

Table of contents

- 1- Introduction
- 2- Specifications to toggle led based on arm-cortex-m4
- 3- Source code
 - 3.1 – main application code
 - 3.2 – startup code with c
 - 3.3 -linker script
 - 3.4- make file
- 4- map file
- 5- run application on uvision keil tool with feature from edx course (shape the world).
- 6– Debug with uvision keil tool with feature from edx course (shape the world).

1- Introduction:

Setting stack top by location counter in linker script file and extern it in startup with c done before but, in this report, I will make in startup.c stack top as a variable not a symbol with no need a linking

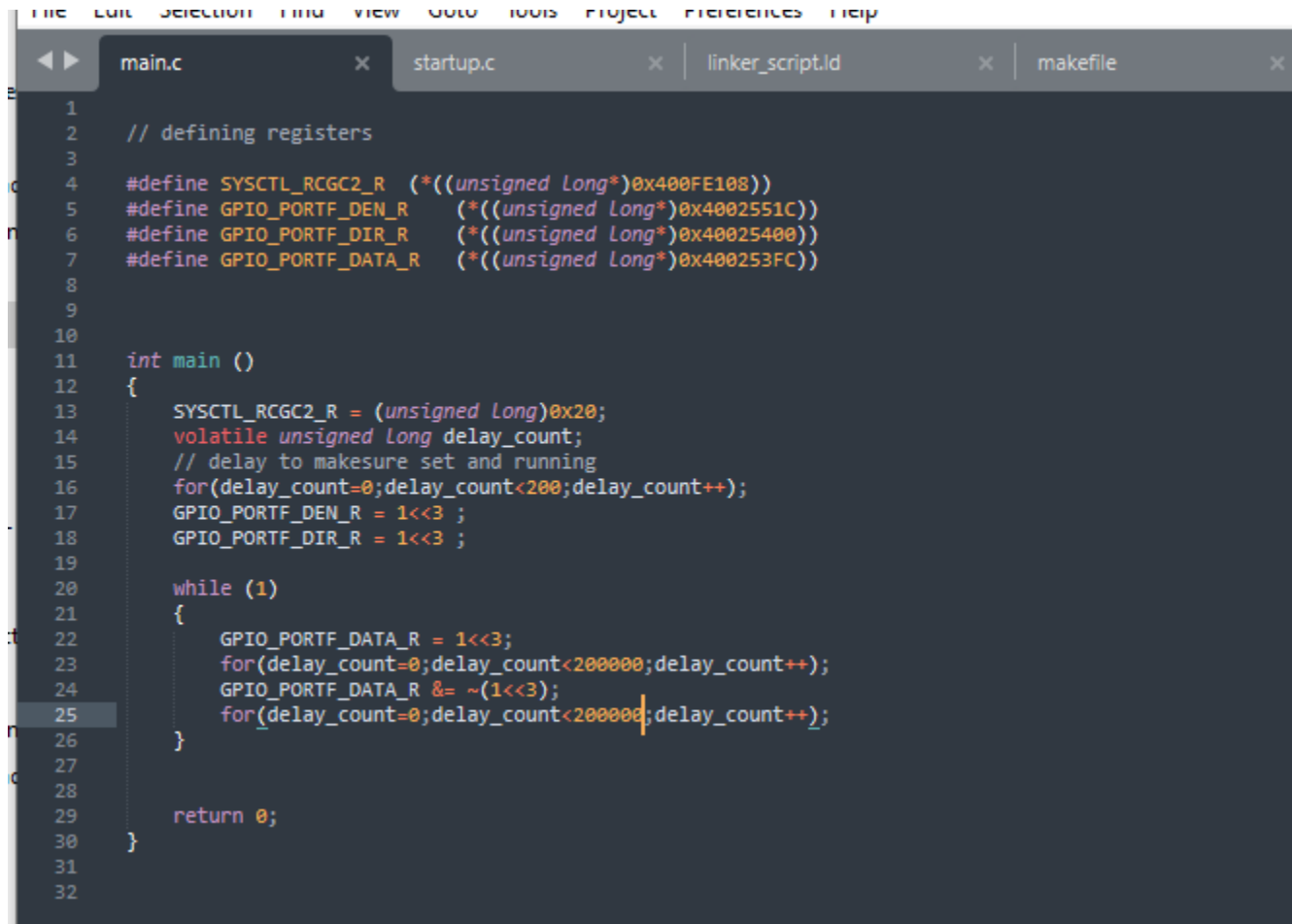
I will Execute this simple application on a virtual board using uvision tool with feature from edx course (shape the world).

