CPE 325: Intro to Embedded Computer System

Lab11

Software Reverse Engineering

Submitted by: Gianna Foti

Date of Experiment:_____7 November 2024_____

Report Deadline:_____12 November 2024_____

Lab Section: _____CPE353-02_____

Demonstration Deadline: _____12 November 2024_____

Introduction

In this lab, I worked with a microcontroller (MSP430) to explore file formats, programming, and reverse engineering. I started by converting the `.out` file from my Lab 7 PWM program into a HEX file and analyzed it using various tools. I then flashed the HEX file to the microcontroller and verified its functionality with the MSP430Flasher tool. Additionally, I reverse-engineered the HEX file back to assembly code and documented the program's behavior. The goal was to deepen my understanding of embedded systems, from code conversion to flashing and debugging.

Theory Topics

1. ELF File Components
    a. In MSP430 development, the Executable and Linkable Format (ELF) file organizes program components into specific sections for efficient loading and execution. The ELF file begins with a header, detailing the architecture and entry point, followed by a program header table mapping sections to memory. Key sections include `.text` for executable code, `.data` for initialized variables, and `.bss` for uninitialized data. Read-only data resides in `.rodata`, while `.stack` and `.heap` manage runtime memory for function calls and dynamic allocations. Additionally, the symbol table aids in debugging, providing addresses for functions and variables. This structure optimizes memory use, critical in the resource-constrained MSP430 environment.
2. Naken Utility:
    a. The Naken Utility is a lightweight tool for working with assembly code on embedded systems, including the MSP430. It offers features like assembling, disassembling, and simulating code, allowing developers to write, test, and debug MSP430 assembly programs efficiently. With Naken Utility, developers can convert assembly code into machine code and vice versa, view memory contents, and test code execution within a

simulator, helping to catch errors early in the development cycle. Its simplicity and focused functionality make it valuable for embedded programming, especially for optimizing code and managing low-level memory operations on devices like the MSP430.

3. MSP430 Flasher
    a. The MSP430 Flasher is a command-line tool provided by Texas Instruments for flashing firmware onto MSP430 microcontrollers. It supports programming, erasing, and verifying the device memory, making it ideal for development and production environments. The utility allows users to interface with the MSP430 through various communication options, such as JTAG and Spy-Bi-Wire, ensuring compatibility across MSP430 devices. Its lightweight design and scripting capabilities streamline the process of updating firmware and automating tasks, making MSP430 Flasher a versatile tool for embedded developers working with MSP430 systems.

Lab 11 Questions:

1. What insights can you glean from your analysis?
    a. The ELF commands produced by the TI compiler differ from those generated by the GNU compiler, leading to variations in the resulting .out files. Although I got similar outputs from both compilers, there were differences in the beginning of the section headers, flags, number of section headers, section header string table index, and the magic number. However, executing the commands outlined in the tutorial gave similar results regardless of whether I used the TI or GCC compiler.
2. Can you find what symbol is associated with that address?
    a. Converting the subroutine number 12914 to hexadecimal gets 0x3272. According to the symbol table, the associated symbol with this address is `memset`.
    b. 

Part 1:

Program Description:

Created a HEX file using the out file.

Program Output:

