

CPE 323
Introduction to Software Reverse Engineering in
Embedded Computer Systems

Aleksandar Milenković

Email: milenka@uah.edu

Web: <http://www.ece.uah.edu/~milenka>

Objective:

Introduce tools and methods for software reverse engineering in embedded systems

Contents

Contents.....	1
1 Introduction.....	2
2 Format of Executable Files	2
3 GNU Utilities	6
4 Deconstructing Executable Files: An Example.....	10
5 Working with HEX Files and MSP430Flasher Utility	25
6 To Learn More	29



1 Introduction

In this section we will introduce basic concepts, tools, and techniques for software reverse engineering with a special emphasis on embedded computer systems.

Reverse engineering in general is a process of deconstructing man-made artifacts with a goal to reveal their designs and architecture or to extract knowledge. It is widely used in many areas of engineering, but here we are focusing on software reverse engineering. Note: hardware reverse engineering is another topic that may be of interest for electrical and computer engineers, but it is out of scope in this tutorial.

Software reverse engineering refers to a process of analyzing a software system in order to identify its components and their interrelationships and to create representations of the system in another form, typically at a higher level of abstraction. Two main components of software reverse engineering are *re-documentation* and *design recovery*. Re-documentation is a process of creating a new representation of the computer code that is easier to understand, often given at a higher level of abstraction. Design recovery is the use of deduction or reasoning from personal experience of the software system to understand its functionality. Software reverse engineering can be used even when the source code is available with the goal to uncover aspects of the program that may be poorly documented or are documented but no longer valid. More often though, software reverse engineering is used when source code is not available.

Software reverse engineering is used for the purpose of:

- Analyzing malware;
- Analyzing closed-source software to uncover vulnerabilities or interoperability issues;
- Analyzing compiler-generated code to validate performance and/or correctness;
- Debugging programs;

This tutorial focuses on reverse engineering of code written for the TI's MSP430 family of microcontrollers. It covers the following topics:

- Format of Executable Files
- GNU binary utilities typically used in software reverse engineering to understand executable files and disassemble executable programs;
- Extracting useful information from binaries;
- Retrieving programs from embedded platforms and analyzing them.

2 Format of Executable Files

In this section we will take a look at the format of executable files. Figure 1 illustrates a generalized flow of source code translation. User created source code written in high-level programming languages or an assembly language is translated into object files that are further linked with library files into executable files that are then loaded onto the target platform.

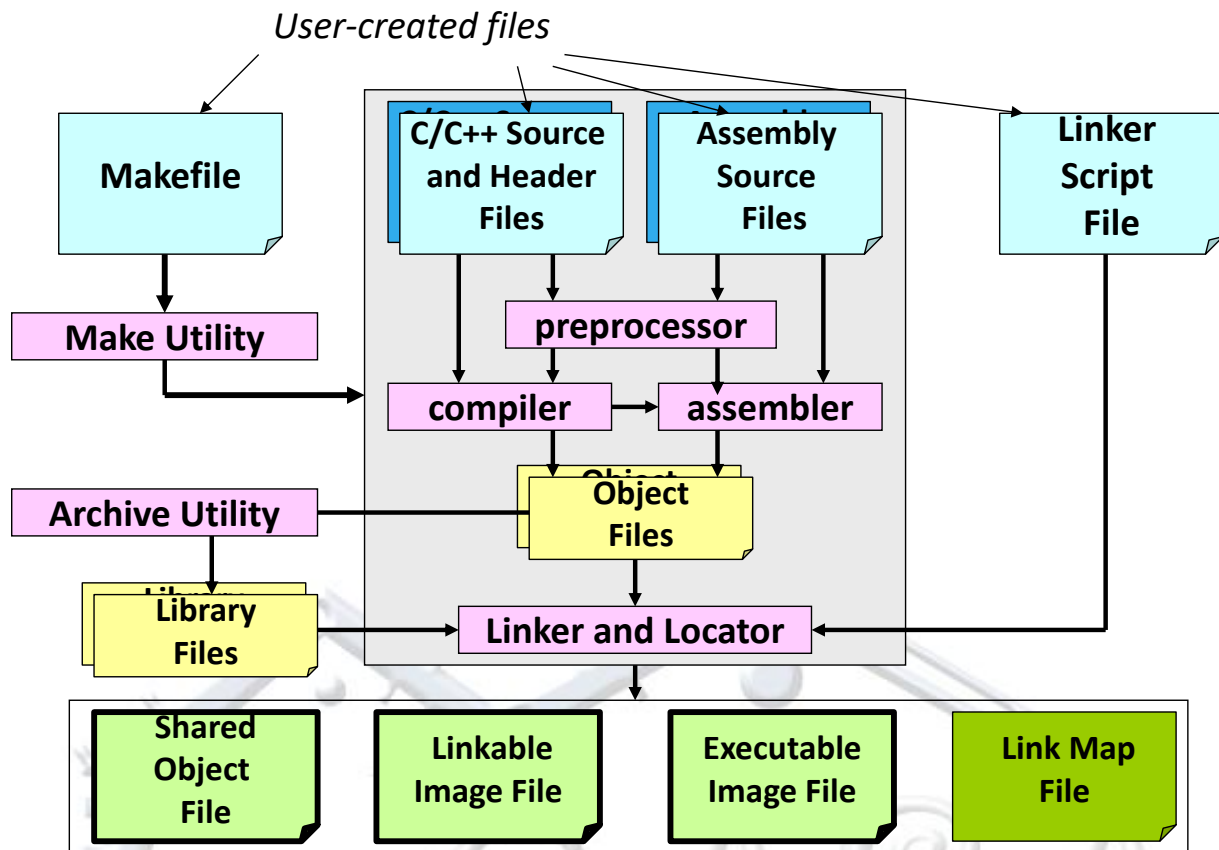


Figure 1. Source Translation Process.

Executable and Linkable File (ELF) format is a common standard file format used for executable files, object files, shared libraries, and core dumps. TI Code Composer Studio produces executable files in the ELF format, regardless of the compiler used (TI compiler or GNU MSP430 GCC compiler). The ELF format is not bound by the Instruction Set Architecture or operating systems. It defines the structure of the file, specifically the headers which describe actual binary content of the file. The structure of the ELF file is well defined and more information can be found at https://en.wikipedia.org/wiki/Executable_and_Linkable_Format. In brief, it is important to recognize the concept of segments and sections. The segments contain information that is needed for run-time execution of the program, while sections contain important data needed for linking and relocation.

Figure 2 illustrates two views of ELF files: linkable and executable file formats. ELF files contain the following components:

- ELF file header
- Program header table: Describes zero or more memory segments; It tells loader how to create a process image in memory;
- Section header table: Describes zero or more sections that contain data referred to by entries in the program header tables and section header tables;
- Segments: contain info needed for run-time execution;

- Sections: contain info for linking and relocation.

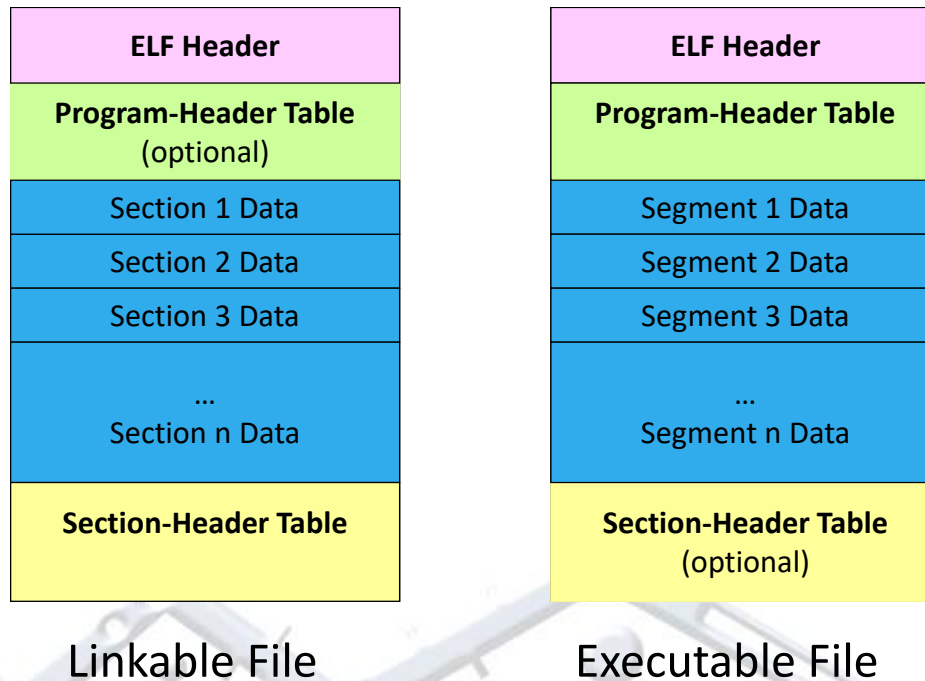


Figure 2. Linkable and Executable Views of ELF Files.

ELF linkable files are divided into a collection of sections. Each section contains a single type of information and can contain flags (writable data, memory space during execution or executable machine instructions). Sections have:

- Name and type
- Requested memory location at run time
- Permissions (R, W, X).

Table 1 shows common sections of ELF linkable files.

Table 1. ELF Linking View: Common Sections.

Sections	Description
.interp	Path name of program interpreter
.text	Code (executable instructions) of a program; Typically stored in read-only memory.
.data	Initialized read/write data (global, static)
.bss	Uninitialized read/write data (global, static) Often it is initialized by the start-up code
.const/.rodata	Read-only data; typically stored in Flash memory

.init	Executable instructions for process initialization
.fini	Executable instructions for process termination
.ptl	Holds the procedure linkage table
.re.[x]	Relocation information for section [x]
.dynamic	Dynamic linking information
.symtab, .dynsym	Symbols (static/dynamic)
.strtab, .dynstr	String table
.stack	Stack

Linker is a utility program that takes one or more object files generated by a compiler and combines them into a single executable file, library file, or another object file. Figure 3 illustrates linking multiple object files into a single executable file. The linker script defines the memory map of the device with respect to the compiled code sections. The linker needs to know where in memory to locate each of the sections of code, based on the type of section and its attributes. Sometimes, these linker scripts can be modified by the developer to add custom sections for very specific purposes, but typically they are provided by software development environments.

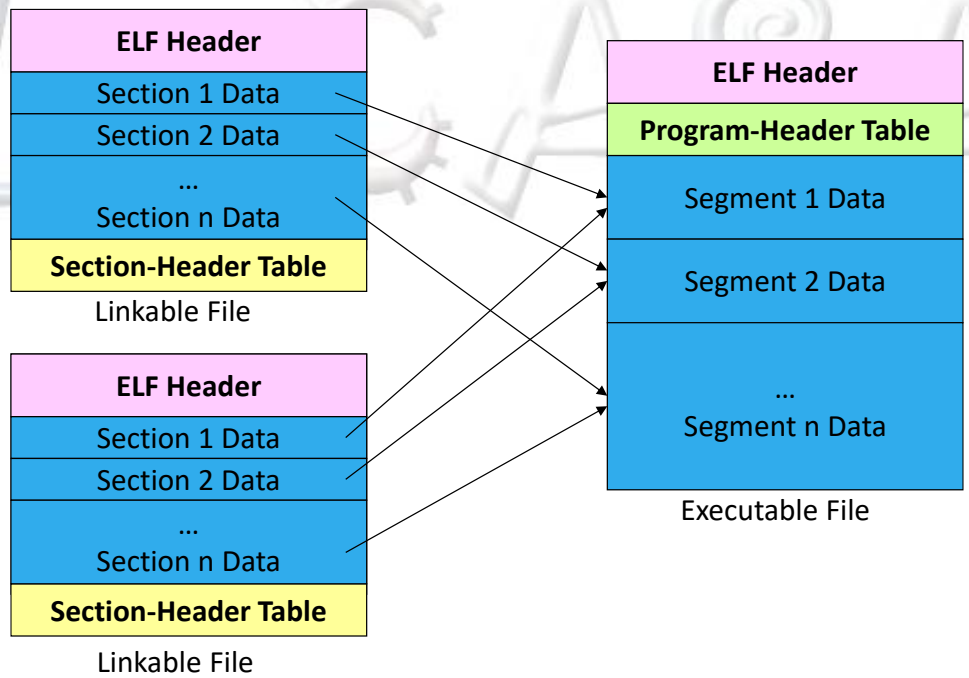


Figure 3. Linking Object Files into an Executable File.

The executable ELF file consists of segments. All loadable sections are packed into segments. Segments are parts with code and data that are loaded into memory at run-time. Utility

programs that load executable files into memory and start program execution are called loaders. Segments have:

- Type
- Requested memory location
- Permissions (R, W, X)
- Size (in file and in memory)

Table 2 shows common segments in ELF executable files.

Table 2. ELF Executable View: Common Segments.

Common Segments	Description
LOAD	Portion of file to be loaded into memory
INTERP	Pointer to dynamic linker for this executable (.interp section)
DYNAMIC	Pointer to dynamic linking information (.dynamic section)

3 GNU Utilities

In this section we will give a brief introduction to GNU Binary Utilities, also known as binutils. Binutils is a set of programming tools for creating and managing binary programs, object files, profile data, and assembly source code. Table 3 shows a list of commonly used binutils.

Table 3. Common GNU utilities

Utility	Description
as	Assembler
elfedit	Edit ELF files
gdb	Debugger
gprof	Profiler
ld	Linker
objcopy	Copy object files, possibly making changes
objdump	Dump information about object files
nm	List symbols from object files
readelf	Display content of ELF files
strings	List printable strings
size	List total and section sizes

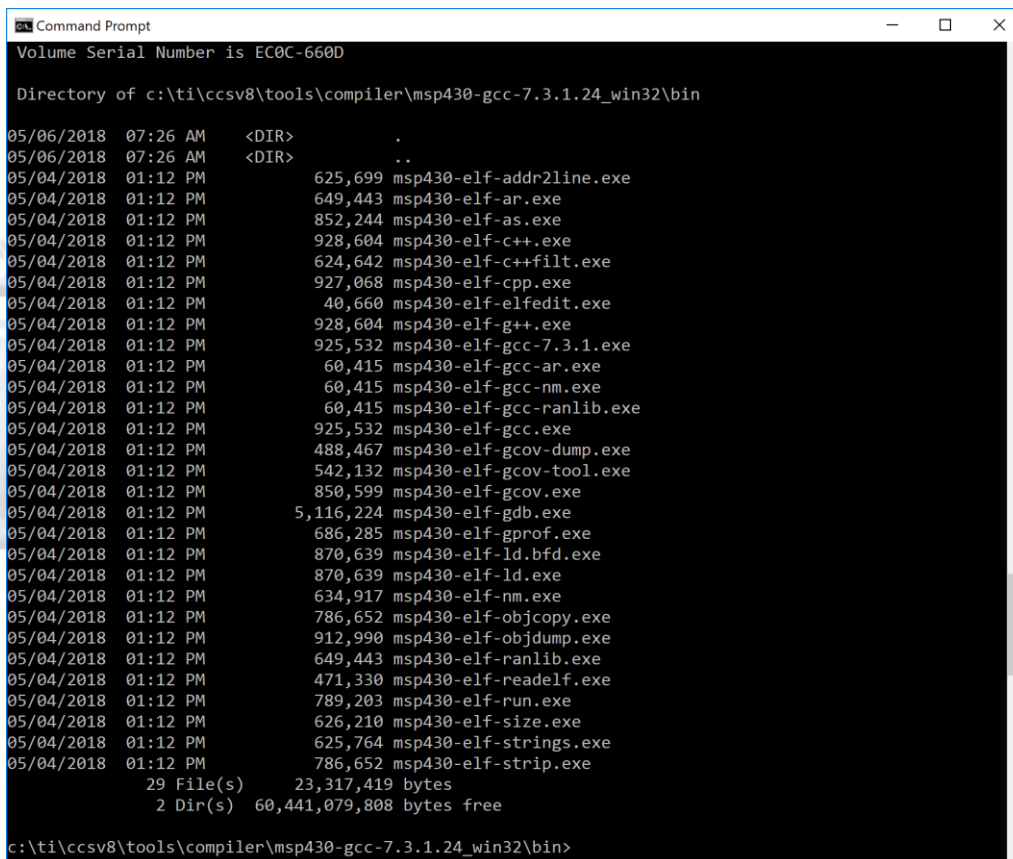
strip

Remove symbols from an object file

Texas Instruments partnered with a third party company to support open-source compiler called MSP430 GCC that originated from a community-driven MSPGCC. MSP430 GCC can be used as a stand-alone package or it can be used within Code Composer Studio (CCS) IDE v6.0 or later as an Add-On through the CCS's App Center.

You can locate various MSP430 GNU utilities from a Windows Command Prompt as shown in Figure 4. To learn more about each utility, run each of them with `--help` switch. Here we will take a closer look at several of these utilities of interest for software reverse engineering tasks:

- `msp430-elf-readelf`: displays information about executable files (Figure 5);
- `msp430-elf-objdump`: disassembler (Figure 6);
- `msp430-elf-strings`: displays printable strings (Figure 7).



```
Command Prompt
Volume Serial Number is EC0C-660D

Directory of c:\ti\ccsv8\tools\compiler\msp430-gcc-7.3.1.24_win32\bin

05/06/2018 07:26 AM <DIR>      .
05/06/2018 07:26 AM <DIR>      ..
05/04/2018 01:12 PM          625,699 msp430-elf-addr2line.exe
05/04/2018 01:12 PM          649,443 msp430-elf-ar.exe
05/04/2018 01:12 PM          852,244 msp430-elf-as.exe
05/04/2018 01:12 PM          928,604 msp430-elf-c++.exe
05/04/2018 01:12 PM          624,642 msp430-elf-c++filt.exe
05/04/2018 01:12 PM          927,068 msp430-elf-cpp.exe
05/04/2018 01:12 PM          40,660 msp430-elf-elfedit.exe
05/04/2018 01:12 PM          928,604 msp430-elf-g++.exe
05/04/2018 01:12 PM          925,532 msp430-elf-gcc-7.3.1.exe
05/04/2018 01:12 PM          60,415 msp430-elf-gcc-ar.exe
05/04/2018 01:12 PM          60,415 msp430-elf-gcc-nm.exe
05/04/2018 01:12 PM          60,415 msp430-elf-gcc-ranlib.exe
05/04/2018 01:12 PM          925,532 msp430-elf-gcc.exe
05/04/2018 01:12 PM          488,467 msp430-elf-gcov-dump.exe
05/04/2018 01:12 PM          542,132 msp430-elf-gcov-tool.exe
05/04/2018 01:12 PM          850,599 msp430-elf-gcov.exe
05/04/2018 01:12 PM      5,116,224 msp430-elf-gdb.exe
05/04/2018 01:12 PM          686,285 msp430-elf-gprof.exe
05/04/2018 01:12 PM          870,639 msp430-elf-ld.bfd.exe
05/04/2018 01:12 PM          870,639 msp430-elf-ld.exe
05/04/2018 01:12 PM          634,917 msp430-elf-nm.exe
05/04/2018 01:12 PM          786,652 msp430-elf-objcopy.exe
05/04/2018 01:12 PM          912,990 msp430-elf-objdump.exe
05/04/2018 01:12 PM          649,443 msp430-elf-ranlib.exe
05/04/2018 01:12 PM          471,330 msp430-elf-readelf.exe
05/04/2018 01:12 PM          789,203 msp430-elf-run.exe
05/04/2018 01:12 PM          626,210 msp430-elf-size.exe
05/04/2018 01:12 PM          625,764 msp430-elf-strings.exe
05/04/2018 01:12 PM          786,652 msp430-elf-strip.exe
                29 File(s)      23,317,419 bytes
                2 Dir(s)      60,441,079,808 bytes free

c:\ti\ccsv8\tools\compiler\msp430-gcc-7.3.1.24_win32\bin>
```

Figure 4. Windows Command Prompt: List of GNU Utilities

```
1 c:\ti\ccsv8\tools\compiler\msp430-gcc-7.3.1.24_win32\bin>msp430-elf-readelf.exe --help
2 Usage: readelf <option(s)> elf-file(s)
3 Display information about the contents of ELF format files
4 Options are:
5 -a --all Equivalent to: -h -l -S -s -r -d -V -A -I
6 -h --file-header Display the ELF file header
7 -l --program-headers Display the program headers
```

```

 8      --segments          An alias for --program-headers
 9      -S --section-headers Display the sections' header
10      --sections          An alias for --section-headers
11      -g --section-groups  Display the section groups
12      -t --section-details Display the section details
13      -e --headers         Equivalent to: -h -l -S
14      -s --syms            Display the symbol table
15      --symbols           An alias for --syms
16      --dyn-syms          Display the dynamic symbol table
17      -n --notes           Display the core notes (if present)
18      -r --relocs          Display the relocations (if present)
19      -u --unwind           Display the unwind info (if present)
20      -d --dynamic          Display the dynamic section (if present)
21      -V --version-info    Display the version sections (if present)
22      -A --arch-specific   Display architecture specific information (if any)
23      -c --archive-index   Display the symbol/file index in an archive
24      -D --use-dynamic     Use the dynamic section info when displaying symbols
25      -x --hex-dump=<number|name>
26                          Dump the contents of section <number|name> as bytes
27      -p --string-dump=<number|name>
28                          Dump the contents of section <number|name> as strings
29      -R --relocated-dump=<number|name>
30                          Dump the contents of section <number|name> as relocated bytes
31      -z --decompress      Decompress section before dumping it
32      -w[llIaprmfFsoRt] or
33      --debug-dump[=rawline,=decodedline,=info,=abbrev,=pubnames,=aranges,=macro,=frames,
34      =frames-interp,=str,=loc,=Ranges,=pubtypes,
35      =gdb_index,=trace_info,=trace_abbrev,=trace_aranges,
36      =addr,=cu_index]
37                          Display the contents of DWARF2 debug sections
38      --dwarf-depth=N      Do not display DIEs at depth N or greater
39      --dwarf-start=N      Display DIEs starting with N, at the same depth
40                          or deeper
41      -I --histogram       Display histogram of bucket list lengths
42      -W --wide             Allow output width to exceed 80 characters
43      @<file>              Read options from <file>
44      -H --help            Display this information
45      -v --version          Display the version number of readelf
46      Report bugs to <http://www.sourceware.org/bugzilla/>
47

```

Figure 5. msp430-elf-readelf Utility: Help System.

```

 1  c:\ti\ccsv8\tools\compiler\msp430-gcc-7.3.1.24_win32\bin>msp430-elf-objdump.exe --help
 2  Usage: msp430-elf-objdump.exe <option(s)> <file(s)>
 3  Display information from object <file(s)>.
 4  At least one of the following switches must be given:
 5  -a, --archive-headers   Display archive header information
 6  -f, --file-headers      Display the contents of the overall file header
 7  -p, --private-headers   Display object format specific file header contents
 8  -P, --private=OPT,OPT... Display object format specific contents
 9  -h, --[section]-headers Display the contents of the section headers
10  -x, --all-headers       Display the contents of all headers
11  -d, --disassemble       Display assembler contents of executable sections
12  -D, --disassemble-all  Display assembler contents of all sections
13  -S, --source            Intermix source code with disassembly
14  -s, --full-contents     Display the full contents of all sections requested
15  -g, --debugging         Display debug information in object file
16  -e, --debugging-tags    Display debug information using ctags style
17  -G, --stabs             Display (in raw form) any STABS info in the file

```



```

18 -W[llIaprmfFsoRt] or
19 --dwarf[=rawline,=decodedline,=info,=abbrev,=pubnames,=aranges,=macro,=frames,
20 =frames-interp,=str,=loc,=Ranges,=pubtypes,
21 =gdb_index,=trace_info,=trace_abbrev,=trace_aranges,
22 =addr,=cu_index]
23 Display DWARF info in the file
24 -t, --syms Display the contents of the symbol table(s)
25 -T, --dynamic-syms Display the contents of the dynamic symbol table
26 -r, --reloc Display the relocation entries in the file
27 -R, --dynamic-reloc Display the dynamic relocation entries in the file
28 @<file> Read options from <file>
29 -v, --version Display this program's version number
30 -i, --info List object formats and architectures supported
31 -H, --help Display this information
32
33 The following switches are optional:
34 -b, --target=BFDNAME Specify the target object format as BFDNAME
35 -m, --architecture=MACHINE Specify the target architecture as MACHINE
36 -j, --section=NAME Only display information for section NAME
37 -M, --disassembler-options=OPT Pass text OPT on to the disassembler
38 -EB --endian=big Assume big endian format when disassembling
39 -EL --endian=little Assume little endian format when disassembling
40 --file-start-context Include context from start of file (with -S)
41 -I, --include=DIR Add DIR to search list for source files
42 -l, --line-numbers Include line numbers and filenames in output
43 -F, --file-offsets Include file offsets when displaying information
44 -C, --demangle[=STYLE] Decode mangled/processed symbol names
45 The STYLE, if specified, can be `auto', `gnu',
46 `lucid', `arm', `hp', `edg', `gnu-v3', `java'
47 or `gnat'
48 -w, --wide Format output for more than 80 columns
49 -z, --disassemble-zeroes Do not skip blocks of zeroes when disassembling
50 --start-address=ADDR Only process data whose address is >= ADDR
51 --stop-address=ADDR Only process data whose address is <= ADDR
52 --prefix-addresses Print complete address alongside disassembly
53 --[no-]show-raw-insn Display hex alongside symbolic disassembly
54 --insn-width=WIDTH Display WIDTH bytes on a single line for -d
55 --adjust-vma=OFFSET Add OFFSET to all displayed section addresses
56 --special-syms Include special symbols in symbol dumps
57 --prefix=PREFIX Add PREFIX to absolute paths for -S
58 --prefix-strip=LEVEL Strip initial directory names for -S
59 --dwarf-depth=N Do not display DIEs at depth N or greater
60 --dwarf-start=N Display DIEs starting with N, at the same depth
61 or deeper
62 --dwarf-check Make additional dwarf internal consistency checks.
63
64 msp430-elf-objdump.exe: supported targets: elf32-msp430 elf32-msp430 elf32-little elf32-big
65 plugin srec symbolsrec verilog tekhex binary ihex
66 msp430-elf-objdump.exe: supported architectures: msp:14 MSP430 MSP430x11x1 MSP430x12 MSP430x13
67 MSP430x14 MSP430x15 MSP430x16 MSP430x20 MSP430x21 MSP430x22 MSP430x23 MSP430x24 MSP430x26
68 MSP430x31 MSP430x32 MSP430x33 MSP430x41 MSP430x42 MSP430x43 MSP430x44 MSP430x46 MSP430x47
69 MSP430x54 MSP430X plugin
70 Report bugs to <http://www.sourceware.org/bugzilla/>.

```

Figure 6. msp430-elf-objdump Utility: Help System.

```

1 c:\ti\ccsv8\tools\compiler\msp430-gcc-7.3.1.24_win32\bin>msp430-elf-strings.exe --help
2 Usage: msp430-elf-strings.exe [option(s)] [file(s)]
3 Display printable strings in [file(s)] (stdin by default)
4 The options are:

```

```

5  -a --all          Scan the entire file, not just the data section [default]
6  -d --data        Only scan the data sections in the file
7  -f --print-file-name Print the name of the file before each string
8  -n --bytes=[number] Locate & print any NUL-terminated sequence of at
9  <number>         least [number] characters (default 4).
10 -t --radix={o,d,x} Print the location of the string in base 8, 10 or 16
11 -w --include-all-whitespace Include all whitespace as valid string characters
12 -o              An alias for --radix=o
13 -T --target=<BFDNAME> Specify the binary file format
14 -e --encoding={s,S,b,l,B,L} Select character size and endianness:
15                s = 7-bit, S = 8-bit, {b,l} = 16-bit, {B,L} = 32-bit
16 -s --output-separator=<string> String used to separate strings in output.
17 @<file>         Read options from <file>
18 -h --help        Display this information
19 -v -V --version  Print the program's version number
20 msp430-elf-strings.exe: supported targets: elf32-msp430 elf32-msp430 elf32-little elf32-big
21 plugin srec symbolsrec verilog tekhex binary ihex
22 Report bugs to <http://www.sourceware.org/bugzilla/>

```

Figure 7. msp430-elf-strings Utility: Help System.

4 Deconstructing Executable Files: An Example

To demonstrate software reverse engineering in practice, let us start from a C program described in Figure 8. This program toggles the LEDs connected to ports P2.1 and P2.2 on the TI Experimenter's board. Our first step is to compile this program using GNU C compiler that comes as an Add-on in TI's Code Composer Studio. To compile this program we select the GNU C compiler and set appropriate compilation flags as shown in Figure 9.

```

1  /*****
2  *   File:          ToggleLEDs.c
3  *   Description:  Program toggles LED1 and LED2 by
4  *               xoring port pins inside of an infinite loop.
5  *   Board:        MSP430FG461x/F20xx Experimenter Board
6  *   Clocks:       ACLK = 32.768kHz, MCLK = SMCLK = default DCO
7  *
8  *               MSP430FG461x
9  *
10 *               /|\
11 *               | |
12 *               --| RST
13 *
14 *               |
15 *               |           P2.1 |--> LED2
16 *               |           P2.2 |--> LED1
17 *
18 *   Author: Alex Milenkovich, milenkovic@computer.org
19 *   Date:   September 2010
20 *****/
21 #include <msp430.h>
22 int main(void) {
23     WDCTL = WDTPW + WDTHOLD;    // Stop watchdog timer
24     P2DIR |= (BIT1 | BIT2);     // Set P2.1 and P2.2 to output direction (0000_0110)
25     P2OUT = 0x00;              // Clear output port P2, P2OUT=0000_0000b
26     for (;;) {

```

```

27     unsigned int i;
28     P2OUT ^= (BIT1 | BIT2);    // Toggle P2.1 and P2.2 using exclusive-OR
29     for(i = 0; i < 50000; i++); // Software delay (13 cc per iteration)
30     /* Total delay on average 13 cc*50,000 = 750,000; 750,000 * 1us = 0.75 s */
31 }
32 return 0;
33 }

```

Figure 8. ToggleLEDs Source Code.

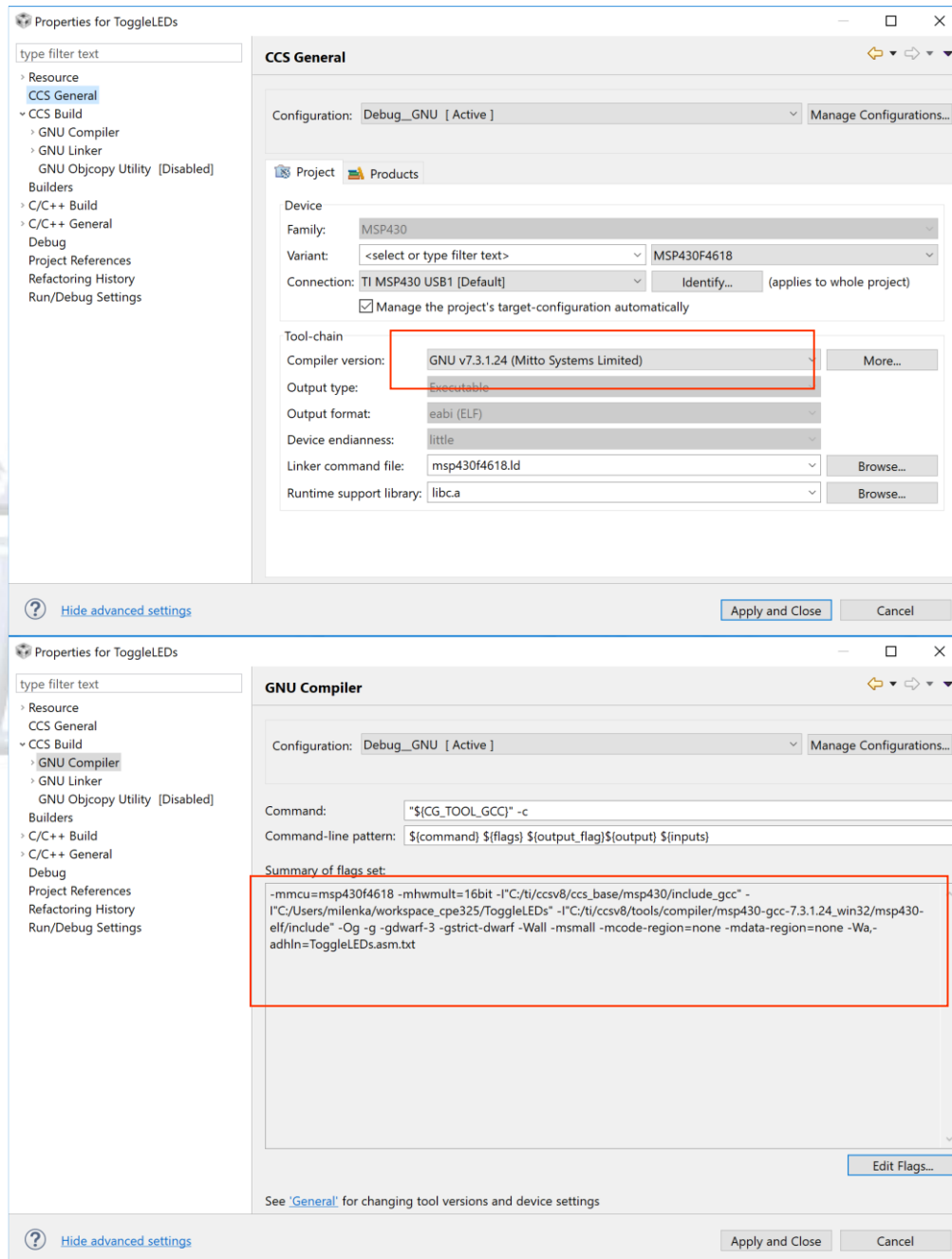


Figure 9. Settings for GNU C Compiler.