2- Specifications to toggle led based on arm-cortex-m4 :

- led is connected to GPIO port F3
- to make a GPIO toggling in stm4c123,you need to work with two peripherals:
 - SYSTCL_RCGC2 (System control register)
 - GPIO F (general purpose i/o)
- to access SYSTCL_RCGC2 (0x400FE108)set to (0x20)
- to enable GPIOF (0x4002551c)set to 1 bit 3
- to enable Direction (0x40025400)set to 1 bit 3
- to write GPIOF (0x400253fc)

3- Source code

3.1 – main application code



```
1 // defining registers
2
3
4 #define SYSTCL_RCGC2_R (*((unsigned Long*)0x400FE108))
5 #define GPIO_PORTF_DEN_R (*((unsigned Long*)0x4002551C))
6 #define GPIO_PORTF_DIR_R (*((unsigned Long*)0x40025400))
7 #define GPIO_PORTF_DATA_R (*((unsigned Long*)0x400253FC))
8
9
10
11 int main ()
12 {
13     SYSTCL_RCGC2_R = (unsigned Long)0x20;
14     volatile unsigned Long delay_count;
15     // delay to makesure set and running
16     for(delay_count=0;delay_count<200;delay_count++);
17     GPIO_PORTF_DEN_R = 1<<3 ;
18     GPIO_PORTF_DIR_R = 1<<3 ;
19
20     while (1)
21     {
22         GPIO_PORTF_DATA_R = 1<<3;
23         for(delay_count=0;delay_count<200000;delay_count++);
24         GPIO_PORTF_DATA_R &= ~(1<<3);
25         for(delay_count=0;delay_count<200000;delay_count++);
26     }
27
28
29     return 0;
30 }
31
32
```

