TOGGLING LED ON TIVA C TM4C123 KIT WITH STARTUP.C

AUTHOR: Hassan Attia

Brief:

- I made a bare metal software that runs on TIVA C microcontroller.
- This SW toggles an embedded LED connected to PORT_F Pin number 3
- TIVA C has ARM cortex M4 processor and it supports writing startup file with C code.
- I'll put some screenshots illustrating the whole processes which include(compiling the files, map file, keil micro-vision simulation and debugging, and analyzing PORT_F pin_3 using logic analyzer).

1-Main.c

```c
//Hassan_Attia
 #include<stdint.h>

#define SYSCTL_RCGC2_R    (*((volatile unsigned long*)0x400FE108))// GPIO PORT F ENABLE
#define GPIO_PORTF_DATA_R (*((volatile unsigned long*)0x400253FC))// WRINTG DATA ON PORT F PIN 3
#define GPIO_PORTF_DIR_R  (*((volatile unsigned long*)0x40025400)) // PORT F PIN 3 DIRECTION
#define GPIO_PORTF_DEN_R  (*((volatile unsigned long*)0x4002551C)) // PORT F PIN 3 ENABLE




int main(){
 volatile unsigned long delay_count;
 SYSCTL_RCGC2_R = 0x00000020;
 //delay to ensure GPIO is turned on and running
for(delay_count = 0; delay_count<200; delay_count++);
GPIO_PORTF_DIR_R |= 1<<3; // set pin 3 to be output
GPIO_PORTF_DEN_R |= 1<<3;// enabling pin 3

while(1){
    //LED turn on
    GPIO_PORTF_DATA_R |= 1<<3;

    for(delay_count = 0; delay_count<200000; delay_count++);

    //LED turn off
    GPIO_PORTF_DATA_R &= ~(1<<3);

    for(delay_count = 0; delay_count<200000; delay_count++);

    }


return 0;
}
```

2-Startup.C:

```c
/* Hassan Attia*/

#include <stdint.h>


void  reset_handler  ();
void Default_Handler ();
void  NMI            () __attribute__((weak, alias ("Default_Handler")));;
void  Hard_Fault     () __attribute__((weak, alias("Default_Handler")));;
void  MemManage      () __attribute__((weak, alias("Default_Handler")));;
void  BusFault       () __attribute__((weak, alias("Default_Handler")));;
void  Usage_Fault    () __attribute__((weak, alias("Default_Handler")));;
void  SV_Call        () __attribute__((weak, alias("Default_Handler")));;



//using extern class for functions and symbols to make linker script links without errors
extern int main(void);
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;

Stack_top defined in startup.c instead of linker_script.ld

unsigned long stack_top [256] = {0}; //creating .bss section to initilize the stack


    void (* g_vectors_arr_of_ptrs_to_fn[])() __attribute__((section(".vectors"))) = {

    (void (*)())((unsigned long)stack_top + sizeof(stack_top)),

    &reset_handler,
    &NMI,
    &Hard_Fault,
    &MemManage,
    &BusFault,
    &Usage_Fault,
    &SV_Call


    };



void  reset_handler  (){
    int j;
    //copying from flash 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;

    for(j = 0 ; j<data_size; j++){

        *((unsigned char*)p_dst++) = *((unsigned char*)p_src++);

    }

    //Initilize .bss with zeros in ram s
    unsigned int bss_size  = (unsigned char*)&_E_bss - (unsigned char*)&_S_bss;

    p_dst = (unsigned char*)&_S_bss;

        for(j = 0 ; j<bss_size; j++){

        *((unsigned char*)p_dst++) = (unsigned char)0;

    }
    //Jumping to the main()


    main();

}



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

3-Linker Script:

```
/*Hassan Attia*/



MEMORY{

    flash(rx): ORIGIN = 0x00000000, LENGTH = 512M
    sram(rwx): ORIGIN = 0x20000000, LENGTH = 512M


}



SECTIONS{

.text :{
    *(.vectors*)
    *(.text*)
    *(.rodata*)
    _E_text = .;
}> flash

.data :{
    _S_data = . ;
    *(.data*)
    _E_data = . ;
}> sram AT> flash

.bss :{
    _S_bss = . ;
    *(.bss*)
    _E_bss = . ;

}> sram



}
```

Stack_top symbol removed and defined in startup.c

4-MakeFile:

```
#Author: Hassan Attia

CC=arm-none-eabi-
CFLAGS= -mcpu=cortex-m4 -mthumb -gdwarf-2 -g
INCS=
LIBS=
SRC = $(wildcard *.c)
OBJ = $(SRC:.c=.o)
As = $(wildcard *.s)
AsOBJ = $(As:.s=.o)
Project_Name=learn-in-depth_cortex_M4

all: $(Project_Name).bin
    @echo "*******Build is Done********"


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

$(Project_Name).bin: $(Project_Name).elf

    $(CC)objcopy.exe -O binary $< $@

$(Project_Name).elf: $(OBJ) $(AsOBJ)
    $(CC)ld.exe -T linker-script.ld $(LIBS) -Map=Map_file.txt $(OBJ) $(AsOBJ) -o $@

    cp $(Project_Name).elf $(Project_Name).axf




clean:
    rm *.o

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

5-Compiling:

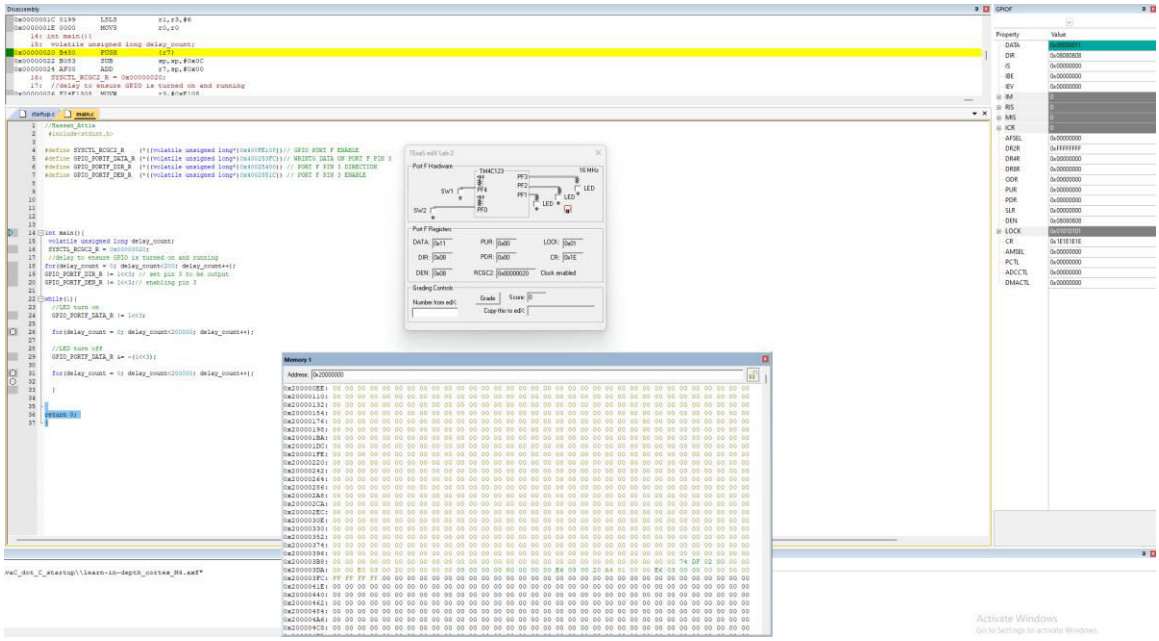
```
$ make
arm-none-eabi-gcc.exe  -c -mcpu=cortex-m4 -mthumb -gdwarf-2 -g  main.c -o main.o
arm-none-eabi-gcc.exe  -c -mcpu=cortex-m4 -mthumb -gdwarf-2 -g  startup.c -o sta
rtup.o
arm-none-eabi-ld.exe -T linker-script.ld  -Map=Map_file.txt main.o startup.o  -o
 learn-in-depth_cortex_M4.elf
cp learn-in-depth_cortex_M4.elf learn-in-depth_cortex_M4.axf
arm-none-eabi-objcopy.exe -O binary learn-in-depth_cortex_M4.elf learn-in-depth_
cortex_M4.bin
```

6-Map FILE:

```
Memory Configuration

Name             Origin             Length             Attributes
flash            0x00000000         0x20000000         xr
sram             0x20000000         0x20000000         xrw
*default*        0x00000000         0xffffffff

Linker script and memory map


.text           0x00000000      0x1a4
 *(.vectors*)
 .vectors       0x00000000       0x20 startup.o
                0x00000000                g_vectors_arr_of_ptrs_to_fn
 *(.text*)
 .text          0x00000020       0xc8 main.o
                0x00000020                main
 .text          0x000000e8       0xbc startup.o
                0x000000e8                reset_handler
                0x00000198                BusFault
                0x00000198                Hard_Fault
                0x00000198                Default_Handler
                0x00000198                Usage_Fault
                0x00000198                SV_Call
                0x00000198                MemManage
                0x00000198                NMI
 *(.rodata*)
                0x000001a4                _E_text = .

.glue_7         0x000001a4        0x0
 .glue_7        0x00000000        0x0 linker stubs

.glue_7t        0x000001a4        0x0
 .glue_7t       0x00000000        0x0 linker stubs

.vfp11_veneer   0x000001a4        0x0
 .vfp11_veneer  0x00000000        0x0 linker stubs

.v4_bx          0x000001a4        0x0
 .v4_bx         0x00000000        0x0 linker stubs

.iplt           0x000001a4        0x0
 .iplt          0x00000000        0x0 main.o

.rel.dyn        0x000001a4        0x0
 .rel.iplt      0x00000000        0x0 main.o

.data           0x20000000        0x0 load address 0x000001a4
                0x20000000                _S_data = .
 *(.data*)
 .data          0x20000000        0x0 main.o
 .data          0x20000000        0x0 startup.o
                0x20000000                _E_data = .

.igot.plt       0x20000000        0x0 load address 0x000001a4
 .igot.plt      0x00000000        0x0 main.o

.bss            0x20000000      0x400 load address
                0x20000000                _S_bss =
 *(.bss*)
 .bss           0x20000000        0x0 main.o
 .bss           0x20000000      0x400 startup.o
                0x20000000                stack_top
                0x20000400                _E_bss = .
```

Size of stack_top reserved In .bss section as shown and it's 1024 bytes

7-Debugging Using Keil micro vision:



8-Anlyzing PORT_F PIN_3 using logic analyzer: