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

Admin

→ HW. 6 11/18/2020

→ Practice Quiz ADC/DAC

→ Quiz. 7 next week (Thursday)

→ Return the boards

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Outline

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

LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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

LaCASA

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

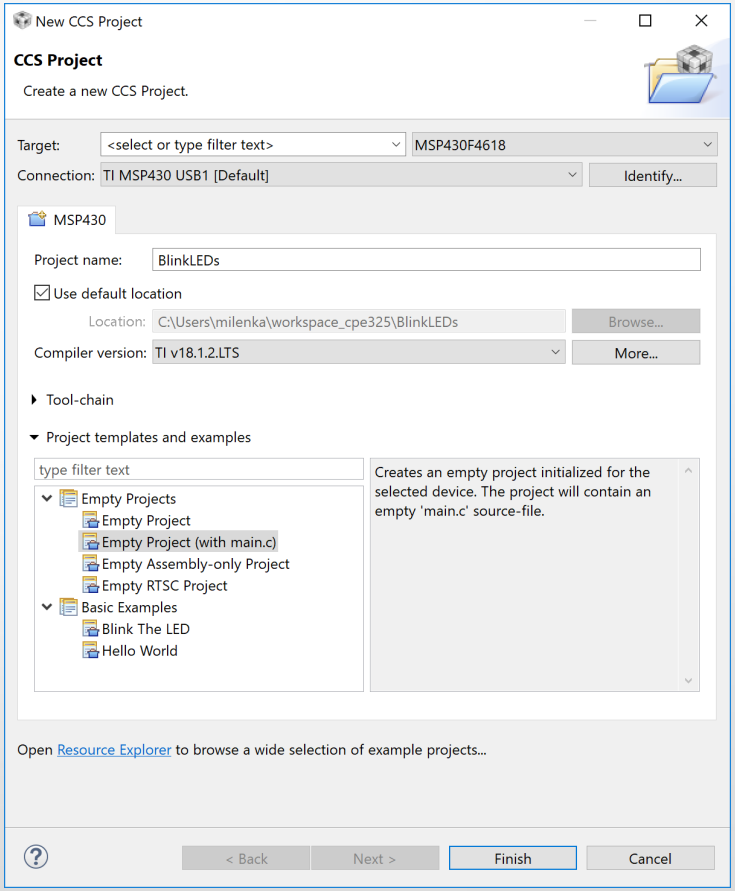
What Will You Learn?

- 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

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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

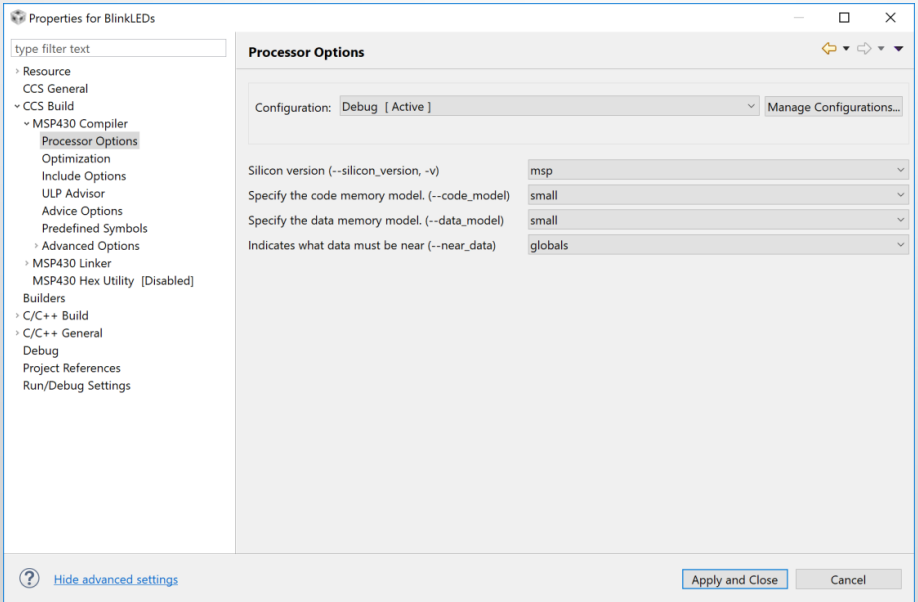


LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Project Options

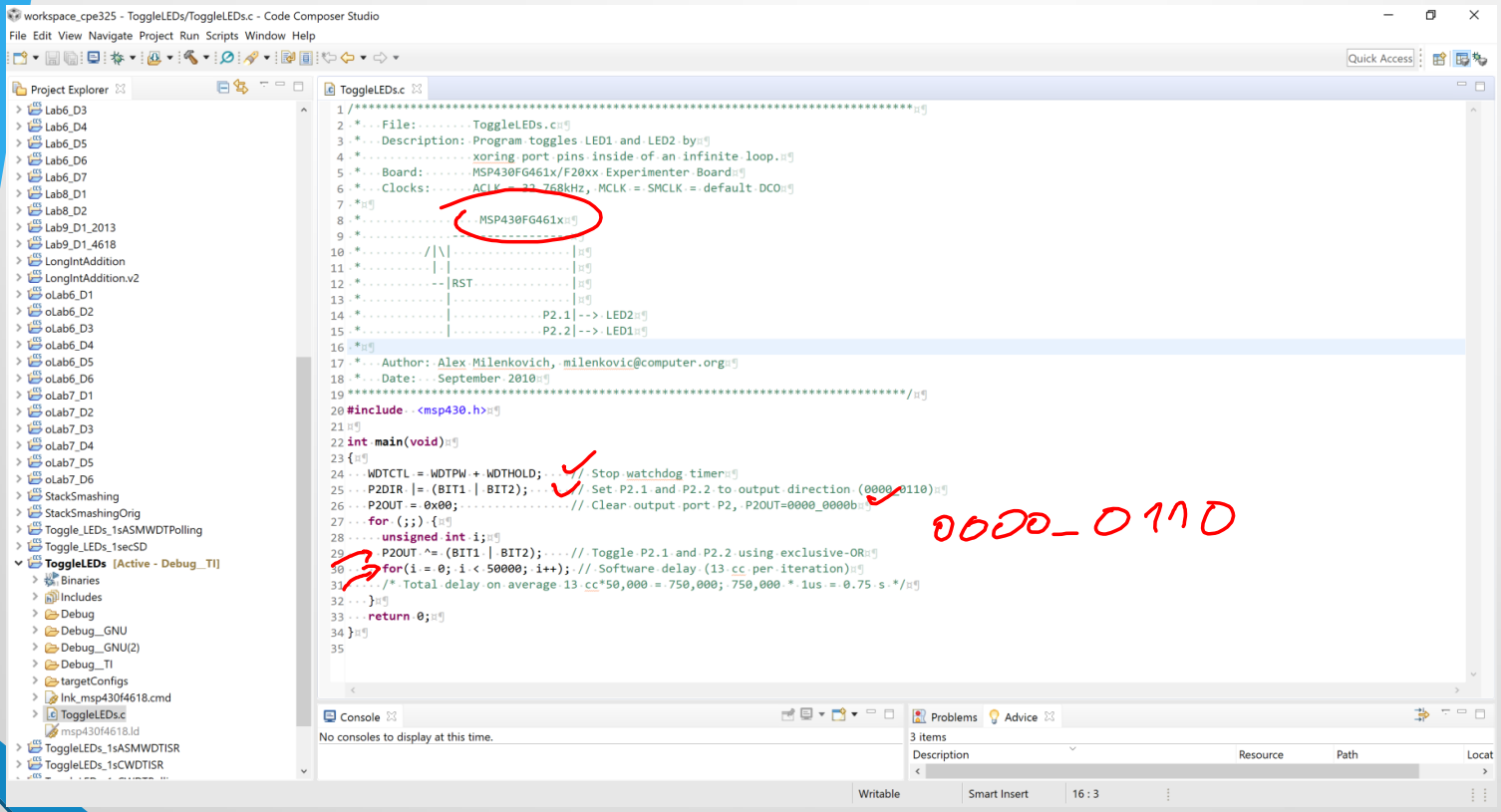
- Right-click on Active Project (ToggleLEDs)
- Select CCS Build>MSP430 Compiler>Processor Options
  - Silicon version: select msp instead of mspx
  - Code model: small; Data model: small (data will be in lower 64KB)
  - Press: Apply and close



LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Code Composer Studio: Edit View

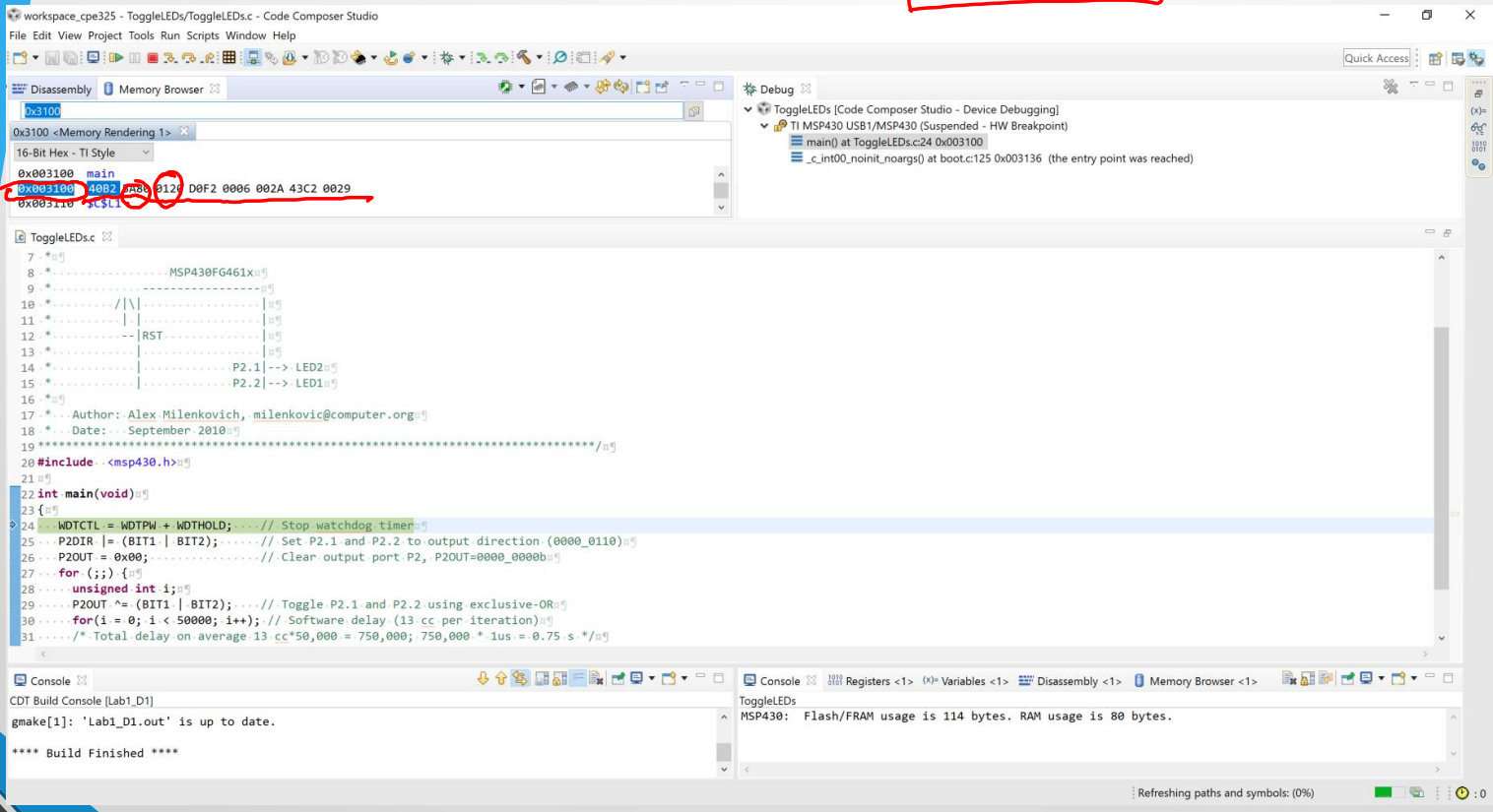


LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Code Composer Studio: Debug View