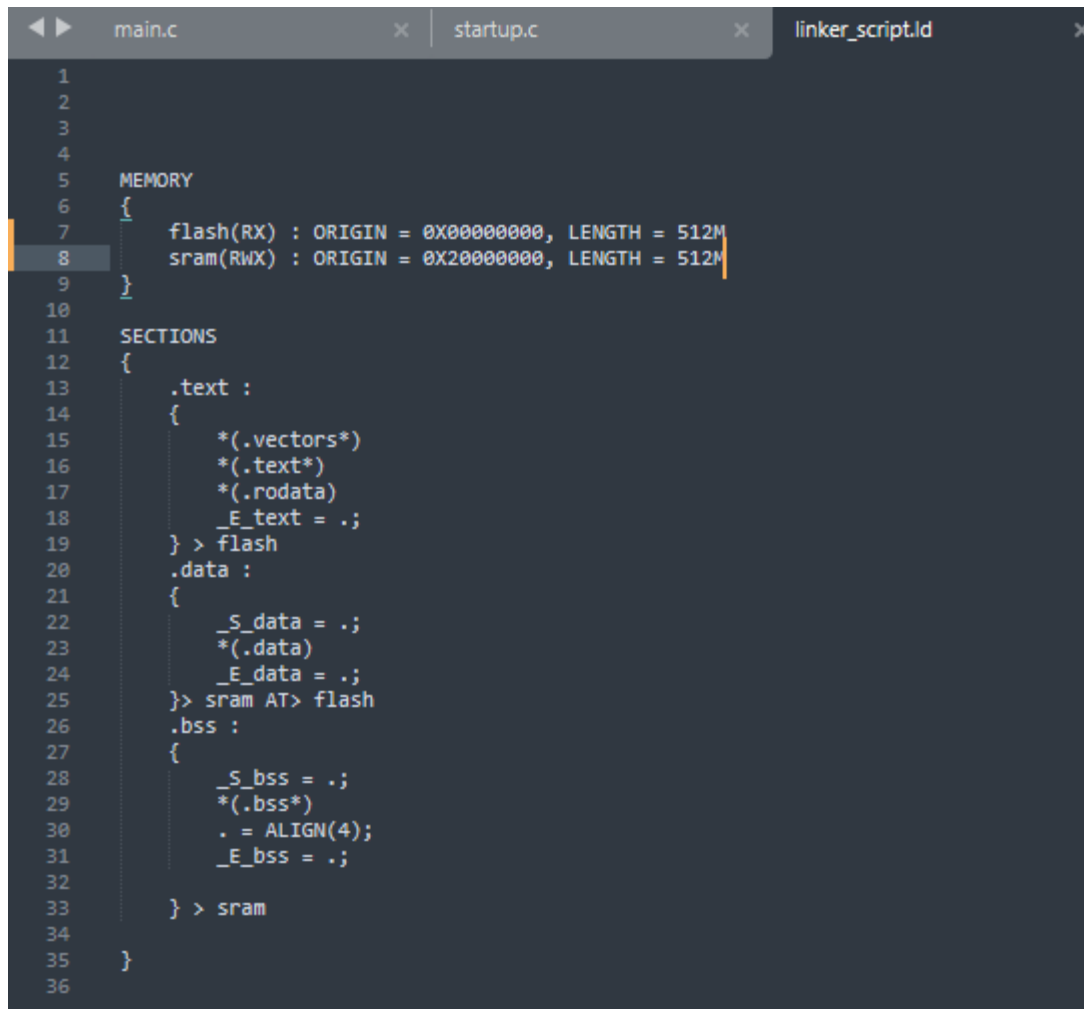
3.2- startup with c :

```
File Edit Selection Find View Goto Tools Project Preferences Help
main.c startup.c linker_script.ld makefile
1 //starup.c
2 //Eng.Ahmed MOhsen
3
4 #include<stdint.h>
5 extern int main();
6 void Reset_Handler();
7
8 void Default_Handler(){
9     Reset_Handler();
10 }
11 void NMI_Handler() __attribute__((weak,alias("Default_Handler")));
12 void H_Fault_Handler() __attribute__((weak,alias("Default_Handler")));
13
14 static unsigned Long stack_top[256]; //reserve stack size (256*4=1024B)
15
16 // making array of pointers to function
17 void (* g_p_fun_vectors[])() __attribute__((section(".vectors"))) = {
18     (void (*)())(stack_top+sizeof(stack_top)),
19     &NMI_Handler,
20     &H_Fault_Handler
21 };
22
23 extern unsigned int _S_data;
24 extern unsigned int _E_data;
25 extern unsigned int _S_bss;
26 extern unsigned int _E_bss;
27 extern unsigned int _E_text;
28
29 void Reset_Handler(){
30     //copy data from rom to ram
31     unsigned int data_size=(unsigned char*)&_E_data - (unsigned char*)&_S_data;
32     unsigned char* p_src =(unsigned char*)&_E_text;
33     unsigned char* p_dst =(unsigned char*)&_S_data;
34     int i;
35     for(i=0;i<data_size;i++){
36         *((unsigned char*)p_dst++)=*((unsigned char*)p_src++);
37     }
38     //init bss with zero
39     unsigned int bss_size=(unsigned char*)&_E_bss - (unsigned char*)&_S_bss;
40
41     p_dst =(unsigned char*)&_S_bss;
42     for(i=0;i<bss_size;i++){
43         *((unsigned char*)p_dst++)=(unsigned char) 0;
44     }
45     // jump to main
46     main();
47 }
48 }
```

I know startup should be written in assembly to set stack pointer and branch label to it then branch label to main but in cortex m4 stack is labeled when power is applied to MCU the (pc) value will be 0 which mapped to (0x00000000) and will start at the same address which point to stack .

In this startup.c , I defined an array of pointers to function which holds every handlers and entry (sp) according to (IVT),and set size of stack to be 1024 byte as I defined it to ununsilzed array to be in .bss ,and put it in vectors section, and I defined handlers to be weak and alias to override in user code and cause declaration to be emitted for another symbol, and I copy data from rom to ram and initialize bss in ram then jump to main.

