Startup code for STM32f756VGTx (GCC based toolchain)

```
.global  g_pfnVectors
.global  Default_Handler
```

- If a label is marked as .global it can be accessed by other files.We will see where this g_pfnVectors is initialized below.

```
/* start address for the initialization values of the .data section.
defined in linker script */
.word  _sidata
/* start address for the .data section. defined in linker script */
.word  _sdata
/* end address for the .data section. defined in linker script */
.word  _edata
/* start address for the .bss section. defined in linker script */
.word  _sbss
/* end address for the .bss section. defined in linker script */
.word  _ebss
/* stack used for SystemInit_ExtMemCtl; always internal RAM used */
```

- The labels _sidata,_sdata,_edata,_sbss and _ebss represents :
    - _sidata:Start address of the data section in the Flash memory.
    - _sdata :Start address of the data section in the RAM.
    - _edata :end address of the data section in the RAM.
    - _sbss and _ebss represents the start and end address of the bss section in the RAM.
- These all are defined in the linker script(FLASH.ld file) as shown below:

```
SECTIONS
{
 /* Used by the startup to initialize data */
 _sidata = LOADADDR(.data);
 /* Initialized data sections into "RAM" Ram type memory */
 .data :
 {
   . = ALIGN(4);
   _sdata = .;        /* create a global symbol at data start */
   *(.data)           /* .data sections */
   *(.data*)          /* .data* sections */
   *(.RamFunc)        /* .RamFunc sections */
   *(.RamFunc*)       /* .RamFunc* sections */
   . = ALIGN(4);
   _edata = .;        /* define a global symbol at data end */
 } >RAM AT> FLASH
 /* Uninitialized data section into "RAM" Ram type memory */
 . = ALIGN(4);
 .bss :
 {
   /* This is used by the startup in order to initialize the .bss section */
   _sbss = .;         /* define a global symbol at bss start */
   __bss_start__ = _sbss;
   *(.bss)
   *(.bss*)
   *(COMMON)
   . = ALIGN(4);
   _ebss = .;         /* define a global symbol at bss end */
   __bss_end__ = _ebss;
 } >RAM
}
```

- MEMORY section is used to place the different types of memory present in the microcontroller
- Its consists of name of the memory,It's starting address and the length of the memory.

```
MEMORY
{
 RAM    (xrw)    : ORIGIN = 0x20000000,   LENGTH = 320K
 FLASH    (rx)    : ORIGIN = 0x08000000,   LENGTH = 1024K
}
```

- SECTIONS consist of different types of sections like text section which consist of code,data section which consists of initialized data,bss section which contain the uninitialized data etc….

- Developer can place different sections at his desired location by placing them sequentially.

For example:If he needs to place the vector table at the start of flash memory he can write the linker script as below:

```
SECTIONS
{
 /* The startup code into "FLASH" Rom type memory */
 .isr_vector :
 {
   . = ALIGN(4);         /*To make sure the vectors are placed at 4 byte
boundary*/
   KEEP(*(.isr_vector)) /* All the files in which this .isr_vector is present
are placed here and KEEP is used so that it is compulsory placed in memory
without opmimizing it and removing it as it may not be referenced in any
program */
   . = ALIGN(4);         /*finally the address is aligned to 4 byte
boundary*/
 } >FLASH                /*These vectors are physically placed at address
0x08000000,which is the flash memory start address.*/
}
```

Coming back to the Startup code:

```
 .section  .text.Reset_Handler
 .weak  Reset_Handler
 .type  Reset_Handler, %function
Reset_Handler:
 ldr   sp, =_estack      /* set stack pointer */
/* Call the clock system initialization function.*/
 bl  SystemInit
```

- A section is defined as .text. Whose name is Reset_Handler.
- It's made as weak so that in any other file if its overridden,the linker links that address to the Reset_Handler.
- The type of label Reset_Handler is made as Function.It is helpful for linker while resolving symbols.
- Reset_Handler: ------>denotes the start of the Reset handler.