0x03100

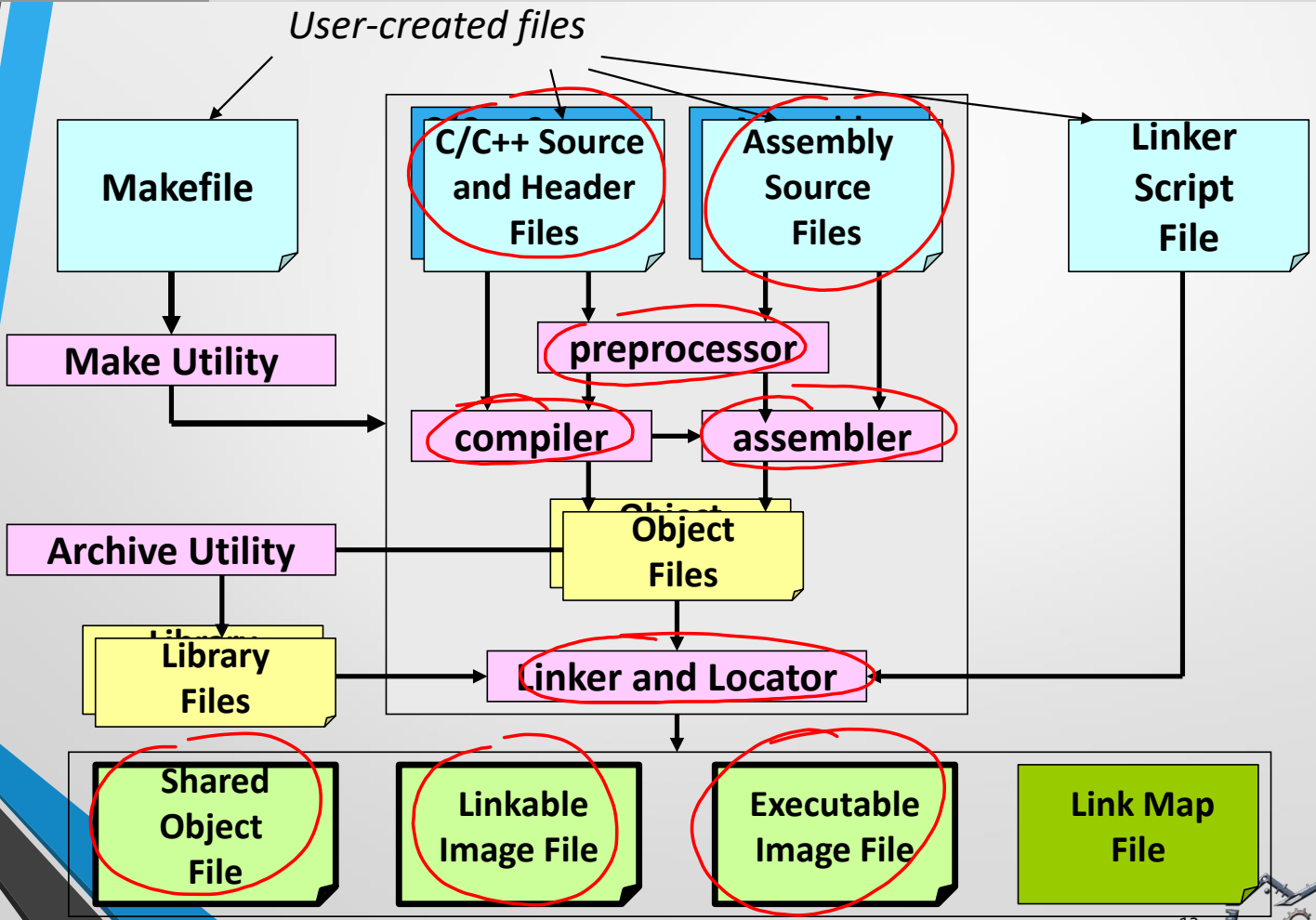


Executable File

- 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

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

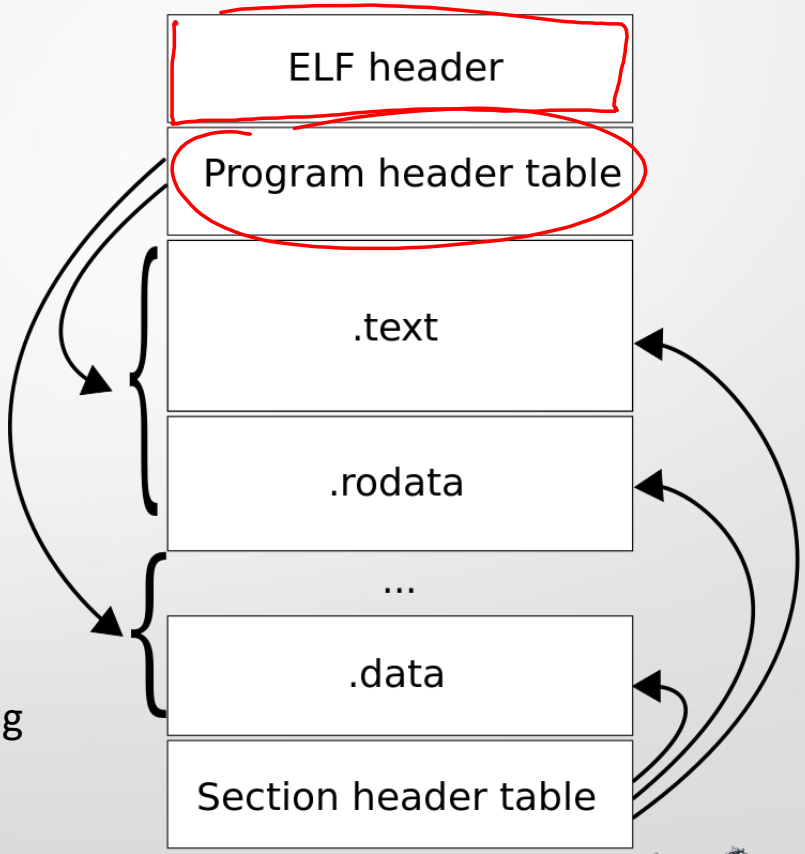
Source Translation



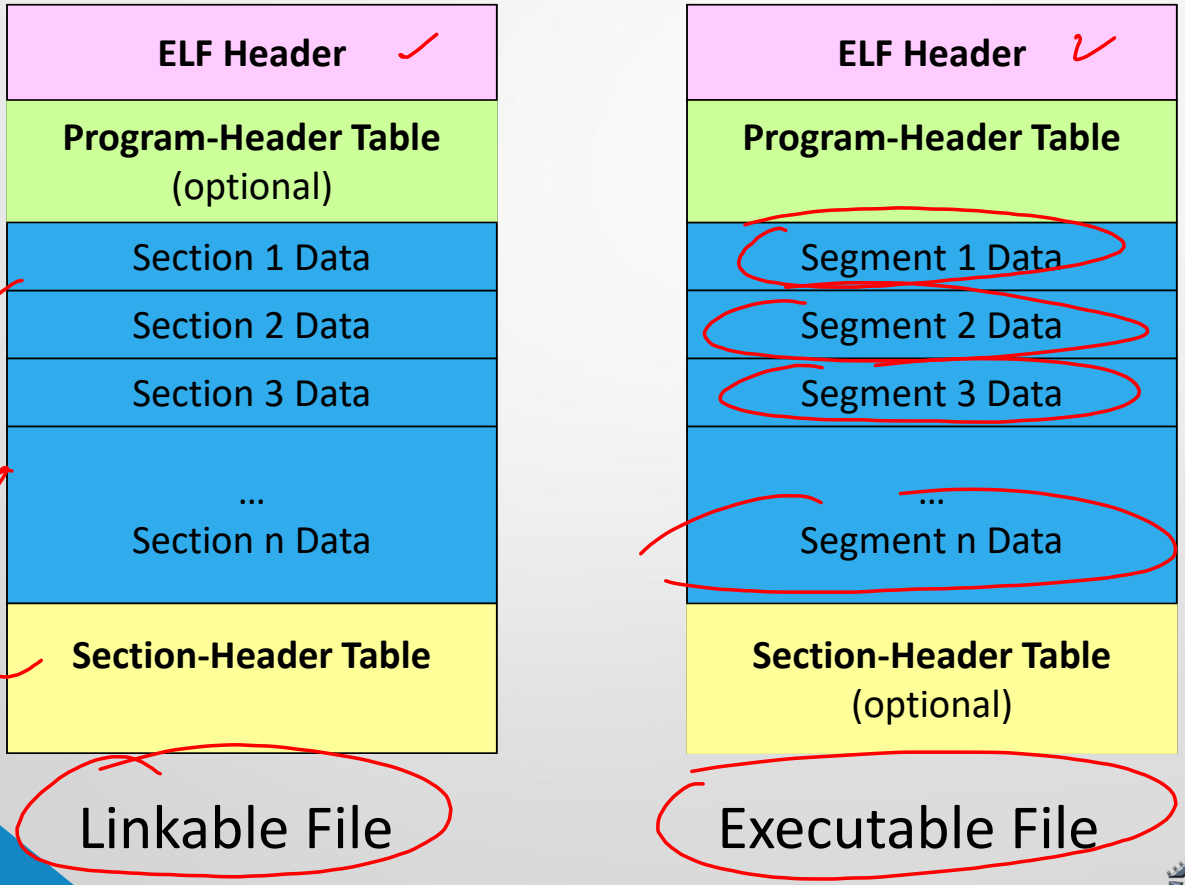
LaCASA

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



