CPE 323
Introduction to Software Reverse Engineering in Embedded Systems
Aleksandar Milenkovic
Electrical and Computer Engineering
The University of Alabama in Huntsville
milenka@ece.uah.edu
http://www.ece.uah.edu/~milenka
Outline

• Introduction
• Format of Executable Files
• GNU Utilities
• Deconstructing Executable Files: An Example
• Working with HEX Files and MSP430 Flasher Utility

Introduction
ToggleLEDs
Executable Files
GNU Utilities
Reversing ToggleLEDs
HEX Files
Conclusions
Introduction

• Objective
  • Introduce tools and methods for software reverse engineering in embedded systems

• What is software reverse engineering?
  • 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

• Main aspects of software reverse engineering
  • Re-documentation: creating a new representation of computer code that is easier to understand
  • Design recovery: use of deduction and reasoning from personal experience of the software systems to understand its functionality
When Is It Used?

• Analyzing malware
• Analyzing closed-source software to uncover vulnerabilities or interoperability issues
• Analyzing compiler-generated code to validate performance and/or correctness
• Debugging programs
What Will You Learn?

• Format of Executable Files
• Common GNU Utilities Used in SWRE
• How to Extract Useful Information from Executables
• How to Retrieve Programs from Platforms (HEX format)
• How to Analyze HEX Files
Create a New CCS Project

- Target: MSP430F4618
- Project name: ToggleLEDs
- Compiler: TI v18.1.2.LTS
- Project templates: Empty
- Click Finish
- Copy Code into main.c
- Set project options
- Build project
- Debug and run
Project Options

- Right-click on Active Project (ToggleLEDs)
- Select CCS Build>MSP430 Compiler>Processor Options
  - Silicon version: select msp instead of mspx
  - Code model: small; Data model: small (data will be in lower 64KB)
- Press: Apply and close
Code Composer Studio: Edit View

![Screenshot of Code Composer Studio interface]

- Introduction
- ToggleLEDs
- Executable Files
- GNU Utilities
- Reversing ToggleLEDs
- HEX Files
- Conclusions
Code Composer Studio: Debug View
 Executable File

• Executable file: ToggleLEDs.out
• How does it get created? => Translation Process
• What does it contain? => Executable and Linkable File (ELF)
  • Common standard file format for executable files, object code, shared libraries, and core dumps
  • Not bound by ISAs and OSes
• What is the format of this file? => ELF Format
• How do we deal with executables? => Utilities
Source Translation

User-created files

Makefile → Make Utility

C/C++ Source and Header Files → preprocessor → compiler → Object Files

Assembly Source Files → assembler → Object Files

Object Files → Linker and Locator

Library Files

Shared Object File

Linkable Image File

Executable Image File

Link Map File

Linker Script File
ELF File Layout

- ELF file header
- Program header table
  - Describes zero or more memory segments; Tells loader how to create a process image in memory
- Section header table
  - Describes zero or more sections
  - 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
ELF Views: Linkable vs. Executable File

**Linkable File**

- ELF Header
  - Program-Header Table (optional)
    - Section 1 Data
    - Section 2 Data
    - Section 3 Data
    - ...
    - Section n Data
  - Section-Header Table

**Executable File**

- ELF Header
  - Program-Header Table
    - Segment 1 Data
    - Segment 2 Data
    - Segment 3 Data
    - ...
    - Segment n Data
  - Section-Header Table (optional)
ELF Linking View

- Object files are divided into a collection of sections
- Sections have
  - Name and type
  - Requested memory location at run time
  - Permissions (R, W, X)
- Each section contains a single type of information and can contain flags (writable data, memory space during execution or executable machine instructions)
# ELF Linking View: Common Sections

<table>
<thead>
<tr>
<th>Sections</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.interp</td>
<td>Path name of program interpreter</td>
</tr>
<tr>
<td>.text</td>
<td>Code (executable instructions) of a program</td>
</tr>
<tr>
<td>.data</td>
<td>Initialized data</td>
</tr>
<tr>
<td>.bss</td>
<td>Uninitialized data</td>
</tr>
<tr>
<td>.init</td>
<td>Executable instructions for process initialization</td>
</tr>
<tr>
<td>.fini</td>
<td>Executable instructions for process termination</td>
</tr>
<tr>
<td>.ptl</td>
<td>Holds the procedure linkage table</td>
</tr>
<tr>
<td>.re.[x]</td>
<td>Relocation information for section [x]</td>
</tr>
<tr>
<td>.dynamic</td>
<td>Dynamic linking information</td>
</tr>
<tr>
<td>.symtab, .dynsym</td>
<td>Symbols (static/dynamic)</td>
</tr>
<tr>
<td>.strtab, .dynstr</td>
<td>String table</td>
</tr>
</tbody>
</table>
What Does a Linker Do?