@4400

0F 12 0E 12 E2 B3 01 02 2C 20 E2 B3 00 02 29 20

82 93 04 24 06 20 92 43 04 24 82 43 06 24 92 43

08 24 92 53 06 24 B2 90 5E 00 06 24 29 38 0F 43

0E 43 82 93 08 24 01 24 1E 43 0E 93 01 20 1F 43

82 4F 08 24 82 93 08 24 07 20 1F 42 00 24 0F 5F

92 4F 34 47 56 03 02 3C 82 43 56 03 82 43 06 24

0F 3C 92 93 04 24 0C 20 82 43 04 24 82 43 06 24

92 43 08 24 1F 42 00 24 0F 5F 92 4F 34 47 56 03

3E 41 3F 41 00 13 0A 12 09 12 08 12 0A 4C 78 4A

09 43 11 3C 0E 4D 0E 8B 1E 83 1D 53 FD 4E FF FF

1F 83 FB 23 03 3C 1D 53 FD 4A FF FF 12 C3 08 10

19 53 39 92 EC 37 18 B3 F6 23 7B 4A 7F 4A 0C 4B

B0 12 C0 46 0B 4C 0C 4F B0 12 04 46 3C F0 0F 00

0B DC 3F F0 0F 00 3F 50 03 00 3F 90 12 00 0C 20

7E 4A 3E B0 80 00 07 24 7C 4A 4C 4C B0 12 BA 46

3E F0 7F 00 0E DC 0F 5E 3B 90 FF 0F CB 23 30 40

12 47 B2 40 80 5A 5C 01 F2 D2 04 02 F2 D2 0A 02

E2 C3 05 02 E2 D3 07 02 E2 D3 03 02 E2 D3 1B 02

E2 D3 19 02 E2 C3 1D 02 E2 C3 04 02 E2 D3 06 02

E2 D3 02 02 E2 D3 1A 02 E2 D3 18 02 E2 C3 1C 02

B2 40 1C 5A 5C 01 92 D3 00 01 B2 40 E7 03 52 03

B2 40 E0 00 46 03 1F 42 00 24 0F 5F 92 4F 34 47

56 03 B2 40 10 02 40 03 32 D2 32 D0 18 00 0C 43

30 41 0A 12 09 12 08 12 19 42 5C 01 B2 40 80 5A

5C 01 3F 40 58 47 3F 90 5C 47 16 24 3F 40 5C 47

3F 90 60 47 11 24 3A 40 60 47 3A 80 5C 47 0A 11

0A 11 38 40 5C 47 3C 48 7F 4C 0F 5F 1F 4F 58 47

3D 48 8F 12 1A 83 F7 23 79 C2 39 D0 08 5A 82 49

5C 01 B0 12 32 47 30 40 12 47 3D F0 0F 00 3D E0

0F 00 0D 5D 0D 5D 00 5D 12 C3 0C 10 12 C3 0C 10

12 C3 0C 10 12 C3 0C 10 12 C3 0C 10 12 C3 0C 10

12 C3 0C 10 12 C3 0C 10 12 C3 0C 10 12 C3 0C 10

12 C3 0C 10 12 C3 0C 10 12 C3 0C 10 12 C3 0C 10

12 C3 0C 10 30 41 0F 12 0E 12 0D 12 0C 12 0B 12

E2 B3 1C 02 12 24 B2 40 A8 61 02 24 82 93 02 24

05 24 92 83 02 24 82 93 02 24 FB 23 82 93 00 24

04 24 92 83 00 24 B0 12 1A 47 E2 C3 1C 02 3B 41

3C 41 3D 41 3E 41 3F 41 00 13 0F 12 0E 12 0D 12

0C 12 0B 12 E2 B3 1D 02 12 24 B2 40 A8 61 02 24

82 93 02 24 05 24 92 83 02 24 82 93 02 24 FB 23

B2 92 00 24 04 2C 92 53 00 24 B0 12 1A 47 E2 C3

1D 02 3B 41 3C 41 3D 41 3E 41 3F 41 00 13 3D F0

0F 00 3D E0 0F 00 0D 5D 00 5D 0C 5C 0C 5C 0C 5C

0C 5C 0C 5C 0C 5C 0C 5C 0C 5C 0C 5C 0C 5C 0C 5C

0C 5C 0C 5C 0C 5C 0C 5C 30 41 31 40 00 44 B0 12

2E 47 0C 93 02 24 B0 12 72 45 0C 43 B0 12 02 45

1C 43 B0 12 28 47 0F 4C 0C 4D 3D 40 03 00 0D 5F

1E 4F 01 00 30 40 F8 46 0E 93 06 24 0F 4C 1F 53

FF 4D FF FF 1E 83 FB 23 30 41 34 41 35 41 36 41

37 41 38 41 39 41 3A 41 30 41 1F 42 00 24 0F 5F

92 4F 34 47 56 03 30 41 03 43 FF 3F 03 43 1C 43

30 41 30 41 00 00 7D 00 FA 00 77 01 F4 01 71 02

EE 02 6B 03 E8 03 32 D0 10 00 FD 3F 03 43 00 1B

04 00 00 03 01 00 FF F0 86 44 E6 46 4E 47 00 24

@ffd2

46 47 5A 46 46 47 46 47 46 47 46 47 16 46 46 47

46 47 46 47 46 47 46 47 46 47 46 47 46 47 46 47

00 44 46 47 46 47 46 47 46 47 46 47 CA 46

q

Part 2:

Question Description:

From the same .out file from Q2, find the following relevant information. What tool did you use? Take a screenshot and put it in your report.

Questions:

1. What is the magic number used?
    a. 7f 45 4c 46 01 01 01 ff 00 00 00 00 00 00 00 00
2. What is the class of this .out file?
    a. ELF32
3. What machine was this file built for?
    a. MSP430 microcontroller by Texas Instrument
4. What is the size of the header?

a. 52 bytes

5. How many section headers are there? Please verify. You may need to run the command again.

   a. 25

```
C:\Users\jkpx1\workspace_v12\Lab11_G_LTS\Debug>cd C:\Users\jkpx1\workspace_v12\Lab11_G_GNU\Debug

C:\Users\jkpx1\workspace_v12\Lab11_G_GNU\Debug>msp430-elf-readelf -h Lab11_G_GNU.out
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 ff 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            Standalone App
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Texas Instruments msp430 microcontroller
  Version:                           0x1
  Entry point address:               0x4416
  Start of program headers:          52 (bytes into file)
  Start of section headers:          30148 (bytes into file)
  Flags:                             0x2d: architecture variant: MSP430X
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         5
  Size of section headers:           40 (bytes)
  Number of section headers:         25
  Section header string table index: 24
```

Part 3:

Question Description:

Use the HEX file you generated in Q.2 to do the followings:

1. Program the given hex file to your microcontroller using the MSP430Flasher tool and paste the output in your report.

a.

```
C:\Users\jkpx1\workspace_v12\Lab11_G_LTS\Debug>MSP430Flasher.exe -n MSP430F5529 -w Lab11_G_LTS.txt -v -z [Vcc]
* -----/|---------------------------------------------------------------- *
*     / |__                                                               *
*    /_   /   MSP Flasher v1.3.20                                         *
*      | /                                                                *
* -----|/---------------------------------------------------------------- *
*
* Evaluating triggers...done
* Checking for available FET debuggers:
* Found USB FET @ COM12 <- Selected
* Initializing interface @ COM12...done
* Checking firmware compatibility:
* FET firmware is up to date.
* Reading FW version...
* Debugger does not support target voltages other than 3000 mV!
* Setting VCC to 3000 mV...done
* Accessing device...done
* Reading device information...done
* Loading file into device...done
* Verifying memory (Lab11_G_LTS.txt)...done
*
* ------------------------------------------------------------------------
* Arguments   : -n MSP430F5529 -w Lab11_G_LTS.txt -v -z [Vcc]
* ------------------------------------------------------------------------
* Driver      : loaded
* Dll Version : 31400000
* FwVersion   : 31200000
* Interface   : TIUSB
* HwVersion   : E 3.0
* JTAG Mode   : AUTO
* Device      : MSP430F5529
* EEM         : Level 7, ClockCntrl 2
* Erase Mode  : ERASE_ALL
* Prog.File   : Lab11_G_LTS.txt
* Verified    : TRUE
* BSL Unlock  : FALSE
* InfoA Access: FALSE
* VCC ON      : 3000 mV
* ------------------------------------------------------------------------
* Starting target code execution...done
* Disconnecting from device...done
*
* ------------------------------------------------------------------------
* Driver      : closed (No error)
* ------------------------------------------------------------------------
*/
```

2. Show that using the Flasher you can change the Brightness Level as you could in the CCS environment.
    a. It does indeed adjust the brightness of LED1. I showed this during my demo.
3. Using the naken utility and the steps shown in Section 5.2 of the tutorial, reverse engineer the hex file to assembly code.
4. Comment on each line of the assembly code generated from Q4c above to describe what each line is doing. Try to identify delay loops, switch inputs, LED outputs
5. Describe what the program is doing in a neat flowchart.

naken_util - by Michael Kohn

Joe Davisson

Web: https://www.mikekohn.net/

Email: mike@mikekohn.net

Version:

Loaded Lab11_G_LTS.txt of type ti_txt / msp430 from 0x4400 to 0xffff

```
Addr    Opcode Instruction                  Comment                                          Cycles
------- ------ ---------------------------------- ------------------------------------------------ ------
0x4400: 0x120F push.w r15                   ; Save r15 on stack                              ; 3
0x4402: 0x120E push.w r14                   ; Save r14 on stack                              ; 3
0x4404: 0xB3E2 bit.b #2, &0x0201            ; Test bit 1 of P2IFG (P2 Interrupt Flag)        ; 4
0x4406: 0x0201                              ; (Operand for previous instruction)
0x4408: 0x202C jne 0x4462  (offset: 88)     ; If Z=0 (bit set), jump to 0x4462               ; 2
0x440A: 0xB3E2 bit.b #2, &0x0200            ; Test bit 1 of P2IE (P2 Interrupt Enable)       ; 4
0x440C: 0x0200                              ; (Operand for previous instruction)
0x440E: 0x2029 jne 0x4462  (offset: 82)     ; If Z=0 (bit set), jump to 0x4462               ; 2
0x4410: 0x9382 cmp.w #0, &0x2404            ; Compare &0x2404 with 0                         ; 4
0x4412: 0x2404                              ; (Operand for previous instruction)
0x4414: 0x2006 jne 0x4422  (offset: 12)     ; If not equal, jump to 0x4422                   ; 2
0x4416: 0x4392 mov.w #1, &0x2404            ; Set &0x2404 to 1                               ; 4
0x4418: 0x2404                              ; (Operand for previous instruction)
0x441A: 0x4382 mov.w #0, &0x2406            ; Clear &0x2406                                  ; 4
0x441C: 0x2406                              ; (Operand for previous instruction)
0x441E: 0x4392 mov.w #1, &0x2408            ; Set &0x2408 to 1                               ; 4
```

```
0x4420: 0x2408                                  ; (Operand for previous instruction)
0x4422: 0x5392 add.w #1, &0x2406                ; Increment &0x2406 by 1                    ; 4
0x4424: 0x2406                                  ; (Operand for previous instruction)
0x4426: 0x90B2 cmp.w #0x005E, &0x2406           ; Compare &0x2406 with 94                   ; 5
0x4428: 0x005E                                  ; (Immediate value 94)
0x442A: 0x2406                                  ; (Operand for previous instruction)
0x442C: 0x3829 jl 0x4480  (offset: 82)          ; If less, jump to 0x4480                   ; 2
0x442E: 0x430F mov.w #0, r15                    ; Clear r15                                 ; 1
0x4430: 0x430E mov.w #0, r14                    ; Clear r14                                 ; 1
0x4432: 0x9382 cmp.w #0, &0x2408                ; Compare &0x2408 with 0                    ; 4
0x4434: 0x2408                                  ; (Operand for previous instruction)
0x4436: 0x2401 jeq 0x443A  (offset: 2)          ; If equal, jump to 0x443A                  ; 2
0x4438: 0x431E mov.w #1, r14                    ; Set r14 to 1                              ; 1
0x443A: 0x930E cmp.w #0, r14                    ; Compare r14 with 0                        ; 1
0x443C: 0x2001 jne 0x4440  (offset: 2)          ; If not equal, jump to 0x4440              ; 2
0x443E: 0x431F mov.w #1, r15                    ; Set r15 to 1                              ; 1
0x4440: 0x4F82 mov.w r15, &0x2408               ; Move r15 to &0x2408                       ; 4
0x4442: 0x2408                                  ; (Operand for previous instruction)
0x4444: 0x9382 cmp.w #0, &0x2408                ; Compare &0x2408 with 0                    ; 4
0x4446: 0x2408                                  ; (Operand for previous instruction)
0x4448: 0x2007 jne 0x4458  (offset: 14)         ; If not equal, jump to 0x4458              ; 2
0x444A: 0x421F mov.w &0x2400, r15               ; Load &0x2400 into r15                     ; 3
0x444C: 0x2400                                  ; (Operand for previous instruction)
0x444E: 0x5F0F add.w r15, r15                   ; Double r15 (r15 = r15 + r15)              ; 1
0x4450: 0x4F92 mov.w 18228(r15), &0x0356        ; Move word from table to &0x0356           ; 6
0x4452: 0x4734                                  ; (Table base address)
```

```
0x4454: 0x0356                                  ; (Destination address)
0x4456: 0x3C02 jmp 0x445C  (offset: 4)          ; Jump to 0x445C                    ; 2
0x4458: 0x4382 mov.w #0, &0x0356                ; Clear &0x0356                     ; 4
0x445A: 0x0356                                  ; (Operand for previous instruction)
0x445C: 0x4382 mov.w #0, &0x2406                ; Clear &0x2406                     ; 4
0x445E: 0x2406                                  ; (Operand for previous instruction)
0x4460: 0x3C0F jmp 0x4480  (offset: 30)         ; Jump to 0x4480                    ; 2
0x4462: 0x9392 cmp.w #1, &0x2404                ; Compare &0x2404 with 1            ; 4
0x4464: 0x2404                                  ; (Operand for previous instruction)
0x4466: 0x200C jne 0x4480  (offset: 24)         ; If not equal, jump to 0x4480      ; 2
0x4468: 0x4382 mov.w #0, &0x2404                ; Clear &0x2404                     ; 4
0x446A: 0x2404                                  ; (Operand for previous instruction)
0x446C: 0x4382 mov.w #0, &0x2406                ; Clear &0x2406                     ; 4
0x446E: 0x2406                                  ; (Operand for previous instruction)
0x4470: 0x4392 mov.w #1, &0x2408                ; Set &0x2408 to 1                  ; 4
0x4472: 0x2408                                  ; (Operand for previous instruction)
0x4474: 0x421F mov.w &0x2400, r15               ; Load &0x2400 into r15             ; 3
0x4476: 0x2400                                  ; (Operand for previous instruction)
0x4478: 0x5F0F add.w r15, r15                   ; Double r15                        ; 1
0x447A: 0x4F92 mov.w 18228(r15), &0x0356        ; Move word from table to &0x0356   ; 6
0x447C: 0x4734                                  ; (Table base address)
0x447E: 0x0356                                  ; (Destination address)
0x4480: 0x413E pop.w r14                        ; Restore r14 from stack            ; 2
0x4482: 0x413F pop.w r15                        ; Restore r15 from stack            ; 2
0x4484: 0x1300 reti                             ; Return from interrupt             ; 5
0x4486: 0x120A push.w r10                       ; Save r10 on stack                 ; 3
```

```
0x4488: 0x1209 push.w r9                    ; Save r9 on stack                    ; 3
0x448A: 0x1208 push.w r8                    ; Save r8 on stack                    ; 3
0x448C: 0x4C0A mov.w r12, r10               ; Copy r12 to r10                     ; 1
0x448E: 0x4A78 mov.b @r10+, r8              ; Move byte from *r10 to r8; increment r10   ; 2
0x4490: 0x4309 mov.w #0, r9                 ; Clear r9                            ; 1
0x4492: 0x3C11 jmp 0x44B6  (offset: 34)     ; Jump to 0x44B6                      ; 2
0x4494: 0x4D0E mov.w r13, r14               ; Copy r13 to r14                     ; 1
0x4496: 0x8B0E sub.w r11, r14               ; Subtract r11 from r14               ; 1
0x4498: 0x831E sub.w #1, r14                ; Decrement r14                       ; 1
0x449A: 0x531D add.w #1, r13                ; Increment r13                       ; 1
0x449C: 0x4EFD mov.b @r14+, -1(r13)         ; Move byte from *r14 to *(r13 -1); increment r14; 5
0x449E: 0xFFFF                              ; (Offset -1 for destination)
0x44A0: 0x831F sub.w #1, r15                ; Decrement r15                       ; 1