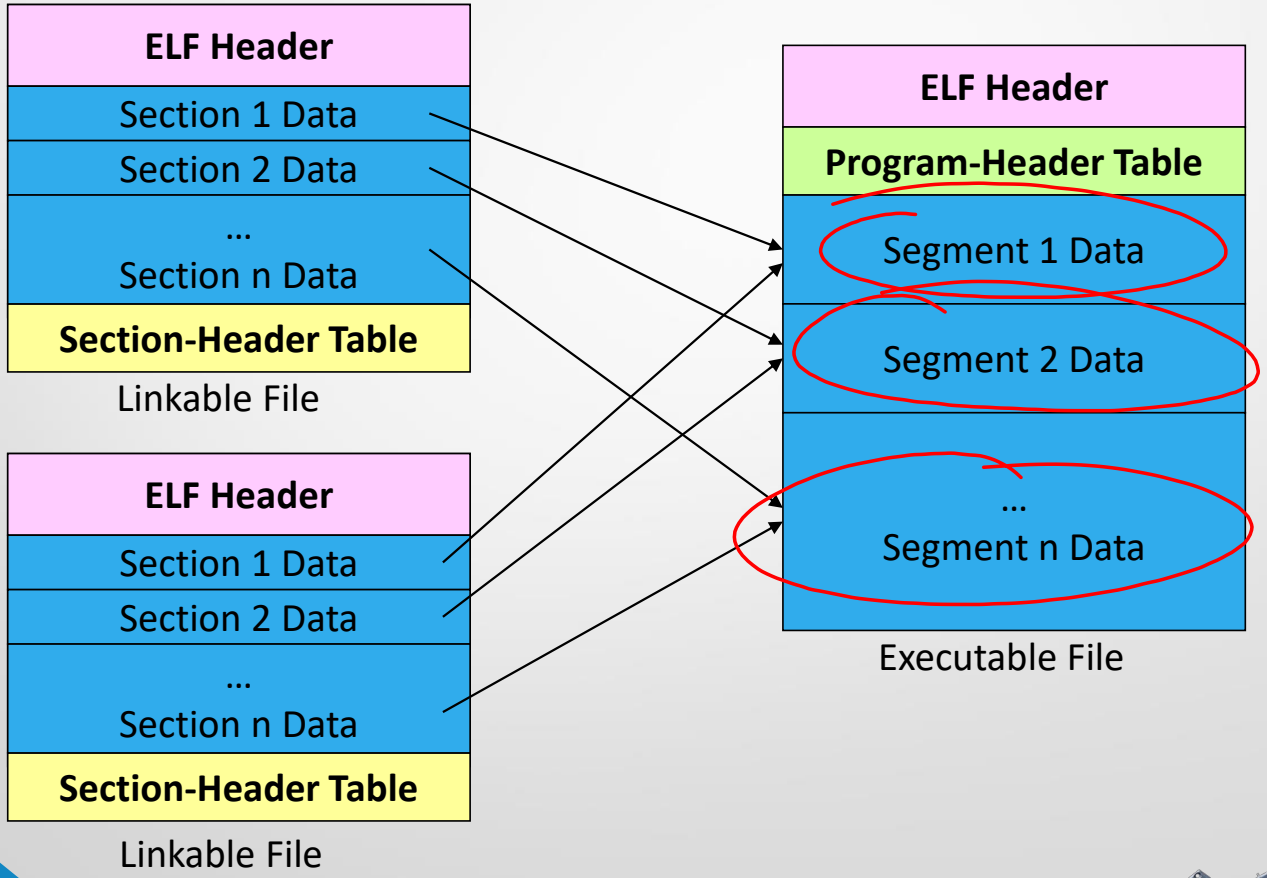
ELF Linking View

- 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

Sections	Description
.interp	Path name of program interpreter
.text	Code (executable instructions) of a program
.data	Initialized data
.bss	Uninitialized data
.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

What Does a Linker Do?



