract the constant value 1 from the value in register
151                                 @ R9 and set the condition flags based on the result
152           BNE WAIT              @ Branch to the label WAIT if the previous subtraction did
153                                 @ not result in R9 being zero
154       MOV R9,#0x00200000
155       MOV PC,LR
156
157   INT_DIRECTOR:
158       STMFD SP!, {R0-R3, R9, LR}     @ Push the contents of registers R0 through R3 and LR onto the stack
159       LDR R0, =0x482000B8       @ Address for INTC_PENDING_IRQ1
160       LDR R1, [R0]              @ Load the content of INTC_PENDING_IRQ1
161       TST R1, #0x00000001       @ Test if the bitwise AND of the content of R1 (Bit 0 = Bit 32)
162       BEQ PASS_ON               @ Branch to the instruction located at the label PASS_ON if the zero flag is set
163       LDR R0, =0x481AC02C       @ Load GPIO_IQRSTATUS_0
164       LDR R1, [R0]              @ Load the content of the memory location pointed to by R0 into R1
165       TST R1, #0x00000002       @ Test if the bitwise AND of the content of R1 (Bit 1 = Bit 32)
166       BNE BUTTON_SVC            @ Branch to the instruction located at the label BUTTON_SVC if the zero flag is clear
167       BEQ PASS_ON               @ Branch to the instruction located at the label PASS_ON if the zero flag is set
```

```
168
169     PASS_ON:
170         LDMFD SP!, {R0-R3, R9, LR}     @ Pop the contents of registers R0 through R3 and LR from the stack
171         SUBS PC, LR, #4           @ Subtract 4 from the content of LR and store the result in PC
172
173     BUTTON_SVC:
174         MOV R1, #0x00000002       @ Value turns off GPI01_14 Interrupt request
175                                   @ Also turns off INTC interrupt request
176         STR R1, [R0]              @ Write to GPIO1_IRQSTATUS_0 register