0x44A2: 0x23FB jne 0x449A  (offset: -10)    ; If r15 != 0, loop back to 0x449A    ; 2
0x44A4: 0x3C03 jmp 0x44AC  (offset: 6)      ; Jump to 0x44AC                      ; 2
0x44A6: 0x531D add.w #1, r13                ; Increment r13                       ; 1
0x44A8: 0x4AFD mov.b @r10+, -1(r13)         ; Move byte from *r10 to *(r13 -1); increment r10 ; 5
0x44AA: 0xFFFF                              ; (Offset -1 for destination)
0x44AC: 0xC312 clrc  --  bic.w #1, SR       ; Clear carry flag                    ; 1
0x44AE: 0x1008 rrc.w r8                     ; Rotate r8 right through carry       ; 1
0x44B0: 0x5319 add.w #1, r9                 ; Increment r9                        ; 1
0x44B2: 0x9239 cmp.w #8, r9                 ; Compare r9 with 8                   ; 1
0x44B4: 0x37EC jge 0x448E  (offset: -20)    ; If r9 >= 8, jump to 0x448E          ; 2
0x44B6: 0xB318 bit.w #1, r8                 ; Test bit 0 of r8                    ; 1
0x44B8: 0x23F6 jne 0x44A6  (offset: -20)    ; If bit is set, jump to 0x44A6       ; 2
0x44BA: 0x4A7B mov.b @r10+, r11             ; Move byte from *r10 to r11; increment r10   ; 2
```

```
0x44BC: 0x4A7F mov.b @r10+, r15                ; Move byte from *r10 to r15; increment r10      ; 2
0x44BE: 0x4B0C mov.w r11, r12                  ; Copy r11 to r12                                ; 1
0x44C0: 0x12B0 call #0x46C0                    ; Call subroutine at 0x46C0                      ; 5
0x44C2: 0x46C0                                 ; (Address of subroutine)
0x44C4: 0x4C0B mov.w r12, r11                  ; Copy r12 back to r11                           ; 1
0x44C6: 0x4F0C mov.w r15, r12                  ; Copy r15 to r12                                ; 1
0x44C8: 0x12B0 call #0x4604                    ; Call subroutine at 0x4604                      ; 5
0x44CA: 0x4604                                 ; (Address of subroutine)
0x44CC: 0xF03C and.w #0x000F, r12              ; Mask lower 4 bits of r12                       ; 2
0x44CE: 0x000F                                 ; (Immediate value)
0x44D0: 0xDC0B bis.w r12, r11                  ; OR r12 with r11                                ; 1
0x44D2: 0xF03F and.w #0x000F, r15              ; Mask lower 4 bits of r15                       ; 2
0x44D4: 0x000F                                 ; (Immediate value)
0x44D6: 0x503F add.w #0x0003, r15              ; Add 3 to r15                                   ; 2
0x44D8: 0x0003                                 ; (Immediate value)
0x44DA: 0x903F cmp.w #0x0012, r15              ; Compare r15 with 18                            ; 2
0x44DC: 0x0012                                 ; (Immediate value)
0x44DE: 0x200C jne 0x44F8  (offset: 24)        ; If not equal, jump to 0x44F8                   ; 2
0x44E0: 0x4A7E mov.b @r10+, r14                ; Move byte from *r10 to r14; increment r10      ; 2
0x44E2: 0xB03E bit.w #0x0080, r14              ; Test bit 7 of r14                              ; 2
0x44E4: 0x0080                                 ; (Immediate value)
0x44E6: 0x2407 jeq 0x44F6  (offset: 14)        ; If zero (bit not set), jump to 0x44F6          ; 2
0x44E8: 0x4A7C mov.b @r10+, r12                ; Move byte from *r10 to r12; increment r10      ; 2
0x44EA: 0x4C4C mov.b r12, r12                  ; Move r12 to r12 (NOP operation)                ; 1
0x44EC: 0x12B0 call #0x46BA                    ; Call subroutine at 0x46BA                      ; 5
0x44EE: 0x46BA                                 ; (Address of subroutine)
```

```
0x44F0: 0xF03E and.w #0x007F, r14          ; Mask bits 0-6 of r14              ; 2
0x44F2: 0x007F                            ; (Immediate value)
0x44F4: 0xDC0E bis.w r12, r14             ; OR r12 with r14                   ; 1
0x44F6: 0x5E0F add.w r14, r15             ; Add r14 to r15                    ; 1
0x44F8: 0x903B cmp.w #0x0FFF, r11         ; Compare r11 with 4095             ; 2
0x44FA: 0x0FFF                            ; (Immediate value)
0x44FC: 0x23CB jne 0x4494  (offset: -106) ; If not equal, loop back to 0x4494 ; 2
0x44FE: 0x4030 mov.w #0x4712, PC          ; Jump to address 0x4712            ; 3
0x4500: 0x4712                            ; (Target address)

; Initialize hardware and configure peripherals
0x4502: 0x40B2 mov.w #0x5A80, &0x015C     ; Stop Watchdog Timer (WDTCTL = WDTPW |
WDTHOLD)  ; 5
0x4504: 0x5A80                            ; (Operand for previous instruction)
0x4506: 0x015C                            ; (Address of WDTCTL)
0x4508: 0xD2F2 bis.b #8, &0x0204          ; Set P1DIR |= BIT3 (P1.3 as output)  ; 4
0x450A: 0x0204                            ; (Address of P1DIR)
0x450C: 0xD2F2 bis.b #8, &0x020A          ; Set P2DIR |= BIT3 (P2.3 as output)  ; 4
0x450E: 0x020A                            ; (Address of P2DIR)
0x4510: 0xC3E2 bic.b #2, &0x0205          ; Clear P1OUT &= ~BIT1 (P1.1 low)     ; 4
0x4512: 0x0205                            ; (Address of P1OUT)
0x4514: 0xD3E2 bis.b #2, &0x0207          ; Set P1REN |= BIT1 (Enable pull-up resistor)  ; 4
0x4516: 0x0207                            ; (Address of P1REN)
0x4518: 0xD3E2 bis.b #2, &0x0203          ; Set P1SEL |= BIT1 (Select peripheral function)  ; 4
0x451A: 0x0203                            ; (Address of P1SEL)
0x451C: 0xD3E2 bis.b #2, &0x021B          ; Set P2SEL |= BIT1 (Select peripheral function)  ; 4
```

```
0x451E: 0x021B                          ; (Address of P2SEL)
0x4520: 0xD3E2 bis.b #2, &0x0219        ; Set P2DIR |= BIT1 (P2.1 as output)        ; 4
0x4522: 0x0219                          ; (Address of P2DIR)
0x4524: 0xC3E2 bic.b #2, &0x021D        ; Clear P2OUT &= ~BIT1 (P2.1 low)           ; 4
0x4526: 0x021D                          ; (Address of P2OUT)
0x4528: 0xC3E2 bic.b #2, &0x0204        ; Set P1DIR &= ~BIT1 (P1.1 as input)        ; 4
0x452A: 0x0204                          ; (Address of P1DIR)
0x452C: 0xD3E2 bis.b #2, &0x0206        ; Set P1IES |= BIT1 (High-to-low transition)  ; 4
0x452E: 0x0206                          ; (Address of P1IES)
0x4530: 0xD3E2 bis.b #2, &0x0202        ; Set P1IE |= BIT1 (Enable interrupt)       ; 4
0x4532: 0x0202                          ; (Address of P1IE)
0x4534: 0xD3E2 bis.b #2, &0x021A        ; Set P2IE |= BIT1 (Enable interrupt)       ; 4
0x4536: 0x021A                          ; (Address of P2IE)
0x4538: 0xD3E2 bis.b #2, &0x0218        ; Set P2IES |= BIT1 (High-to-low transition)  ; 4
0x453A: 0x0218                          ; (Address of P2IES)
0x453C: 0xC3E2 bic.b #2, &0x021C        ; Clear P2IFG &= ~BIT1 (Clear interrupt flag) ; 4
0x453E: 0x021C                          ; (Address of P2IFG)
0x4540: 0x40B2 mov.w #0x5A1C, &0x015C       ; Start WDT in interval mode (WDTCTL)       ;
5
0x4542: 0x5A1C                          ; (Operand for previous instruction)
0x4544: 0x015C                          ; (Address of WDTCTL)
0x4546: 0xD392 bis.w #1, &0x0100        ; Enable NMI interrupts (IE1 |= NMIIE)      ; 4
0x4548: 0x0100                          ; (Address of IE1)
0x454A: 0x40B2 mov.w #0x03E7, &0x0352       ; Set Timer_B CCR0 to 999               ; 5
0x454C: 0x03E7                          ; (Value for CCR0)
0x454E: 0x0352                          ; (Address of TBCCR0)
```

```
0x4550: 0x40B2 mov.w #0x00E0, &0x0346             ; Set Timer_B CCR1 to 224                    ; 5
0x4552: 0x00E0                                    ; (Value for CCR1)
0x4554: 0x0346                                    ; (Address of TBCCR1)
0x4556: 0x421F mov.w &0x2400, r15                 ; Load &0x2400 into r15                      ; 3