3.3- linker script

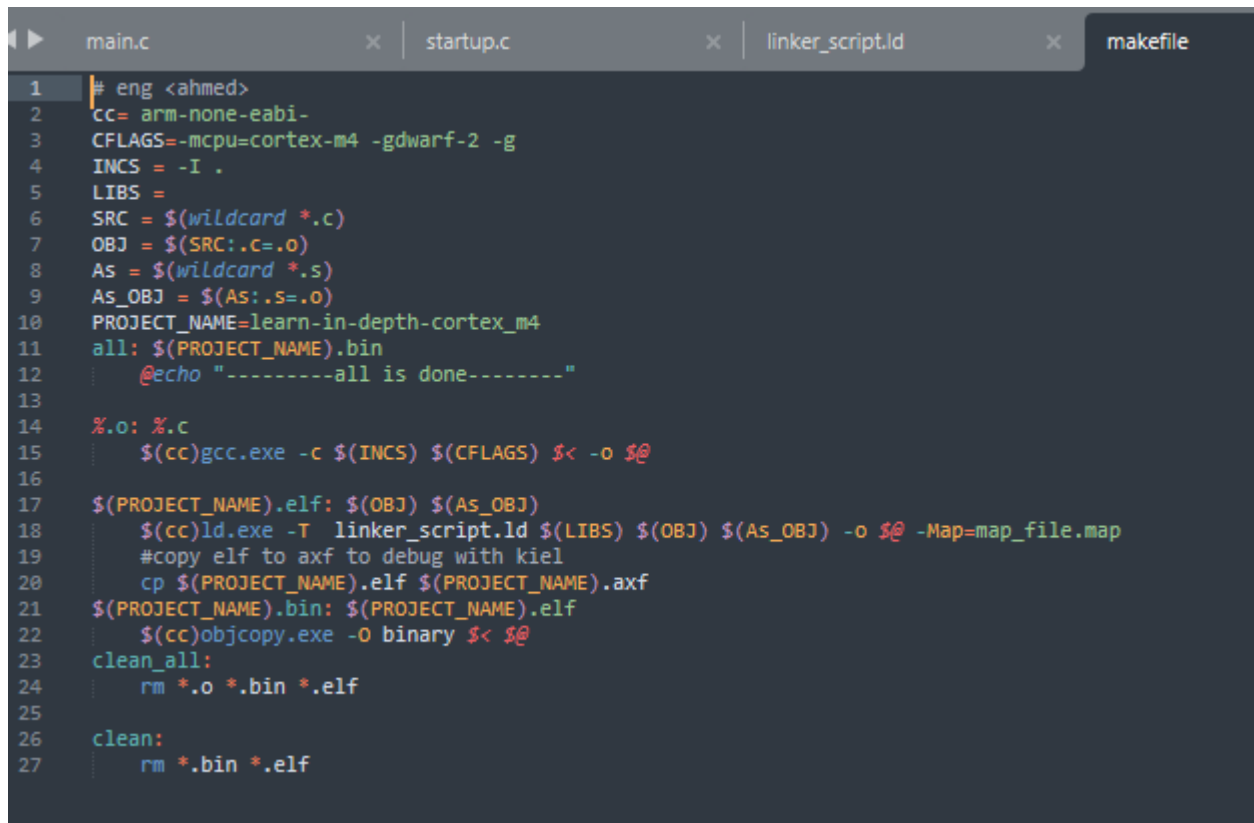


```
1
2
3
4
5 MEMORY
6 {
7     flash(RX) : ORIGIN = 0x00000000, LENGTH = 512M
8     sram(RWX) : ORIGIN = 0x20000000, LENGTH = 512M
9 }
10
11 SECTIONS
12 {
13     .text :
14     {
15         *(.vectors*)
16         *(.text*)
17         *(.rodata)
18         _E_text = .;
19     } > flash
20     .data :
21     {
22         _S_data = .;
23         *(.data)
24         _E_data = .;
25     } > sram AT> flash
26     .bss :
27     {
28         _S_bss = .;
29         *(.bss*)
30         . = ALIGN(4);
31         _E_bss = .;
32     } > sram
33 }
34
35
36
```

In this linker script file I defined two memory flash and sram with its addresses and lengths, then I made sections like (.text)

Which contain (. vectors,.text,rodata) and mapped it flash ,and then made (.data) section of all initialized data and (.bss) for all uninitialized data , and I made a locator counter to count addresses to end of .bss which I reference to top of stack in starting of .bss .

3.4 – make file :



```
1 # eng <ahmed>
2 cc= arm-none-eabi-
3 CFLAGS=-mcpu=cortex-m4 -gdwarf-2 -g
4 INCS = -I .
5 LIBS =
6 SRC = $(wildcard *.c)
7 OBJ = $(SRC:.c=.o)
8 As = $(wildcard *.s)
9 As_OBJ = $(As:.s=.o)
10 PROJECT_NAME=learn-in-depth-cortex_m4
11 all: $(PROJECT_NAME).bin
12     @echo "-----all is done-----"
13
14 %.o: %.c
15     $(cc)gcc.exe -c $(INCS) $(CFLAGS) $< -o $@
16
17 $(PROJECT_NAME).elf: $(OBJ) $(As_OBJ)
18     $(cc)ld.exe -T linker_script.ld $(LIBS) $(OBJ) $(As_OBJ) -o $@ -Map=map_file.map
19     #copy elf to axf to debug with kiel
20     cp $(PROJECT_NAME).elf $(PROJECT_NAME).axf
21 $(PROJECT_NAME).bin: $(PROJECT_NAME).elf
22     $(cc)objcopy.exe -O binary $< $@
23 clean_all:
24     rm *.o *.bin *.elf
25
26 clean:
27     rm *.bin *.elf
```

This make file is optimize compiling the program , so I used some make feature to do it like simplifaction dry and wildcards.

4-map file

```
1
2 Memory Configuration
3
4 Name          Origin          Length          Attributes
5 flash         0x00000000      0x20000000      xR
6 sram          0x20000000      0x20000000      xRW
7 *default*     0x00000000      0xffffffff
8
9 Linker script and memory map
10
11
12 .text         0x00000000      0x174
13 *(.vectors*) 0x00000000      0xc startup.o
14 .vectors      0x00000000      g_p_fun_vectors
15
16 *(.text*)
17 .text        0x0000000c      0xac main.o
18              0x0000000c      main
19 .text        0x000000b8      0xbc startup.o
20              0x000000b8      H_Fault_Handler
21              0x000000b8      Default_Handler
22              0x000000b8      NMI_Handler
23              0x000000c4      Reset_Handler
24
25 *(.rodata)    0x00000174      _E_text = .
26
27
28 .data         0x20000000      0 load address 0x00000174
29              0x20000000      _S_data = .
30
31 *(.data)
32 .data        0x20000000      0 main.o
33 .data        0x20000000      0 startup.o
34              0x20000000      _E_data = .
35
36 .igot.plt     0x20000000      0 load address 0x00000174
37 .igot.plt     0x00000000      0 main.o
38
39 .bss          0x20000000      0x400 load address 0x00000174
40              0x20000000      _S_bss = .
41
42 *(.bss*)
43 .bss         0x20000000      0 main.o
44 .bss         0x20000000      0x400 startup.o
45              0x20000400      . = ALIGN (0x4)
46              0x20000400      _E_bss = .
```

Handwritten annotations in blue:

- Arrows pointing from line 12 to line 13 and from line 14 to line 15.
- A bracket under lines 28-33.
- A bracket under lines 39-46.
- Handwritten text "1024B" next to line 40.

5- run application on uvision keil tool with feature from edx course (shape the world).

The screenshot shows the Keil uVision IDE with the Logic Analyzer window open. The Logic Analyzer window displays a waveform for the GPIOF signal, with a time scale of 73.75 us. The waveform shows a series of pulses, indicating the signal is active. The GPIOF hardware configuration window is also open, showing the TM4C123 microcontroller with pins PF3, PF2, PF1, PF0, and SW1, SW2 connected to LEDs. The GPIOF registers are also visible, showing the current state of the pins.

Logic Analyzer Setup:

- Min Time: 0 s
- Max Time: 1.476062 ms
- Grid: 1 ms
- Zoom: In
- Min/Max: Auto
- Update Screen: Stop
- Transition: Prev
- Jump to: Code
- Signal Info: Show Cycles

GPIOF Hardware Configuration:

- TM4C123
- PF3, PF2, PF1, PF0
- SW1, SW2
- LEDs
- 16 MHz

GPIOF Registers:

- DATA: 0x00000000
- PUR: 0x00000000
- LOCK: 0x00000000
- DIR: 0x00000000
- PDR: 0x00000000
- CR: 0x00000000
- DEN: 0x00000000
- RCGC2: 0x00000000
- Clock disabled

The screenshot shows the Keil uVision IDE with the Setup Logic Analyzer dialog box open. The dialog box allows configuring the Logic Analyzer signals, including the signal name, display type, color, and range. The GPIOF hardware configuration window is also open, showing the TM4C123 microcontroller with pins PF3, PF2, PF1, PF0, and SW1, SW2 connected to LEDs. The GPIOF registers are also visible, showing the current state of the pins.