177
178     @ Turn off NEWIRQA bit in INT_CONTROL, so processor can respond to new IRQ
179         LDR R0, =0x48200048       @ Address of INTC__CONTROL register
180         MOV R1, #01               @ Value to clear bit 0
181         STR R1, [R0]              @ Write to INTC__CONTROL register
182
183         LDR R0,=0x48042038      @Address for Timer TCLR
184         LDR R2,[R0]             @ Start and reload timer
185         TST R2,#0x03            @ Check if timer is running and reload
186         BEQ ON_LEDS             @ if z flag is clear
187         B OFF_LEDS              @ if z flag is set
188
189
190     @ Turn on LED ON GPIO1_12
191         LDR R3, =SWITCH           @ Load the address of the label SWITCH into R3
192         LDR R9, [R3]              @ Load value of SWITCH register
193         CMP R9, #0x0              @ Compare the content of R9 with the hexadecimal value 0x0
194         MOVEQ R9, #0x1            @ If the zero flag is set, move the hexadecimal value 0x1 into R9
195         MOVNE R9, #0x0
196         STR R9, [R3]
197
198     @ Return to wait loop
199         LDMFD SP!, {R0-R3, R9, LR}   @ Restore registers
200         SUBS PC, LR, #4           @ Return from IRQ interrupt procedure
201
202     .align 2
203     SYS_IRQ:        .word 0       @ Location to store systems IRQ address
204     .data
205     .align 2
206     STACK1:         .rept 1024
207                     .word 0x00
208                     .endr
209     STACK2:         .rept 1024
210                     .word 0x00
211                     .endr
212     SWITCH:         .word 0x01
213 .END
```

Figure3: Part3, Using a Programmable Timer to control LED switching times

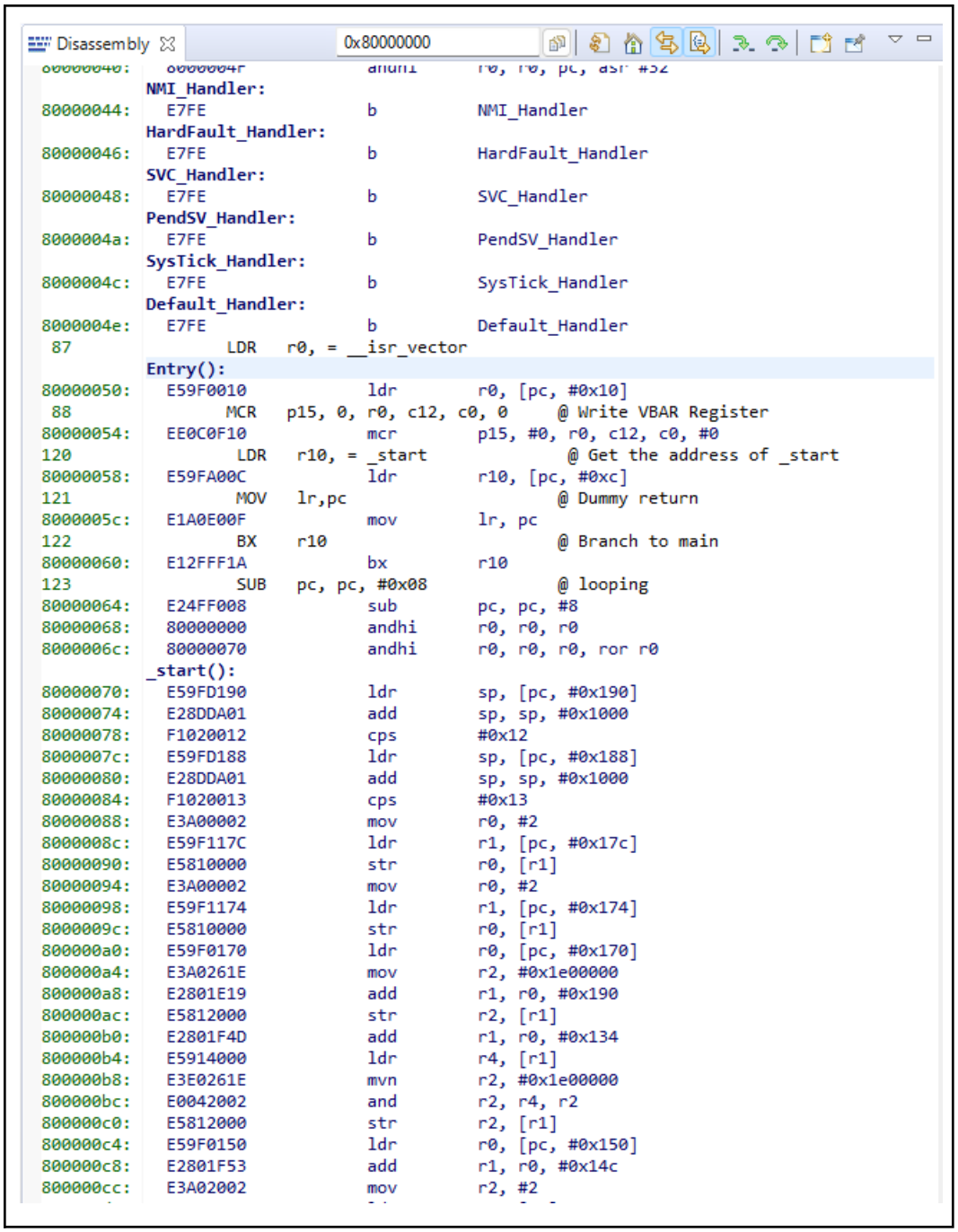
E. Debugger Memory and Registers

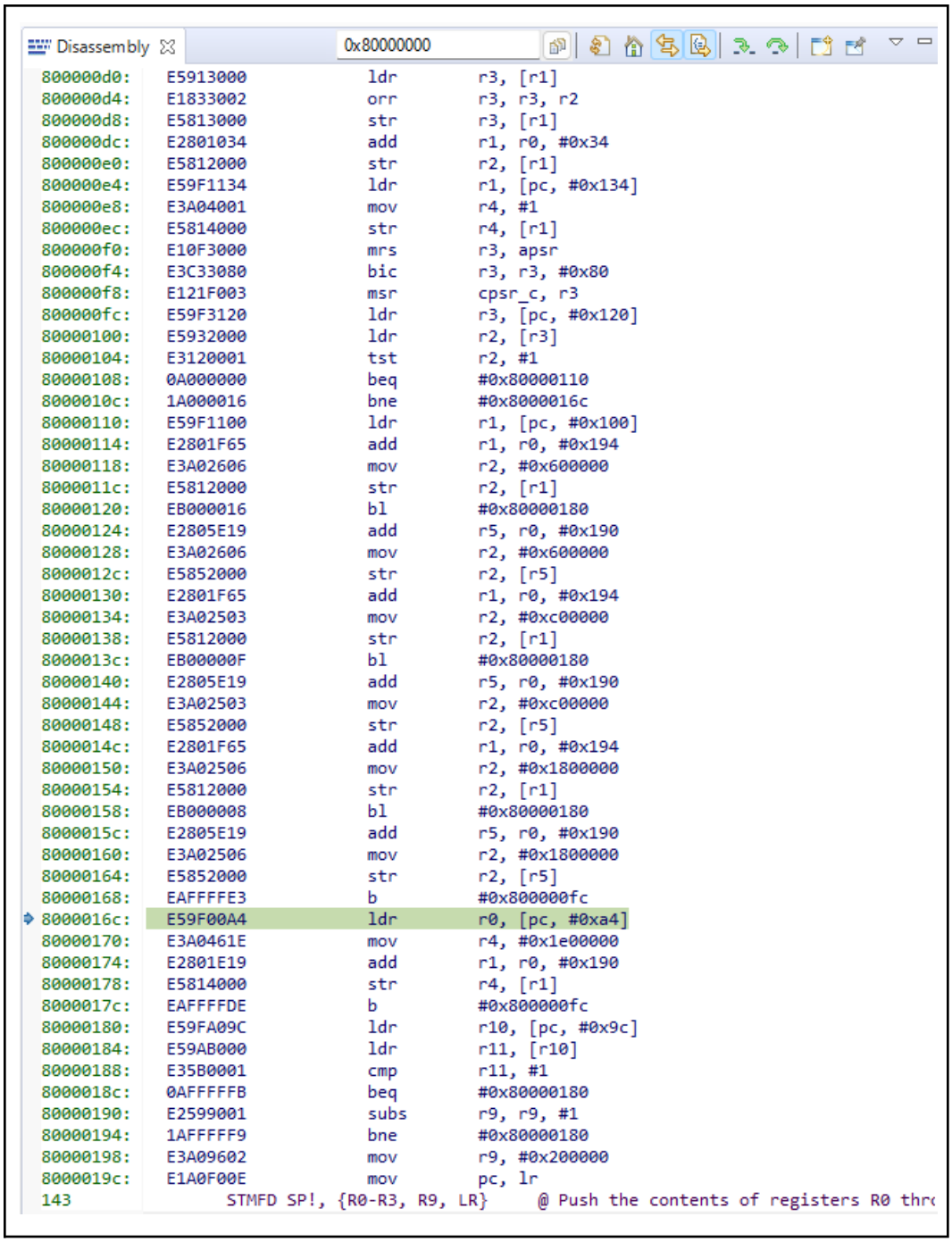


