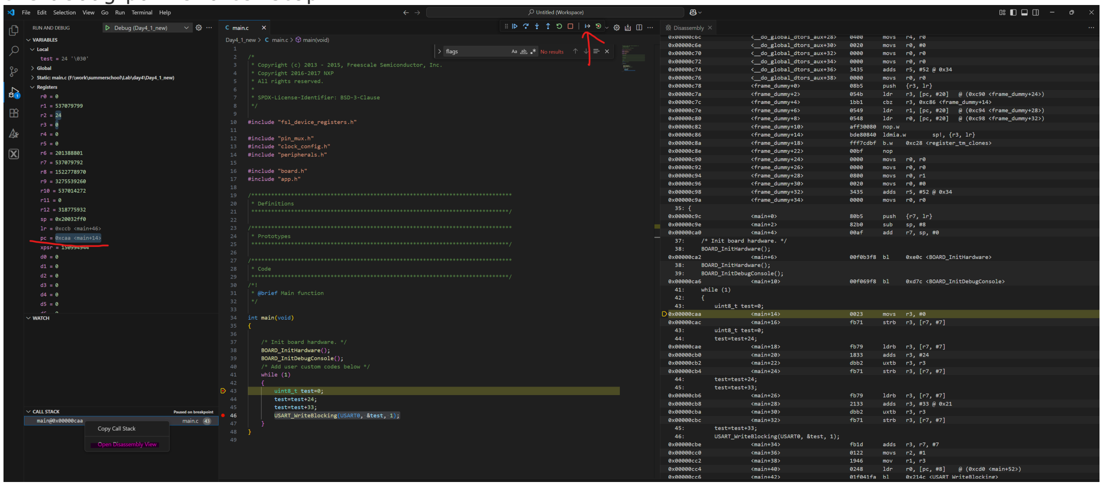
Lab 4 exercises

June 25, 2024

1. Manual exercise: Write a simple program that does some aditions(make sure they are not optimized). For example: ~~C uint8_t test=0; test=test+24; test=test+33;~~

   Send `test` over UART. Using the dissasembly view, the registers and breakpoints, modify the program counter to skip the: +33 adition. To do this, you will need to identify the `adds` command in the dissasembly view that must be skipped and skip it by modifying the PC register. Check in Hercules if you receive the modified result. This showcases a basic attack on an embedded target.