Setup Logic Analyzer Dialog:

- Current Logic Analyzer Signals: PORTF
- Signal Display: Display Type: Bit, Color: Red, Hexadecimal Display: ☐
- Display Range: Max: 0xFFFFFFFF, Min: 0x0
- Display Formula (Signal & Mask) >> Shift: And Mask: 0x0000000F, Shift Right: 3
- Export / Import: Export Signal Definitions..., Import Signal Definitions..., Delete actual Signals: ☐
- Kill All, Close, Help

GPIOF Hardware Configuration:

- TM4C123
- PF3, PF2, PF1, PF0
- SW1, SW2
- LEDs
- 16 MHz

GPIOF Registers:

- DATA: 0x11
- PUR: 0x00
- LOCK: 0x01
- DIR: 0x08
- PDR: 0x00
- CR: 0x1E
- DEN: 0x08
- RCGC2: 0x00000020
- Clock enabled

6- Debug application on uvision keil tool with feature from edx course (shape the world).

Registers: R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, xPSR

Logic Analyzer: Setup..., Load..., Save..., Min Time: 27.61327 s, Max Time: 50.30181 s, Grid: 0.5 s, Zoom: In, Out, All, Min/Max: Auto, Undo, Update Screen: Stop, Clear, Transition: Prev, Next, Jump to: Code, Trace, Signal Info: Show Cycles

PORTF: 40.55727 s, 43.14727 s, 47.05727 s

```
17 GPIO_PORTF_DEN_R = 1<<3 ;
18 GPIO_PORTF_DIR_R = 1<<3 ;
19
20 while (1)
21 {
22     GPIO_PORTF_DATA_R = 1<<3;
23     for(delay_count=0;delay_count<200000;delay_count++)
24         GPIO_PORTF_DATA_R &= ~(1<<3);
25     for(delay_count=0;delay_count<200000;delay_count++)
26         GPIO_PORTF_DATA_R &= (1<<3);
27 }
```

GPIOF: Pro... Value, D... 0x00000011, D... 0x08080808, IS 0x00000000, IBE 0x00000000, IEV 0x00000000, IM 0, RIS 0, M... 0, L... 0, A... 0x00000000, D... 0xFFFFFFFF, D... 0x00000000, D... 0x00000000, O... 0x00000000

Command: Load "...\\learn-in-depth-cortex_m4.axf", LA ^PORTF, ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet, Start code execution

Port F Hardware: TM4C123, SW1, SW2, PF3, PF2, PF1, PF4, PF0, 16 MHz, LED, LED, LED, LED

Port F Registers: DATA: 0x11, PUR: 0x00, LOCK: 0x01, DIR: 0x08, PDR: 0x00, CR: 0x1E, DEN: 0x08, RCGC2: 0x00000020, Clock enabled

Registers: R0, R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, xPSR

Logic Analyzer: Setup..., Load..., Save..., Min Time: 27.61327 s, Max Time: 60.83741 s, Grid: 0.5 s, Zoom: In, Out, All, Min/Max: Auto, Undo, Update Screen: Stop, Clear, Transition: Prev, Next, Jump to: Code, Trace, Signal Info: Show Cycles

PORTF: 40.55727 s, 43.14727 s, 47.05727 s

```
17 GPIO_PORTF_DEN_R = 1<<3 ;
18 GPIO_PORTF_DIR_R = 1<<3 ;
19
20 while (1)
21 {
22     GPIO_PORTF_DATA_R = 1<<3;
23     for(delay_count=0;delay_count<200000;delay_count++)
24         GPIO_PORTF_DATA_R &= ~(1<<3);
25     for(delay_count=0;delay_count<200000;delay_count++)
26         GPIO_PORTF_DATA_R &= (1<<3);
27 }
```

GPIOF: Pro... Value, D... 0x08080819, D... 0x08080808, IS 0x00000000, IBE 0x00000000, IEV 0x00000000, IM 0, RIS 0, M... 0, L... 0, A... 0x00000000, D... 0xFFFFFFFF, D... 0x00000000, D... 0x00000000, O... 0x00000000

Command: Load "...\\learn-in-depth-cortex_m4.axf", LA ^PORTF, ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet, Reset the CPU

Port F Hardware: TM4C123, SW1, SW2, PF3, PF2, PF1, PF4, PF0, 16 MHz, LED, LED, LED, LED

Port F Registers: DATA: 0x19, PUR: 0x00, LOCK: 0x01, DIR: 0x08, PDR: 0x00, CR: 0x1E, DEN: 0x08, RCGC2: 0x00000020, Clock enabled