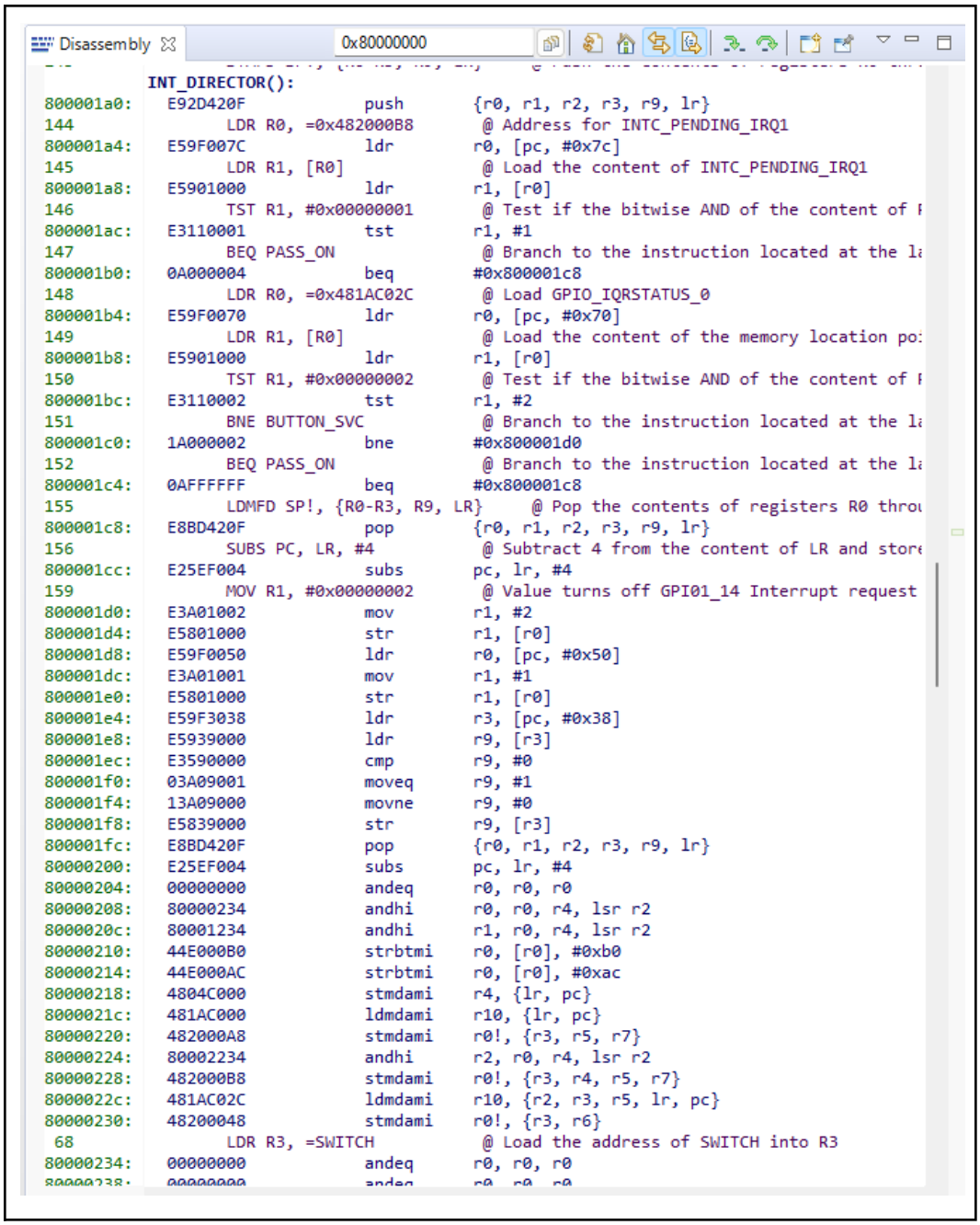

Disassembly
0x80000000
80000000: E59FF018 ldr pc, [pc, #0x18]
80000004: E51FF008 ldr pc, [pc, #-8]
80000008: E59FF018 ldr pc, [pc, #0x18]
8000000c: E51FF008 ldr pc, [pc, #-8]
80000010: E51FF008 ldr pc, [pc, #-8]
80000014: E51FF008 ldr pc, [pc, #-8]
80000018: E51FF008 ldr pc, [pc, #-8]
8000001c: E51FF008 ldr pc, [pc, #-8]
80000020: 80000050 andhi r0, r0, r0, asr r0
80000024: 00000000 andeq r0, r0, r0
80000028: 80000049 andhi r0, r0, r9, asr #32
8000002c: 00000000 andeq r0, r0, r0
80000030: 00000000 andeq r0, r0, r0
80000034: 00000000 andeq r0, r0, r0
80000038: 00000000 andeq r0, r0, r0
8000003c: 00000000 andeq r0, r0, r0
80000040: 8000004F andhi r0, r0, pc, asr #32
NMI_Handler:
80000044: E7FE b NMI_Handler
HardFault_Handler:
80000046: E7FE b HardFault_Handler
SVC_Handler:
80000048: E7FE b SVC_Handler
PendSV_Handler:
8000004a: E7FE b PendSV_Handler
SysTick_Handler:
8000004c: E7FE b SysTick_Handler
Default_Handler:
8000004e: E7FE b Default_Handler
87 LDR r0, = __isr_vector
Entry():
80000050: E59F0010 ldr r0, [pc, #0x10]
88 MCR p15, 0, r0, c12, c0, 0 @ Write VBAR Register
80000054: EE0C0F10 mcr p15, #0, r0, c12, c0, #0
120 LDR r10, = _start @ Get the address of _start
80000058: E59FA00C ldr r10, [pc, #0xc]
121 MOV lr,pc @ Dummy return
8000005c: E1A0E00F mov lr, pc
122 BX r10 @ Branch to main
80000060: E12FFF1A bx r10
123 SUB pc, pc, #0x08 @ looping
80000064: E24FF008 sub pc, pc, #8
80000068: 80000000 andhi r0, r0, r0
8000006c: 80000070 andhi r0, r0, r0, ror r0
_start():
80000070: E3A00002 mov r0, #2
80000074: E59F1090 ldr r1, [pc, #0x90]
80000078: E5810000 str r0, [r1]
8000007c: E59F008C ldr r0, [pc, #0x8c]
80000080: E3A0261E mov r2, #0x1e00000
80000084: E2801E19 add r1, r0, #0x190
80000088: E5812000 str r2, [r1]
8000008c: E2801F4D add r1, r0, #0x134