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

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)

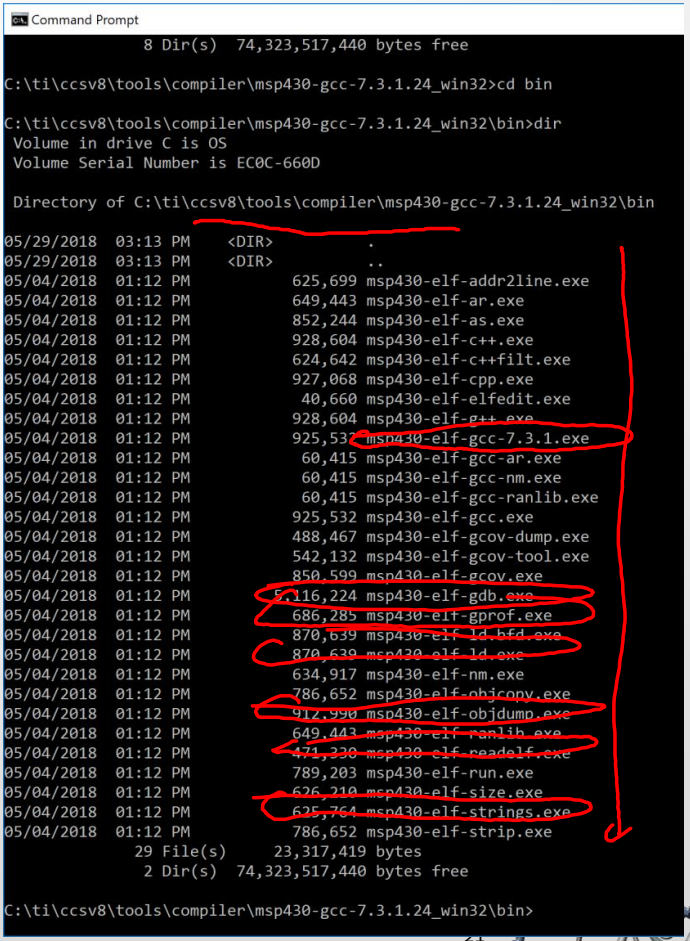
ELF Loaders

- 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

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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 to the Path System Environment Variable
  - Go to the project's working directory where the ELF file is located (e.g., C:\Users\milenka\workspace_cpe325\ToggleLEDs\Debug)



GNU Binary Utilities (Binutils)

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

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

readelf Utility

```
C:\Users\milenka\workspace_cpe325\BlinkLEDs\Debug>msp430-elf-readelf --help
Usage: readelf <option(s)> elf-file(s)
 Display information about the contents of ELF format files
 Options are:
  -a --all               Equivalent to: -h -l -S -s -r -d -V -A -I
  -h --file-header       Display the ELF file header
  -l --program-headers   Display the program headers
     --segments          An alias for --program-headers
  -S --section-headers   Display the sections' header
     --sections          An alias for --section-headers
  -g --section-groups    Display the section groups
  -t --section-details   Display the section details
  -e --headers           Equivalent to: -h -l -S
  -s --syms              Display the symbol table
     --symbols           An alias for --syms
  --dyn-syms             Display the dynamic symbol table
  -n --notes             Display the core notes (if present)
  -r --relocs            Display the relocations (if present)
  -u --unwind            Display the unwind info (if present)
  -d --dynamic           Display the dynamic section (if present)
  -V --version-info      Display the version sections (if present)
  -A --arch-specific     Display architecture specific information (if any)
  -c --archive-index     Display the symbol/file index in an archive
  -D --use-dynamic       Use the dynamic section info when displaying symbols
  . . . (continued)
```


LaCASA

readelf Utility (cont'd)

```
C:\Users\milenka\workspace_cpe325\BlinkLEDs\Debug>msp430-elf-readelf --help
...
  -D --use-dynamic       Use the dynamic section info when displaying symbols
  -x --hex-dump=<number|name>
                         Dump the contents of section <number|name> as bytes
  -p --string-dump=<number|name>
                         Dump the contents of section <number|name> as strings
  -R --relocated-dump=<number|name>
                         Dump the contents of section <number|name> as relocated bytes
  -z --decompress        Decompress section before dumping it
  -w[lLiaprmfFsoRt] or
  --debug-dump[=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 the contents of DWARF2 debug sections
  --dwarf-depth=N        Do not display DIEs at depth N or greater
  --dwarf-start=N        Display DIEs starting with N, at the same depth
                         or deeper
  -I --histogram         Display histogram of bucket list lengths
  -W --wide              Allow output width to exceed 80 characters
  @<file>                Read options from <file>
  -H --help              Display this information
  -v --version           Display the version number of readelf
Report bugs to <http://www.sourceware.org/bugzilla/>
```

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

objdump Utility

```
C:\Users\milenka\workspace_cpe325\BlinkLEDs\Debug__TI>msp430-elf-objdump --help
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[lLiaprmfFsoRt] 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
```


LaCASA

objdump Utility (cont'd)

```
C:\Users\milenka\workspace_cpe325\BlinkLEDs\Debug__TI>msp430-elf-objdump --help
. . .
-t, --syms                Display the contents of the symbol table(s)
  -T, --dynamic-syms       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
  -l, --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'
```

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

objdump Utility

```
C:\Users\milenka\workspace_cpe325\BlinkLEDs\Debug__TI>msp430-elf-objdump --help
. . .
-w, --wide                      Format output for more than 80 columns
  -z, --disassemble-zeroes        Do not skip blocks of zeroes when disassembling
      --start-address=ADDR        Only process data whose address is >= ADDR
      --stop-address=ADDR         Only process data whose address is <= ADDR
      --prefix-addresses          Print complete address alongside disassembly
      --[no-]show-raw-insn        Display hex alongside symbolic disassembly
      --insn-width=WIDTH          Display WIDTH bytes on a single line for -d
      --adjust-vma=OFFSET         Add OFFSET to all displayed section addresses
      --special-syms              Include special symbols in symbol dumps
      --prefix=PREFIX             Add PREFIX to absolute paths for -S
      --prefix-strip=LEVEL        Strip initial directory names for -S
      --dwarf-depth=N         Do not display DIEs at depth N or greater
      --dwarf-start=N         Display DIEs starting with N, at the same depth
                              or deeper
      --dwarf-check           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
Report bugs to <http://www.sourceware.org/bugzilla/>.
```


LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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
  -<number>                   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/>
```


LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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 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
```

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x000034 0x000030b0 0x000030b0 0x00000 0x00050 RW  0x4
  LOAD           0x000034 0x00003100 0x00003100 0x0004e 0x0004e R E 0x2
  LOAD           0x000084 0x0000ffbe 0x0000ffbe 0x00002 0x00002 R   0x1
  LOAD           0x000088 0x0000ffde 0x0000ffde 0x00022 0x00022 R   0x2

 Section to Segment mapping:
  Segment Sections...
   00     .stack
   01     .text .text:_isr
   02     $fill000
   03     DMA BASICTIMER PORT2 USART1TX USART1RX PORT1 TIMERA1 TIMERA0 ADC12 USCIAB0TX
USCIAB0RX WDT COMPARATORA TIMERB1 TIMERB0 NMI .reset
```


LaCASA

Reversing the Main

- Go step-by-step through the main code
- What does it do?
- How does it do it?

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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:

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

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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
```

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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
```


LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Objectives

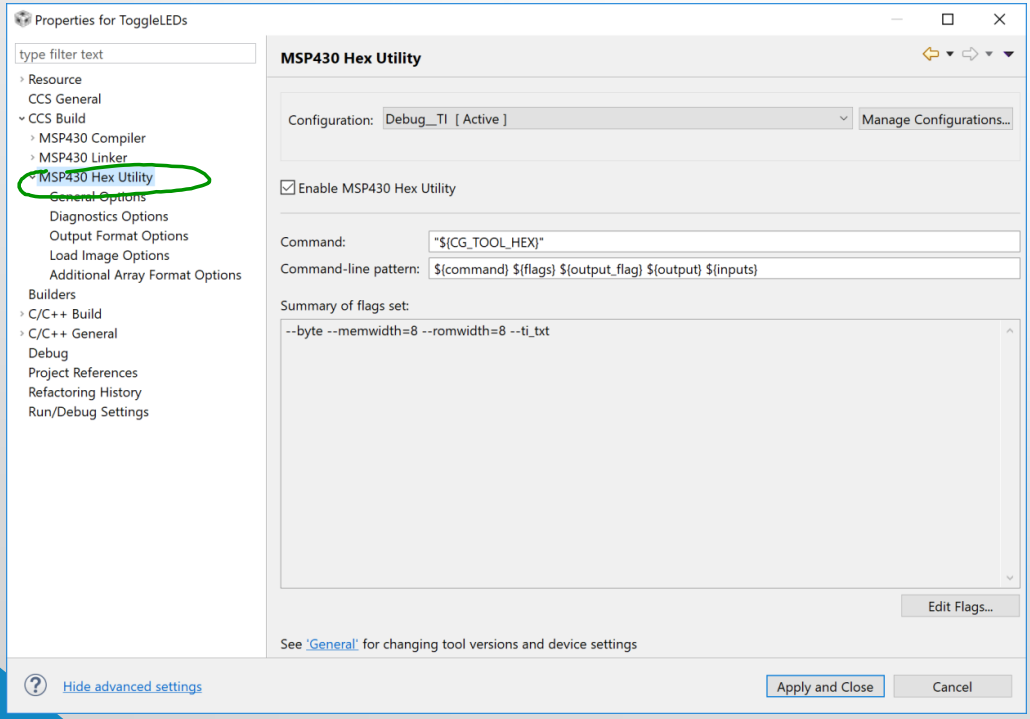
- Objectives
  - *Learn How to Create a HEX File using TI Composer Studio*
  - *Learn How to Program the Board Using MSP430 Flasher and HEX File*
  - *Learn How to Retrieve Code from the Board*
  - *Learn How to Disassemble the Retrieved Code*
- Software resources
  - *TI Code Composer with GNU tools*
  - *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*

LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Enable MSP430 Hex Utility

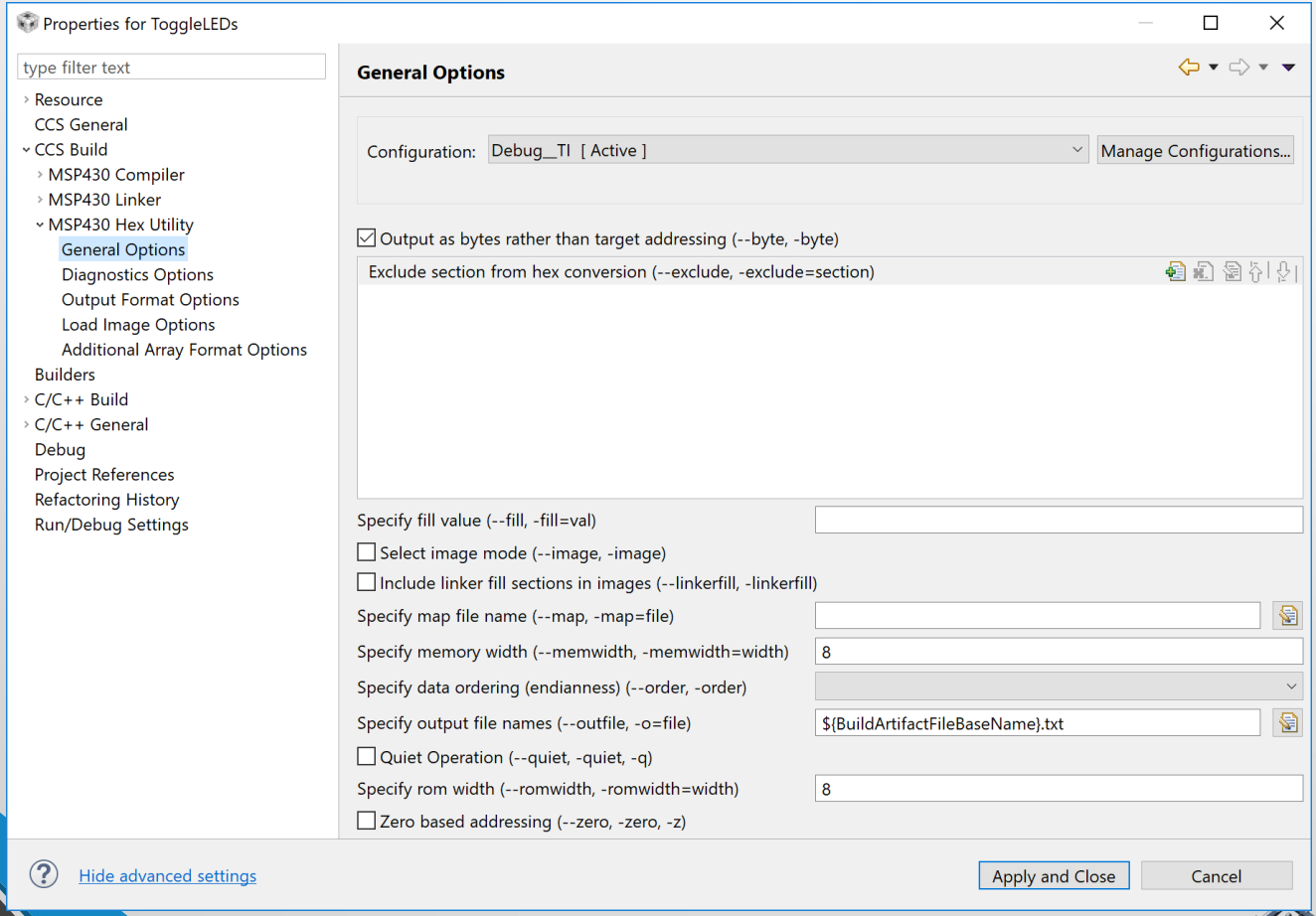
- CCS-General: Select TI compiler
- CCS-Build: Check "Enable MSP430 Hex Utility"



LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

General Options

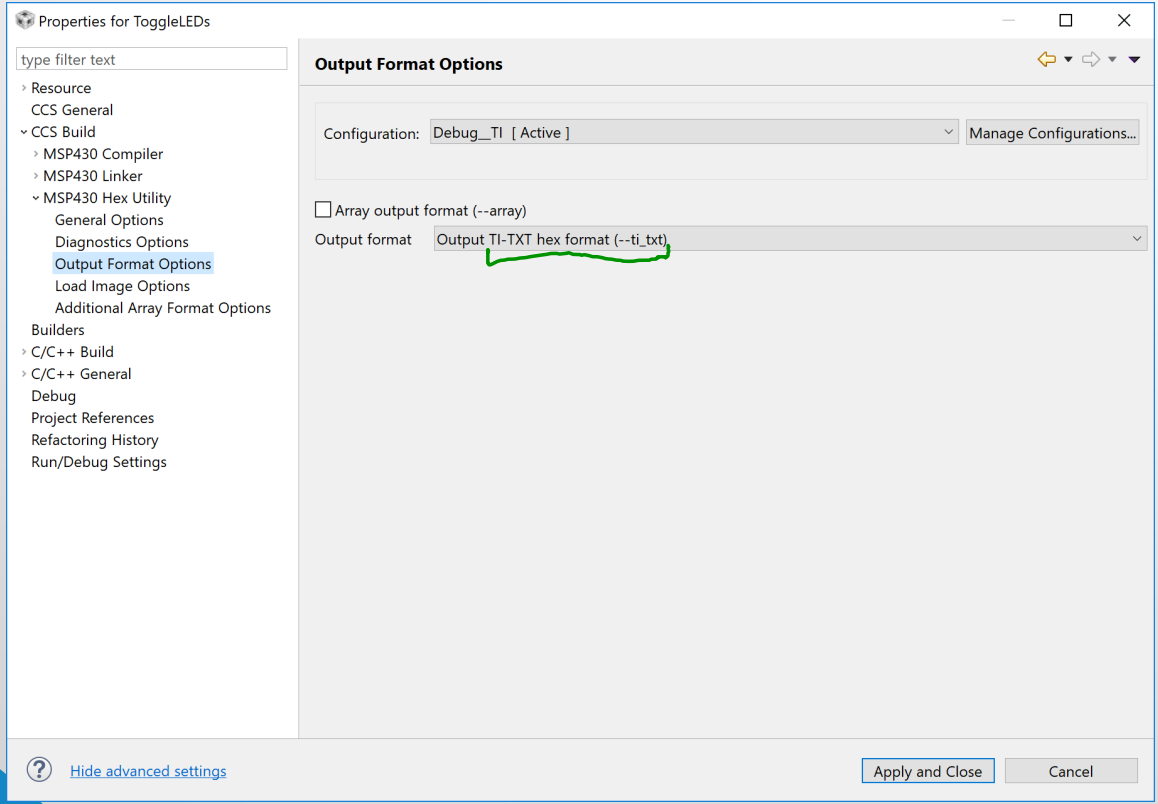


LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Output Format Options