Note: To open the dissasembly view right click on any entry in CALL STACK. In order to step through assembly instructions you will need to activate the Instruction Stepping Mode found on the debug pannel after stop.



```
test = 24 '\030'
r0 = 0
r1 = 537079799
r2 = 24
r3 = 0
r4 = 0
r5 = 0
r6 = 201388801
r7 = 537079792
r8 = 1522778970
r9 = 3275539260
r10 = 537014272
r11 = 0
r12 = 318775932
sp = 0x20032ff0
lr = 0xccb <main+46>
pc = 0xcaa <main+14>
xpsr = 150994944
d0 = 0
d1 = 0
d2 = 0
d3 = 0
d4 = 0
d5 = 0

main@0x00000caa                main.c 43
```

```c
/*
 * Copyright (c) 2013 - 2015, Freescale Semiconductor, Inc.
 * Copyright 2016-2017 NXP
 * All rights reserved.
 *
 * SPDX-License-Identifier: BSD-3-Clause
 */

#include "fsl_device_registers.h"

#include "pin_mux.h"
#include "clock_config.h"
#include "peripherals.h"

#include "board.h"
#include "app.h"

/*******************************************************************************
 * Definitions
 ******************************************************************************/

/*******************************************************************************
 * Prototypes
 ******************************************************************************/

/*******************************************************************************
 * Code
 ******************************************************************************/
/*!
 * @brief Main function
 */

int main(void)
{

    /* Init board hardware. */
    BOARD_InitHardware();
    BOARD_InitDebugConsole();
    /* Add user custom codes below */
    while (1)
    {
        uint8_t test=0;
        test=test+24;
        test=test+33;
        USART_WriteBlocking(USART0, &test, 1);
    }
}
```

```
0x00000c6c    <__do_global_dtors_aux+28>    0400      movs   r4, r0
0x00000c6e    <__do_global_dtors_aux+30>    0020      movs   r0, #0
0x00000c70    <__do_global_dtors_aux+32>    0000      movs   r0, r0
0x00000c72    <__do_global_dtors_aux+34>    0000      movs   r0, r0
0x00000c74    <__do_global_dtors_aux+36>    3435      adds   r5, #52 @ 0x34
0x00000c76    <__do_global_dtors_aux+38>    0000      movs   r0, r0
0x00000c78    <frame_dummy+0>               08b5      push   {r3, lr}
0x00000c7a    <frame_dummy+2>               054b      ldr    r3, [pc, #20]   @ (0xc90 <frame_dummy+24>)
0x00000c7c    <frame_dummy+4>               1bb1      cbz    r3, 0xc86 <frame_dummy+14>
0x00000c7e    <frame_dummy+6>               0549      ldr    r1, [pc, #20]   @ (0xc94 <frame_dummy+28>)
0x00000c80    <frame_dummy+8>               0548      ldr    r0, [pc, #20]   @ (0xc98 <frame_dummy+32>)
0x00000c82    <frame_dummy+10>              aff30080  nop.w
0x00000c86    <frame_dummy+14>              bde80840  ldmia.w       sp!, {r3, lr}
0x00000c8a    <frame_dummy+18>              fff7cdbf  b.w    0xc28 <register_tm_clones>
0x00000c8e    <frame_dummy+22>              00bf      nop
0x00000c90    <frame_dummy+24>              0000      movs   r0, r0
0x00000c92    <frame_dummy+26>              0000      movs   r0, r0
0x00000c94    <frame_dummy+28>              0800      movs   r0, r1
0x00000c96    <frame_dummy+30>              0020      movs   r0, #0
0x00000c98    <frame_dummy+32>              3435      adds   r5, #52 @ 0x34
0x00000c9a    <frame_dummy+34>              0000      movs   r0, r0
  35: {
0x00000c9c    <main+0>                      80b5      push   {r7, lr}
0x00000c9e    <main+2>                      82b0      sub    sp, #8
0x00000ca0    <main+4>                      00af      add    r7, sp, #0
  37:     /* Init board hardware. */
  38:     BOARD_InitHardware();
0x00000ca2    <main+6>                      00f0b3f8  bl     0xe0c <BOARD_InitHardware>
  38:     BOARD_InitHardware();
  39:     BOARD_InitDebugConsole();
0x00000ca6    <main+10>                     00f069f8  bl     0xd7c <BOARD_InitDebugConsole>
  41:     while (1)
  42:     {
  43:         uint8_t test=0;
0x00000caa    <main+14>                     0023      movs   r3, #0
0x00000cac    <main+16>                     fb71      strb   r3, [r7, #7]
  43:         uint8_t test=0;
  44:         test=test+24;
0x00000cae    <main+18>                     fb79      ldrb   r3, [r7, #7]
0x00000cb0    <main+20>                     1833      adds   r3, #24
0x00000cb2    <main+22>                     dbb2      uxtb   r3, r3
0x00000cb4    <main+24>                     fb71      strb   r3, [r7, #7]
  44:         test=test+24;
  45:         test=test+33;
0x00000cb6    <main+26>                     fb79      ldrb   r3, [r7, #7]
0x00000cb8    <main+28>                     2133      adds   r3, #33 @ 0x21
0x00000cba    <main+30>                     dbb2      uxtb   r3, r3
0x00000cbc    <main+32>                     fb71      strb   r3, [r7, #7]
  45:         test=test+33;
  46:         USART_WriteBlocking(USART0, &test, 1);
0x00000cbe    <main+34>                     fb1d      adds   r3, r7, #7
0x00000cc0    <main+36>                     0122      movs   r2, #1
0x00000cc2    <main+38>                     1946      mov    r1, r3
0x00000cc4    <main+40>                     0248      ldr    r0, [pc, #8]    @ (0xcd0 <main+52>)
0x00000cc6    <main+42>                     01f041fa  bl     0x214c <USART_WriteBlocking>
```

2. Implement a non time constant memory compare. For example, as an input we receive a input via UART of 3 bytes, where each element can have values between 0 and 2. Non time constant means that we check each byte and and the first mismatch we return "FAIL" on UART, if all bytes pass we return "PASS". Add a "secret key" to compare against on the microcontroller.

   1. Add a delay (1-2 seconds) such that we can distinguish when the function fails after 1 byte or 2 or 3. This is the non time constant part.
   2. Try to see if you can guess the secret password just by using Hercules and seeing how fast it fails. How many attempts do we need to crack the code?
   3. Change the implementtion to be time constant.

3. Implement the Caesar cipher using a TLV encoding on UART. Have separate commands for encryption and decryption. Shift by 3 characters.

4. Use source files from aes to implement a chipher with commands for encryption and decryption. In `Project Files\armgcc\CMakeLists.txt` add `"${ProjDirPath}/../AES.c"` to `add_executable(${MCUX_SDK_PROJECT_NAME}`. Use https://www.toolhelper.cn/en/SymmetricEncryption/AES to create a reference to compare against.

   The AES algorithm works on blocks of 16 bytes (input/output): Data must be sliced into 16 byte chunks.

5. Implement CBC mode of operation.

   In CBC mode, each plaintext block is XORed with the previous ciphertext block before encryption. The first block uses an IV (Initialization Vector) instead. This helps hide patterns in the data. Encrypt the message below.

   ```
   See you at 13 o'clock in class, do not be late!!!
   ```

   Modify input data for decryption algorithm in such way the decrypted text still appears valid.

!Hint if you change one byte from IV only corresponding byte from first block will be altered.

6. Implement CBC MAC

   CBC-MAC (Cipher Block Chaining Message Authentication Code) is a MAC algorithm based on the CBC mode of a block cipher. It always uses an Initialization Vector (IV) set to zero.

7. Implement CTR mode of operation. (Optional can be done in day5)

   CTR (Counter) mode uses a counter value that is encrypted with a block cipher. The resulting ciphertext is then XORed with the plaintext to produce the final encrypted output. The counter is incremented for each block of plaintext.