Registers

Name	Value
Core Registers	
PC	0x80000184
SP	0x80001234
LR	0x80000140
CPSR	0x20000113
N	0
Z	0
C	1
V	0
Q	0
IT_1_0	00
J	0
Reserved	0000
GE	0000
IT_7_2	000000
E	0
A	1
I	0
F	0
T	0
M	10011
R0	0x4804C000
R1	0x4804C194
R2	0x00C00000
R3	0x80002234
R4	0x01E00000
R5	0x4804C190
R6	0x0000550B
R7	0x48040000
R8	0x40FF8000
R9	0x001FD682
R10	0x80002234
R11	0x00000000
R12	0x00000CCC
R13	0x80001234
R14	0x80000140


Disassembly
0x80000000
NMI_Handler:
80000044: E7FE b NMI_Handler
HardFault_Handler:
80000046: E7FE b HardFault_Handler
SVC_Handler:
80000048: E7FE b SVC_Handler
PendSV_Handler:
8000004a: E7FE b PendSV_Handler
SysTick_Handler:
8000004c: E7FE b SysTick_Handler
Default_Handler:
8000004e: E7FE b Default_Handler
87 LDR r0, = __isr_vector
Entry():
80000050: E59F0010 ldr r0, [pc, #0x10]
88 MCR p15, 0, r0, c12, c0, 0 @ Write VBAR Register
80000054: EE0C0F10 mcr p15, #0, r0, c12, c0, #0
120 LDR r10, = _start @ Get the address of _start
80000058: E59FA00C ldr r10, [pc, #0xc]
121 MOV lr,pc @ Dummy return
8000005c: E1A0E00F mov lr, pc
122 BX r10 @ Branch to main
80000060: E12FFF1A bx r10
123 SUB pc, pc, #0x08 @ looping
80000064: E24FF008 sub pc, pc, #8
80000068: 80000000 andhi r0, r0, r0
8000006c: 80000070 andhi r0, r0, r0, ror r0
_start():
80000070: E59FD190 ldr sp, [pc, #0x190]
80000074: E28DDA01 add sp, sp, #0x1000
80000078: F1020012 cps #0x12
8000007c: E59FD188 ldr sp, [pc, #0x188]
80000080: E28DDA01 add sp, sp, #0x1000
80000084: F1020013 cps #0x13
80000088: E3A00002 mov r0, #2
8000008c: E59F117C ldr r1, [pc, #0x17c]
80000090: E5810000 str r0, [r1]
80000094: E3A00002 mov r0, #2
80000098: E59F1174 ldr r1, [pc, #0x174]
8000009c: E5810000 str r0, [r1]
800000a0: E59F0170 ldr r0, [pc, #0x170]
800000a4: E3A0261E mov r2, #0x1e00000
800000a8: E2801E19 add r1, r0, #0x190
800000ac: E5812000 str r2, [r1]