- `ldr sp, =_estack` ---->The stack pointer is been initialized to the value _estack.The _estack value is is been initialized in the linker script as below:

    - `_estack = ORIGIN(RAM) + LENGTH(RAM);`

- It contains the top address of the RAM.
- `bl SystemInit /*Branch and link to SystemInit() function */`
- After this the execution jumps to SystemInit() function defined in system_stm32f7xx.c.It is as follows:

```
void SystemInit(void)
{
 /* FPU settings
------------------------------------------------------------*/
#if (__FPU_PRESENT == 1) && (__FPU_USED == 1)
 SCB->CPACR |= ((3UL << 10*2)|(3UL << 11*2));  /* set CP10 and CP11 Full
Access */
#endif
 /* Configure the Vector Table location
--------------------------------------*/
#if defined(USER_VECT_TAB_ADDRESS)
 SCB->VTOR = VECT_TAB_BASE_ADDRESS | VECT_TAB_OFFSET; /* Vector Table
Relocation in Internal SRAM */
#endif /* USER_VECT_TAB_ADDRESS */
}
```

Decoding of the SystemInit() function:

- The __FPU_PRESENT is been defined with 1U in the STM32f756xx.h and __FPU_USED is also made as 1U in core_cm7.h.So the control goes inside the #if directive and makes the floating point unit to be fully accessed by making CP10 and CP11 bits set(Bits 20,21,22,23 are set).

```
SCB->CPACR |= ((3UL << 10*2)|(3UL << 11*2)); /*This CPACR is a memory mapped
register in the system control space.The core_cm7.h provides abstraction in
which these registers are mapped to physical addresses so that we can
directly access and modify them easily.*/
```

- Next is the relocation of the vector table if necessary by the user.He can relocate the vector table to the RAM memory by changing the SCB->VTOR which is called as vector table offset register.
- In the same file system_stm32f7xx.c,we have these lines of code:

```
/*!< Uncomment the following line if you need to relocate the vector table
    anywhere in Flash or Sram, else the vector table is kept at the automatic
remap of boot address selected */
/* #define USER_VECT_TAB_ADDRESS */
#if defined(USER_VECT_TAB_ADDRESS)
/*!< Uncomment the following line if you need to relocate your vector Table
    in Sram else user remap will be done in Flash. */
/* #define VECT_TAB_SRAM */
#if defined(VECT_TAB_SRAM)
#define VECT_TAB_BASE_ADDRESS   RAMDTCM_BASE    /*!< Vector Table base
address field.This value must be a multiple of 0x200.This is defined in
core_cm7.h whose value is equal to base address of RAM i.e 0x20000000
,#define RAMDTCM_BASE           0x20000000UL */
#define VECT_TAB_OFFSET         0x00000000U     /*!< Vector Table base
offset field.This value must be a multiple of 0x200. */
```

```
#else
#define VECT_TAB_BASE_ADDRESS   FLASH_BASE      /*!< Vector Table base
address field.This value must be a multiple of 0x200. */
#define VECT_TAB_OFFSET         0x00000000U     /*!< Vector Table base
offset field.This value must be a multiple of 0x200. */
#endif /* VECT_TAB_SRAM */
#endif /* USER_VECT_TAB_ADDRESS */
```

- **After the SystemInit function is completed the execution returns to reset handler by using the link register and the execution continues as follows:**

```
/* Copy the data segment initializers from
flash to SRAM */
 ldr r0, =_sdata
 ldr r1, =_edata
 ldr r2, =_sidata
 movs r3, #0
 b LoopCopyDataInit
CopyDataInit:
 ldr r4, [r2, r3]
 str r4, [r0, r3]
 adds r3, r3, #4
LoopCopyDataInit:
 adds r4, r0, r3
 cmp r4, r1
 bcc CopyDataInit
```

Decoding the above code:

- At first the variables present in the data section of the FLASH memory are been copied to the Data section of the RAM by using the start address of the data section in FLASH,Start and end address of the data memory in RAM.

- After copying the processor can access all the data from the RAM.

- This copying is required because after reset,all the variables should be initialized to their default values for the execution of the program in the known state.

```
/* Zero fill the bss segment. */
 ldr r2, =_sbss
 ldr r4, =_ebss
 movs r3, #0
 b LoopFillZerobss
FillZerobss:
 str  r3, [r2]
 adds r2, r2, #4
LoopFillZerobss:
 cmp r2, r4
 bcc FillZerobss
```

- In the similar way the next step is to ZERO the entire bss section so that they can be initialized again in the program run time.

- The above two steps can be possible only if we know the addresses of the data section and bss section.These are provided by the linker script as we have seen in the beginning.

- Finally the main() function is called:

```
/* Call the application's entry point.*/
 bl  main
```

- Now the execution goes to the main function where the developer application code gets executed.

Now let us see what is present in the startup file other than the Reset_Handler which is very important:

```
 .section  .text.Default_Handler,"ax",%progbits
Default_Handler:
Infinite_Loop:
 b  Infinite_Loop
 .size  Default_Handler, .-Default_Handler
```

- This is the definition for any handler which is aliased to Default_Handler which will be seeing below.Its just an infinite loop.

```
  .section  .isr_vector,"a",%progbits
 .type  g_pfnVectors, %object
 .size  g_pfnVectors, .-g_pfnVectors


g_pfnVectors:
 .word  _estack
 .word  Reset_Handler
 .word  NMI_Handler
 .word  HardFault_Handler
 .word  MemManage_Handler
 .word  BusFault_Handler
 .word  UsageFault_Handler
 .word  0
 .word  0
 .word  0
 .word  0
 .word  SVC_Handler
 .word  DebugMon_Handler
 .word  0
 .word  PendSV_Handler
 .word  SysTick_Handler
```

