Table of contents

1- Introduction :

Writing a bare-metal application from scratch is not hard that you think, in this report I will build a simble bare-metal application in cortex-m3 micro-controller chip which is based on stm32f103c6 chip to toogle led using gpio ports, I will bulild everything from scratch including startup,linker script,make file and source code, and compile them using arm-non-eabi cross tool chain for arm processors.

I will Execute this simple application on a virtual board using protus tool .

2- Specifications to toggle led based on arm-cortex-m3 :

-led is connected to gpio port A13

- to make a gpio toggling in stm32,you need to work with two peripherals:

- RCC (reset and clock control)

-GPIO A (general purpose i/o)

- the Rcc is must due to the gpio has disabled clock by default.

- base address RCC (0x40021000)

- GPIO port A (0x40010800)

- APB2ENR (0x18) which is responsible for enable port A on its pin 2.

- CRH Register (0x04) to start signal to write on portA

- ODR Register (0x0c) to write 1 or 0 on bit 13.

3- Source code

3.1 – main application code

app.c | linker_script.ld | startup.c

```c
typedef volatile unsigned int vuint32_t;

#include<stdint.h>

#define RCC_BASE     0x40021000
#define GPIOA_BASE   0x40010800

#define RCC_APB2ENR     *(volatile uint32_t *) (RCC_BASE + 0x18)
#define GPIOA_CRH       *(volatile uint32_t *) (GPIOA_BASE + 0x04)
#define GPIOA_ODR       *(volatile uint32_t *) (GPIOA_BASE + 0x0c)
////////
#define RCC_IOPAEN  (1<<2)

typedef union {
    vuint32_t       all_field;
    struct{
        vuint32_t   reserved:13;
        vuint32_t   p_13:1;
    }Spin;
} U_R_ODR_t ;

volatile U_R_ODR_t*  R_ODR=  (volatile U_R_ODR_t*)(GPIOA_BASE+0x0c);

int main(void)
{
    int i;
    RCC_APB2ENR |= RCC_IOPAEN;
    GPIOA_CRH   &= 0xFF0FFFFF;
    GPIOA_CRH   |= 0x00200000;
    while(1){
        R_ODR->Spin.p_13=1;
        for(i=0;i<5000;i++);
        R_ODR->Spin.p_13=0;
        for(i=0;i<5000;i++);
    }
    return 0;
}
```

In the main application code , which I called “app.c” I defined RCC_base and GPIOPA addresses , and defined some registers to reference to base addresses and offset to drive to its destination , and make an union variable to control bits of port A , in the main I set RCC_register with bit 2 to 1,make CRH Register from bit 20-24 set to 2 in binary, and make pin13 to be on set it by 1 and make a delay then make pin 13 to be on clear by 0 and then delay again this is in continuous switching.

3.2- startup with c :

app.c | linker_script.ld | startup.c

```c
extern int main();
extern int _stack_top;
void Reset_Handeler();

void Default_Handeler(){
    Reset_Handeler();
}
void NMI_Handeler() __attribute__((weak,alias("Default_Handeler")));;
void H_Fault_Handeler() __attribute__((weak,alias("Default_Handeler")));;
void MM_Fault_Handeler()  __attribute__((weak,alias("Default_Handeler")));;
void Bus_Fault()  __attribute__((weak,alias("Default_Handeler")));;
void Usage_Fault_Handeler() __attribute__((weak,alias("Default_Handeler")));;

uint32_t vectors[] __attribute__((section(".vectors"))) = {
    (uint32_t)&_stack_top,
    (uint32_t)&NMI_Handeler,
    (uint32_t)&H_Fault_Handeler,
    (uint32_t)&MM_Fault_Handeler,
    (uint32_t)&Bus_Fault,
    (uint32_t)&Usage_Fault_Handeler
};

extern unsigned int _S_data;
extern unsigned int _E_data;
extern unsigned int _S_bss;
extern unsigned int _E_bss;
extern unsigned int _E_text;

void Reset_Handeler(){
    //copy data from rom to ram
    unsigned int data_size=(unsigned char*)&_E_data - (unsigned char*)&_S_data;
    unsigned char* p_src =(unsigned char*)&_E_text;
    unsigned char* p_dst =(unsigned char*)&_S_data;
    int i;
    for(i=0;i<data_size;i++){
        *((unsigned char*)p_dst++)=*((unsigned char*)p_src++);
    }
    //init bss with zero
    unsigned int bss_size=(unsigned char*)&_E_bss - (unsigned char*)&_S_bss;

    p_dst =(unsigned char*)&_S_bss;

    for(i=0;i<bss_size;i++){
        *((unsigned char*)p_dst++)=(unsigned char) 0;
    }
    // jump to main
    main();
```

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 m3 stack is labeled when power is applied to MCU the (pc) value will be 0 which mapped to (0x08000000) and will start at the same address which point to stack .

