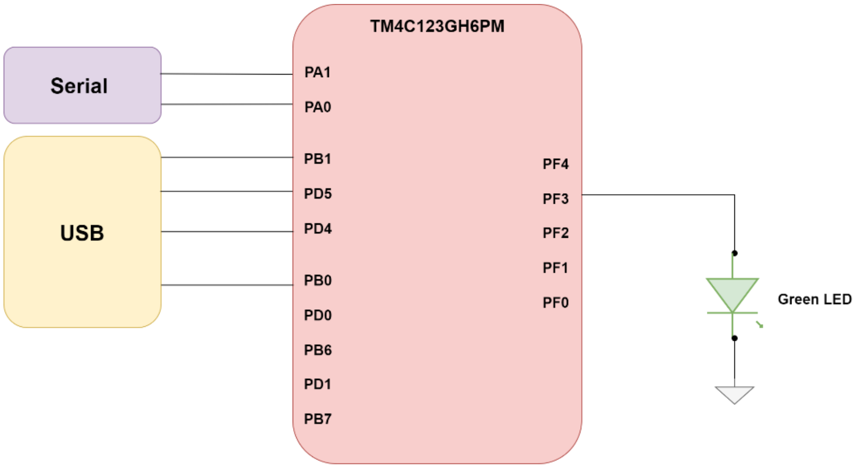
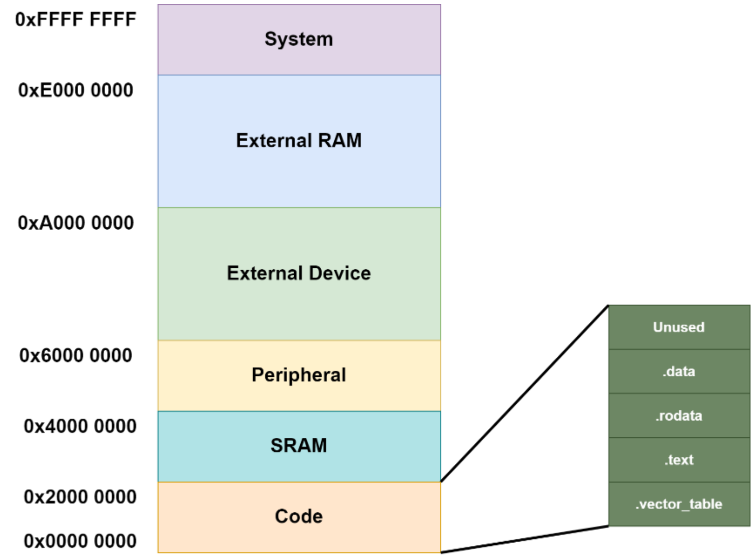
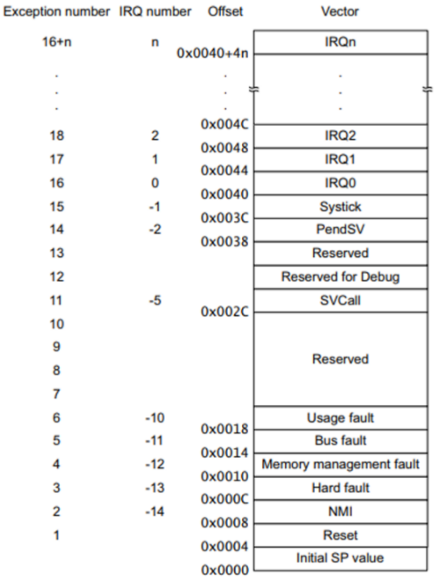
# Lab 03 Bare-metal software on TM4C123 ARM CORTEXM4:

We will write a bare-metal SW to toggle **PF3** which relates to green LED.