- This is the .isr_vector section where all the interrupt handlers addresses are placed so that when an interrupt occurs,by filling the Program Counter register with the interrupt address which has occurred,it executes the interrupt handler.
- We have seen at the beginning,in the linker script we have placed this section at the base address of FLASH memory.So,these are stored at that address one after the other.
- The above are only stack pointer and the system exceptions.We also have external interrupts as shown below:

```
 /* External Interrupts */
 .word     WWDG_IRQHandler                  /* Window WatchDog
*/
 .word     PVD_IRQHandler                   /* PVD through EXTI Line
detection */
 .word     TAMP_STAMP_IRQHandler            /* Tamper and TimeStamps through
the EXTI line */
 .word     RTC_WKUP_IRQHandler              /* RTC Wakeup through the EXTI
line */
 .word     FLASH_IRQHandler                 /* FLASH
*/
 .word     RCC_IRQHandler                   /* RCC
*/
 .word     EXTI0_IRQHandler                 /* EXTI Line0
*/
 .word     EXTI1_IRQHandler                 /* EXTI Line1
*/
 .word     EXTI2_IRQHandler                 /* EXTI Line2
*/
 .word     EXTI3_IRQHandler                 /* EXTI Line3
*/
```

```
 .word     EXTI4_IRQHandler                  /* EXTI Line4
*/
 .word     DMA1_Stream0_IRQHandler           /* DMA1 Stream 0
*/
 .word     DMA1_Stream1_IRQHandler           /* DMA1 Stream 1
*/
 .word     DMA1_Stream2_IRQHandler           /* DMA1 Stream 2
*/
 .word     DMA1_Stream3_IRQHandler           /* DMA1 Stream 3
*/
 .word     DMA1_Stream4_IRQHandler           /* DMA1 Stream 4
*/
 .word     DMA1_Stream5_IRQHandler           /* DMA1 Stream 5
*/
 .word     DMA1_Stream6_IRQHandler           /* DMA1 Stream 6
*/
 .word     ADC_IRQHandler                    /* ADC1, ADC2 and ADC3s
*/
 .word     CAN1_TX_IRQHandler                /* CAN1 TX
*/
 .word     CAN1_RX0_IRQHandler               /* CAN1 RX0
*/
 .word     CAN1_RX1_IRQHandler               /* CAN1 RX1
*/
 .word     CAN1_SCE_IRQHandler               /* CAN1 SCE
*/
 .word     EXTI9_5_IRQHandler                /* External Line[9:5]s
*/
 .word     TIM1_BRK_TIM9_IRQHandler          /* TIM1 Break and TIM9
*/
 .word     TIM1_UP_TIM10_IRQHandler          /* TIM1 Update and TIM10
*/
 .word     TIM1_TRG_COM_TIM11_IRQHandler     /* TIM1 Trigger and Commutation
and TIM11 */
 .word     TIM1_CC_IRQHandler                /* TIM1 Capture Compare
*/
 .word     TIM2_IRQHandler                   /* TIM2
*/
 .word     TIM3_IRQHandler                   /* TIM3
*/
 .word     TIM4_IRQHandler                   /* TIM4
*/
 .word     I2C1_EV_IRQHandler                /* I2C1 Event
*/
 .word     I2C1_ER_IRQHandler                /* I2C1 Error
*/
 .word     I2C2_EV_IRQHandler                /* I2C2 Event
*/
 .word     I2C2_ER_IRQHandler                /* I2C2 Error
*/
 .word     SPI1_IRQHandler                   /* SPI1
*/
```

```
  .word     SPI2_IRQHandler                   /* SPI2
*/
  .word     USART1_IRQHandler                 /* USART1
*/
  .word     USART2_IRQHandler                 /* USART2
*/
  .word     USART3_IRQHandler                 /* USART3
*/
  .word     EXTI15_10_IRQHandler              /* External Line[15:10]s
*/
  .word     RTC_Alarm_IRQHandler              /* RTC Alarm (A and B) through
EXTI Line */
  .word     OTG_FS_WKUP_IRQHandler            /* USB OTG FS Wakeup through
EXTI line */
  .word     TIM8_BRK_TIM12_IRQHandler         /* TIM8 Break and TIM12
*/
  .word     TIM8_UP_TIM13_IRQHandler          /* TIM8 Update and TIM13
*/
  .word     TIM8_TRG_COM_TIM14_IRQHandler     /* TIM8 Trigger and Commutation
and TIM14 */
  .word     TIM8_CC_IRQHandler                /* TIM8 Capture Compare
*/
  .word     DMA1_Stream7_IRQHandler           /* DMA1 Stream7
*/
  .word     FMC_IRQHandler                    /* FMC
*/
  .word     SDMMC1_IRQHandler                 /* SDMMC1
*/
  .word     TIM5_IRQHandler                   /* TIM5
*/
  .word     SPI3_IRQHandler                   /* SPI3
*/
  .word     UART4_IRQHandler                  /* UART4