In this startup.c , I defined an array which holds every handlers and entry (sP) according to (IVT),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

app.c | linker_script.ld | startup.c

```
1
2
3
4
5   MEMORY
6   {
7       flash(RX) : ORIGIN = 0X08000000, LENGTH = 128k
8       sram(RWX) : ORIGIN = 0X20000000, LENGTH = 20k
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
33          . = ALIGN(4);
34          . = . + 0x1000 ;
35          _stack_top = .;
36
37      } > sram
38
39  }
40
```

:

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 jump stack and take some memory .

3.4 – make file :

app.c | linker_script.ld | makefile | startup.c

```
# eng <ahmed>
cc= arm-none-eabi-
CFLAGS=-mcpu=cortex-m3 -gdwarf-2
INCS = -I .
LIBS =
SRC = $(wildcard *.c)
OBJ = $(SRC:.c=.o)
As = $(wildcard *.s)
As_OBJ = $(As:.s=.o)
PROJECT_NAME=learn-in-depth-cortexm3
all: $(PROJECT_NAME).bin
    @echo "---------all is done--------"

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

$(PROJECT_NAME).elf: $(OBJ) $(As_OBJ)
    $(cc)ld.exe -T  linker_script.ld $(LIBS) $(OBJ) $(As_OBJ) -o $@ -Map=map_file.map
$(PROJECT_NAME).bin: $(PROJECT_NAME).elf
    $(cc)objcopy.exe -O binary $< $@
clean_all:
    rm *.o *.bin *.elf

clean:
    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- symbols:

Using nm binary utility, I can hack every binary file and see its symbols and which section every symbol belongs to and address of every symbol. In object files there is only virtual addresses, and every symbol will take a real load address after linking process in the elf image.

4.1 app.o symbols

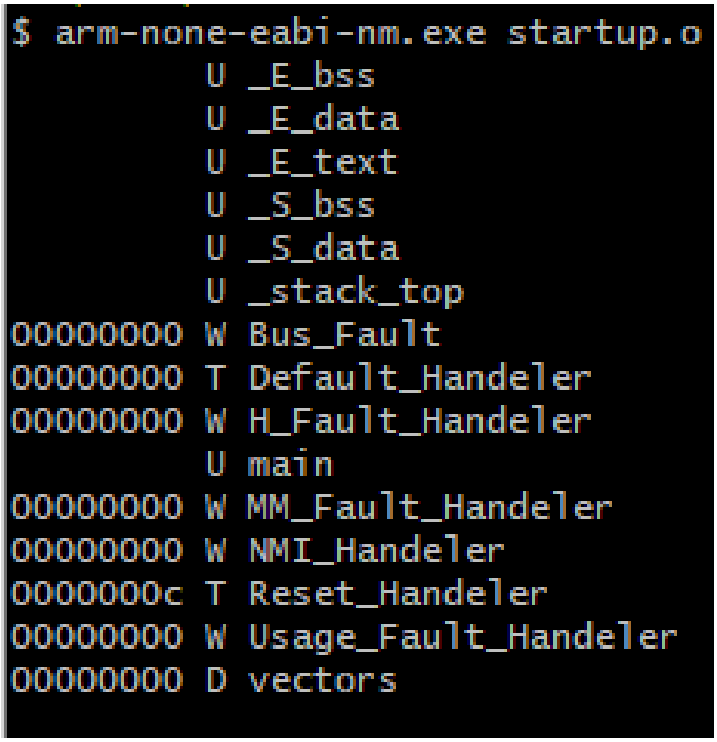
```
$ arm-none-eabi-nm.exe app.o
00000000 T main
00000000 D R_ODR
```

this object file contains two symbols,

1. Main: which is in text section.
2. R_ODR: which is in data section.

4.2 elf image symbols

4.3 startup.o symbols



5- Sections Headers:

In this section we just care about .text, .data, .bss and .rodata sections, linker may add some other sections like .ARM.attributes and .comment but we don't care about these sections as it will be excluded in the final executable file and wont be loaded the micro-controller.

5.1 app.o sections header