- Select Output TI-TXT hex format



LaCASA

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

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

MSP430Flasher.exe

```
C:\ti\MSPFlasher_1.3.18>MSP430Flasher.exe
* -----/|---------------------------------------------------------------- *
*      / |__                                                              *
*     /_   /   MSP Flasher v1.3.18                                       *
*       | /                                                              *
* -----|/---------------------------------------------------------------- *
*
* Evaluating triggers...done
  No arguments. Aborting.
*******************************************************************************

 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 |       specifies the COM port of a connected debug tool
    COMn (Win) |             (default: TIUSB/USB = first detected FET tool is
    ttyACMn (Linux) |        used) For info on how to connect to specific eZ-
    usbmodem* (OSX)          tools, see the MSP Flasher manual.
                             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)

`-r [filenname,mem_sect]`	specifies a memory section to read from and a file to write to. mem_sect: 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
`-w filenname`	specifies a file for device programming. Supported data formats: TI-TXT (.txt), Intel-Hex (.hex/.a43)
`-b`	unlocks BSL memory for writing (use only with -w switch)
`-u`	unlocks InfoA memory for writing (use only with -w switch)
`-e ERASE_ALL | ERASE_MAIN | ERASE_SEGMENT | ERASE_TOTAL | ERASE_USER_CODE | NO_ERASE`	Erase memory. Default: ERASE_ALL (INFO&MAIN) ERASE_MAIN: erase MAIN memory only ERASE_SEGMENT: erase segment to be programmed only ERASE_TOTAL: applicable for FR5xx/FR6xx only! ERASE_USER_CODE: applicable for FR4xx only! NO_ERASE: use only with -w switch
`-v filename (optional)`	triggers verification of the target memory against a target code file. If -w is used, no argument is required. For a stand-alone verify, provide the path to a target code file as an argument.
`-z [exit_spec,...]`	specifies state of device on exit (view available 'exit_spec's using -x switch)
`-g`	switches log OFF (default: ON)
`-q`	triggers QUIET MODE (no system messages displayed)