*/
  .word     UART5_IRQHandler                  /* UART5
*/
  .word     TIM6_DAC_IRQHandler               /* TIM6 and DAC1&2 underrun
errors */
  .word     TIM7_IRQHandler                   /* TIM7
*/
  .word     DMA2_Stream0_IRQHandler           /* DMA2 Stream 0
*/
  .word     DMA2_Stream1_IRQHandler           /* DMA2 Stream 1
*/
  .word     DMA2_Stream2_IRQHandler           /* DMA2 Stream 2
*/
  .word     DMA2_Stream3_IRQHandler           /* DMA2 Stream 3
*/
  .word     DMA2_Stream4_IRQHandler           /* DMA2 Stream 4
*/
  .word     ETH_IRQHandler                    /* Ethernet
*/
```

```
  .word     ETH_WKUP_IRQHandler              /* Ethernet Wakeup through EXTI
line */
  .word     CAN2_TX_IRQHandler               /* CAN2 TX
*/
  .word     CAN2_RX0_IRQHandler              /* CAN2 RX0
*/
  .word     CAN2_RX1_IRQHandler              /* CAN2 RX1
*/
  .word     CAN2_SCE_IRQHandler              /* CAN2 SCE
*/
  .word     OTG_FS_IRQHandler                /* USB OTG FS
*/
  .word     DMA2_Stream5_IRQHandler          /* DMA2 Stream 5
*/
  .word     DMA2_Stream6_IRQHandler          /* DMA2 Stream 6
*/
  .word     DMA2_Stream7_IRQHandler          /* DMA2 Stream 7
*/
  .word     USART6_IRQHandler                /* USART6
*/
  .word     I2C3_EV_IRQHandler               /* I2C3 event
*/
  .word     I2C3_ER_IRQHandler               /* I2C3 error
*/
  .word     OTG_HS_EP1_OUT_IRQHandler        /* USB OTG HS End Point 1 Out
*/
  .word     OTG_HS_EP1_IN_IRQHandler         /* USB OTG HS End Point 1 In
*/
  .word     OTG_HS_WKUP_IRQHandler           /* USB OTG HS Wakeup through
EXTI */
  .word     OTG_HS_IRQHandler                /* USB OTG HS
*/
  .word     DCMI_IRQHandler                  /* DCMI
*/
  .word     CRYP_IRQHandler                  /* Crypto
*/
  .word     HASH_RNG_IRQHandler              /* Hash and Rng
*/
  .word     FPU_IRQHandler                   /* FPU
*/
  .word     UART7_IRQHandler                 /* UART7
*/
  .word     UART8_IRQHandler                 /* UART8
*/
  .word     SPI4_IRQHandler                  /* SPI4
*/
  .word     SPI5_IRQHandler                  /* SPI5
*/
  .word     SPI6_IRQHandler                  /* SPI6
*/
  .word     SAI1_IRQHandler                  /* SAI1
*/
```

```
 .word     LTDC_IRQHandler                  /* LTDC
*/
 .word     LTDC_ER_IRQHandler               /* LTDC error
*/
 .word     DMA2D_IRQHandler                 /* DMA2D
*/
 .word     SAI2_IRQHandler                  /* SAI2
*/
 .word     QUADSPI_IRQHandler               /* QUADSPI
*/
 .word     LPTIM1_IRQHandler                /* LPTIM1
*/
 .word     CEC_IRQHandler                   /* HDMI_CEC
*/
 .word     I2C4_EV_IRQHandler               /* I2C4 Event
*/
 .word     I2C4_ER_IRQHandler               /* I2C4 Error
*/
 .word     SPDIF_RX_IRQHandler              /* SPDIF_RX
*/


*****************************************************************************
**
*
* Provide weak aliases for each Exception handler to the Default_Handler.
* As they are weak aliases, any function with the same name will override
* this definition.
*
*****************************************************************************
**/
  .weak      NMI_Handler
  .thumb_set NMI_Handler,Default_Handler
   .weak      HardFault_Handler
  .thumb_set HardFault_Handler,Default_Handler
   .weak      MemManage_Handler
  .thumb_set MemManage_Handler,Default_Handler
   .weak      BusFault_Handler
  .thumb_set BusFault_Handler,Default_Handler
```

- Like this for all the handlers weak alias is provided to the default handler which we have seen above.So all the exceptions and interrupts except Reset handler when occurred goes to the default handler for the execution.

- By setting `all handlers` to `Default_Handler`, you ensure that if no specific handler is provided for the interrupts(i.e they are not been overridden by their definition), the default handler will be used instead. This improves the robustness of the application by ensuring that all possible interrupts or exceptions have a defined behaviour, even if it's just a default response.