800000b0: E2801F4D add r1, r0, #0x134
800000b4: E5914000 ldr r4, [r1]
800000b8: E3E0261E mvn r2, #0x1e00000
800000bc: E0042002 and r2, r4, r2
800000c0: E5812000 str r2, [r1]
800000c4: E59F0150 ldr r0, [pc, #0x150]
800000c8: E2801F53 add r1, r0, #0x14c
800000cc: E3A02002 mov r2, #2


Disassembly
0x80000000
800000d0: E5913000 ldr r3, [r1]
800000d4: E1833002 orr r3, r3, r2
800000d8: E5813000 str r3, [r1]
800000dc: E2801034 add r1, r0, #0x34
800000e0: E5812000 str r2, [r1]
800000e4: E59F1134 ldr r1, [pc, #0x134]
800000e8: E3A04001 mov r4, #1
800000ec: E5814000 str r4, [r1]
800000f0: E10F3000 mrs r3, apsr
800000f4: E3C33080 bic r3, r3, #0x80
800000f8: E121F003 msr cpsr_c, r3
800000fc: E59F3120 ldr r3, [pc, #0x120]
80000100: E5932000 ldr r2, [r3]
80000104: E3120001 tst r2, #1
80000108: 0A000000 beq #0x80000110
8000010c: 1A000016 bne #0x8000016c
80000110: E59F1100 ldr r1, [pc, #0x100]
80000114: E2801F65 add r1, r0, #0x194
80000118: E3A02606 mov r2, #0x600000
8000011c: E5812000 str r2, [r1]
80000120: EB000016 bl #0x80000180
80000124: E2805E19 add r5, r0, #0x190
80000128: E3A02606 mov r2, #0x600000
8000012c: E5852000 str r2, [r5]
80000130: E2801F65 add r1, r0, #0x194
80000134: E3A02503 mov r2, #0xc00000
80000138: E5812000 str r2, [r1]
8000013c: EB00000F bl #0x80000180
80000140: E2805E19 add r5, r0, #0x190
80000144: E3A02503 mov r2, #0xc00000
80000148: E5852000 str r2, [r5]
8000014c: E2801F65 add r1, r0, #0x194
80000150: E3A02506 mov r2, #0x1800000
80000154: E5812000 str r2, [r1]
80000158: EB000008 bl #0x80000180
8000015c: E2805E19 add r5, r0, #0x190
80000160: E3A02506 mov r2, #0x1800000
80000164: E5852000 str r2, [r5]
80000168: EAFFFFE3 b #0x800000fc
8000016c: E59F00A4 ldr r0, [pc, #0xa4]
80000170: E3A0461E mov r4, #0x1e00000
80000174: E2801E19 add r1, r0, #0x190
80000178: E5814000 str r4, [r1]