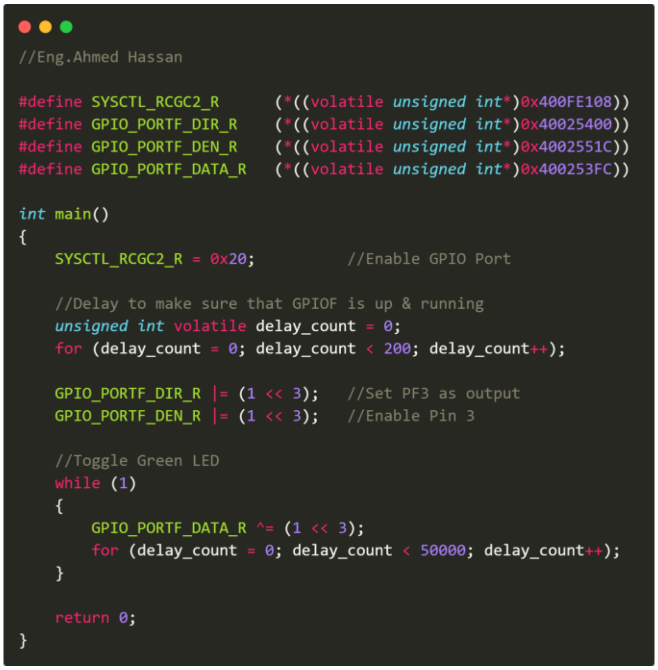
## CortexM4 Memory map:





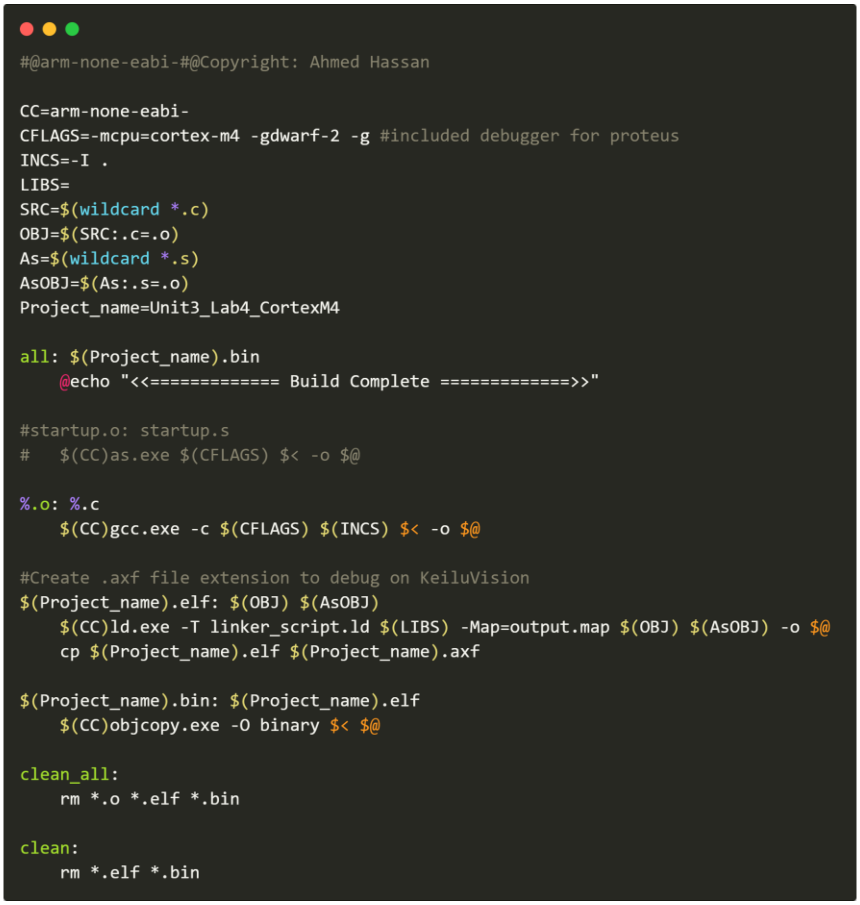
| Exception number | IRQ number | Offset | Vector |
|---|---|---|---|
| 16+n | n | 0x0040+4n | IRQn |
| . |  | . | . |
| . |  | . | . |
| . |  | . 0x004C | . |
| 18 | 2 | 0x0048 | IRQ2 |
| 17 | 1 | 0x0044 | IRQ1 |
| 16 | 0 | 0x0040 | IRQ0 |
| 15 | -1 | 0x003C | Systick |
| 14 | -2 | 0x0038 | PendSV |
| 13 |  |  | Reserved |
| 12 |  |  | Reserved for Debug |
| 11 | -5 | 0x002C | SVCall |
| 10 |  |  | Reserved |
| 9 |  |  |  |
| 8 |  |  |  |
| 7 |  |  |  |
| 6 | -10 | 0x0018 | Usage fault |
| 5 | -11 | 0x0014 | Bus fault |
| 4 | -12 | 0x0010 | Memory management fault |
| 3 | -13 | 0x000C | Hard fault |
| 2 | -14 | 0x0008 | NMI |
| 1 |  | 0x0004 | Reset |
|  |  | 0x0000 | Initial SP value |