As the result of compilation you will notice ToggleLEDs.o (object file), ToggleLEDs.out (executable file), and ToggleLEDs.asm.txt (assembly code created by the compiler switches – Wa,-adhIn=ToggleLEDs.asm.txt). Figure 10 shows the output list file with assembly code for each line of the source code in C.

```

1      1      .file "ToggleLEDs.c"
2      2      .text
3      3      .Ltext0:
4      4      .balign 2
5      5      .globalmain
6      6      main:
7      7      .LFB0:
8      8      .file 1 "../ToggleLEDs.c"
9      9      1:../ToggleLEDs.c ****
10     /*****
11     2:../ToggleLEDs.c **** *   File:      ToggleLEDs.c
12     3:../ToggleLEDs.c **** *   Description: Program toggles LED1 and LED2 by
13     4:../ToggleLEDs.c **** *   xoring port pins inside of an infinite loop.
14     5:../ToggleLEDs.c **** *   Board:      MSP430FG461x/F20xx Experimenter Board
15     6:../ToggleLEDs.c **** *   Clocks:     ACLK = 32.768kHz, MCLK = SMCLK = default DCO
16     7:../ToggleLEDs.c **** *
17     8:../ToggleLEDs.c **** *
18     9:../ToggleLEDs.c **** *
19     10:../ToggleLEDs.c **** *
20     11:../ToggleLEDs.c **** *
21     12:../ToggleLEDs.c **** *
22     13:../ToggleLEDs.c **** *
23     14:../ToggleLEDs.c **** *
24     15:../ToggleLEDs.c **** *
25     16:../ToggleLEDs.c **** *
26     17:../ToggleLEDs.c **** *
27     18:../ToggleLEDs.c **** *
28     19:../ToggleLEDs.c **** *
29     *****/
30     20:../ToggleLEDs.c **** #include <msp430.h>
31     21:../ToggleLEDs.c ****
32     22:../ToggleLEDs.c **** int main(void)
33     23:../ToggleLEDs.c **** {
34     10     .loc 1 23 0
35     11     ; start of function
36     12     ; framesize_regs:    0
37     13     ; framesize_locals:  0
38     14     ; framesize_outgoing: 0
39     15     ; framesize:         0
40     16     ; elim ap -> fp      2
41     17     ; elim fp -> sp      0
42     18     ; saved regs:(none)
43     19     ; start of prologue
44     20     ; end of prologue
45     24:../ToggleLEDs.c ****   WDCTL = WDTPW + WDTHOLD;    // Stop watchdog timer
46     21     .loc 1 24 0
47     22 0000 B240 805A   MOV.W #23168, &WDCTL
48     22     0000
49     25:../ToggleLEDs.c ****   P2DIR |= (BIT1 | BIT2);    // Set P2.1 and P2.2 to output
50     direction (0000_0110)
51     23     .loc 1 25 0
52     24 0006 F2D0 0600   BIS.B #6, &P2DIR
53     24     0000

```

```

54 26:../ToggleLEDs.c **** P2OUT = 0x00; // Clear output port P2,
55 P2OUT=0000_0000b
56 25 .loc 1 26 0
57 26 000c C243 0000 MOV.B #0, &P2OUT
58 27 0010 3040 0000 BR #.L4
59 28 .LVL0:
60 29 .L3:
61 30 .LBB2:
62 27:../ToggleLEDs.c **** for (;;) {
63 28:../ToggleLEDs.c **** unsigned int i;
64 29:../ToggleLEDs.c **** P2OUT ^= (BIT1 | BIT2); // Toggle P2.1 and P2.2 using
65 exclusive-OR
66 30:../ToggleLEDs.c **** for(i = 0; i < 50000; i++); // Software delay (13 cc per
67 iteration)
68 31 .loc 1 30 0
69 32 0014 1C53 ADD.W #1, R12
70 33 .LVL1:
71 34 .L2:
72 35 .loc 1 30 0 is_stmt 0
73 36 0016 3D40 4FC3 MOV.W #-15537, R13
74 37 001a 0D9C 002C CMP.W R12, R13 { JHS .L3
75 38 .LVL2:
76 39 .L4:
77 29:../ToggleLEDs.c **** for(i = 0; i < 50000; i++); // Software delay (13 cc per
78 iteration)
79 40 .loc 1 29 0 is_stmt 1
80 41 001e F2E0 0600 XOR.B #6, &P2OUT
81 41 0000
82 42 .LVL3:
83 43 .loc 1 30 0
84 44 0024 4C43 MOV.B #0, R12
85 45 0026 3040 0000 BR #.L2
86 46 .LBE2:
87 47 .LFE0:
88 75 .letext0:
89 76 .file 2 "C:/ti/ccsv8/ccs_base/msp430/include_gcc/msp430f4618.h"

```

Figure 10. Output Assembly Code Generated by GNU GCC Compiler

For the moment, let us assume that we are given the executable file and that we have no prior knowledge what that executable code is doing. Here we will demonstrate steps we can take to deconstruct or reverse engineer code from the executable file.

Step #1: Examine ELF header to determine type of machine code, data representation, entry points and more. We can use `msp430-elf-readelf` to learn more about the executable file. Switch `--file-header` displays information about the ELF header: this is an ELF32 executable file, containing code for MSP430 microcontroller, the entry program point is at address `0x310c`, and so on (see Figure 11).

```

1 C:\Users\milenka\workspace_cpe325\ToggleLEDs\Debug_GNU>msp430-elf-readelf -h ToggleLEDs.out
2 ELF Header:
3 Magic: 7f 45 4c 46 01 01 01 ff 00 00 00 00 00 00 00
4 Class: ELF32
5 Data: 2's complement, little endian
6 Version: 1 (current)
7 OS/ABI: Standalone App

```

```

8     ABI Version:                0
9     Type:                      EXEC (Executable file)
10    Machine:                   Texas Instruments msp430 microcontroller
11    Version:                   0x1
12    Entry point address:       0x310c
13    Start of program headers:  52 (bytes into file)
14    Start of section headers:  12920 (bytes into file)
15    Flags:                      0x2d: architecture variant: MSP430X
16    Size of this header:       52 (bytes)
17    Size of program headers:    32 (bytes)
18    Number of program headers:  5
19    Size of section headers:    40 (bytes)
20    Number of section headers:  25
21    Section header string table index: 22

```

Figure 11. msp430-elf-readelf --file-header (-h): ELF Header Content for ToggleLEDs.out

Step #2. Examine ELF file sections.

We can use msp430-elf-readelf utility with --section-headers switch to display information about all sections. Figure 12 shows the output of this command for ToggleLEDs.out. A similar information can be obtained using objdump utility with -h switch as shown in Figure 13. The list of sections includes the section name, the starting addresses (VMA – virtual and LMA – load memory address), the offset of the section in the actual file, the size of the section, the section attributes, and the alignment in memory.

The __reset_vector, .rodata, and .text sections reside in the Flash memory (read only). The .rodata2 starts at the address 0x3100 and the size is 12 bytes (0x000c). The .lowtext starts at 0x310c (right after that) and has the size of 0x66 bytes (102 bytes). It is followed by the .text section that starts at 0x3172 and contain 0x146 bytes. The RAM memory region consists of .data and .bss sections. The .bss section starts at address 0x1100 and occupies 0x12 (18) bytes, followed by the .heap section at 0x1112. Another noteworthy entry is __reset_vector that sits at the address 0xFFFFE.

```

1 C:\Users\milenka\workspace_cpe325\ToggleLEDs\Debug__GNU>msp430-elf-readelf --section-headers
2 ToggleLEDs.out
3 There are 25 section headers, starting at offset 0x3278:
4
5 Section Headers:
6 [Nr] Name                Type                Addr    Off    Size  ES Flg Lk Inf Al
7 [ 0]                      NULL               00000000 000000 000000 00   0  0  0  0
8 [ 1] __reset_vector         PROGBITS           0000ffff 00028e 000002 00   A  0  0  1
9 [ 2] .lower.rodata          PROGBITS           00003100 000290 000000 00   W  0  0  1
10 [ 3] .rodata                PROGBITS           00003100 000290 000000 00   WA 0  0  1
11 [ 4] .rodata2               PROGBITS           00003100 0000d4 00000c 00   WA 0  0  4
12 [ 5] .data                  PROGBITS           00001100 000290 000000 00   WA 0  0  1
13 [ 6] .bss                   NOBITS             00001100 0000e0 000012 00   WA 0  0  2
14 [ 7] .noinit                PROGBITS           00001112 000290 000000 00   W  0  0  1
15 [ 8] .heap                  NOBITS             00001112 0000e2 000004 00   WA 0  0  1
16 [ 9] .lowtext               PROGBITS           0000310c 0000e0 000066 00   AX 0  0  1
17 [10] .lower.text            PROGBITS           00003172 000290 000000 00   W  0  0  1
18 [11] .text                  PROGBITS           00003172 000146 000146 00   AX 0  0  2
19 [12] .upper.text            PROGBITS           00010000 000290 000000 00   W  0  0  1
20 [13] .MSP430.attribute     MSP430_ATTRIBUT   00000000 000290 000017 00   0  0  0  1
21 [14] .comment               PROGBITS           00000000 0002a7 000039 01   MS 0  0  1
22 [15] .debug_aranges         PROGBITS           00000000 0002e0 000020 00   0  0  0  1

```

```

23 [16] .debug_info      PROGBITS      00000000 000300 000d42 00      0 0 1
24 [17] .debug_abbrev      PROGBITS      00000000 001042 0000a6 00      0 0 1
25 [18] .debug_line        PROGBITS      00000000 0010e8 0000a5 00      0 0 1
26 [19] .debug_frame       PROGBITS      00000000 001190 000024 00      0 0 4
27 [20] .debug_str         PROGBITS      00000000 0011b4 0007f8 01 MS  0 0 1
28 [21] .debug_loc         PROGBITS      00000000 0019ac 000013 00      0 0 1
29 [22] .shstrtab          STRTAB        00000000 003187 0000ef 00      0 0 1
30 [23] .symtab            SYMTAB        00000000 0019c0 000f20 10      24 196 4
31 [24] .strtab            STRTAB        00000000 0028e0 0008a7 00      0 0 1