8000017c: EAFFFFDE b #0x800000fc
80000180: E59FA09C ldr r10, [pc, #0x9c]
80000184: E59AB000 ldr r11, [r10]
80000188: E35B0001 cmp r11, #1
8000018c: 0AFFFFFB beq #0x80000180
80000190: E2599001 subs r9, r9, #1
80000194: 1AFFFFF9 bne #0x80000180
80000198: E3A09602 mov r9, #0x200000
8000019c: E1A0F00E mov pc, lr
143 STMFD SP!, {R0-R3, R9, LR} @ Push the contents of registers R0 thr



F. Statement of Originality

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 provide help to anyone else.

Conclusion

In conclusion, this report has detailed the successful development of a program that controls the lighting of four USR LEDs on the BeagleBone Black microprocessor. The project was divided into three parts, each building on the previous one, with the final outcome being a practical demonstration of the application of assembly language programming and embedded systems concepts. The step-by-step guide provided in this report, along with the explanations of programming concepts and techniques used, serve as a useful resource for those interested in developing similar projects.

Appendix. A: Design specifications and requirements

ECE371 – DESIGN PROJECT #2, WINTER 2023

Electrical and Computer Engineering Department
Portland State University
Portland, OR 97207

REFERENCES

Hall Text Chapter 4 and Chapter 5; The BeagleBone Black System C.1 Reference Manual on http://elinux.org/Beagleboard:BeagleBoneBlack; and the TI Sitara AM 335X manual at www.ti.com/lit/ug/spruh73p/spruh73p.pdf

IMPORTANT FIRST STEP

As with any design project, it is very important to read through the entire project description to get an overview before starting to work through the project. Make notes on the requirements, deliverables, and timelines, if given, etc. These notes are the start of your design logbook.