## Main.c code:



```c
//Eng.Ahmed Hassan

#define SYSCTL_RCGC2_R      (*((volatile unsigned int*)0x400FE108))
#define GPIO_PORTF_DIR_R    (*((volatile unsigned int*)0x40025400))
#define GPIO_PORTF_DEN_R    (*((volatile unsigned int*)0x4002551C))
#define GPIO_PORTF_DATA_R   (*((volatile unsigned int*)0x400253FC))

int main()
{
    SYSCTL_RCGC2_R = 0x20;          //Enable GPIO Port

    //Delay to make sure that GPIOF is up & running
    unsigned int volatile delay_count = 0;
    for (delay_count = 0; delay_count < 200; delay_count++);

    GPIO_PORTF_DIR_R |= (1 << 3);   //Set PF3 as output
    GPIO_PORTF_DEN_R |= (1 << 3);   //Enable Pin 3

    //Toggle Green LED
    while (1)
    {
        GPIO_PORTF_DATA_R ^= (1 << 3);
        for (delay_count = 0; delay_count < 50000; delay_count++);
    }

    return 0;
}
```

## Makefile code:



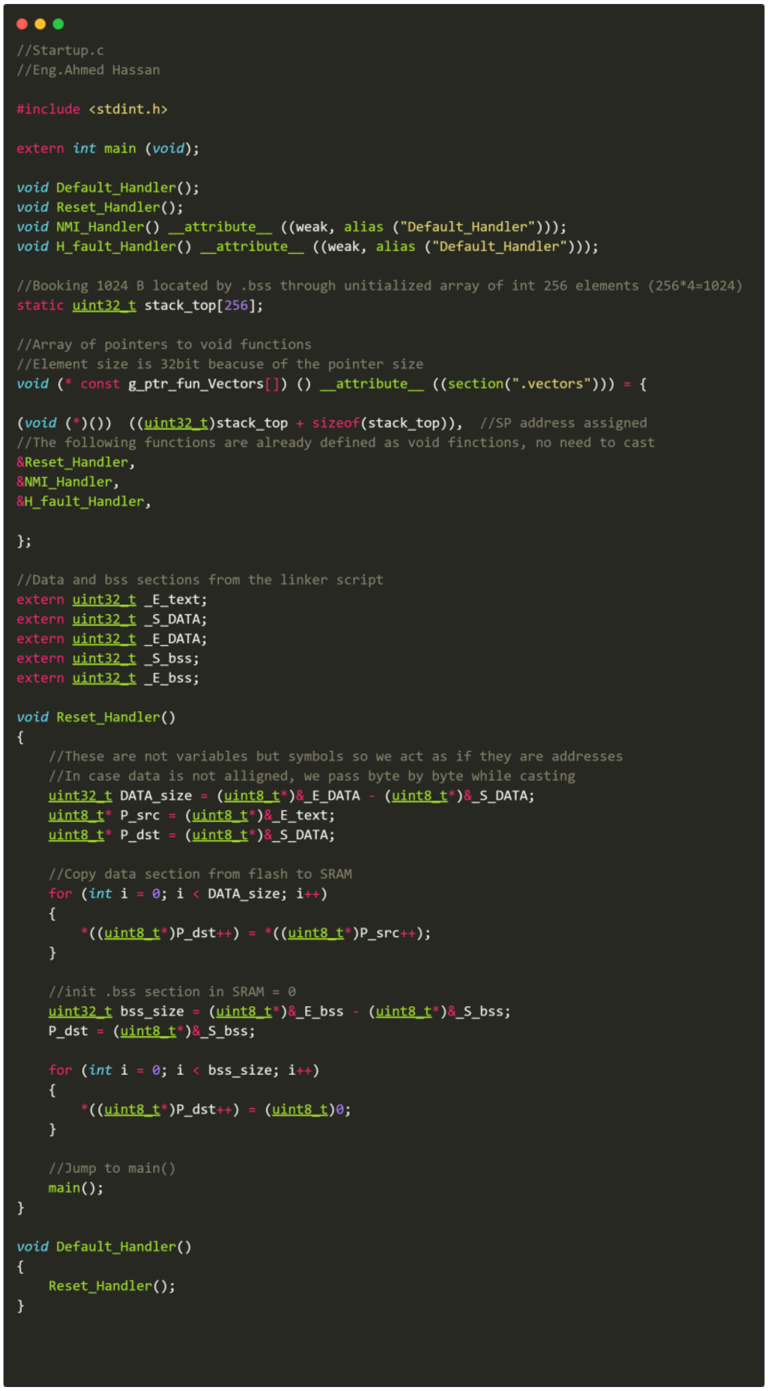
```makefile
#@arm-none-eabi-#@Copyright: Ahmed Hassan

CC=arm-none-eabi-
CFLAGS=-mcpu=cortex-m4 -gdwarf-2 -g #included debugger for proteus
INCS=-I .
LIBS=
SRC=$(wildcard *.c)
OBJ=$(SRC:.c=.o)
As=$(wildcard *.s)
AsOBJ=$(As:.s=.o)
Project_name=Unit3_Lab4_CortexM4

all: $(Project_name).bin
    @echo "<<============= Build Complete =============>>"

#startup.o: startup.s
#   $(CC)as.exe $(CFLAGS) $< -o $@

%.o: %.c
    $(CC)gcc.exe -c $(CFLAGS) $(INCS) $< -o $@

#Create .axf file extension to debug on KeiluVision
$(Project_name).elf: $(OBJ) $(AsOBJ)
    $(CC)ld.exe -T linker_script.ld $(LIBS) -Map=output.map $(OBJ) $(AsOBJ) -o $@
    cp $(Project_name).elf $(Project_name).axf

$(Project_name).bin: $(Project_name).elf
    $(CC)objcopy.exe -O binary $< $@

clean_all:
    rm *.o *.elf *.bin

clean:
    rm *.elf *.bin
```