```

32 Key to Flags:

```

33 W (write), A (alloc), X (execute), M (merge), S (strings)
34 I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
35 0 (extra OS processing required) o (OS specific), p (processor specific)

```

Figure 12. msp430-elf-readelf -section-headers (-S): ELF section headers for ToggleLEDs.out

```

1 C:\Users\milenka\workspace_cpe325\ToggleLEDs\Debug__GNU>msp430-elf-objdump -h ToggleLEDs.out
2
3 ToggleLEDs.out:      file format elf32-msp430
4
5 Sections:
6 Idx Name              Size      VMA      LMA      File off  Algn
7  0 __reset_vector     00000002  0000ffff  0000ffff  0000028e  2**0
8                      CONTENTS, ALLOC, LOAD, READONLY, DATA
9  1 .lower.rodata       00000000  00003100  00003100  00000290  2**0
10                     CONTENTS
11  2 .rodata            00000000  00003100  00003100  00000290  2**0
12                     CONTENTS, ALLOC, LOAD, DATA
13  3 .rodata2           0000000c  00003100  00003100  000000d4  2**2
14                     CONTENTS, ALLOC, LOAD, DATA
15  4 .data              00000000  00001100  00001100  00000290  2**0
16                     CONTENTS, ALLOC, LOAD, DATA
17  5 .bss               00000012  00001100  0000310c  000000e0  2**1
18                     ALLOC
19  6 .noinit            00000000  00001112  00001112  00000290  2**0
20                     CONTENTS
21  7 .heap              00000004  00001112  0000310c  000000e2  2**0
22                     ALLOC
23  8 .lowtext           00000066  0000310c  0000310c  000000e0  2**0
24                     CONTENTS, ALLOC, LOAD, READONLY, CODE
25  9 .lower.text        00000000  00003172  00003172  00000290  2**0
26                     CONTENTS
27 10 .text              00000146  00003172  00003172  00000146  2**1
28                     CONTENTS, ALLOC, LOAD, READONLY, CODE
29 11 .upper.text        00000000  00010000  00010000  00000290  2**0
30                     CONTENTS
31 12 .MSP430.attributes 00000017  00000000  00000000  00000290  2**0
32                     CONTENTS, READONLY
33 13 .comment           00000039  00000000  00000000  000002a7  2**0
34                     CONTENTS, READONLY
35 14 .debug_aranges     00000020  00000000  00000000  000002e0  2**0
36                     CONTENTS, READONLY, DEBUGGING
37 15 .debug_info        00000d42  00000000  00000000  00000300  2**0
38                     CONTENTS, READONLY, DEBUGGING
39 16 .debug_abbrev      000000a6  00000000  00000000  00001042  2**0
40                     CONTENTS, READONLY, DEBUGGING
41 17 .debug_line        000000a5  00000000  00000000  000010e8  2**0
42                     CONTENTS, READONLY, DEBUGGING
43 18 .debug_frame       00000024  00000000  00000000  00001190  2**2
44                     CONTENTS, READONLY, DEBUGGING

```

```

45 19 .debug_str 000007f8 00000000 00000000 000011b4 2**0
46          CONTENTS, READONLY, DEBUGGING
47 20 .debug_loc 00000013 00000000 00000000 000019ac 2**0
48          CONTENTS, READONLY, DEBUGGING

```

Figure 13. msp430-elf-objdump -h: ELF section headers for ToggleLEDs.out

Step #3. Display ELF symbols.

We use msp430-elf-readelf utility with --symbols switch (or -s) to display all symbols in the ELF file. Figure 15 shows a filtered output of this utility for ToggleLEDs.out (the full list contains 241 symbols). A similar output can be obtained by using msp430-elf-objdump utility with switch -t or by using a separate binutils utility msp430-elf-nm. By searching the output you can identify important symbols such as '_start', '__stack', '__heap_start__', '_bssstart', 'main', and others. These sections and their locations are defined in the linker script file for the given microcontroller as the placement is a function of the size and mapping of Flash and RAM memory.

```

1
2 Symbol table '.symtab' contains 242 entries:
3   Num:   Value   Size Type   Bind   Vis     Ndx Name
4     0: 00000000    0 NOTYPE LOCAL  DEFAULT UND
5     1: 0000ffff    0 SECTION LOCAL  DEFAULT 1
6     2: 00003100    0 SECTION LOCAL  DEFAULT 2
7     3: 00003100    0 SECTION LOCAL  DEFAULT 3
8   .....
9   195: 0000310c    0 NOTYPE LOCAL  DEFAULT 9 _start
10  196: 0000323e   56 FUNC   GLOBAL DEFAULT 11 memmove
11  197: 000032b8    0 OBJECT GLOBAL  HIDDEN 11 __TMC_END__
12  198: 0000310a    0 OBJECT GLOBAL  HIDDEN 4  __DTOR_END__
13  199: 00003100    0 NOTYPE GLOBAL  DEFAULT 3  __fini_array_end
14  200: 0000002a    0 NOTYPE GLOBAL  DEFAULT ABS P2DIR
15  201: 00000120    0 NOTYPE GLOBAL  DEFAULT ABS WDTCTL
16  202: 0000310c    4 FUNC   GLOBAL  DEFAULT 9  __crt0_start
17  203: 00001116    0 NOTYPE GLOBAL  DEFAULT 8  __HeapLimit
18  204: 00001116    0 NOTYPE GLOBAL  DEFAULT 8  __heap_end__
19  205: 00003110   14 FUNC   GLOBAL  DEFAULT 9  __crt0_init_bss
20  206: 00000012    0 NOTYPE GLOBAL  DEFAULT ABS __bsssize
21  207: 00003132   10 FUNC   GLOBAL  DEFAULT 9  __crt0_call_init_then_mai
22  208: 00000000    0 NOTYPE WEAK    DEFAULT UND __deregister_frame_info
23  209: 00000000    0 NOTYPE WEAK    DEFAULT UND __ITM_registerTMCloneTable
24  210: 00003154    0 FUNC   GLOBAL  DEFAULT 9  _msp430_run_fini_array
25  211: 00000000    0 NOTYPE GLOBAL  DEFAULT ABS __romdatacopysize
26  212: 00000029    0 NOTYPE GLOBAL  DEFAULT ABS P2OUT
27  213: 00000000    0 NOTYPE WEAK    DEFAULT UND __ITM_deregisterTMCloneTab
28  214: 00003100    0 NOTYPE GLOBAL  DEFAULT 3  __fini_array_start
29  215: 00000000    0 NOTYPE WEAK    DEFAULT ABS __rom_highdatacopysize
30  216: 0000329c    0 NOTYPE GLOBAL  DEFAULT 11 __msp430_init
31  217: 00003272   20 FUNC   GLOBAL  DEFAULT 11 memset
32  218: 00003218   42 FUNC   GLOBAL  DEFAULT 11 main
33  219: 00003100    0 NOTYPE GLOBAL  DEFAULT 3  __init_array_end
34  220: 00001112    0 NOTYPE GLOBAL  DEFAULT 8  __heap_start__
35  221: 00000000    0 NOTYPE WEAK    DEFAULT ABS __high_bsssize
36  222: 00000000    0 NOTYPE WEAK    DEFAULT ABS __rom_highdatastart
37  223: 000032b8    0 NOTYPE GLOBAL  DEFAULT 11 __msp430_fini_end
38  224: 0000310c    0 NOTYPE GLOBAL  DEFAULT ABS __romdatastart
39  225: 0000313c    0 FUNC   GLOBAL  DEFAULT 9  _msp430_run_init_array

```



```

40      226: 00003100      0 NOTYPE GLOBAL DEFAULT      3 __preinit_array_end
41      227: 00000000      0 NOTYPE WEAK  DEFAULT ABS  __high_datastart
42      228: 00000000      0 NOTYPE WEAK  DEFAULT ABS  __upper_data_init
43      229: 00001100      0 NOTYPE GLOBAL DEFAULT      6 __bssstart
44      230: 00003100      0 NOTYPE GLOBAL DEFAULT      8 __stack
45      231: 00001100      0 NOTYPE GLOBAL DEFAULT      5 _edata
46      232: 00001112      0 NOTYPE GLOBAL DEFAULT      8 _end
47      233: 000032ae      0 NOTYPE GLOBAL DEFAULT     11 __msp430_init_end
48      234: 00000000      0 NOTYPE WEAK  DEFAULT ABS  __high_bssstart
49      235: 00003100      0 NOTYPE GLOBAL DEFAULT      3 __init_array_start
50      236: 00001100      0 NOTYPE GLOBAL DEFAULT      5 _datastart
51      237: 00003100      0 NOTYPE GLOBAL DEFAULT      3 __preinit_array_start
52      238: 0000311e     20 FUNC  GLOBAL DEFAULT      9 __crt0_movedata
53      239: 00000000      0 NOTYPE WEAK  DEFAULT UND  __register_frame_info
54      240: 00003148      0 FUNC  GLOBAL DEFAULT      9 _msp430_run_preinit_array
55      241: 000032ae      0 NOTYPE GLOBAL DEFAULT     11 __msp430_fini

```

Figure 14. msp430-elf-readelf --symbols: ELF symbols for ToggleLEDs.out

Step #4. Display ELF segments.

We use msp430-elf-readelf utility with --program-headers switch (or --segments) to display all segments that are loadable into the memory. Figure 15 shows the output of this utility for ToggleLEDs.out. It shows information about loadable segments in the memory, namely .rodata2 (Flash memory), .bss and .heap (RAM memory), and .text and __reset_vector (Flash memory).

```

1 C:\Users\milenka\workspace_cpe325\ToggleLEDs\Debug__GNU>msp430-elf-readelf --program-headers
2 ToggleLEDs.out
3
4 Elf file type is EXEC (Executable file)
5 Entry point 0x310c
6 There are 5 program headers, starting at offset 52
7
8 Program Headers:
9   Type      Offset      VirtAddr    PhysAddr    FileSiz MemSiz  Flg Align
10  LOAD       0x000000    0x0000302c  0x0000302c  0x000e0 0x000e0 RW  0x4
11  LOAD       0x0000e0    0x00001100  0x0000310c  0x00000 0x00012 RW  0x4
12  LOAD       0x0000e2    0x00001112  0x0000310c  0x00000 0x00004 RW  0x4
13  LOAD       0x0000e0    0x0000310c  0x0000310c  0x001ac 0x001ac R E 0x4
14  LOAD       0x00028e    0x0000fffe  0x0000fffe  0x00002 0x00002 R   0x4
15
16 Section to Segment mapping:
17 Segment Sections...
18  00  .rodata2
19  01  .bss
20  02  .heap
21  03  .lowtext .text
22  04  __reset_vector

```

Figure 15. msp430-elf-readelf: ELF Program Headers or Segments for ToggleLEDs.out (-l or --program-headers)

Step #5. Disassemble the code.