**ELF Header**

- Section 1 Data
- Section 2 Data
- ...  
- Section n Data

**Section-Header Table**

- Linkable File

**ELF Header**

- Program-Header Table

- Segment 1 Data
- Segment 2 Data
- ...  
- Segment n Data

**Segment-Header Table**

- Linkable File

**ELF Header**

- Executable File
Execution View

• Simpler view that divides the object file into segments
  • Parts of file to be loaded into memory at run time
  • Locations of important data at run time

• Segments have:
  • Simple type
  • Requested memory location
  • Permissions (R, W, X)
  • Size (in file and in memory)

• All loadable sections are packed into segments so that file mapping is easier
# Execution View

<table>
<thead>
<tr>
<th>Common Segments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD</td>
<td>Portion of file to be loaded into memory</td>
</tr>
<tr>
<td>INTERP</td>
<td>Pointer to dynamic linker for this executable (.interp section)</td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>Pointer to dynamic linking information (.dynamic section)</td>
</tr>
</tbody>
</table>
ELF Loaders

• ELF Loaders are OS routines that
  • Load executable in memory
  • Begin execution

• Steps
  • Open ELF file
  • Map LOAD segments into the memory
  • Calls the dynamic linker specified in the INTERP segment, passing information about the executable
GNU Binary Utilities

- Code composer comes with GNU utilities that can be used to inspect and reverse engineer the code
- Working with them:
  - Include the bin directory into the Path System Environment Variable
  - Go to the project’s working directory where the ELF file is located (e.g., C:sers\milenka\workspace\cpe325\ToggleLEDs\Debug)
# GNU Binary Utilities (Binutils)

<table>
<thead>
<tr>
<th>Utility</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>as</td>
<td>Assembler</td>
</tr>
<tr>
<td>elfedit</td>
<td>Edit ELF files</td>
</tr>
<tr>
<td>gdb</td>
<td>Debugger</td>
</tr>
<tr>
<td>gprof</td>
<td>Profiler</td>
</tr>
<tr>
<td>ld</td>
<td>Linker</td>
</tr>
<tr>
<td>objcopy</td>
<td>Copy object files, possibly making changes</td>
</tr>
<tr>
<td>objdump</td>
<td>Dump information about object files</td>
</tr>
<tr>
<td>nm</td>
<td>List symbols from object files</td>
</tr>
<tr>
<td>readelf</td>
<td>Display content of ELF files</td>
</tr>
<tr>
<td>strings</td>
<td>List printable strings</td>
</tr>
<tr>
<td>size</td>
<td>List total and section sizes</td>
</tr>
<tr>
<td>strip</td>
<td>Remove symbols from an object file</td>
</tr>
</tbody>
</table>
readelf Utility

Usage: readelf <option(s)> elf-file(s)
Display information about the contents of ELF format files
Options are:
- a --all
  -h --file-header
  -l --program-headers
    --segments
  -S --section-headers
    --sections
  -g --section-groups
  -t --section-details
  -e --headers
  -s --syms
    --symbols
  --dyn-syms
  -n --notes
  -r --relocs
  -u --unwind
  -d --dynamic
  -V --version-info
  -A --arch-specific
  -c --archive-index
  -D --use-dynamic
Use the dynamic section info when displaying symbols
... (continued)
readelf Utility (cont’d)

C:\Users\milenka\workspace_cpe325\BlinkLEDs\Debug>msp430-elf-readelf --help
...
-D --use-dynamic
-x --hex-dump=<number|name>
-p --string-dump=<number|name>
-R --relocated-dump=<number|name>
-z --decompress
-w[lLiaprmFSoRt] or
--debug-dump=[rawline,=decodedline,=info,=abbrev,=pubnames,=ranges,=macro,=frames,=frames-interp,=str,=loc,=Ranges,=pubtypes,=gdb_index,=trace_info,=trace_abbrev,=trace_ranges,=addr,=cu_index]
--dwarf-depth=N
--dwarf-start=N
-I --histogram
-W --wide
@<file>
-H --help
-v --version

Report bugs to <http://www.sourceware.org/bugzilla/>
**objdump Utility**

Usage: msp430-elf-objdump <option(s)> <file(s)>

Display information from object <file(s)>

At least one of the following switches must be given:

- `-a`, `--archive-headers` Display archive header information
- `-f`, `--file-headers` Display the contents of the overall file header
- `-p`, `--private-headers` Display object format specific file header contents
- `-P`, `--private=OPT,OPT...` Display object format specific contents
- `-h`, `--[section-]headers` Display the contents of the section headers
- `-x`, `--all-headers` Display the contents of all headers
- `-d`, `--disassemble` Display assembler contents of executable sections
- `-D`, `--disassemble-all` Display assembler contents of all sections
- `-S`, `--source` Intermix source code with disassembly
- `-s`, `--full-contents` Display the full contents of all sections requested
- `-g`, `--debugging` Display debug information in object file
- `-e`, `--debugging-tags` Display debug information using ctags style
- `-G`, `--stabs` Display (in raw form) any STABS info in the file

```
-W[lLiaprmFsoRt] or
--dwarf[=rawline,=decodedline,=info,=abbrev,=pubnames,=aranges,=macro,=frames,
    =frames-interp,=str,=loc,=Ranges,=pubtypes,
    =gdb_index,=trace_info,=trace_abbrev,=trace_aranges,
    =addr,=cu_index]
```

Display DWARF info in the file
objdump Utility (cont’d)

C:\Users\milenka\workspace_cpe325\BlinkLEDs\Debug__TI>msp430-elf-objdump --help

... 
-t, --sym Display the contents of the symbol table(s)
-T, --dynamic-sym Display the contents of the dynamic symbol table
-r, --reloc Display the relocation entries in the file
-R, --dynamic-reloc Display the dynamic relocation entries in the file
@<file> Read options from <file>
-v, --version Display this program's version number
-i, --info List object formats and architectures supported
-H, --help Display this information