INTRODUCTION

You will be developing this project for one of the BeagleBone Black boards connected to stations on the west end of the Tek Lab and in the Intel Lab. There are three major parts to this design project:

1. In the first part you will learn is to learn how to control GPIO pins and how to turn the 4 BeagleBone Black USR LEDs on and off in a specified pattern with delay loop timing.
2. In the second part of this project, you will develop an interrupt procedure that services an interrupt request from a push-button switch connected to GPIO2_1. The first time the button is pushed, the interrupt procedure will start the LED display pulsing. The next time the button is pushed, the interrupt procedure will stop the pulsing LED display.
3. In the third part you will use a Sitara AM3358 Timer on an interrupt basis to time the LED lighting, instead of using a delay loop

GOALS AND OBJECTIVES

The major goals of this exercise are:

1. To give you some practice following the "Fast is Slow" rule, after you use up the "5-minute rule". In fact this exercise is specifically designed to be relatively easy, if you follow the "Fast is Slow" rule but, due to the amount of detail, very difficult and prone to many errors, if you try to just speed through it without carefully studying all the reference material and developing detailed task lists as described in the text.
2. To give you some practice in working through and using text examples and data sheets to quickly and efficiently develop programs that work with GPIO pins, an interrupt controller and a timer.
3. To show you how to successfully develop a multi-part program by systematically building and testing one piece at a time.

The specific objectives of this exercise are that at the end of this exercise you should be able to:

1. Output highs or lows on GPIO pins.
2. Initialize a GPIO pin as an input or as an output
3. Set up a "flag" that can be used to switch back and forth between two different actions.
4. Utilize AM3358 GPIO pins as interrupt inputs.
5. Initialize and use an AM 335X Interrupt Controller.
6. Write an interrupt service procedure that services an IRQ interrupt request.
7. Use a programmable timer to generate interrupts at desired time intervals.

DELIVERABLES

A. An **as-you-go** design log that shows your thinking, all the steps you took, and results at each step. **Make sure to create quality documentation that you would like to receive, if you were assigned to take over a project halfway through it and successfully complete the project.**
B. A clearly written algorithm for each of the development sections of your program,:for the final LED pulsing program, for the final button driven LED program, and for the final version that uses the button interrupts, and a timer instead of a delay loop. Note that you need high level and low-level algorithms/tasklists for each section
C. A signed off copy of the .s file for the button activated flasher and a signed off copy of the .s file for the timer LED controlled LED flasher. Note that your .s files should have full headers and comments. (Be prepared to answer questions at signoff.)
D. A written and signed statement that, "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." (Any evidence of joint work will result in project grades of zeros for all parties involved.)

Grading rubric is as follows:

GRADING KEY FOR ECE 371 DESIGN PROJECTS WINTER 2020

LOG (DETAILED AND "AS YOU GO" 20 MAX _____

TASK LISTS/ALGORITHMS
(DETAILED AND COMPLETE) 20 MAX _____

WORKING PROGRAMS (Demos required) 40 MAX _____
COMMENTS (CLEAR AND USEFUL) 10 MAX _____
OVERALL ORGANIZATION 10 MAX _____
TOTAL 100 MAX _____

PROCEDURE OVERVIEW

Part 1 Lighting LEDs

1. Study the BeagleBone Black System Manual to determine which GPIO pins are connected to the 4 USR LEDs and the logic level required to turn on one of the LEDs. In the manual you need to look under Hardware Files, Latest Production files for C Version, and System Reference Manual. Skim through the User Manual until you find the GPIO connection for the 4 User LEDs and the logic level required to turn on an LED.
2. Write the **high level algorithm** for a program that lights LED0 and LED 1 for 1 second, then switches off LED 0 and Led 1 and lights LED 1 and LED 2 for one second. Finally, it switches off LED 1 and LED 2 and turns on LED 2 and LED 3 for one second. Set this pattern to repeat.
