

# BAREMETAL APP. FROM SCRATCH WITHOUT IDE BY A STARTUP.C

Author: Eng. Mustafa Hafez | Date: July-7-2024

## OVERVIEW

### 1. Description

- ✓ The arm-cortex-m Family has a wonderful feature, The address at the entry point of the processor will be used to initiate the stack pointer (sp). so, we can write the startup file by C-language as the stack will be initiated.

### 2. Requirements:

- ✓ main.c – Startup.c – Linkerscript.ld - Makefile ---→ arm-none-eabi cross toolchain.

### 3. Procedure

Imagine that we have the ability to make an array of type `uint_32` as each elements will take a 4 byte , as we know the array elements are arranged sequentially in the memory, so if we make the first element of our array equal to the address we want to give to the SP, and by someway put this array at the entry point of processor, so we can make the startup file and the next element in array will be next to the SP address.

```
#include <stdint.h>

extern void main(void);
void _Reset_Handler(void);

void Default_Handler(void) {
    _Reset_Handler();
}

void _NMI_Handler(void) __attribute__((weak,alias("Default_Handler")));
void _Hard_Fault_Handler(void) __attribute__((weak,alias("Default_Handler")));
void _MM_Fault_Handler(void) __attribute__((weak,alias("Default_Handler")));
void _Bus_Fault(void) __attribute__((weak,alias("Default_Handler")));
void _Usage_Fault_Handler(void) __attribute__((weak,alias("Default_Handler")));

//__attribute__((section(".vectors"))) put the array at vector section
// .vectors we will be at the beginning of .text section which will be at
extern uint32_t _stack_top;

//Array of pointer to functions that take no arguments and return void.
void (* const G_PFun_Vectors[])(void) __attribute__((section(".vectors"))) = {
    (void (*)()) &_stack_top,
    _Reset_Handler,
    _NMI_Handler,
    _Hard_Fault_Handler,
    _MM_Fault_Handler,
    _Bus_Fault,
    _Usage_Fault_Handler
};
```

The command `__attribute__((section(".vectors")))` is used to put some lines at a desired specified section. so we put our arrays at a vector section

Here we used an array of pointer to functions to hold the handlers addresses

The other Handler is arranged according to the vendor vector table.

```

1  /*
2  Startup.c
3  Author: Eng.Mustafa Hafez Eldeeb
4  Date: 30/7/2023
5  Family: Cortex-m
6  */
7
8  /* This Family has a wonderful features, The addresse at the entry point of the processor will be used to initiate the stack pointer (sp).
9
10 /* Imagine with me that we have the ability to make an array of type uint_32 as each element will take a 4 byte , as we know the array elements
11
12
13
14
15
16
17 #include <stdint.h>
18
19 extern void main(void);
20 void _Reset_Handler(void);
21
22
23 void Default_Handler(void) {
24     _Reset_Handler();
25 }
26
27 void _NMI_Handler(void) __attribute__((weak,alias("Default_Handler")));
28 void _Hard_Fault_Handler(void) __attribute__((weak,alias("Default_Handler")));
29 void _MM_Fault_Handler(void) __attribute__((weak,alias("Default_Handler")));
30 void _Bus_Fault(void) __attribute__((weak,alias("Default_Handler")));
31 void _Usage_Fault_Handler(void) __attribute__((weak,alias("Default_Handler")));
32
33 // __attribute__((section(".vectors"))) put the array at vectors section
34
35 // .vectors we will be at the beginning of .text section which will be at the entry
36 extern uint32_t _stack_top;
37

```

when we define the vector table of our MC, it may be having a huge number of interrupts, so it's not sensible to define all of them, in addition, the user itself may not use all of them so why occupies an address although the user doesn't use them?!!

The weak and alias help to avoid that, as we give the function a weak attribute to make it overridden and an alias to a default handler definition by this way, a user can override the function definition and if it not defined, it takes the default aliased definition.

```
3 Author: Eng.Mustafa Hafez Eldeep
4 Date: 30/7/2023
5 Family: Cortex-m
6 */
7
8 /* This Family has a wonderful
9 so, we can write the startup
10 */
11
12 /* Imagine with me that we have
13 are arranged squantily in the
14 processor, so we can make the
15
16
17
18 #include <stdint.h>
19
20 extern void main(void);
21
22 void _Reset_Handler(void);
23
24 void Default_Handler(void) {
25
26     _Reset_Handler();
27 }
28 void NMI_Handler(void) {}
29 void Hard_Fault_Handler(void) {}
30 void MM_Fault_Handler(void) __attribute__((weak, alias("Default_Handler")));
31 void Bus_Fault(void) __attribute__((weak, alias("Default_Handler")));
32 void Usage_Fault_Handler(void) __attribute__((weak, alias("Default_Handler")));
33
34 //__attribute__((section(".vectors"))) put the array at vector section
35
```