The following switches are optional:
-b, --target=BFDNAME Specify the target object format as BFDNAME
-m, --architecture=MACHINE Specify the target architecture as MACHINE
-j, --section=NAME Only display information for section NAME
-M, --disassembler-options=OPT Pass text OPT on to the disassembler
--eb Endian=big Assume big endian format when disassembling
--el Endian=little Assume little endian format when disassembling
--file-start-context Include context from start of file (with -S)
-I, --include=DIR Add DIR to search list for source files
-1, --line-numbers Include line numbers and filenames in output
-F, --file-offsets Include file offsets when displaying information
-C, --demangle[=STYLE] Decode mangled/processed symbol names
The STYLE, if specified, can be `auto', `gnu', `lucid', `arm', `hp', `edg', `gnu-v3', `java' or `gnat'
objdump Utility

C:\Users\milenka\workspace_cpe325\BlinkLEDs\Debug_TI>msp430-elf-objdump --help

... 
-w, --wide
-z, --disassemble-zeroes
--start-address=ADDR
--stop-address=ADDR
--prefix-addresses
--[no-]show-raw-instr
--insn-width=WIDTH
--adjust-vma=OFFSET
--special-syms
--prefix=PREFIX
--prefix-strip=LEVEL
--dwarf-depth=LEVEL
--dwarf-start=LEVEL
--dwarf-check

Format output for more than 80 columns
Do not skip blocks of zeroes when disassembling
Only process data whose address is >= ADDR
Only process data whose address is <= ADDR
Print complete address alongside disassembly
Display hex alongside symbolic disassembly
Display WIDTH bytes on a single line for -d
Add OFFSET to all displayed section addresses
Include special symbols in symbol dumps
Add PREFIX to absolute paths for -S
Strip initial directory names for -S
Do not display DIEs at depth N or greater
Display DIEs starting with N, at the same depth or deeper
Make additional dwarf internal consistency checks.

msp430-elf-objdump: supported targets: elf32-msp430 elf32-msp430 elf32-little elf32-big
plugin srec symbolsrec verilog tekhex binary ihex
msp430-elf-objdump: supported architectures: msp:14 MSP430 MSP430x11x1 MSP430x12
MSP430x13 MSP430x14 MSP430x15 MSP430x16 MSP430x20 MSP430x21 MSP430x22 MSP430x23
MSP430x24 MSP430x26 MSP430x31 MSP430x32 MSP430x33 MSP430x41 MSP430x42 MSP430x43
MSP430x44 MSP430x46 MSP430x47 MSP430x54 MSP430X plugin
strings Utility

- Extracts printable strings from binary and display them

C:\Users\milenka\workspace_cpe325\ToggleLEDs\Debug_TI>msp430-elf-strings --help

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

```
C:\Users\milenka\workspace_cpe325\ToggleLEDs\Debug__TI>msp430-elf-readelf -h ToggleLEDs.out
ELF Header:
  Magic:    7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00
  Class:    ELF32
  Data:    2's complement, little endian
  Version: 1 (current)
  OS/ABI:  UNIX - System V
  ABI Version: 0
  Type:    EXEC (Executable file)
  Machine: Texas Instruments msp430 microcontroller
  Version: 0x1
  Entry point address: 0x3128
  Start of program headers: 15260 (bytes into file)
  Start of section headers: 15388 (bytes into file)
  Flags: 0x0
  Size of this header: 52 (bytes)
  Size of program headers: 32 (bytes)
  Number of program headers: 4
  Size of section headers: 40 (bytes)
  Number of section headers: 67
  Section header string table index: 66
```
readelf Utility: Display Program Headers

C:\Users\milenka\workspace_cpe325\ToggleLEDs\Debug_TI>msp430-elf-readelf -l ToggleLEDs.out

Elf file type is EXEC (Executable file)
Entry point 0x3128
There are 4 program headers, starting at offset 15260

Program Headers:

<table>
<thead>
<tr>
<th>Type</th>
<th>Offset</th>
<th>VirtAddr</th>
<th>PhysAddr</th>
<th>FileSiz</th>
<th>MemSiz</th>
<th>Flg</th>
<th>Align</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD</td>
<td>0x000034</td>
<td>0x000030b0</td>
<td>0x000030b0</td>
<td>0x00000</td>
<td>0x0050</td>
<td>RW</td>
<td>0x4</td>
</tr>
<tr>
<td>LOAD</td>
<td>0x000034</td>
<td>0x00003100</td>
<td>0x00003100</td>
<td>0x004e</td>
<td>0x004e</td>
<td>R E</td>
<td>0x2</td>
</tr>
<tr>
<td>LOAD</td>
<td>0x000084</td>
<td>0x0000ffbe</td>
<td>0x0000ffbe</td>
<td>0x0002</td>
<td>0x0002</td>
<td>R</td>
<td>0x1</td>
</tr>
<tr>
<td>LOAD</td>
<td>0x000088</td>
<td>0x0000ffde</td>
<td>0x0000ffde</td>
<td>0x00022</td>
<td>0x00022</td>
<td>R</td>
<td>0x2</td>
</tr>
</tbody>
</table>

Section to Segment mapping:

Segment Sections...
00 .stack
01 .text .text:_isr
02 $fill1000
03 DMA BASICTIMER PORT2 USART1TX USART1RX PORT1 TIMERA1 TIMERA0 ADC12 USCIAB0TX USCIAB0RX WDT COMPARATORA TIMERB1 TIMERB0 NMI .reset
Reversing the Main