```
$ arm-none-eabi-objdump.exe -h app.o

app.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000000a8  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000004  00000000  00000000  000000dc  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000e0  2**0
                  ALLOC
  3 .debug_info   00000124  00000000  00000000  000000e0  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  4 .debug_abbrev 000000bf  00000000  00000000  00000204  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    00000038  00000000  00000000  000002c3  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000020  00000000  00000000  000002fb  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  7 .debug_line   0000009d  00000000  00000000  0000031b  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_str    00000123  00000000  00000000  000003b8  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000012  00000000  00000000  000004db  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  000004ed  2**0
                  CONTENTS, READONLY
 11 .debug_frame  0000002c  00000000  00000000  00000520  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

5.3 startup.o sections

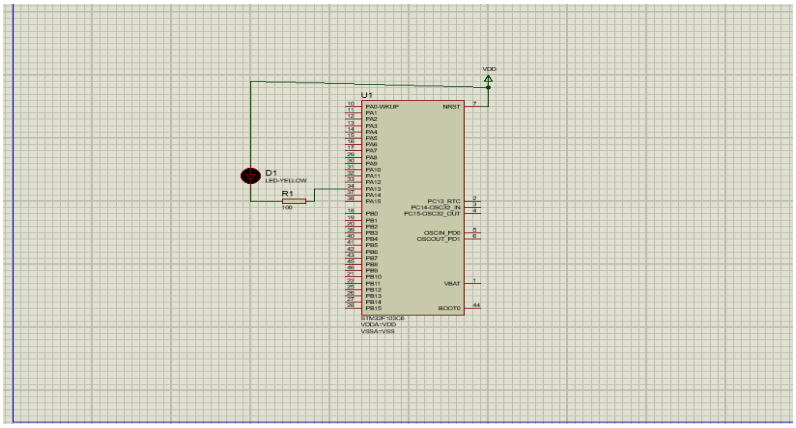
```
$ arm-none-eabi-objdump.exe -h startup.o

startup.o:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         000000bc  00000000  00000000  00000034  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data         00000000  00000000  00000000  000000f0  2**0
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00000000  00000000  00000000  000000f0  2**0
                  ALLOC
  3 .vectors      00000018  00000000  00000000  000000f0  2**2
                  CONTENTS, ALLOC, LOAD, RELOC, DATA
  4 .debug_info   00000167  00000000  00000000  00000108  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  5 .debug_abbrev 000000c0  00000000  00000000  0000026f  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_loc    00000064  00000000  00000000  0000032f  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_aranges 00000020  00000000  00000000  00000393  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  8 .debug_line   000000af  00000000  00000000  000003b3  2**0
                  CONTENTS, RELOC, READONLY, DEBUGGING
  9 .debug_str    0000016c  00000000  00000000  00000462  2**0
                  CONTENTS, READONLY, DEBUGGING
 10 .comment      00000012  00000000  00000000  000005ce  2**0
                  CONTENTS, READONLY
 11 .ARM.attributes 00000033  00000000  00000000  000005e0  2**0
                  CONTENTS, READONLY
 12 .debug_frame  0000004c  00000000  00000000  00000614  2**2
                  CONTENTS, RELOC, READONLY, DEBUGGING
```