0x4558: 0x2400                                    ; (Operand for previous instruction)
0x455A: 0x5F0F add.w r15, r15                     ; Double r15                                 ; 1
0x455C: 0x4F92 mov.w 18228(r15), &0x0356          ; Move word from table to &0x0356 (TBCCTL1)
; 6
0x455E: 0x4734                                    ; (Table base address)
0x4560: 0x0356                                    ; (Destination address)
0x4562: 0x40B2 mov.w #0x0210, &0x0340             ; Configure Timer_B control register (TBCTL)      ;
5
0x4564: 0x0210                                    ; (Value for TBCTL)
0x4566: 0x0340                                    ; (Address of TBCTL)
0x4568: 0xD232 eint  --  bis.w #8, SR             ; Enable interrupts (GIE bit in SR)          ; 1
0x456A: 0xD032 bis.w #0x0018, SR                  ; Enter LPM0 (Low-power mode 0)              ; 2
0x456C: 0x0018                                    ; (Bits to set in SR)
0x456E: 0x430C mov.w #0, r12                      ; Clear r12                                  ; 1
0x4570: 0x4130 ret   --  mov.w @SP+, PC           ; Return from subroutine                     ; 3

; Subroutine to handle certain operation
0x4572: 0x120A push.w r10                         ; Save r10 on stack                          ; 3
0x4574: 0x1209 push.w r9                          ; Save r9 on stack                           ; 3
0x4576: 0x1208 push.w r8                          ; Save r8 on stack                           ; 3
0x4578: 0x4219 mov.w &0x015C, r9                  ; Load WDTCTL into r9                        ; 3
0x457A: 0x015C                                    ; (Address of WDTCTL)
```

```
0x457C: 0x40B2 mov.w #0x5A80, &0x015C          ; Stop Watchdog Timer                    ; 5
0x457E: 0x5A80                                 ; (Operand for previous instruction)
0x4580: 0x015C                                 ; (Address of WDTCTL)
0x4582: 0x403F mov.w #0x4758, r15              ; Load address 0x4758 into r15           ; 2
0x4584: 0x4758                                 ; (Immediate value)
0x4586: 0x903F cmp.w #0x475C, r15              ; Compare r15 with 0x475C                ; 2
0x4588: 0x475C                                 ; (Operand for comparison)
0x458A: 0x2416 jeq 0x45B8  (offset: 44)        ; If equal, jump to 0x45B8               ; 2
0x458C: 0x403F mov.w #0x475C, r15              ; Load address 0x475C into r15           ; 2
0x458E: 0x475C                                 ; (Immediate value)
0x4590: 0x903F cmp.w #0x4760, r15              ; Compare r15 with 0x4760                ; 2
0x4592: 0x4760                                 ; (Operand for comparison)
0x4594: 0x2411 jeq 0x45B8  (offset: 34)        ; If equal, jump to 0x45B8               ; 2
0x4596: 0x403A mov.w #0x4760, r10              ; Load address 0x4760 into r10           ; 2
0x4598: 0x4760                                 ; (Immediate value)
0x459A: 0x803A sub.w #0x475C, r10              ; Subtract 0x475C from r10               ; 2
0x459C: 0x475C                                 ; (Immediate value)
0x459E: 0x110A rra.w r10                       ; Right arithmetic shift r10 (divide by 2)  ; 1
0x45A0: 0x110A rra.w r10                       ; Right arithmetic shift r10 again (divide by 2)  ; 1
0x45A2: 0x4038 mov.w #0x475C, r8               ; Load address 0x475C into r8            ; 2
0x45A4: 0x475C                                 ; (Immediate value)
0x45A6: 0x483C mov.w @r8+, r12                 ; Load word from *r8 into r12; increment r8  ; 2
0x45A8: 0x4C7F mov.b @r12+, r15                ; Move byte from *r12 to r15; increment r12  ; 2
0x45AA: 0x5F0F add.w r15, r15                  ; Double r15                             ; 1
0x45AC: 0x4F1F mov.w 18264(r15), r15           ; Load word from table into r15          ; 3
0x45AE: 0x4758                                 ; (Table base address)
```

```
0x45B0: 0x483D mov.w @r8+, r13          ; Load word from *r8 into r13; increment r8      ; 2
0x45B2: 0x128F call r15                 ; Call subroutine at address in r15              ; 4
0x45B4: 0x831A sub.w #1, r10            ; Decrement r10                                  ; 1
0x45B6: 0x23F7 jne 0x45A6  (offset: -18)  ; If r10 != 0, loop back to 0x45A6             ; 2
0x45B8: 0xC279 bic.b #8, r9             ; Clear bit 3 in r9                              ; 1
0x45BA: 0xD039 bis.w #0x5A08, r9        ; Set bits in r9 for WDTCTL                      ; 2
0x45BC: 0x5A08                          ; (Immediate value)
0x45BE: 0x4982 mov.w r9, &0x015C        ; Write r9 to WDTCTL                             ; 4
0x45C0: 0x015C                          ; (Address of WDTCTL)
0x45C2: 0x12B0 call #0x4732             ; Call subroutine at 0x4732                      ; 5
0x45C4: 0x4732                          ; (Address of subroutine)
0x45C6: 0x4030 mov.w #0x4712, PC        ; Jump to 0x4712                                 ; 3
0x45C8: 0x4712                          ; (Target address)
0x45CA: 0xF03D and.w #0x000F, r13       ; Mask lower 4 bits of r13                       ; 2
0x45CC: 0x000F                          ; Immediate value for the mask
0x45CE: 0xE03D xor.w #0x000F, r13       ; XOR lower 4 bits of r13 with 0x000F            ; 2
0x45D0: 0x000F                          ; Immediate value for XOR
0x45D2: 0x5D0D add.w r13, r13           ; Double r13 (r13 = r13 * 2)                     ; 1
0x45D4: 0x5D0D add.w r13, r13           ; Double r13 again (r13 = r13 * 4)               ; 1
0x45D6: 0x5D00 add.w r13, PC            ; Adjust PC by r13 (jump to computed address)    ; 2
0x45D8: 0xC312 clrc                     ; Clear carry flag                               ; 1
0x45DA: 0x100C rrc.w r12                ; Rotate r12 right through carry                 ; 1
0x45DC: 0xC312 clrc                     ; Clear carry flag                               ; 1
0x45DE: 0x100C rrc.w r12                ; Rotate r12 right through carry                 ; 1
0x45E0: 0xC312 clrc                     ; Clear carry flag                               ; 1
0x45E2: 0x100C rrc.w r12                ; Rotate r12 right through carry                 ; 1
```

```
0x45E4: 0xC312 clrc                    ; Clear carry flag                    ; 1
0x45E6: 0x100C rrc.w r12               ; Rotate r12 right through carry      ; 1
0x45E8: 0xC312 clrc                    ; Clear carry flag                    ; 1
0x45EA: 0x100C rrc.w r12               ; Rotate r12 right through carry      ; 1
0x45EC: 0xC312 clrc                    ; Clear carry flag                    ; 1
0x45EE: 0x100C rrc.w r12               ; Rotate r12 right through carry      ; 1
0x45F0: 0xC312 clrc                    ; Clear carry flag                    ; 1
0x45F2: 0x100C rrc.w r12               ; Rotate r12 right through carry      ; 1
0x45F4: 0xC312 clrc                    ; Clear carry flag                    ; 1
0x45F6: 0x100C rrc.w r12               ; Rotate r12 right through carry      ; 1
0x45F8: 0xC312 clrc                    ; Clear carry flag                    ; 1
0x45FA: 0x100C rrc.w r12               ; Rotate r12 right through carry      ; 1
0x45FC: 0xC312 clrc                    ; Clear carry flag                    ; 1
0x45FE: 0x100C rrc.w r12               ; Rotate r12 right through carry      ; 1
0x4600: 0xC312 clrc                    ; Clear carry flag                    ; 1
0x4602: 0x100C rrc.w r12               ; Rotate r12 right through carry      ; 1
0x4604: 0xC312 clrc                    ; Clear carry flag                    ; 1