• Go step-by-step through the main code
• What does it do?
• How does it do it?
readelf Utility: Display Section Headers

C:\Users\milenka\workspace_cpe325\ToggleLEDs\Debug_TI> msp430-elf-readelf -S ToggleLEDs.out

There are 67 section headers, starting at offset 0x3c1c:

<table>
<thead>
<tr>
<th>Nr</th>
<th>Name</th>
<th>Type</th>
<th>Addr</th>
<th>Off</th>
<th>Size</th>
<th>ES</th>
<th>Flg</th>
<th>Lk</th>
<th>Inf</th>
<th>Al</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NULL</td>
<td>NULL</td>
<td>00000000</td>
<td>000000</td>
<td>000000</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>.bss</td>
<td>NOBITS</td>
<td>00000000</td>
<td>0000aa</td>
<td>000000</td>
<td>00</td>
<td>WA</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>.data</td>
<td>NOBITS</td>
<td>00000000</td>
<td>0000aa</td>
<td>000000</td>
<td>00</td>
<td>WA</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>.TI.noinit</td>
<td>NOBITS</td>
<td>00000000</td>
<td>000000</td>
<td>000000</td>
<td>00</td>
<td>p</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>.sysmem</td>
<td>NOBITS</td>
<td>00000000</td>
<td>000000</td>
<td>000000</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>.stack</td>
<td>NOBITS</td>
<td>000030b0</td>
<td>000034</td>
<td>000050</td>
<td>00</td>
<td>WA</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>.text</td>
<td>PROGBITS</td>
<td>00003100</td>
<td>000034</td>
<td>000046</td>
<td>00</td>
<td>AX</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>.text:_isr</td>
<td>PROGBITS</td>
<td>00003146</td>
<td>00007a</td>
<td>000008</td>
<td>00</td>
<td>AX</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>WDT</td>
<td>PROGBITS</td>
<td>0000fff4</td>
<td>00009e</td>
<td>000002</td>
<td>00</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>47</td>
<td>COMPARATORA</td>
<td>PROGBITS</td>
<td>0000fff6</td>
<td>0000a0</td>
<td>000002</td>
<td>00</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>48</td>
<td>TIMERB1</td>
<td>PROGBITS</td>
<td>0000fff8</td>
<td>0000a2</td>
<td>000002</td>
<td>00</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>49</td>
<td>TIMERB0</td>
<td>PROGBITS</td>
<td>0000fff0</td>
<td>0000a4</td>
<td>000002</td>
<td>00</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>NMI</td>
<td>PROGBITS</td>
<td>0000fff0</td>
<td>0000a6</td>
<td>000002</td>
<td>00</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>51</td>
<td>.reset</td>
<td>PROGBITS</td>
<td>0000ffe0</td>
<td>0000a8</td>
<td>000002</td>
<td>00</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>.symtab</td>
<td>SYMTAB</td>
<td>00000000</td>
<td>0010d0</td>
<td>001b20</td>
<td>10</td>
<td>65</td>
<td>168</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>.TI.section.flags</td>
<td>MSP430_SEC_FLAG</td>
<td>00000000</td>
<td>002bf0</td>
<td>00001a</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>.strtab</td>
<td>STRTAB</td>
<td>00000000</td>
<td>002c0a</td>
<td>000d37</td>
<td>01</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>66</td>
<td>.shstrtab</td>
<td>STRTAB</td>
<td>00000000</td>
<td>003941</td>
<td>00025a</td>
<td>01</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Key to Flags:
- W (write), A (alloc), X (execute), M (merge), S (strings)
- I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
- O (extra OS processing required) o (OS specific), p (processor specific)
objdump Utility: Disassembling

C:\Users\milenka\workspace_cpe325\ToggleLEDs\Debug_TI>msp430-elf-objdump -d ToggleLEDs.out
msp430-elf-objdump: ToggleLEDs.out: warning: sh_link not set for section `.mspabi.exidx'

ToggleLEDs.out: file format elf32-msp430

Disassembly of section .text:

00003100 <main>:
   3100:   b2 40 80 5a    mov    #23168, &0x0120 ;#0x5a80
   3104:   20 01
   3106:   f2 d0 06 00    bis.b    #6, &0x002a ;
   310a:   2a 00
   310c:   c2 43 29 00    mov.b    #0, &0x0029 ;r3 As==00

00003110 <$C$L1>:
   3110:   f2 e0 06 00    xor.b    #6, &0x0029 ;
   3114:   29 00
   3116:   0f 43          clr    r15    ;
   3118:   3f 90 50 c3    cmp    #50000, r15    ;#0xc350
   311c:   f9 2f          jc     $-12    ;abs 0x3110

0000311e <$C$L2>:
   311e:   1f 53          inc    r15    ;
   3120:   3f 90 50 c3    cmp    #50000, r15    ;#0xc350
   3124:   f5 2f          jc     $-20    ;abs 0x3110
   3126:   fb 3f          jmp     $-8    ;abs 0x311e