## Startup C code:

- **Interview Question: Modify the startup code so it does not use the extern symbol of the stack_top from the linker script and the stack_top is located 1024 bytes after .bss section in the SRAM.**

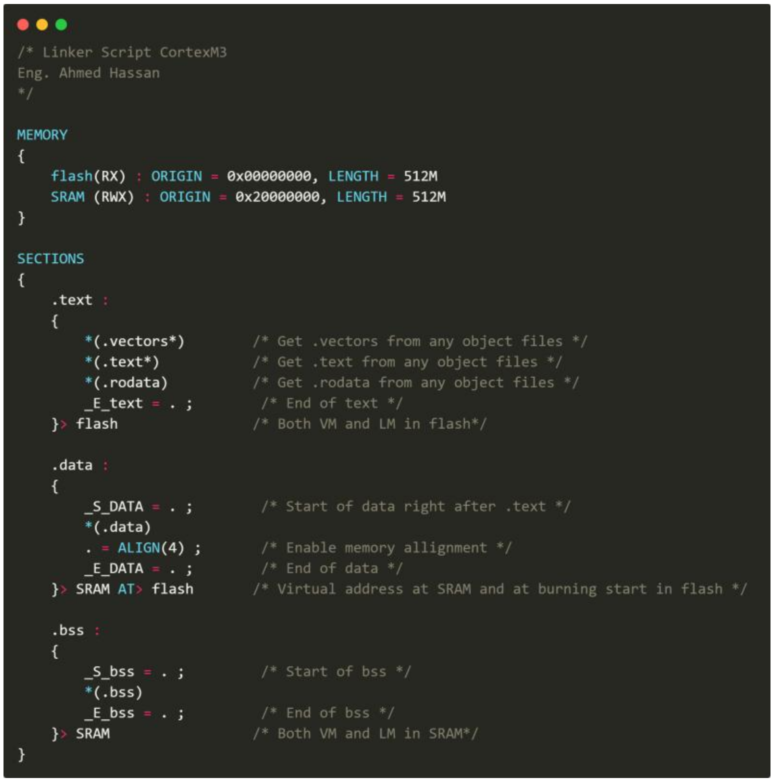
  **- Solution:** Create an **global uninitialized array** of 256 elements (1024/4) and let **SP = (array[0] + arr_size)**