3. Carefully work through the section of Hall Chapter 4 that describes the AM3358 GPIO pins and the memory mapped registers that control them. As part of this, determine the registers that control the GPIO pins connected to the LEDs. Also study the text section that shows how to set up bit templates for working with the GPIO registers.
4. Set up the templates needed for the GPIO pins you are using.
5. Determine values and addresses you need to output a high or output a low on a GPIO pin.
6. Determine the RMW sequence of instructions and addresses required to program the GPIO pins for the LEDs as outputs and initialize the LEDs in the off state.
7. Develop the **low level algorithm**, as shown in chapter 4, for your program, including the delay loop
8. Write and carefully check the assembly language program.
9. Build, Load, Run, and Debug the program. Note that to single step through the program easily you can initially use a very small delay constant, so you don't have to single step forever to get from one LED to the next.
10. When your program works, make an electronic copy that you will use for developing the next part of the project. Key point is that once you get something working, you save it and use a copy to develop the next stage. That way you always have a working program section to go back to.

Part 2 Creating an Interrupt Procedure for Servicing a Button Push

1. Carefully read through Hall Chapter 5 and work through the development of the button service program. Your log should parallel the development steps described in the text section for the example program. Develop an initialization list, such as those in the text, with the values needed for this project where the pushbutton is on GPIO2_1. (Show all thinking, labeled templates, etc.)
2. Use this detailed initialization list to modify the required steps of the assembly language program in Figure 5-14 as needed for this project. (Note that, as discussed in the text, you have to modify the startup file to intercept

the system IRQ response, instead of using the usual method of hooking the interrupt vector, due to the way the BeagleBone Black is set up.)

3. Write an algorithm for the Button Service procedure that will start the LED strobing pattern the first time it is called and turn the LED strobing pattern off the next time it is called. In general, one way to do this is to set aside a memory location that you toggle back and forth to keep track of whether the LEDs are pulsing or not. You then use this to determine whether to start or stop the pulsing when a button press produces an IRQ interrupt request.
4. Write the assembly language program.
5. To test the program, set a breakpoint at the start of the INT_DIRECTOR procedure and run the program. If all is well, execution should go to the breakpoint when the button is pushed. If execution doesn't make it to the breakpoint, mentally work through your code very carefully again to make sure all the bits in the control words, etc. are initialized as they should be. Your log should be helpful in doing this.
6. When execution gets to the INT_DIRECTOR procedure correctly, set a breakpoint at the start of the BUTTON_SVC procedure and step execution to that point.
7. If execution gets to the start of BUTTON_SVC correctly, then you can step through the rest of the program and back to the wait loop to wait for the next button push.
8. When this is all working, save it and make a copy that you will use for developing the next addition to the program.
9. Take a break and give a couple of cheers!

Part 3 Using a Programmable Timer to control LED switching times

For this part of your program, you will use interrupts from Timer 4 to determine when to switch from one LED to the next, instead of using a delay loop. The Timer 2 program discussed in text chapter 5 shows you how to set up a timer to produce interrupts at desired time intervals. In ECE 371 Design Project #2 part 1, you learned how to use a dedicated memory location to keep track of whether an LED is on or off. For this program you need to figure out some simple way to keep track of which of the 4 LEDs is on when a timer interrupt occurs and take appropriate action.

The timer section of Chapter 5 tells you almost everything you need to add the timer capability to your button program, if you study it VERY carefully. Of course, you are using Timer 4 instead of Timer 2, so you have to modify those parts as needed. You also have to integrate your rotating LED actions into the Program.

Signoff

1. Demonstrate the basic LED rotate with that is started and stopped with a button push. Be prepared to answer a question or 2 from Luis and her sign, if successful. Likewise, demonstrate your final, button-push/timer program to Luis or Instructor

for signoff on .s file. Be prepared to answer questions about the details of this program.

2. Congratulate yourself. You have successfully created a very real program that uses multiple interrupts and a timer as is done in almost every embedded system.

3. Turn in a hard copy of your documentation for the project to Luis or Jonathan. You do not need to put it in a fancy binder. Just fasten all the pages together in some way so they do not become separated during grading, etc. Projects will be available in first week of ECE 372 classes or they can be picked up from my office during winter quarter.