// continued
objdump Utility: Disassembling

// continued from the previous page

00003128 <_c_int00_noinit_noargs>:
  3128:   31 40 00 31     mov   #12544, r1   ;#0x3100
  312c:   b0 12 42 31     call  #12610   ;#0x3142
  3130:   0c 43           clr   r12
  3132:   b0 12 00 31     call  #12544   ;#0x3100
  3136:   1c 43           mov   #1, r12   ;r3 As==01
  3138:   b0 12 3c 31     call  #12604   ;#0x313c

0000313c <C$EXIT>:
  313c:   03 43           nop

0000313e <$C$L1>:
  313e:   ff 3f           jmp   $+0   ;abs 0x313e
  3140:   03 43           nop

00003142 <_system_pre_init>:
  3142:   1c 43           mov   #1, r12   ;r3 As==01
  3144:   30 41           ret

Disassembly of section .text:_isr:

00003146 <__TI_ISR_TRAP>:
  3146:   32 d0 10 00     bis   #16, r2   ;#0x0010
  314a:   fd 3f           jmp   $-4   ;abs 0x3146
  314c:   03 43           nop
Objectives

• Objectives
  • Learn How to Create a HEX File using TI Composer Studio
  • Learn How to Program the Board Using MSP430 Flasher and HEX File
  • Learn How to Retrieve Code from the Board
  • Learn How to Disassemble the Retrieved Code

• Software resources
  • TI Code Composer with GNU tools
  • 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)
  • 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)

• Hardware resources
  • TI MSP430 Experimenter’s Board
Enable MSP430 Hex Utility

• CCS-General: Select TI compiler
• CCS-Build: Check “Enable MSP430 Hex Utility”
Output Format Options

• Select Output TI-TXT hex format
MSP430Flasher Utility

- Shell-based interface that provides easy access to MSP devices through JTAG or Spy-By-Wire (SBW)
  - Ports the most common functions of the MSP Debug Stack to the command line
MSP430 Flasher Functions

- 1. Initialize FET debugger
- 2. Perform FET recovery (if a corrupted FET firmware is detected)
- 3. Update FET firmware (if a mismatch between firmware and MSP Debug Stack versions is detected)
- 4. Power up the target MSP device
- 5. Configure the target MSP for JTAG or SBW communication
- 6. Connect to the target MSP and display device information
- 7. Optional: Erase (parts of) the target device memory
- 8. Optional: Load target code into the device from a TXT or HEX file
- 9. Optional: Verify target code transfer
- **10. Optional: Read device memory and write it to a TXT or HEX file**
- 11. Optional: Reset the device
- 12. Optional: Lock JTAG access
- 13. Optional: Reset the device
- 14. Optional: Power down the device
- 15. Optional: Start target code execution
- 16. Disconnect from the target MSP device
- 17. Close the FET connection
MSP430Flasher.exe

Usage: MSP430Flasher [OPTIONS]

-n DEVICE_NAME  (optional for MSP430, required for MSP432)
specifies the name of the target MSP - prompt in
case of mismatch
Use -n NO_TARGET to run MSP Flasher without
attempting a target connection (FET detection or
FET firmware update only)

-i (TI)USB | DETECT | (default: TIUSB/USB = first detected FET tool is
COMn (Win) | used) For info on how to connect to specific eZ-
ttyACMn (Linux) | tools, see the MSP Flasher manual.
usbmodem* (OSX) Use -i DETECT to execute a FET detection sweep,
displaying detailed info about all connected debug
tools. User is prompted to pick a FET.

-j fast | medium | slow sets FET speed for JTAG/SBW - only applicable for
MSP-FET! Option will be ignored for all other FET
tools. Default = medium.

-a non-intrusive target connection: use this switch
if no reset should be applied to the target device
on start-up. Correct target device name needs to
be specified using the -n switch!
### MSP430Flasher.exe (cont’d)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-r [filename,mem_sect]</code></td>
<td>Specifies a memory section to read from and a file to write to. <code>mem_sect</code>: RAM, INFO, MAIN, BSL or specific memory areas: 0x****-0x*****. The file extension of 'filename' determines the data format: .txt = TI-TXT, .hex/.a43 = Intel-Hex</td>
</tr>
<tr>
<td><code>-w filename</code></td>
<td>Specifies a file for device programming. Supported data formats: TI-TXT (.txt), Intel-Hex (.hex/.a43)</td>
</tr>
<tr>
<td><code>-b</code></td>
<td>Unlocks BSL memory for writing (use only with <code>-w</code> switch)</td>
</tr>
<tr>
<td><code>-u</code></td>
<td>Unlocks InfoA memory for writing (use only with <code>-w</code> switch)</td>
</tr>
<tr>
<td>`-e ERASE_ALL</td>
<td>ERASE/Main</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td>`</td>
<td>`</td>
</tr>
<tr>
<td><code>-v filename (optional)</code></td>
<td>Triggers verification of the target memory against a target code file. If <code>-w</code> is used, no argument is required. For a stand-alone verify, provide the path to a target code file as an argument.</td>
</tr>
<tr>
<td><code>-z [exit_spec,...]</code></td>
<td>Specifies state of device on exit (view available 'exit_spec's using <code>-x</code> switch)</td>
</tr>
<tr>
<td><code>-g</code></td>
<td>Switches log OFF (default: ON)</td>
</tr>
<tr>
<td><code>-q</code></td>
<td>Triggers QUIET MODE (no system messages displayed)</td>
</tr>
</tbody>
</table>
MSP430Flasher.exe (cont’d)

-d [breakpoint addresses] specifies addresses for hardware breakpoints and triggers RUN_TO_BREAKPOINT mode
WARNING! This option is deprecated and will no longer be maintained. All breakpoint functionality will be removed in a future version of MSP Flasher

-t timeout_in_ms specifies the breakpoint timeout (in milliseconds)
WARNING! This option is deprecated and will no longer be maintained. All breakpoint functionality will be removed in a future version of MSP Flasher

-p JTAG PASSWORD specifies the JTAG password (hex format: 0x...).
(if any, default: no password)

-s suppresses the FET firmware update prompt

-o L | C specifies operation mode for L092 or RF430 devices
(L = normal mode, C = ROM development mode)

-l PASSWORD LENGTH OBSOLETE! If used, this option will be ignored!
JTAG password length is determined automatically

-m JTAG | SBW2 | SBW4 | AUTO OBSOLETE! If used, this option will be ignored!
Communication protocol is determined automatically

For a GUI-based alternative, check out UniFlash: http://ti.com/tool/uniflash
*******************************************************************************
Press ENTER to continue.
HEX File: ToggleLEDs.txt

@3100
B2 40 80 5A 20 01 F2 D0 06 00 2A 00 C2 43 29 00
F2 E0 06 00 29 00 0F 43 3F 90 50 C3 F9 2F 1F 53
3F 90 50 C3 F5 2F FB 3F 31 40 00 31 B0 12 42 31
0C 43 B0 12 00 31 1C 43 B0 12 3C 31 03 43 FF 3F
03 43 1C 43 30 41 32 D0 10 00 FD 3F 03 43
@ffbe
FF FF
@ffde
46 31 46 31 46 31 46 31 46 31 46 31 46 31 46 31
46 31 46 31 46 31 46 31 46 31 46 31 46 31 46 31
28 31
q
Download HEX File Using MSP430Flasher

C:\Users\milenka\workspace_cpe325\ToggleLEDs\Debug__TI>MSP430Flasher.exe -n MSP430FG4618 -w ToggleLEDs.txt -v -z [VCC]
* -----|--------------------------------------------------------------------- *
* / |__                                                                 *
* /_ /   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
* Loading file into device...done
* Verifying memory (ToggleLEDs.txt)...done
*
Download HEX File Using MSP430Flasher

* Arguments   : -n MSP430FG4618 -w ToggleLEDs.txt -v -z [VCC]
* Driver      : loaded
* Dll Version : 31300001
* FwVersion   : 31200000
* Interface   : TIUSB
* HwVersion   : U 3.0
* JTAG Mode   : AUTO
* Device      : MSP430FG4618
* EEM         : Level 3, ClockCtrl 2
* Erase Mode  : ERASE_ALL
* Prog.File   : ToggleLEDs.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)
*/
Retrieving Flash Image From the Platform

- Problem: You need to retrieve a program from the Experimenter Board and reverse engineer it to understand what does it do
Reading Device Memory

• MSP430Flasher can read out any section of the device memory and write it to a file

• Memory sectors
  • MAIN
  • INFO
  • RAM
  • BSL

• Make sure debug interface is not locked by other applications (e.g., debugger in Code Composer)

• Read sector MAIN using the following command
  • `MSP430Flasher.exe -r [output.txt,MAIN]`
Reading MAIN sector

C:\Users\milenka\workspace_cpe325\MSP430Flasher>MSP430Flasher.exe -r [output.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 output.txt...done
*
* -----------------------------------------------------*
* Arguments   : -r [output.txt,MAIN]
* -----------------------------------------------------*
* Driver      : loaded
* Dll Version : 31300001
* FwVersion   : 31200000
* Interface   : TIUSB
* HwVersion   : U 3.0
* JTAG Mode   : AUTO
* Device      : MSP430FG4618
* EEM         : Level 3, ClockCtrl 2
* Read File   : output.txt (memory segment = MAIN)
* VCC OFF
* -----------------------------------------------------*
* Powering down...done
* Disconnecting from device...done
*
* -----------------------------------------------------*
* Driver      : closed (No error)
* -----------------------------------------------------*/
HEX Content of MAIN Sector

- Output.txt contains hexadecimal content of flash memory starting from the address 0x3100
- Note: output.txt is relatively big as it includes the content of the entire Flash memory
- Flash memory locations with 0xFF are erased bytes and thus do not contain useful code (can be actually removed from the file)
Disassembling HEX File

• Create stripped version of the HEX file (output_Stripped.txt) by removing erased flash locations

• Run disassembler

  naken_util -msp430 -disasm output_Stripped.txt > ToggleLEDs_Reversed.txt

• Inspect code
Disassembled Code

naken_util - by Michael Kohn
  Joe Davisson
  Web: http://www.mikekohn.net/
  Email: mike@mikekohn.net

Version: April 23, 2018

Loaded ti_txt output_Stripped.txt from 0x3100 to 0x314f
Type help for a list of commands.