  **- Explanation:** The **uninitialized array** will be stored in the **.bss** section by default.



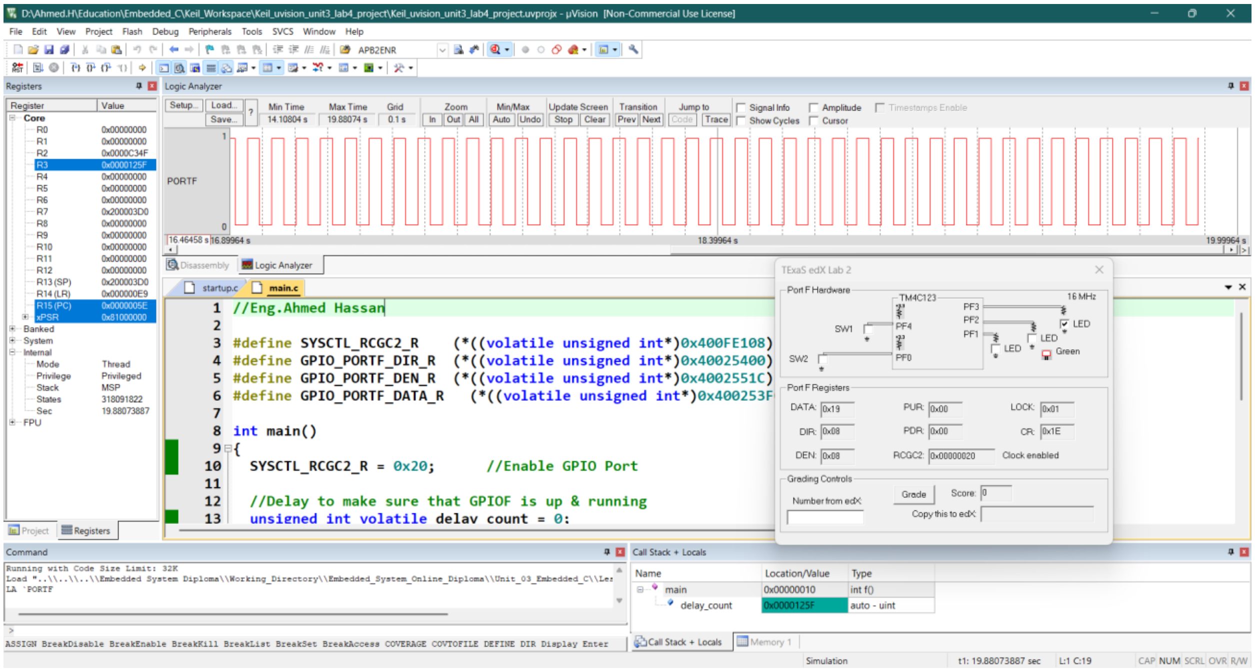
```c
//Startup.c
//Eng.Ahmed Hassan

#include <stdint.h>

extern int main (void);

void Default_Handler();
void Reset_Handler();
void NMI_Handler() __attribute__ ((weak, alias ("Default_Handler")));
void H_fault_Handler() __attribute__ ((weak, alias ("Default_Handler")));

//Booking 1024 B located by .bss through unitialized array of int 256 elements (256*4=1024)
static uint32_t stack_top[256];

//Array of pointers to void functions
//Element size is 32bit beacuse of the pointer size
void (* const g_ptr_fun_Vectors[]) () __attribute__ ((section(".vectors"))) = {

(void (*)())  ((uint32_t)stack_top + sizeof(stack_top)),  //SP address assigned
//The following functions are already defined as void finctions, no need to cast
&Reset_Handler,
&NMI_Handler,
&H_fault_Handler,

};

//Data and bss sections from the linker script
extern uint32_t _E_text;
extern uint32_t _S_DATA;
extern uint32_t _E_DATA;
extern uint32_t _S_bss;
extern uint32_t _E_bss;

void Reset_Handler()
{
    //These are not variables but symbols so we act as if they are addresses
    //In case data is not alligned, we pass byte by byte while casting
    uint32_t DATA_size = (uint8_t*)&_E_DATA - (uint8_t*)&_S_DATA;
    uint8_t* P_src = (uint8_t*)&_E_text;
    uint8_t* P_dst = (uint8_t*)&_S_DATA;

    //Copy data section from flash to SRAM
    for (int i = 0; i < DATA_size; i++)
    {
        *((uint8_t*)P_dst++) = *((uint8_t*)P_src++);
    }

    //init .bss section in SRAM = 0
    uint32_t bss_size = (uint8_t*)&_E_bss - (uint8_t*)&_S_bss;
    P_dst = (uint8_t*)&_S_bss;

    for (int i = 0; i < bss_size; i++)
    {
        *((uint8_t*)P_dst++) = (uint8_t)0;
    }

    //Jump to main()
    main();
}

void Default_Handler()
{
    Reset_Handler();
}
```