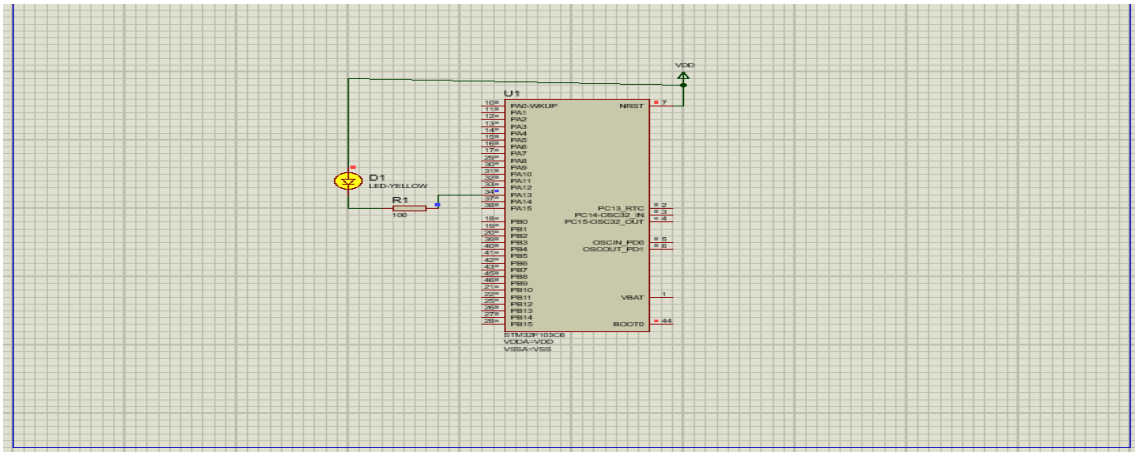
5.4 final elf image sections

```
$ arm-none-eabi-objdump.exe -h learn-indepth-cortex-m3.elf

learn-indepth-cortex-m3.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000017c  08000000  08000000  00008000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  20000000  0800017c  00010000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          00001000  20000004  08000180  00010004  2**0
                  ALLOC
  3 .debug_info   0000028b  00000000  00000000  00010004  2**0
                  CONTENTS, READONLY, DEBUGGING
  4 .debug_abbrev 0000017f  00000000  00000000  0001028f  2**0
                  CONTENTS, READONLY, DEBUGGING
  5 .debug_loc    0000009c  00000000  00000000  0001040e  2**0
                  CONTENTS, READONLY, DEBUGGING
  6 .debug_aranges 00000040  00000000  00000000  000104aa  2**0
                  CONTENTS, READONLY, DEBUGGING
  7 .debug_line   0000014c  00000000  00000000  000104ea  2**0
                  CONTENTS, READONLY, DEBUGGING
  8 .debug_str    00000171  00000000  00000000  00010636  2**0
                  CONTENTS, READONLY, DEBUGGING
  9 .comment      00000011  00000000  00000000  000107a7  2**0
                  CONTENTS, READONLY
 10 .ARM.attributes 00000033  00000000  00000000  000107b8  2**0
                  CONTENTS, READONLY
 11 .debug_frame  00000078  00000000  00000000  000107ec  2**2
                  CONTENTS, READONLY, DEBUGGING
```

All sections now have the real load memory addresses which is given to every section in linker script.