<table>
<thead>
<tr>
<th>Addr</th>
<th>Opcode</th>
<th>Instruction</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x3100:</td>
<td>0x40b2</td>
<td>mov.w #0x5a80, &amp;0x0120</td>
<td>5</td>
</tr>
<tr>
<td>0x3102:</td>
<td>0x5a80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3104:</td>
<td>0x0120</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3106:</td>
<td>0xd0f2</td>
<td>bis.b #0x06, &amp;0x002a</td>
<td>5</td>
</tr>
<tr>
<td>0x3108:</td>
<td>0x0006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x310a:</td>
<td>0x002a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x310c:</td>
<td>0x43c2</td>
<td>mov.b #0, &amp;0x0029</td>
<td>4</td>
</tr>
<tr>
<td>0x310e:</td>
<td>0x0029</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3110:</td>
<td>0xe0f2</td>
<td>xor.b #0x06, &amp;0x0029</td>
<td>5</td>
</tr>
<tr>
<td>0x3112:</td>
<td>0x0006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3114:</td>
<td>0x0029</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3116:</td>
<td>0x430f</td>
<td>mov.w #0, r15</td>
<td>1</td>
</tr>
<tr>
<td>0x3118:</td>
<td>0x903f</td>
<td>cmp.w #0xc350, r15</td>
<td>2</td>
</tr>
<tr>
<td>0x311a:</td>
<td>0xc350</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x311c:</td>
<td>0xff9</td>
<td>jhs 0x3110 (offset: -14)</td>
<td>2</td>
</tr>
</tbody>
</table>
Disassembled Code (cont’d)

naken_util - by Michael Kohn
   Joe Davisson
   Web: http://www.mikekohn.net/
   Email: mike@mikekohn.net

Version: April 23, 2018

Loaded ti_txt output_Stripped.txt from 0x3100 to 0x314f
Type help for a list of commands.

<table>
<thead>
<tr>
<th>Addr</th>
<th>Opcode</th>
<th>Instruction</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x3126:</td>
<td>0x3ffb</td>
<td>jmp 0x311e (offset: -10)</td>
<td>2</td>
</tr>
<tr>
<td>0x3128:</td>
<td>0x4031</td>
<td>mov.w #0x3100, SP</td>
<td>2</td>
</tr>
<tr>
<td>0x312a:</td>
<td>0x3100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x312c:</td>
<td>0x12b0</td>
<td>call #0x3142</td>
<td>5</td>
</tr>
<tr>
<td>0x312e:</td>
<td>0x3142</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3130:</td>
<td>0x430c</td>
<td>mov.w #0, r12</td>
<td>1</td>
</tr>
<tr>
<td>0x3132:</td>
<td>0x12b0</td>
<td>call #0x3100</td>
<td>5</td>
</tr>
<tr>
<td>0x3134:</td>
<td>0x3100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3136:</td>
<td>0x431c</td>
<td>mov.w #1, r12</td>
<td>1</td>
</tr>
<tr>
<td>0x3138:</td>
<td>0x12b0</td>
<td>call #0x313c</td>
<td>5</td>
</tr>
<tr>
<td>0x313a:</td>
<td>0x313c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x313c:</td>
<td>0x4303</td>
<td>nop -- mov.w #0, CG</td>
<td>1</td>
</tr>
<tr>
<td>0x313e:</td>
<td>0x3fff</td>
<td>jmp 0x313e (offset: -2)</td>
<td>2</td>
</tr>
<tr>
<td>0x3140:</td>
<td>0x4303</td>
<td>nop -- mov.w #0, CG</td>
<td>1</td>
</tr>
<tr>
<td>0x3142:</td>
<td>0x431c</td>
<td>mov.w #1, r12</td>
<td>1</td>
</tr>
<tr>
<td>0x3144:</td>
<td>0x4130</td>
<td>ret -- mov.w @SP+, PC</td>
<td>3</td>
</tr>
<tr>
<td>0x3146:</td>
<td>0xd032</td>
<td>bis.w #0x0010, SR</td>
<td>2</td>
</tr>
<tr>
<td>0x3148:</td>
<td>0x0010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x314a:</td>
<td>0x3ffd</td>
<td>jmp 0x3146 (offset: -6)</td>
<td>2</td>
</tr>
<tr>
<td>0x314c:</td>
<td>0x4303</td>
<td>nop -- mov.w #0, CG</td>
<td>1</td>
</tr>
<tr>
<td>0x314e:</td>
<td>0xffff</td>
<td>and.b @r15+, 0(r15)</td>
<td>5</td>
</tr>
<tr>
<td>0x3150:</td>
<td>0x0000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What is in IVT?

```
FFD0: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 46 31
FFE0: 46 31 46 31 46 31 46 31 46 31 46 31 46 31 46 31
FFF0: 46 31 46 31 46 31 46 31 46 31 46 31 46 31 46 31 28 31
```

=> RESET VECTOR contains 0x3128
# Reversing Code

naken_util - by Michael Kohn  
   Joe Davison  
   Web: http://www.mikekohn.net/  
   Email: mike@mikekohn.net

Version: April 23, 2018

Loaded ti_txt output_Stripped.txt from 0x3100 to 0x314f  
Type help for a list of commands.