## Linker Script Code:



```
/* Linker Script CortexM3
Eng. Ahmed Hassan
*/

MEMORY
{
    flash(RX) : ORIGIN = 0x00000000, LENGTH = 512M
    SRAM (RWX) : ORIGIN = 0x20000000, LENGTH = 512M
}

SECTIONS
{
    .text :
    {
        *(.vectors*)        /* Get .vectors from any object files */
        *(.text*)           /* Get .text from any object files */
        *(.rodata)          /* Get .rodata from any object files */
        _E_text = . ;        /* End of text */
    }> flash                /* Both VM and LM in flash*/

    .data :
    {
        _S_DATA = . ;        /* Start of data right after .text */
        *(.data)
        . = ALIGN(4) ;       /* Enable memory allignment */
        _E_DATA = . ;        /* End of data */
    }> SRAM AT> flash       /* Virtual address at SRAM and at burning start in flash */

    .bss :
    {
        _S_bss = . ;         /* Start of bss */
        *(.bss)
        _E_bss = . ;         /* End of bss */
    }> SRAM                 /* Both VM and LM in SRAM*/
}
```

## Keil uVision Debug:



D:\Ahmed.H\Education\Embedded_C\Keil_Workspace\Keil_uvision_unit3_lab4_project\Keil_uvision_unit3_lab4_project.uvprojx - µVision [Non-Commercial Use License]

| Register | Value |
|---|---|
| Core | |
| R0 | 0x00000000 |
| R1 | 0x00000000 |
| R2 | 0x0000C34F |
| R3 | 0x0000125F |
| R4 | 0x00000000 |
| R5 | 0x00000000 |
| R6 | 0x00000000 |
| R7 | 0x200003D0 |
| R8 | 0x00000000 |
| R9 | 0x00000000 |
| R10 | 0x00000000 |
| R11 | 0x00000000 |
| R12 | 0x00000000 |
| R13 (SP) | 0x200003D0 |
| R14 (LR) | 0x000000E9 |
| R15 (PC) | 0x0000005E |
| xPSR | 0x81000000 |
| Banked | |
| System | |
| Internal | |
| Mode | Thread |
| Privilege | Privileged |
| Stack | MSP |
| States | 318091822 |
| Sec | 19.88073887 |
| FPU | |

```c
//Eng.Ahmed Hassan

#define SYSCTL_RCGC2_R    (*((volatile unsigned int*)0x400FE108)
#define GPIO_PORTF_DIR_R  (*((volatile unsigned int*)0x40025400)
#define GPIO_PORTF_DEN_R  (*((volatile unsigned int*)0x4002551C)
#define GPIO_PORTF_DATA_R   (*((volatile unsigned int*)0x400253F

int main()
{
  SYSCTL_RCGC2_R = 0x20;      //Enable GPIO Port

  //Delay to make sure that GPIOF is up & running
  unsigned int volatile delay_count = 0;
```

TExaS edX Lab 2

Port F Registers

| | | | | | |
|---|---|---|---|---|---|
| DATA: | 0x19 | PUR: | 0x00 | LOCK: | 0x01 |
| DIR: | 0x08 | PDR: | 0x00 | CR: | 0x1E |
| DEN: | 0x08 | RCGC2: | 0x00000020 | Clock enabled | |

Grading Controls

Number from edX:

Grade

Score: 0

Copy this to edX:

Command

```
Running with Code Size Limit: 32K
Load "..\\..\\..\\Embedded System Diploma\\Working_Directory\\Embedded_System_Online_Diploma\\Unit_03_Embedded_C\\Le
LA `PORTF
>
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE DEFINE DIR Display Enter
```

Call Stack + Locals

| Name | Location/Value | Type |
|---|---|---|
| main | 0x00000010 | int f() |
| delay_count | 0x0000125F | auto - uint |

Simulation    t1: 19.88073887 sec    L:1 C:19    CAP NUM SCRL OVR R/W