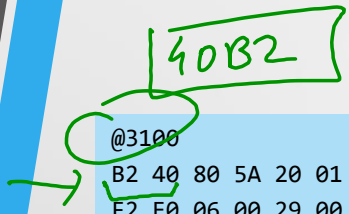
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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 |     OBSOLETE! If used, this option will be ignored!
    AUTO                     Communication protocol is determined automatically


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

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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
```


LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Download HEX File Using MSP430Flasher

F5529

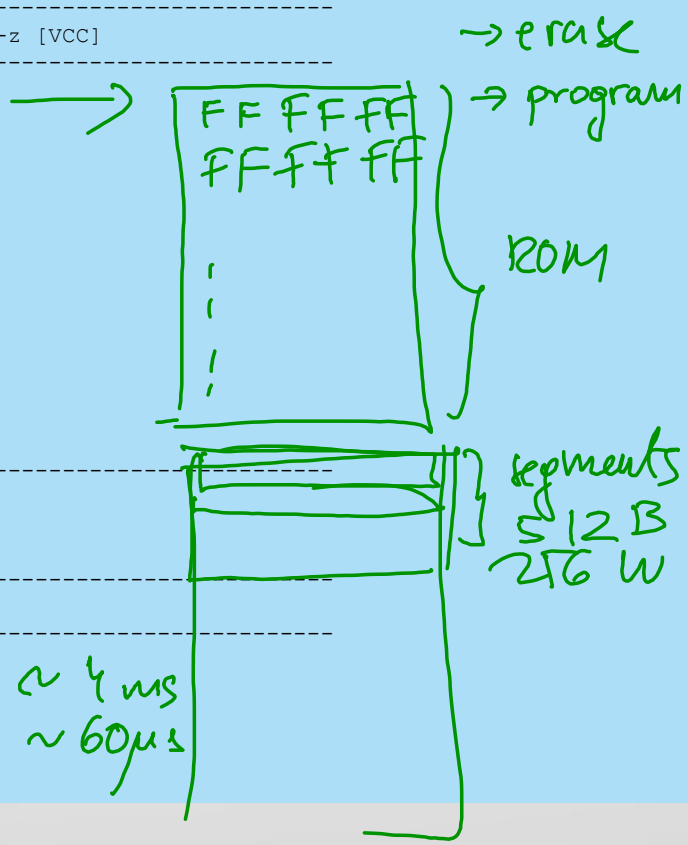
```
C:\Users\milenka\workspace_cpe325\ToggleLEDs\Debug__TI>MSP430Flasher.exe -n MSP430FG4618 -w
ToggleLEDs.txt -v -z [VCC]
* ------/|-------------------------------------------------------------------- *
*      / |__                                                                   *
*     /_   /   MSP 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
*
```


LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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, ClockCntrl 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)
* ------------------------------------------------------------------------------
*/
```



LaCASA

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]`

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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, ClockCntrl 2
* Read File   : output.txt (memory segment = MAIN)
* VCC OFF
* ------------------------------------------------------------------------------
* Powering down...done
* Disconnecting from device...done
*
* ------------------------------------------------------------------------------
* Driver      : closed (No error)
* ------------------------------------------------------------------------------
*/
```

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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)

```
@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 FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
. . .
```


LaCASA

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

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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.

Addr    Opcode Instruction                          Cycles
------- ------ -----------------------------------  ------
0x3100: 0x40b2 mov.w #0x5a80, &0x0120               5
0x3102: 0x5a80
0x3104: 0x0120
0x3106: 0xd0f2 bis.b #0x06, &0x002a                 5
0x3108: 0x0006
0x310a: 0x002a
0x310c: 0x43c2 mov.b #0, &0x0029                    4
0x310e: 0x0029
0x3110: 0xe0f2 xor.b #0x06, &0x0029                 5
0x3112: 0x0006
0x3114: 0x0029
0x3116: 0x430f mov.w #0, r15                        1
0x3118: 0x903f cmp.w #0xc350, r15                   2
0x311a: 0xc350
0x311c: 0x2ff9 jhs 0x3110  (offset: -14)            2
```


LaCASA

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.

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

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

What is in IVT?

```
FFD0:   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 28 31
```

=> RESET VECTOR contains 0x3128

LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Reversing 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.

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


LaCASA

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

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.

Addr    Opcode Instruction                          Cycles
------- ------ -----------------------------------  ------
. . .
0x3128: 0x4031 mov.w #0x3100, SP                    2      // initialize SP
0x312a: 0x3100
0x312c: 0x12b0 call #0x3142                         5      // call a subroutine at 0x3142
0x312e: 0x3142
0x3130: 0x430c mov.w #0, r12                        1      // r12 <= 0
0x3132: 0x12b0 call #0x3100                         5      // call 0x3100 (main program)
0x3134: 0x3100
0x3136: 0x431c mov.w #1, r12                        1      // r12 <= 1
0x3138: 0x12b0 call #0x313c                         5      // call a subroutine at 0x313c
0x313a: 0x313c
0x313c: 0x4303 nop    --  mov.w #0, CG              1      // nop
0x313e: 0x3fff jmp 0x313e  (offset: -2)             2      // jump to itself
0x3140: 0x4303 nop    --  mov.w #0, CG              1
0x3142: 0x431c mov.w #1, r12                        1      // r12 <= 1
0x3144: 0x4130 ret    --  mov.w @SP+, PC            3      // return
0x3146: 0xd032 bis.w #0x0010, SR                    2
0x3148: 0x0010
0x314a: 0x3ffd jmp 0x3146  (offset: -6)             2
0x314c: 0x4303 nop    --  mov.w #0, CG              1
0x314e: 0xffff and.b @r15+, 0(r15)                  5
0x3150: 0x0000
```

What Does the Code Do?

- Blink LEDs connected on Port2 pins 1 and 2
- Delay: 50,000*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