<table>
<thead>
<tr>
<th>Addr</th>
<th>Opcode</th>
<th>Instruction</th>
<th>Cycles</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x3100</td>
<td>0x40b2</td>
<td>mov.w #0x5a80, &amp;0x0120</td>
<td>5</td>
<td>// 0x0120 - WDTCTL; STOP WDT</td>
</tr>
<tr>
<td>0x3102</td>
<td>0x5a80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3104</td>
<td>0x0120</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3106</td>
<td>0x0d0f2</td>
<td>bis.b #0x06, &amp;0x002a</td>
<td>5</td>
<td>// P2DIR to output</td>
</tr>
<tr>
<td>0x3108</td>
<td>0x0006</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x310a</td>
<td>0x0002a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x310c</td>
<td>0x43c2</td>
<td>mov.b #0, &amp;0x0029</td>
<td>4</td>
<td>// P2OUT is cleared</td>
</tr>
<tr>
<td>0x310e</td>
<td>0x0029</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3110</td>
<td>0xe0f2</td>
<td>xor.b #0x06, &amp;0x0029</td>
<td>5</td>
<td>// xor P2OUT with 0x06</td>
</tr>
<tr>
<td>0x3112</td>
<td>0x0006</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3114</td>
<td>0x0029</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3116</td>
<td>0x430f</td>
<td>mov.w #0, r15</td>
<td>1</td>
<td>// clear r15</td>
</tr>
<tr>
<td>0x3118</td>
<td>0x903f</td>
<td>cmp.w #0xc350, r15</td>
<td>2</td>
<td>// compare r15 to 50,000</td>
</tr>
<tr>
<td>0x311a</td>
<td>0xc350</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x311c</td>
<td>0x2ff9</td>
<td>jhs 0x3110  (offset: -14)</td>
<td>2</td>
<td>// jump if carry to 0x3110</td>
</tr>
<tr>
<td>0x311e</td>
<td>0x531f</td>
<td>add.w #1, r15</td>
<td>1</td>
<td>// add #1 to r15</td>
</tr>
<tr>
<td>0x3120</td>
<td>0x903f</td>
<td>cmp.w #0xc350, r15</td>
<td>2</td>
<td>// compare r15 to 50,000</td>
</tr>
<tr>
<td>0x3122</td>
<td>0xc350</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3124</td>
<td>0x2ff5</td>
<td>jhs 0x3110  (offset: -22)</td>
<td>2</td>
<td>// jump if carry to 0x3110 (xoring)</td>
</tr>
<tr>
<td>0x3126</td>
<td>0x3ff8</td>
<td>jmp 0x311e  (offset: -10)</td>
<td>2</td>
<td>// jmp to 0x311e (incrementing)</td>
</tr>
</tbody>
</table>
### Disassembled Code (cont’d)

**naken_util** - by Michael Kohn  
Joe Davisson  
Web: http://www.mikekohn.net/  
Email: mike@mikekohn.net

**Version:** April 23, 2018

**Loaded ti_txt output_Stripped.txt from 0x3100 to 0x314f**  
**Type help for a list of commands.**

<table>
<thead>
<tr>
<th>Addr</th>
<th>Opcode</th>
<th>Instruction</th>
<th>Cycles</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x3128</td>
<td>0x4031</td>
<td>mov.w #0x3100, SP</td>
<td>2</td>
<td>// initialize SP</td>
</tr>
<tr>
<td>0x312a</td>
<td>0x3100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x312c</td>
<td>0x12b0</td>
<td>call #0x3142</td>
<td>5</td>
<td>// call a subroutine at 0x3142</td>
</tr>
<tr>
<td>0x312e</td>
<td>0x3142</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3130</td>
<td>0x430c</td>
<td>mov.w #0, r12</td>
<td>1</td>
<td>// r12 &lt;= 0</td>
</tr>
<tr>
<td>0x3132</td>
<td>0x12b0</td>
<td>call #0x3100</td>
<td>5</td>
<td>// call 0x3100 (main program)</td>
</tr>
<tr>
<td>0x3134</td>
<td>0x3100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x3136</td>
<td>0x431c</td>
<td>mov.w #1, r12</td>
<td>1</td>
<td>// r12 &lt;= 1</td>
</tr>
<tr>
<td>0x3138</td>
<td>0x12b0</td>
<td>call #0x313c</td>
<td>5</td>
<td>// call a subroutine at 0x313c</td>
</tr>
<tr>
<td>0x313a</td>
<td>0x313c</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x313c</td>
<td>0x4303</td>
<td>nop</td>
<td>1</td>
<td>// nop</td>
</tr>
<tr>
<td>0x313e</td>
<td>0x3fff</td>
<td>jmp 0x313e (offset: -2)</td>
<td>2</td>
<td>// jump to itself</td>
</tr>
<tr>
<td>0x3140</td>
<td>0x4303</td>
<td>nop</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0x3142</td>
<td>0x431c</td>
<td>mov.w #1, r12</td>
<td>1</td>
<td>// r12 &lt;= 1</td>
</tr>
<tr>
<td>0x3144</td>
<td>0x4130</td>
<td>ret</td>
<td>3</td>
<td>// return</td>
</tr>
<tr>
<td>0x3146</td>
<td>0xd032</td>
<td>bis.w #0x0010, SR</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0x3148</td>
<td>0x0010</td>
<td>jmp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x314a</td>
<td>0x3fff</td>
<td>jmp 0x3146 (offset: -6)</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>0x314c</td>
<td>0x4303</td>
<td>nop</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0x314e</td>
<td>0xffff</td>
<td>and.b @r15+, 0(r15)</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>0x3150</td>
<td>0x0000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What Does the Code Do?

- Blink LEDs connected on Port2 pins 1 and 2
- Delay: $50,000 \times 7cc = 0.35$ s
Conclusions

• Software reverse engineering flows
• Code compilation and executable file formats
• Binary utilities for reverse engineering
• Generating HEX File
• Downloading HEX File
• Retrieving HEX File
• Reversing code using naken_asm