6- run application on protus





7 – Debug with protus

lab2 - Proteus 8 Professional - Source Code

File Project Build Edit Debug System Help

Schematic Capture | Source Code

CM3 Source Code - U1

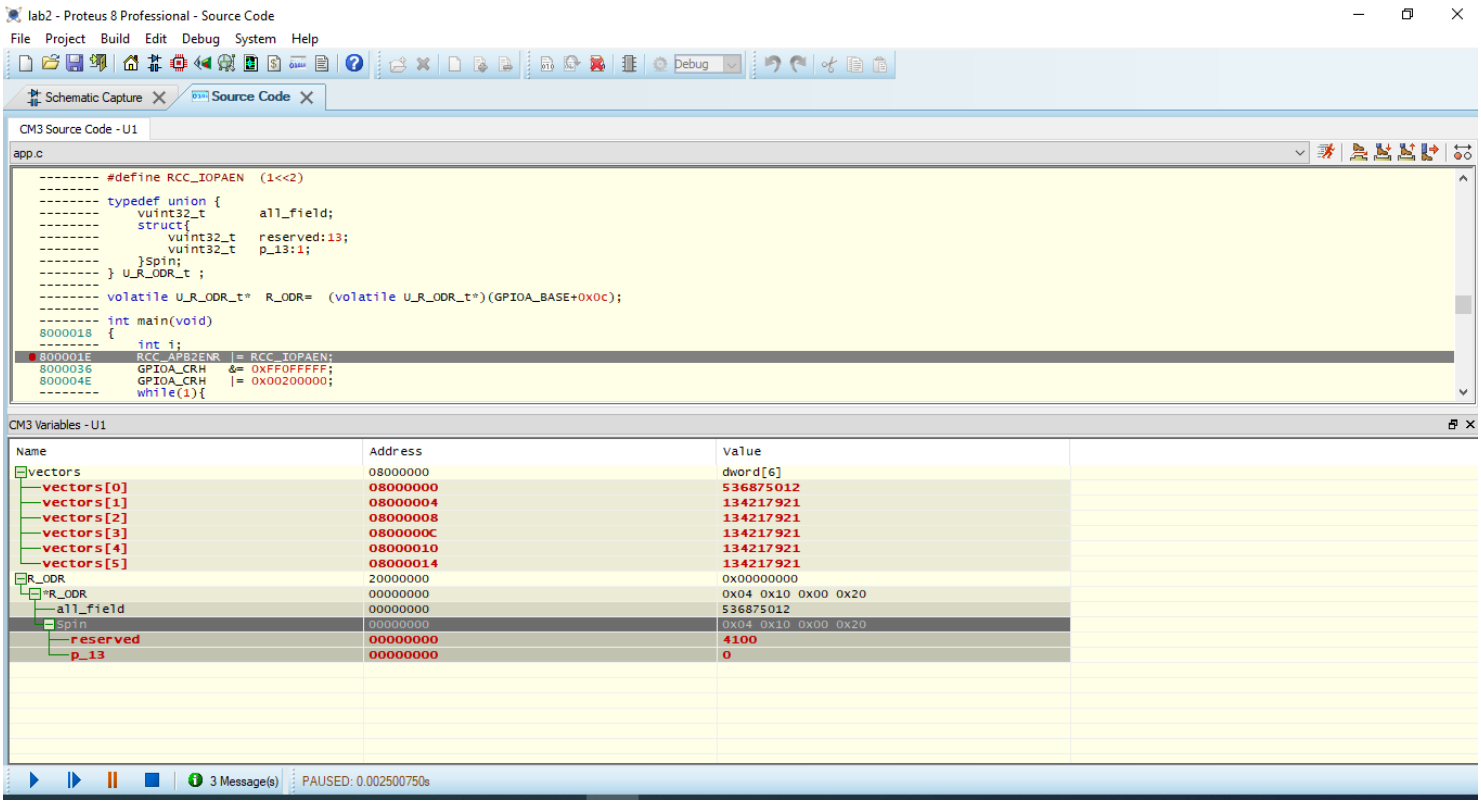
startup.c

```
--------
-------- #include<stdint.h>
-------- extern int main();
-------- extern int _stack_top;
-------- void Reset_Handeler();
--------
80000C0  void Default_Handeler(){
80000C4      Reset_Handeler();
80000C8  }
-------- void NMI_Handeler() __attribute__((weak,alias("Default_Handeler")));;
-------- void H_Fault_Handeler() __attribute__((weak,alias("Default_Handeler")));;
-------- void MM_Fault_Handeler() __attribute__((weak,alias("Default_Handeler")));;
-------- void Bus_Fault() __attribute__((weak,alias("Default_Handeler")));;
-------- void Usage_Fault_Handeler() __attribute__((weak,alias("Default_Handeler")));;
--------
-------- uint32_t vectors[] __attribute__((section(".vectors"))) = {
--------     (uint32_t)&_stack_top,
--------     (uint32_t)&NMI_Handeler,
--------     (uint32_t)&H_Fault_Handeler,
```

CM3 Variables - U1

Name	Address	Value
vectors	08000000	dword[6]
vectors[0]	08000000	536875012
vectors[1]	08000004	134217921
vectors[2]	08000008	134217921
vectors[3]	0800000C	134217921
vectors[4]	08000010	134217921
vectors[5]	08000014	134217921
R_ODR	20000000	0x00000000
*R_ODR	00000000	0x04 0x10 0x00 0x20
all_field	00000000	536875012
Spin	00000000	0x04 0x10 0x00 0x20
reserved	00000000	4100
p_13	00000000	0

3 Message(s) PAUSED: 0.002500750s


lab2 - Proteus 8 Professional - Source Code
File Project Build Edit Debug System Help
Debug
Schematic Capture
Source Code
CM3 Source Code - U1
app.c
-------- #define RCC_IOPAEN (1<<2)
-------- typedef union {
-------- vuint32_t all_field;
-------- struct{
-------- vuint32_t reserved:13;
-------- vuint32_t p_13:1;
-------- }Spin;
-------- } U_R_ODR_t ;
-------- volatile U_R_ODR_t* R_ODR= (volatile U_R_ODR_t*)(GPIOA_BASE+0x0c);
-------- int main(void)
8000018 {
-------- int i;
800001E RCC_APB2ENR |= RCC_IOPAEN;
8000036 GPIOA_CRH &= 0xFF0FFFFF;
800004E GPIOA_CRH |= 0x00200000;
-------- while(1){
CM3 Variables - U1
Name
Address
Value
vectors
08000000
dword[6]
vectors[0]
08000000
536875012
vectors[1]
08000004
134217921
vectors[2]
08000008
134217921
vectors[3]
0800000C
134217921
vectors[4]
08000010
134217921
vectors[5]
08000014
134217921
R_ODR
20000000
0x00000000
*R_ODR
00000000
0x04 0x10 0x00 0x20
all_field
00000000
536875012
Spin
00000000
0x04 0x10 0x00 0x20
reserved
00000000
4100
p_13
00000000
0
3 Message(s)
PAUSED: 0.002500750s