0x4606: 0x100C rrc.w r12               ; Rotate r12 right through carry      ; 1
0x4608: 0xC312 clrc                    ; Clear carry flag                    ; 1
0x460A: 0x100C rrc.w r12               ; Rotate r12 right through carry      ; 1
0x460C: 0xC312 clrc                    ; Clear carry flag                    ; 1
0x460E: 0x100C rrc.w r12               ; Rotate r12 right through carry      ; 1
0x4610: 0xC312 clrc                    ; Clear carry flag                    ; 1
0x4612: 0x100C rrc.w r12               ; Rotate r12 right through carry      ; 1
0x4614: 0x4130 ret                     ; Return from subroutine              ; 3
```

```
; Subroutine starting at 0x4616
0x4616: 0x120F push.w r15                    ; Save r15 on stack                    ; 3
0x4618: 0x120E push.w r14                    ; Save r14 on stack                    ; 3
0x461A: 0x120D push.w r13                    ; Save r13 on stack                    ; 3
0x461C: 0x120C push.w r12                    ; Save r12 on stack                    ; 3
0x461E: 0x120B push.w r11                    ; Save r11 on stack                    ; 3
0x4620: 0xB3E2 bit.b #2, &0x021C             ; Test bit 1 of P3IFG (P3 Interrupt Flag)      ; 4
0x4622: 0x021C                               ; Address of P3IFG
0x4624: 0x2412 jeq 0x464A  (offset: 36)      ; If zero (bit not set), jump to 0x464A        ; 2
0x4626: 0x40B2 mov.w #0x61A8, &0x2402        ; Load 0x61A8 into &0x2402                     ; 5
0x4628: 0x61A8                               ; Immediate value
0x462A: 0x2402                               ; Destination address
0x462C: 0x9382 cmp.w #0, &0x2402             ; Compare &0x2402 with 0                       ; 4
0x462E: 0x2402                               ; Operand address
0x4630: 0x2405 jeq 0x463C  (offset: 10)      ; If equal, jump to 0x463C                     ; 2
0x4632: 0x8392 sub.w #1, &0x2402             ; Decrement &0x2402                            ; 4
0x4634: 0x2402                               ; Operand address
0x4636: 0x9382 cmp.w #0, &0x2402             ; Compare &0x2402 with 0                       ; 4
0x4638: 0x2402                               ; Operand address
0x463A: 0x23FB jne 0x4632  (offset: -10)     ; Loop back if not equal                       ; 2
0x463C: 0x9382 cmp.w #0, &0x2400             ; Compare &0x2400 with 0                       ; 4
0x463E: 0x2400                               ; Operand address
0x4640: 0x2404 jeq 0x464A  (offset: 8)       ; If equal, jump to 0x464A                     ; 2
0x4642: 0x8392 sub.w #1, &0x2400             ; Decrement &0x2400                            ; 4
0x4644: 0x2400                               ; Operand address
0x4646: 0x12B0 call #0x471A                  ; Call subroutine at 0x471A                    ; 5
```

```
0x4648: 0x471A                          ; Subroutine address
0x464A: 0xC3E2 bic.b #2, &0x021C        ; Clear bit 1 of P3IFG (clear interrupt flag)   ; 4
0x464C: 0x021C                          ; Address of P3IFG
0x464E: 0x413B pop.w r11                ; Restore r11 from stack                        ; 2
0x4650: 0x413C pop.w r12                ; Restore r12 from stack                        ; 2
0x4652: 0x413D pop.w r13                ; Restore r13 from stack                        ; 2
0x4654: 0x413E pop.w r14                ; Restore r14 from stack                        ; 2
0x4656: 0x413F pop.w r15                ; Restore r15 from stack                        ; 2
0x4658: 0x1300 reti                     ; Return from interrupt                         ; 5

; Another interrupt service routine starting at 0x465A
0x465A: 0x120F push.w r15               ; Save r15 on stack                             ; 3
0x465C: 0x120E push.w r14               ; Save r14 on stack                             ; 3
0x465E: 0x120D push.w r13               ; Save r13 on stack                             ; 3
0x4660: 0x120C push.w r12               ; Save r12 on stack                             ; 3
0x4662: 0x120B push.w r11               ; Save r11 on stack                             ; 3
0x4664: 0xB3E2 bit.b #2, &0x021D        ; Test bit 1 of P3IFG (P3 Interrupt Flag)       ; 4
0x4666: 0x021D                          ; Address of P3IFG
0x4668: 0x2412 jeq 0x468E  (offset: 36) ; If zero (bit not set), jump to 0x468E         ; 2
0x466A: 0x40B2 mov.w #0x61A8, &0x2402   ; Load 0x61A8 into &0x2402                      ; 5
0x466C: 0x61A8                          ; Immediate value
0x466E: 0x2402                          ; Destination address
0x4670: 0x9382 cmp.w #0, &0x2402        ; Compare &0x2402 with 0                        ; 4
0x4672: 0x2402                          ; Operand address
0x4674: 0x2405 jeq 0x4680  (offset: 10) ; If equal, jump to 0x4680                      ; 2
0x4676: 0x8392 sub.w #1, &0x2402        ; Decrement &0x2402                             ; 4
```

```
0x4678: 0x2402                          ; Operand address
0x467A: 0x9382 cmp.w #0, &0x2402        ; Compare &0x2402 with 0          ; 4
0x467C: 0x2402                          ; Operand address
0x467E: 0x23FB jne 0x4676  (offset: -10)  ; Loop back if not equal        ; 2
0x4680: 0x92B2 cmp.w #8, &0x2400        ; Compare &0x2400 with 8          ; 4
0x4682: 0x2400                          ; Operand address
0x4684: 0x2C04 jhs 0x468E  (offset: 8)  ; If greater or equal, jump to 0x468E  ; 2
0x4686: 0x5392 add.w #1, &0x2400        ; Increment &0x2400               ; 4
0x4688: 0x2400                          ; Operand address
0x468A: 0x12B0 call #0x471A             ; Call subroutine at 0x471A       ; 5
0x468C: 0x471A                          ; Subroutine address
0x468E: 0xC3E2 bic.b #2, &0x021D        ; Clear bit 1 of P3IFG (clear interrupt flag)  ; 4
0x4690: 0x021D                          ; Address of P3IFG
0x4692: 0x413B pop.w r11                ; Restore r11 from stack          ; 2
0x4694: 0x413C pop.w r12                ; Restore r12 from stack          ; 2
0x4696: 0x413D pop.w r13                ; Restore r13 from stack          ; 2
0x4698: 0x413E pop.w r14                ; Restore r14 from stack          ; 2
0x469A: 0x413F pop.w r15                ; Restore r15 from stack          ; 2
0x469C: 0x1300 reti                     ; Return from interrupt           ; 5

; Subroutine starting at 0x469E
0x469E: 0xF03D and.w #0x000F, r13        ; Mask lower 4 bits of r13        ; 2
0x46A0: 0x000F                          ; Immediate value
0x46A2: 0xE03D xor.w #0x000F, r13        ; XOR lower 4 bits of r13 with 0x000F  ; 2
0x46A4: 0x000F                          ; Immediate value
0x46A6: 0x5D0D add.w r13, r13           ; Double r13                      ; 1
```

```
0x46A8: 0x5D00 add.w r13, PC                ; Adjust PC by r13 (jump to computed address)      ; 2

; The following instructions are part of a computed jump table or delay loop
0x46AA: 0x5C0C add.w r12, r12               ; Double r12                                       ; 1
0x46AC: 0x5C0C add.w r12, r12               ; Double r12                                       ; 1
0x46AE: 0x5C0C add.w r12, r12               ; Double r12                                       ; 1
0x46B0: 0x5C0C add.w r12, r12               ; Double r12                                       ; 1
0x46B2: 0x5C0C add.w r12, r12               ; Double r12                                       ; 1
0x46B4: 0x5C0C add.w r12, r12               ; Double r12                                       ; 1