MINGW32/d/KS\_ES/Unit3\_Lesson3\_Embedded\_C/Lab2\_Staetup\_C

```
Mustafa Hafez@DESKTOP-FFV8M55 MINGW32 /d/KS_ES/Unit3_Lesson3_Embedded_C/Lab2_Staetup_C
$ mingw32-make.exe clean_all
rm *.elf *.bin *.o

Mustafa Hafez@DESKTOP-FFV8M55 MINGW32 /d/KS_ES/Unit3_Lesson3_Embedded_C/Lab2_Staetup_C
$ mingw32-make.exe
arm-none-eabi-gcc.exe -c -std=c99 -I. -mcpu=cortex-m3 -mthumb -gdwarf-2 startup.c -o startup.o
arm-none-eabi-gcc.exe -c -std=c99 -I. -mcpu=cortex-m3 -mthumb -gdwarf-2 main.c -o main.o
arm-none-eabi-ld.exe -T linker_script.ld -Map=map_file.map startup.o main.o -o Toggle_LED_Cortex_m3.elf
arm-none-eabi-objcopy.exe -O binary Toggle_LED_Cortex_m3.elf Toggle_LED_Cortex_m3.bin
___Building process is done: Mustafa Hafez___

Mustafa Hafez@DESKTOP-FFV8M55 MINGW32 /d/KS_ES/Unit3_Lesson3_Embedded_C/Lab2_Staetup_C
$ arm-none-eabi-nm.exe Toggle_LED_Cortex_m3.elf
0800001c W _Bus_Fault
08000034 T _Hard_Fault_Handler
0800001c W _MM_Fault_Handler
08000028 T _NMI_Handler
08000040 T _Reset_Handler
0800001c W _Usage_Fault_Handler
08000000 T _vectors
0800001c T Default_Handler
08000034 T main
080000dc D R_ODR
```

Bus\_Fault and Usage\_Fault\_Handler  
Have a weak and alias to Default handler so that each one of them has the same address.

NMI\_Handler and Hard\_Fault\_Handler  
Have not a weak neither alias so that each one of

MINGW32/d/KS\_ES/Unit3\_Lesson3\_Embedded\_C/Lab2\_Staetup\_C

```
Mustafa Hafez@DESKTOP-FFV8M55 MINGW32 /d/KS_ES/Unit3_Lesson3_Embedded_C/Lab2_Staetup_C
$ mingw32-make.exe clean_all
rm *.elf *.bin *.o

Mustafa Hafez@DESKTOP-FFV8M55 MINGW32 /d/KS_ES/Unit3_Lesson3_Embedded_C/Lab2_Staetup_C
$ mingw32-make.exe
arm-none-eabi-gcc.exe -c -std=c99 -I. -mcpu=cortex-m3 -mthumb -gdwarf-2 startup.c -o startup.o
arm-none-eabi-gcc.exe -c -std=c99 -I. -mcpu=cortex-m3 -mthumb -gdwarf-2 main.c -o main.o
arm-none-eabi-ld.exe -T linker_script.ld -Map=map_file.map startup.o main.o -o Toggle_LED_Cortex_m3.elf
arm-none-eabi-objcopy.exe -O binary Toggle_LED_Cortex_m3.elf Toggle_LED_Cortex_m3.bin
___Building process is done: Mustafa Hafez___

Mustafa Hafez@DESKTOP-FFV8M55 MINGW32 /d/KS_ES/Unit3_Lesson3_Embedded_C/Lab2_Staetup_C
$ arm-none-eabi-nm.exe Toggle_LED_Cortex_m3.elf
0800001c W _Bus_Fault
0800001c W _Hard_Fault_Handler
0800001c W _MM_Fault_Handler
0800001c W _NMI_Handler
08000028 T _Reset_Handler
0800001c W _Usage_Fault_Handler
08000000 T _vectors
0800001c T Default_Handler
08000034 T main
080000dc D R_ODR
```

Weak and alias to the Default\_Handler

```

/*Extern the symbol from linker file so that the linker dosen't give an error while
compiling the startup.c*/
extern uint32_t _S_TEXT;
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(void)
{
    /* 1- Copy Data from ROM to SRAM.*/
    uint32_t DATA_SIZE = (uint32_t)&_E_DATA - (uint32_t)&_S_DATA;
    unsigned char* P_SOURCE = (unsigned char*)&_E_TEXT;
    unsigned char* P_DESTINATION = (unsigned char*)&_S_DATA;

    for(uint32_t i = 0; i<DATA_SIZE;i++)
    {
        *((unsigned char*)P_DESTINATION++) = *((unsigned char*)P_SOURCE++);
    }

    /* 2- Reserve a .bss with its size in the SRAM.*/
    uint32_t BSS_SIZE = (uint32_t)&_E_BSS - (uint32_t)&_S_BSS;
    unsigned char* P_BSS = (unsigned char*)&_S_BSS;
    for(uint32_t i = 0; i<BSS_SIZE;i++)
    {
        *((unsigned char*)P_BSS++)= (unsigned char)0;
    }

    /* 3- Jump to the main function.*/
    main();
}

```

#### 1-Copying data from ROM to SRAM:

As we know the .data section is burned on the flash but during the startup, it's copied to the SRAM ,we can do that if we know:

- what's the start address of .data section
- what's the end address of .data section.
- As a result, the size of .data is known.

#### 2-Intiate the .bss with zero at the SRAM:

As we know the .bss section doesn't exist at the flash but with an information of its size we initialize the SRAM with it.

we can do that if we know:

- what's the start address of .bss sectoin
- what's the end address of .bss sectoin.
- As a result the size of .data is known.

#### Important Note:

- 1-We use VMA (Virtual memory address) while copying the data to SRAM while LMA (Load Memory Address) is the physical address at burning.
- 2-Don't forget to **check the map file** to make sure everything is correct, and the memory is aligned (no odd address as its effect on performance).
- 3-Use ALIGN (4) to align your memory with 4 Bytes if you need.