Now we are ready to take additional steps toward deconstructing the text segment that contains the code. We use msp430-elf-objdump -S to dump source code together with disassembly. Note: this is a slight deviation from our assumption that source code is not available. Similar results can be obtained using -d (disassembly) that does not assume that

source code is present. Figure 16 shows the result of disassembling operation of the text segment of ToggleLEDs.out executable file. The first thing we can notice is that the first instruction differs from the one shown in Figure 10. The entry point in the program is as expected 0x310c, but the first instruction is the one to initialize the stack pointer, rather than to stop the watchdog timer. This is because the compiler inserts so-called start-up code that proceeds the main code. Thus, first instruction is actually moving the symbol that corresponds to the label `__stack` (the location above physical RAM) into R1 (stack pointer). Then comes the first label `'__crt0_init_bss'`. The code following this label does three things, moves `__bssstart` to R12, clears R13, and then moves `__bsssize` to R14. Next the subroutine at #0x3272 is called. Can you find what symbol is associated with that address?

This is the `memset` function. Its prototype is as follows:

```
int memset(void *ptr, int fill, size_t nbytes)
```

We can deduce that the `memset` is called with the following parameters:

```
int memset(__bssstart, 0, __bsssize)
```

Thus, this function clears the `.bss` section, as the function name indicates.

Next we have the label `'__crt0_movedata'`. The symbol `__datastart` is moved to r12, `__romdatastart` is moved into r13 and `__romdatacopysize` is moved into r14. Then `memmove` is called that will copy the data section from Flash to RAM, so it can be accessed and modified as required. The `memmove` is called with the following parameters:

```
memmove(__datastart, __romdatastart, __romdatacopysize);
```

Next, `'__crt0_call_init_then_main'` is called, which sets up some C++ exception handlers or may perform initialization of the standard C library tasks. Finally, the main function is executed, starting at the address 0x3218.

```
1
2 ToggleLEDs.out:      file format elf32-msp430
3
4
5 Disassembly of section .lowtext:
6
7 0000310c <__crt0_start>:
8   310c:    31 40 00 31    mov     #12544, r1      ;#0x3100
9
10 00003110 <__crt0_init_bss>:
11   3110:    3c 40 00 11    mov     #4352, r12     ;#0x1100
12
13 00003114 <.Loc.74.1>:
14   3114:    0d 43         clr     r13           ;
15
16 00003116 <.Loc.75.1>:
17   3116:    3e 40 12 00    mov     #18, r14      ;#0x0012
18
```

```

19 0000311a <.Loc.79.1>:
20     311a:    b0 12 72 32    call   #12914        ;#0x3272
21
22 0000311e <__crt0_movedata>:
23     311e:    3c 40 00 11    mov    #4352, r12    ;#0x1100
24
25 00003122 <.Loc.116.1>:
26     3122:    3d 40 0c 31    mov    #12556, r13   ;#0x310c
27
28 00003126 <.Loc.119.1>:
29     3126:    0d 9c          cmp    r12, r13     ;
30
31 00003128 <.Loc.120.1>:
32     3128:    04 24          jz     $+10         ;abs 0x3132
33
34 0000312a <.Loc.122.1>:
35     312a:    3e 40 00 00    mov    #0, r14       ;
36
37 0000312e <.Loc.124.1>:
38     312e:    b0 12 3e 32    call   #12862        ;#0x323e
39
40 00003132 <__crt0_call_init_then_main>:
41     3132:    b0 12 9c 32    call   #12956        ;#0x329c
42
43 00003136 <.Loc.196.1>:
44     3136:    0c 43          clr   r12           ;
45
46 00003138 <.Loc.197.1>:
47     3138:    b0 12 18 32    call   #12824        ;#0x3218
48
49 0000313c <_msp430_run_init_array>:
50     313c:    34 40 00 31    mov    #12544, r4    ;#0x3100
51
52 00003140 <.Loc.224.1>:
53     3140:    35 40 00 31    mov    #12544, r5    ;#0x3100
54
55 00003144 <.Loc.225.1>:
56     3144:    26 43          mov    #2, r6        ;r3 As==10
57
58 00003146 <.Loc.226.1>:
59     3146:    0d 3c          jmp    $+28         ;abs 0x3162
60
61 00003148 <_msp430_run_preinit_array>:
62     3148:    34 40 00 31    mov    #12544, r4    ;#0x3100
63
64 0000314c <.Loc.232.1>:
65     314c:    35 40 00 31    mov    #12544, r5    ;#0x3100
66
67 00003150 <.Loc.233.1>:
68     3150:    26 43          mov    #2, r6        ;r3 As==10
69
70 00003152 <.Loc.234.1>:
71     3152:    07 3c          jmp    $+16         ;abs 0x3162
72
73 00003154 <_msp430_run_fini_array>:
74     3154:    34 40 00 31    mov    #12544, r4    ;#0x3100
75
76 00003158 <.Loc.240.1>:
77     3158:    35 40 00 31    mov    #12544, r5    ;#0x3100
78
79 0000315c <.Loc.241.1>:

```

```

80      315c:    36 40 fe ff  mov    #65534, r6    ;#0xffffe
81
82      00003160 <.Loc.242.1>:
83      3160:    00 3c          jmp    $+2          ;abs 0x3162
84
85      00003162 <_msp430_run_array>:
86      3162:    05 94          cmp    r4, r5      ;
87
88      00003164 <.Loc.246.1>:
89      3164:    05 24          jz     $+12         ;abs 0x3170
90
91      00003166 <.Loc.247.1>:
92      3166:    27 44          mov    @r4, r7     ;
93
94      00003168 <.Loc.248.1>:
95      3168:    04 56          add   r6, r4       ;
96
97      0000316a <.Loc.249.1>:
98      316a:    a7 12          call  @r7          ;
99
100     0000316c <.Loc.250.1>:
101     316c:    10 40 f4 ff  br     0xffff4     ;PC rel. 0x3162
102
103     00003170 <_msp430_run_done>:
104     3170:    30 41          ret
105
106     Disassembly of section .text:
107
108     00003172 <deregister_tm_clones>:
109     3172:    3c 40 b8 32  mov    #12984, r12   ;#0x32b8
110     3176:    3c 90 b8 32  cmp    #12984, r12   ;#0x32b8
111     317a:    07 24          jz     $+16         ;abs 0x318a
112     317c:    3d 40 00 00  mov    #0, r13      ;
113     3180:    0d 93          cmp    #0, r13      ;r3 As==00
114     3182:    03 24          jz     $+8          ;abs 0x318a
115     3184:    3c 40 b8 32  mov    #12984, r12   ;#0x32b8
116     3188:    8d 12          call  r13          ;
117
118     0000318a <.L1>:
119     318a:    30 41          ret
120
121     0000318c <register_tm_clones>:
122     318c:    3d 40 b8 32  mov    #12984, r13   ;#0x32b8
123     3190:    3d 80 b8 32  sub    #12984, r13   ;#0x32b8
124     3194:    0d 11          rra    r13          ;
125     3196:    0c 4d          mov    r13, r12     ;
126     3198:    5c 03          rrum   #1, r12      ;
127     319a:    4d 18 0c 11  rpt   #14 { rrax.w  r12          ;
128     319e:    0d 5c          add   r12, r13     ;
129     31a0:    0d 11          rra    r13          ;
130     31a2:    0d 93          cmp    #0, r13      ;r3 As==00
131     31a4:    07 24          jz     $+16         ;abs 0x31b4
132     31a6:    3e 40 00 00  mov    #0, r14      ;
133     31aa:    0e 93          cmp    #0, r14      ;r3 As==00
134     31ac:    03 24          jz     $+8          ;abs 0x31b4
135     31ae:    3c 40 b8 32  mov    #12984, r12   ;#0x32b8
136     31b2:    8e 12          call  r14          ;
137
138     000031b4 <.L9>:
139     31b4:    30 41          ret
140

```

```

141 000031b6 <__do_global_dtors_aux>:
142 31b6: 1a 15      pushm #2, r10 ;16-bit words
143 31b8: c2 93 00 11  cmp.b #0, &0x1100;r3 As==00
144 31bc: 17 20      jnz $+48 ;abs 0x31ec
145 31be: 3a 40 0a 31  mov #12554, r10 ;#0x310a
146 31c2: 3a 80 08 31  sub #12552, r10 ;#0x3108
147 31c6: 0a 11      rra r10 ;
148 31c8: 3a 53      add #-1, r10 ;r3 As==11
149 31ca: 39 40 08 31  mov #12552, r9 ;#0x3108
150
151 000031ce <.L19>:
152 31ce: 1c 42 02 11  mov &0x1102, r12 ;0x1102
153 31d2: 0c 9a      cmp r10, r12 ;
154 31d4: 0d 28      jnc $+28 ;abs 0x31f0
155 31d6: b0 12 72 31  call #12658 ;#0x3172
156 31da: 3d 40 00 00  mov #0, r13 ;
157 31de: 0d 93      cmp #0, r13 ;r3 As==00
158 31e0: 03 24      jz $+8 ;abs 0x31e8
159 31e2: 3c 40 00 31  mov #12544, r12 ;#0x3100
160 31e6: 8d 12      call r13 ;
161
162 000031e8 <.L21>:
163 31e8: d2 43 00 11  mov.b #1, &0x1100;r3 As==01
164
165 000031ec <.L17>:
166 31ec: 19 17      popm #2, r10 ;16-bit words
167 31ee: 30 41      ret
168
169 000031f0 <.L20>:
170 31f0: 1c 53      inc r12 ;
171 31f2: 82 4c 02 11  mov r12, &0x1102;
172 31f6: 0c 5c      rla r12 ;
173 31f8: 0c 59      add r9, r12 ;
174 31fa: 2c 4c      mov @r12, r12 ;
175 31fc: 8c 12      call r12 ;
176 31fe: e7 3f      jmp $-48 ;abs 0x31ce
177
178 00003200 <frame_dummy>:
179 3200: 3e 40 00 00  mov #0, r14 ;
180 3204: 0e 93      cmp #0, r14 ;r3 As==00
181 3206: 05 24      jz $+12 ;abs 0x3212
182 3208: 3d 40 04 11  mov #4356, r13 ;#0x1104
183 320c: 3c 40 00 31  mov #12544, r12 ;#0x3100
184 3210: 8e 12      call r14 ;
185
186 00003212 <.L27>:
187 3212: b0 12 8c 31  call #12684 ;#0x318c
188 3216: 30 41      ret
189
190 00003218 <main>:
191 *****/
192 #include <msp430.h>
193
194 int main(void)
195 {
196     WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
197     3218: b2 40 80 5a  mov #23168, &0x0120;#0x5a80
198     321c: 20 01
199
200 0000321e <.Loc.25.1>:
201     P2DIR |= (BIT1 | BIT2); // Set P2.1 and P2.2 to output direction (0000_0110)

```

```

202      321e:    f2 d0 06 00    bis.b #6,    &0x002a;
203      3222:    2a 00
204
205 00003224 <.Loc.26.1>:
206      P2OUT = 0x00;           // Clear output port P2, P2OUT=0000_0000b
207      3224:    c2 43 29 00    mov.b #0,    &0x0029;r3 As==00
208      3228:    05 3c          jmp    $+12    ;abs 0x3234
209
210 0000322a <.L3>:
211      for (;;) {
212          unsigned int i;
213          P2OUT ^= (BIT1 | BIT2); // Toggle P2.1 and P2.2 using exclusive-OR
214          for(i = 0; i < 50000; i++); // Software delay (13 cc per iteration)
215      322a:    1c 53          inc    r12    ;
216
217 0000322c <.L2>:
218      322c:    3d 40 4f c3    mov    #49999,r13    ;#0xc34f
219      3230:    0d 9c          cmp    r12,    r13    ;
220      3232:    fb 2f          jc     $-8        ;abs 0x322a
221
222 00003234 <.L4>:
223      WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
224      P2DIR |= (BIT1 | BIT2); // Set P2.1 and P2.2 to output direction (0000_0110)
225      P2OUT = 0x00;           // Clear output port P2, P2OUT=0000_0000b
226      for (;;) {
227          unsigned int i;
228          P2OUT ^= (BIT1 | BIT2); // Toggle P2.1 and P2.2 using exclusive-OR
229      3234:    f2 e0 06 00    xor.b #6,    &0x0029;
230      3238:    29 00
231
232 0000323a <.Loc.30.1>:
233      for(i = 0; i < 50000; i++); // Software delay (13 cc per iteration)
234      323a:    4c 43          clr.b r12    ;
235      323c:    f7 3f          jmp    $-16    ;abs 0x322c
236
237 0000323e <memmove>:
238      323e:    0d 9c          cmp    r12,    r13    ;
239      3240:    09 28          jnc   $+20    ;abs 0x3254
240
241 00003242 <L0>:
242      3242:    0f 4c          mov    r12,    r15    ;
243      3244:    0e 5c          add    r12,    r14    ;
244
245 00003246 <.L3>:
246      3246:    0e 9f          cmp    r15,    r14    ;
247      3248:    0c 24          jz    $+26    ;abs 0x3262
248
249 0000324a <.LVL3>:
250      324a:    ef 4d 00 00    mov.b @r13,  0(r15) ;
251      324e:    1f 53          inc    r15    ;
252
253 00003250 <.LVL4>:
254      3250:    1d 53          inc    r13    ;
255      3252:    f9 3f          jmp    $-12    ;abs 0x3246
256
257 00003254 <.L2>:
258      3254:    0f 4d          mov    r13,    r15    ;
259      3256:    0f 5e          add    r14,    r15    ;
260      3258:    0c 9f          cmp    r15,    r12    ;
261      325a:    f3 2f          jc     $-24    ;abs 0x3242
262

```

```

263 0000325c <.L4>:
264     325c:    3e 53      add    #-1,   r14    ;r3 As==11
265
266 0000325e <.LVL7>:
267     325e:    3e 93      cmp    #-1,   r14    ;r3 As==11
268     3260:    01 20      jnz   $+4      ;abs 0x3264
269
270 00003262 <.L10>:
271     3262:    30 41      ret
272
273 00003264 <.L6>:
274     3264:    0b 4c      mov    r12,   r11    ;
275     3266:    0b 5e      add    r14,   r11    ;
276     3268:    0f 4d      mov    r13,   r15    ;
277     326a:    0f 5e      add    r14,   r15    ;
278     326c:    eb 4f 00 00 mov.b  @r15,  0(r11) ;
279     3270:    f5 3f      jmp    $-20     ;abs 0x325c
280
281 00003272 <memset>:
282     3272:    0f 4c      mov    r12,   r15    ;
283     3274:    0e 5c      add    r12,   r14    ;
284
285 00003276 <L0>:
286     3276:    0f 9e      cmp    r14,   r15    ;
287     3278:    01 20      jnz   $+4      ;abs 0x327c
288
289 0000327a <.Loc.104.1>:
290     327a:    30 41      ret
291
292 0000327c <.L3>:
293     327c:    cf 4d 00 00 mov.b  r13,   0(r15) ;
294     3280:    1f 53      inc    r15     ;
295
296 00003282 <.LVL4>:
297     3282:    f9 3f      jmp    $-12     ;abs 0x3276
298
299 00003284 <__do_global_ctors_aux>:
300     3284:    0a 15      pushm #1,    r10    ;16-bit words
301
302 00003286 <L0>:
303     3286:    3a 40 04 31 mov    #12548, r10   ;#0x3104
304
305 0000328a <.L2>:
306     328a:    2c 4a      mov    @r10,  r12    ;
307     328c:    3c 93      cmp    #-1,   r12    ;r3 As==11
308     328e:    02 20      jnz   $+6      ;abs 0x3294
309     3290:    0a 17      popm  #1,    r10    ;16-bit words
310     3292:    30 41      ret
311
312 00003294 <.L3>:
313     3294:    8c 12      call   r12      ;
314     3296:    3a 50 fe ff add    #65534, r10   ;#0xffff
315     329a:    f7 3f      jmp    $-16     ;abs 0x328a
316
317 0000329c <__msp430_init>:
318     329c:    b0 12 00 32 call   #12800    ;#0x3200
319     32a0:    b0 12 84 32 call   #12932    ;#0x3284
320
321 000032a4 <L0>:
322     32a4:    b0 12 48 31 call   #12616    ;#0x3148
323

```

```

324 000032a8 <.Loc.19.1>:
325     32a8:    b0 12 3c 31    call   #12604          ;#0x313c
326
327 000032ac <.Loc.20.1>:
328     32ac:    30 41          ret
329
330 000032ae <__msp430_fini>:
331     32ae:    b0 12 54 31    call   #12628          ;#0x3154
332
333 000032b2 <L0>:
334     32b2:    b0 12 b6 31    call   #12726          ;#0x31b6
335
336 000032b6 <L0>:
337     32b6:    30 41          ret

```

Figure 16. msp430-elf-objdump -S for ToggleLEDs.out.

By analyzing the sequence of instructions in the main code, we should deduce what our program is doing. Figure 17 shows the disassembled code for the main code using msp430-objdump -d (there is no C statements displayed in the disassembled code). We can walk through the code one by one instruction, write comments, and then tie everything together into a functional description of what this code does. Line 2 is a MOV instruction that moves immediate #0x5a80 into the address 0x0120 in the address space. This address represents the control register of the watchdog timer. By analyzing the format of this register we can deduce that this instruction stops the watchdog timer. The next instruction is bis.b #6, &0x002a. At the address 0x002a we have a P2DIR register, and this instruction will set port pins at bit positions 1 and 2 to be outputs. The next instruction at 0x3224 clears a byte at the address 0x0029, which represents P2OUT. The following instruction is an unconditional jump to address 0x3234, where an XOR instruction performs a logical XOR operation on P2OUT (effectively toggling bits BIT1 and BIT2 of P2OUT). Then, R13 is cleared and a jump performed to 0x322c, where a loop is executed for 50,000 iterations to implement a software delay. After that P2OUT is toggled again and the entire sequence repeats. Thus, we can finally deduce that this code periodically toggles port pins P2.1 and P2.2.

```

1 00003218 <main>:
2   3218:    b2 40 80 5a    mov   #23168, &0x0120 ;#0x5a80
3   321c:    20 01
4
5 0000321e <.Loc.25.1>:
6   321e:    f2 d0 06 00    bis.b #6,    &0x002a;
7   3222:    2a 00
8
9 00003224 <.Loc.26.1>:
10  3224:    c2 43 29 00    mov.b #0,    &0x0029; r3 As==00
11  3228:    05 3c          jmp   $+12      ;abs 0x3234
12
13 0000322a <.L3>:
14  322a:    1c 53          inc   r12      ;
15
16 0000322c <.L2>:
17  322c:    3d 40 4f c3    mov   #49999, r13   ;#0xc34f
18  3230:    0d 9c          cmp   r12, r13    ;
19  3232:    fb 2f          jc    $-8        ;abs 0x322a

```



```

20
21 00003234 <.L4>:
22   3234:    f2 e0 06 00   xor.b  #6,    &0x0029;
23   3238:    29 00
24
25 0000323a <.Loc.30.1>:
26   323a:    4c 43        clr.b  r12    ;
27   323c:    f7 3f        jmp   $-16    ;abs 0x322c

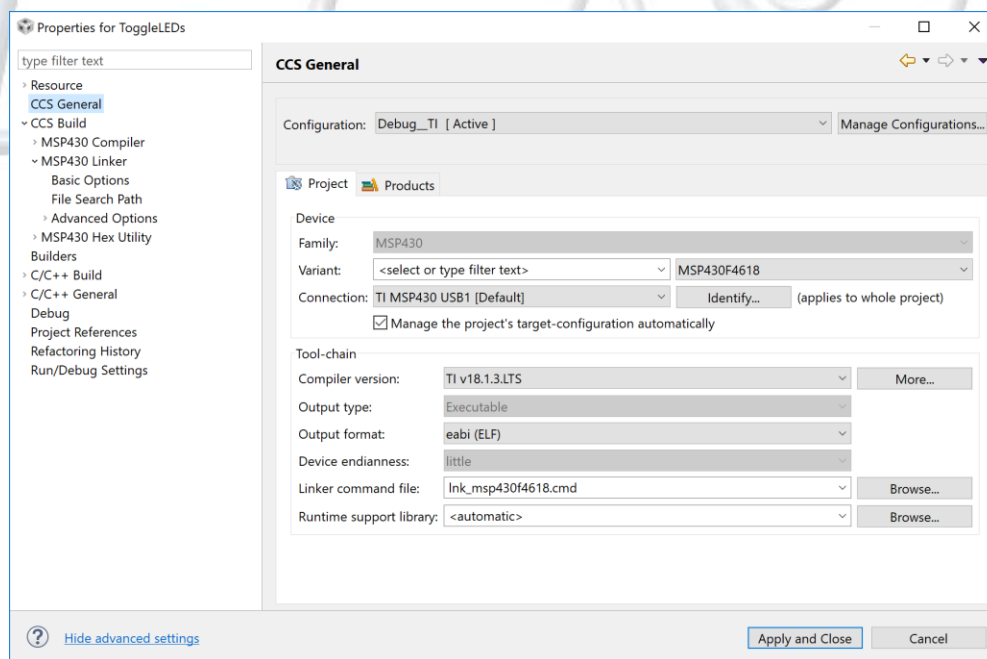
```

Figure 17. msp430-elf-objdump -d for ToggleLEDs.out (main section)

A useful exercise is to select the TI compiler instead of MSP430 GCC, create a new executable file, and repeat the analysis of the executable using utilities discussed above: msp430-elf-readelf, msp430-elf-nm, msp430-elf-symbols, and msp430-elf-objdump. What insights can you glean from your analysis?

5 Working with HEX Files and MSP430Flasher Utility

In this section we use ToggleLEDs.c program to demonstrate how to create a HEX file with executable and how to flash it on the target platform using TI's MSP430Flasher utility program. We select the TI compiler in the CCS, enable MSP430 HEX Utility, set General Options and Output Format Options as shown in Figure 18. An output HEX file is created (ToggleLEDs.txt) and its content is shown in Figure 19. This file can be downloaded on the target platform using a TI utility called MSP430Flasher as shown in Figure 20. If everything goes all right, we should see the green and yellow LEDs flashing together.