0x46B6: 0x5C0C add.w r12, r12               ; Double r12                                       ; 1
0x46B8: 0x5C0C add.w r12, r12               ; Double r12                                       ; 1
0x46BA: 0x5C0C add.w r12, r12               ; Double r12                                       ; 1
0x46BC: 0x5C0C add.w r12, r12               ; Double r12                                       ; 1
0x46BE: 0x5C0C add.w r12, r12               ; Double r12                                       ; 1
0x46C0: 0x5C0C add.w r12, r12               ; Double r12                                       ; 1
0x46C2: 0x5C0C add.w r12, r12               ; Double r12                                       ; 1
0x46C4: 0x5C0C add.w r12, r12               ; Double r12                                       ; 1
0x46C6: 0x5C0C add.w r12, r12               ; Double r12                                       ; 1
0x46C8: 0x4130 ret                          ; Return from subroutine                           ; 3

; Main program starting at 0x46CA
0x46CA: 0x4031 mov.w #0x4400, SP            ; Initialize stack pointer                         ; 2
0x46CC: 0x4400                              ; Immediate value (stack start address)
0x46CE: 0x12B0 call #0x472E                 ; Call subroutine at 0x472E                        ; 5
0x46D0: 0x472E                              ; Subroutine address
0x46D2: 0x930C cmp.w #0, r12                ; Compare r12 with 0                               ; 1
```

```
0x46D4: 0x2402 jeq 0x46DA  (offset: 4)          ; If equal, jump to 0x46DA                    ; 2
0x46D6: 0x12B0 call #0x4572                     ; Call subroutine at 0x4572                   ; 5
0x46D8: 0x4572                                  ; Subroutine address
0x46DA: 0x430C mov.w #0, r12                    ; Clear r12                                   ; 1
0x46DC: 0x12B0 call #0x4502                     ; Call subroutine at 0x4502                   ; 5
0x46DE: 0x4502                                  ; Subroutine address
0x46E0: 0x431C mov.w #1, r12                    ; Set r12 to 1                                ; 1
0x46E2: 0x12B0 call #0x4728                     ; Call subroutine at 0x4728                   ; 5
0x46E4: 0x4728                                  ; Subroutine address
0x46E6: 0x4C0F mov.w r12, r15                   ; Copy r12 to r15                             ; 1
0x46E8: 0x4D0C mov.w r13, r12                   ; Copy r13 to r12                             ; 1
0x46EA: 0x403D mov.w #0x0003, r13               ; Load 3 into r13                             ; 2
0x46EC: 0x0003                                  ; Immediate value
0x46EE: 0x5F0D add.w r15, r13                   ; Add r15 to r13                              ; 1
0x46F0: 0x4F1E mov.w 1(r15), r14                ; Load word from address (r15 + 1) into r14   ; 3
0x46F2: 0x0001                                  ; Offset
0x46F4: 0x4030 mov.w #0x46F8, PC                ; Jump to address 0x46F8                      ; 3
0x46F6: 0x46F8                                  ; Target address

; Loop starting at 0x46F8
0x46F8: 0x930E cmp.w #0, r14                    ; Compare r14 with 0                          ; 1
0x46FA: 0x2406 jeq 0x4708  (offset: 12)         ; If equal, jump to 0x4708                    ; 2
0x46FC: 0x4C0F mov.w r12, r15                   ; Copy r12 to r15                             ; 1
0x46FE: 0x531F add.w #1, r15                    ; Increment r15                               ; 1
0x4700: 0x4DFD mov.b @r13+, -1(r15)             ; Move byte from *r13 to *(r15 -1); increment r13    ;
5
```

```
0x4702: 0xFFFF                                  ; Offset -1
0x4704: 0x831E sub.w #1, r14                    ; Decrement r14                 ; 1
0x4706: 0x23FB jne 0x46FE  (offset: -10)        ; Loop back if not zero         ; 2
0x4708: 0x4130 ret                              ; Return from subroutine        ; 3

; Subroutine starting at 0x470A
0x470A: 0x4134 pop.w r4                         ; Restore r4 from stack         ; 2
0x470C: 0x4135 pop.w r5                         ; Restore r5 from stack         ; 2
0x470E: 0x4136 pop.w r6                         ; Restore r6 from stack         ; 2
0x4710: 0x4137 pop.w r7                         ; Restore r7 from stack         ; 2
0x4712: 0x4138 pop.w r8                         ; Restore r8 from stack         ; 2
0x4714: 0x4139 pop.w r9                         ; Restore r9 from stack         ; 2
0x4716: 0x413A pop.w r10                        ; Restore r10 from stack        ; 2
0x4718: 0x4130 ret                              ; Return from subroutine        ; 3

; Subroutine starting at 0x471A
0x471A: 0x421F mov.w &0x2400, r15               ; Load value from &0x2400 into r15      ; 3
0x471C: 0x2400                                  ; Address
0x471E: 0x5F0F add.w r15, r15                   ; Double r15                    ; 1
0x4720: 0x4F92 mov.w 18228(r15), &0x0356        ; Load word from table into &0x0356     ;
6
0x4722: 0x4734                                  ; Table base address
0x4724: 0x0356                                  ; Destination address
0x4726: 0x4130 ret                              ; Return from subroutine        ; 3

; Infinite loop starting at 0x4728
```

```
0x4728: 0x4303 nop                          ; No operation (NOP)                    ; 1
0x472A: 0x3FFF jmp 0x472A  (offset: -2)     ; Jump to 0x472A (infinite loop)        ; 2
0x472C: 0x4303 nop                          ; No operation (NOP)                    ; 1

; Subroutine starting at 0x472E
0x472E: 0x431C mov.w #1, r12                ; Set r12 to 1                          ; 1
0x4730: 0x4130 ret                          ; Return from subroutine                ; 3

; Subroutine starting at 0x4732
0x4732: 0x4130 ret                          ; Return from subroutine                ; 3

; Data table starting at 0x4734
0x4734: 0x0000                              ; Data word
0x4736: 0x007D                              ; Data word
0x4738: 0x00FA                              ; Data word
0x473A: 0x0177                              ; Data word
0x473C: 0x01F4                              ; Data word
0x473E: 0x0271                              ; Data word
0x4740: 0x02EE                              ; Data word
0x4742: 0x036B                              ; Data word
0x4744: 0x03E8                              ; Data word

; Infinite loop starting at 0x4746
0x4746: 0xD032 bis.w #0x0010, SR            ; Enable general interrupts (GIE bit)   ; 2
0x4748: 0x0010                              ; Immediate value
0x474A: 0x3FFD jmp 0x4746  (offset: -6)     ; Jump back to 0x4746 (infinite loop)   ; 2
```

```
0x474C: 0x4303 nop                          ; No operation (NOP)                    ; 1

; The following addresses may contain invalid instructions or data
0x474E: 0x1B00 mova @PC, r4                 ; Invalid instruction or placeholder    ; ?
0x4750: 0x0004                              ; Data word or operand
0x4752: 0x0300 mova @CG, PC                 ; Invalid instruction or placeholder    ; ?
0x4754: 0x0001 mova @PC, SP                 ; Invalid instruction or placeholder    ; ?
0x4756: 0xF0FF and.b #0x4486, 18150(r15)    ; AND operation with immediate and memory ; 5
0x4758: 0x4486                              ; Immediate value
0x475A: 0x46E6                              ; Memory offset
0x475C: 0x474E mov.b r7, r14                ; Move byte from r7 to r14              ; 1
0x475E: 0x2400 jeq 0x4760  (offset: 0)      ; If equal, jump to 0x4760              ; 2
```