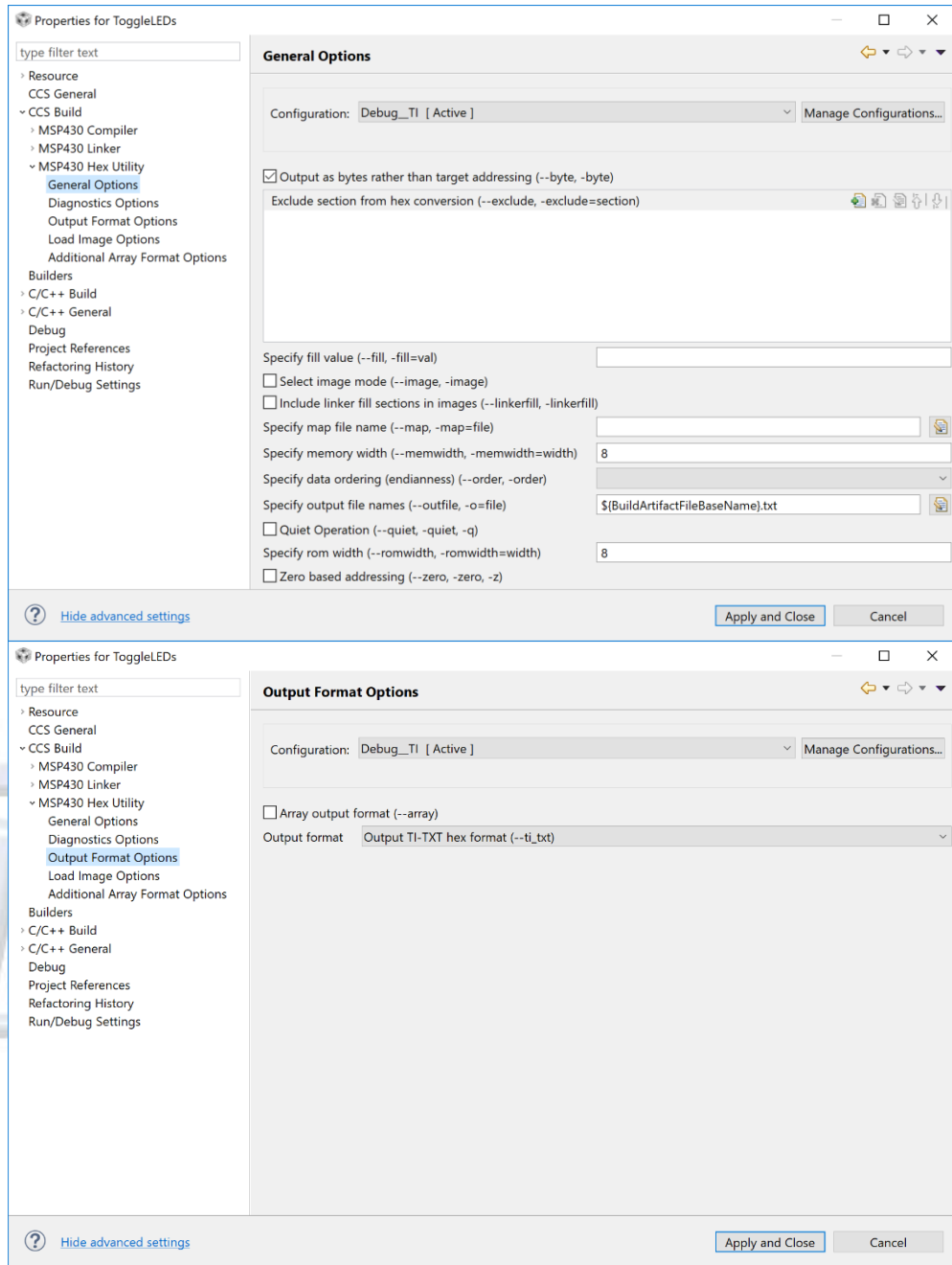


Figure 18. ToggleLEDs Project Properties for Generating HEX files

```

1  @3100
2  B2 40 80 5A 20 01 F2 D0 06 00 2A 00 C2 43 29 00
3  F2 E0 06 00 29 00 0F 43 3F 90 50 C3 F9 2F 1F 53
4  3F 90 50 C3 F5 2F FB 3F 31 40 00 31 B0 12 42 31
5  0C 43 B0 12 00 31 1C 43 B0 12 3C 31 03 43 FF 3F
6  03 43 1C 43 30 41 32 D0 10 00 FD 3F 03 43
7  @ffbe
8  FF FF
9  @ffde
10 46 31 46 31 46 31 46 31 46 31 46 31 46 31 46 31

```



```

Command Prompt
C:\Users\milenka\workspace_cpe325\ToggleLEDs\Debug_TI>MSP430Flasher.exe -r [RetrievedHEX.txt, MAIN]
-----*
*
*  /  \
*  /  \  MSP Flasher v1.3.18
*  /  \
*  /  \
*-----*
* Evaluating triggers...done
* Checking for available FET debuggers:
* Found USB FET @ COM7 <- Selected
* Initializing interface @ COM7...done
* Checking firmware compatibility:
* FET firmware is up to date.
* Reading FW version...done
* Setting VCC to 3000 mV...done
* Accessing device...done
* Reading device information...done
* Dumping memory from MAIN into RetrievedHEX.txt...done
-----*
* Arguments : -r [RetrievedHEX.txt, MAIN]
-----*
* Driver : loaded
* DLL Version : 31300001
* FWVersion : 31200000
* Interface : TIUSB
* HWVersion : U 3.0
* JTAG Mode : AUTO
* Device : MSP430FG4618
* EEM : Level 3, ClockCnt1 2
* Read File : RetrievedHEX.txt (memory segment = MAIN)
* VCC OFF
-----*
* Powering down...done
* Disconnecting from device...done
-----*
* Driver : closed (No error)
-----*
C:\Users\milenka\workspace_cpe325\ToggleLEDs\Debug_TI>

```

Figure 21. Running MSP430Flasher to Retrieve Code from the Platform

```

1 naken_util - by Michael Kohn
2             Joe Davisson
3             Web: http://www.mikekohn.net/
4             Email: mike@mikekohn.net
5
6 Version: April 23, 2018
7
8 Loaded ti_txt RetrievedHEX_Stripped.txt from 0x3100 to 0x314f
9 Type help for a list of commands.
10
11 Addr      Opcode Instruction                               Cycles
12 -----
13 0x3100: 0x40b2 mov.w #0x5a80, &0x0120                    5
14 0x3102: 0x5a80
15 0x3104: 0x0120
16 0x3106: 0xd0f2 bis.b #0x06, &0x002a                    5
17 0x3108: 0x0006
18 0x310a: 0x002a
19 0x310c: 0x43c2 mov.b #0, &0x0029                    4
20 0x310e: 0x0029
21 0x3110: 0xe0f2 xor.b #0x06, &0x0029                    5
22 0x3112: 0x0006
23 0x3114: 0x0029
24 0x3116: 0x430f mov.w #0, r15                            1
25 0x3118: 0x903f cmp.w #0xc350, r15                          2
26 0x311a: 0xc350
27 0x311c: 0x2ff9 jhs 0x3110 (offset: -14)                2
28 0x311e: 0x531f add.w #1, r15                            1
29 0x3120: 0x903f cmp.w #0xc350, r15                          2
30 0x3122: 0xc350
31 0x3124: 0x2ff5 jhs 0x3110 (offset: -22)                2
32 0x3126: 0x3ffb jmp 0x311e (offset: -10)

```

```

33 0x3128: 0x4031 mov.w #0x3100, SP          2
34 0x312a: 0x3100
35 0x312c: 0x12b0 call #0x3142              5
36 0x312e: 0x3142
37 0x3130: 0x430c mov.w #0, r12             1
38 0x3132: 0x12b0 call #0x3100              5
39 0x3134: 0x3100
40 0x3136: 0x431c mov.w #1, r12             1
41 0x3138: 0x12b0 call #0x313c              5
42 0x313a: 0x313c
43 0x313c: 0x4303 nop -- mov.w #0, CG          1
44 0x313e: 0x3fff jmp 0x313e (offset: -2)    2
45 0x3140: 0x4303 nop -- mov.w #0, CG          1
46 0x3142: 0x431c mov.w #1, r12             1
47 0x3144: 0x4130 ret -- mov.w @SP+, PC      3
48 0x3146: 0xd032 bis.w #0x0010, SR         2
49 0x3148: 0x0010
50 0x314a: 0x3ffd jmp 0x3146 (offset: -6)    2
51 0x314c: 0x4303 nop -- mov.w #0, CG          1
52 0x314e: 0xffff and.b @r15+, 0(r15)        5
53 0x3150: 0x0000

```

Figure 22. Disassembled Code in ReverseMe.asm.txt Created Using naked_util

1	Addr	Opcode	Instruction	Cycles
2	-----	-----	-----	-----
3	0x3100:	0x40b2	mov.w #0x5a80, &0x0120	5 // 0x0120 - WDTCTL; STOP WDT
4	0x3102:	0x5a80		
5	0x3104:	0x0120		
6	0x3106:	0xd0f2	bis.b #0x06, &0x002a	5 // P2DIR to output
7	0x3108:	0x0006		
8	0x310a:	0x002a		
9	0x310c:	0x43c2	mov.b #0, &0x0029	4 // P2OUT is cleared
10	0x310e:	0x0029		
11	0x3110:	0xe0f2	xor.b #0x06, &0x0029	5 // xor P2OUT with 0x06
12	0x3112:	0x0006		
13	0x3114:	0x0029		
14	0x3116:	0x430f	mov.w #0, r15	1 // clear r15
15	0x3118:	0x903f	cmp.w #0xc350, r15	2 // compare r15 to 50,000
16	0x311a:	0xc350		
17	0x311c:	0x2ff9	jhs 0x3110 (offset: -14)	2 // jump if carry to 0x3110
18	0x311e:	0x531f	add.w #1, r15	1 // add #1 to r15
19	0x3120:	0x903f	cmp.w #0xc350, r15	2 // compare r15 to 50,000
20	0x3122:	0xc350		
21	0x3124:	0x2ff5	jhs 0x3110 (offset: -22)	2 // jump if carry to 0x3110
22	(xoring)			
23	0x3126:	0x3ffb	jmp 0x311e (offset: -10)	2 // jmp to 0x311e (incrementing)
24				

Figure 23. Reverse Engineering of the Code in Disassembled Code Using naked_util

6 To Learn More

1. Texas Instruments, MSP430 GCC User's Guide:
<http://www.ti.com/lit/ug/slau646c/slau646c.pdf>

2. MSP430 Flasher: <http://www.ti.com/tool/MSP430-FLASHER>
(should be installed on your workstation and its exe directory, e.g. c:\ti\MSP430Flasher_1.3.18, should be in the PATH system environment variable)
3. Mike Kohn's Naken_asm: https://www.mikekohn.net/micro/naken_asm.php
(should be installed on your workstation and its exe directory, e.g., c:\ti\naken_asm, should be in the PATH system environment variable)