Flowchart:

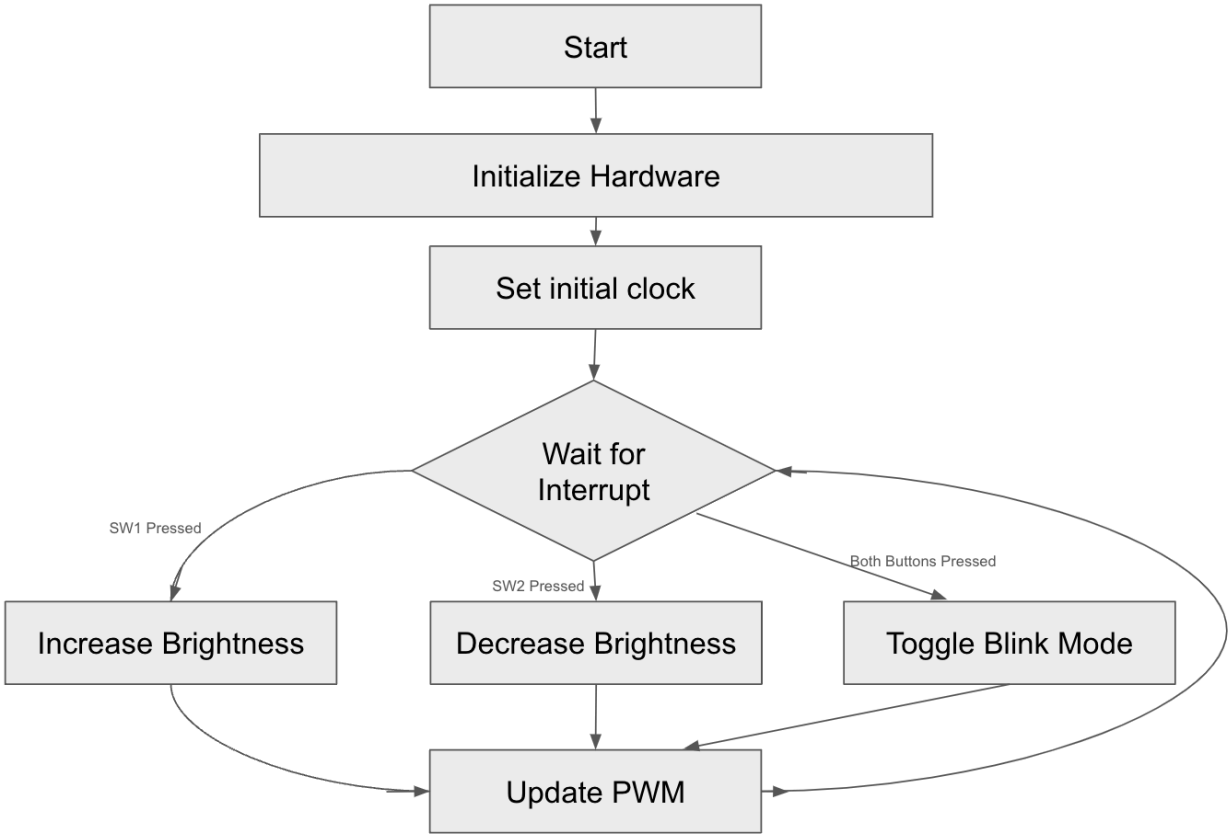

Figure 1: Program 2 Flowchart

Conclusion

In conclusion, I successfully worked through the steps of converting an `.out` file from a PWM program into a HEX file, analyzing it with various tools, and flashing it to the MSP430 microcontroller using the MSP430Flasher. By verifying the functionality of the flashed program, I was able to ensure it operated as expected, allowing control over the brightness level as in the original CCS environment. I also reverse-engineered the HEX file back to assembly code and documented the program's behavior. This lab reinforced my understanding of file formats, the process of flashing firmware to embedded systems, and the techniques used for reverse engineering machine code. Through these exercises, I gained valuable insights into the inner workings of embedded system development and debugging. I am so glad that the labs are done!