Hello everyone!

I'm doing a project for STM32F411RE where I have three different applications. The first one is the bootloader. It's placed at the beggining of FLASH memory and its job is to read the state of one input pin and depending on that,, copy one of the two applications from FLASH to RAM, and then jump to RAM to start executing it. The two applications are very simple - they are just blinking LEDs. The only difference between application 1 and 2 is the frequency of the blinking.

The projects works as expected when I'm using simple for() loop as a delay between blinks. But the next step is to make it work with interrupts.

The applications have their own vector table that is also copied to RAM. After the jump I'm setting the SCB->VTOR register according to the vector's location in memory. I'm able to do this using external variable from linker script. I can see that the location that is being set is correct using GDB and objdump of an .elf file. I am also sure that the code used for enabling the timer interrupt and the ISR is correct because the same code is used in another one of my projects. But in this project, the interrupts doesn't work at all after the jump to RAM.

The memory map is looks like this:

FLASH_BOOTLOADER (rx) : ORIGIN = 0x8000000, LENGTH = 64K

FLASH_APP1 (rx) : ORIGIN = 0x8010000, LENGTH = 64K

FLASH_APP2 (rx) : ORIGIN = 0x8020000, LENGTH = 64K

RAM (xrw) : ORIGIN = 0x20000000, LENGTH = 4K **<- small section of RAM reserved for the bootloader**
RAMAPP (xrw) : ORIGIN = 0x20001000, LENGTH = 124K **<- RAM where the applications are copied**

Here is the code used for the three applications:

bootloader.c - reads input pin, copies application1/2 to RAM and jumps to it. For initialisation it uses the default stm32f411re startup code taken from CubeMX (startup_stm32f411xe.s)

```
#include "stm32f411xe.h"

typedef  void (*pFunction)(void);

static inline void __initialize_data (unsigned int* from, unsigned int* region_begin, unsigned
int* region_end)
{
   // Copy data from flash section of fw1/fw2 to RAM
   // Iterate and copy word by word.
   // It is assumed that the pointers are word aligned.
   unsigned int *p = region_begin;
   while (p < region_end)
      *p++ = *from++;
}

int main()
{
   pFunction Jump_To_Fixed;
   unsigned int * __textdata1__ = (unsigned int *)0x08010000;  //start address of application 1
in flash memory
   unsigned int * __textdata2__ = (unsigned int *)0x08020000;    //start address of application 2
in flash memory
   unsigned int * __scode__ = (unsigned int *)0x20001000;
```

```
    unsigned int * __ecode__ = (unsigned int *)0x20003000;
    //configuration of PB1 pin as input
    RCC->AHB1ENR  |= RCC_AHB1ENR_GPIOBEN; //clock enable for GPIO port B
    GPIOB->MODER &= 0xFFFFFFF3;
    GPIOB->MODER |= 0x00000000; //configure PB1 as input - corresponding MODER bits set
  to 00

    if ((GPIOB->IDR & 0x00000002) > 0)     //read pin state
    {
       __initialize_data(__textdata1__, __scode__, __ecode__);
       Jump_To_Fixed = (pFunction) __scode__;
       Jump_To_Fixed++; //LSB set to 1 (thumb instruction)
       Jump_To_Fixed();
    }
    else
    {
       __initialize_data(__textdata2__, __scode__, __ecode__);
       Jump_To_Fixed = (pFunction) __scode__;
       Jump_To_Fixed++;
       Jump_To_Fixed();
    }
    while(1); //should never reach here
    return 0;
  }
```

linker_bootloader.ld - linker script used for bootloader

```
/* Entry Point */
ENTRY(Reset_Handler)

/* Highest address of the user mode stack */
_estack = 0x20020000;    /* end of RAM */
/* Generate a link error if heap and stack don't fit into RAM */
_Min_Heap_Size = 0x200;      /* required amount of heap  */
_Min_Stack_Size = 0x400; /* required amount of stack */

/* Specify the memory areas */
MEMORY
{
RAM (xrw)                  : ORIGIN = 0x20000000, LENGTH = 4K
FLASH_BOOTLOADER (rx)         : ORIGIN = 0x8000000, LENGTH = 64K
}

/* Define output sections */
SECTIONS
{
 /* The startup code goes first into FLASH */
 .isr_vector :
 {
   . = ALIGN(4);
   KEEP(*(.isr_vector)) /* Startup code */
   . = ALIGN(4);
 } >FLASH_BOOTLOADER

 /* The program code and other data goes into FLASH */
 .text :
 {
   . = ALIGN(4);
```

```
  *(.text)          /* .text sections (code) */
  *(.text*)         /* .text* sections (code) */
  *(.glue_7)        /* glue arm to thumb code */
  *(.glue_7t)       /* glue thumb to arm code */
  *(.eh_frame)

  KEEP (*(.init))
  KEEP (*(.fini))

  . = ALIGN(4);
  _etext = .;       /* define a global symbols at end of code */
} >FLASH_BOOTLOADER

/* Constant data goes into FLASH */
.rodata :
{
  . = ALIGN(4);
  *(.rodata)        /* .rodata sections (constants, strings, etc.) */
  *(.rodata*)       /* .rodata* sections (constants, strings, etc.) */
  . = ALIGN(4);
} >FLASH_BOOTLOADER

.ARM.extab   : { *(.ARM.extab* .gnu.linkonce.armextab.*) } >FLASH_BOOTLOADER
.ARM : {
  __exidx_start = .;
  *(.ARM.exidx*)
  __exidx_end = .;
} >FLASH_BOOTLOADER

/* used by the startup to initialize data */
_sidata = LOADADDR(.data);

/* Initialized data sections goes into RAM, load LMA copy after code */
.data :
{
  . = ALIGN(4);
  _sdata = .;       /* create a global symbol at data start */
  *(.data)          /* .data sections */
  *(.data*)         /* .data* sections */

  . = ALIGN(4);
  _edata = .;       /* define a global symbol at data end */

} >RAM AT> FLASH_BOOTLOADER


/* Uninitialized data section */
. = ALIGN(4);
.bss :
{
  /* This is used by the startup in order to initialize the .bss secion */
  _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


 /* User_heap_stack section, used to check that there is enough RAM left */
 ._user_heap_stack :
 {
  . = ALIGN(8);
  PROVIDE ( end = . );
  PROVIDE ( _end = . );
  . = . + _Min_Heap_Size;
  . = . + _Min_Stack_Size;
  . = ALIGN(8);
 } >RAM



 /* Remove information from the standard libraries */
 /DISCARD/ :
 {
  libc.a ( * )
  libm.a ( * )
  libgcc.a ( * )
 }

 .ARM.attributes 0 : { *(.ARM.attributes) }
}
```

app1.c - code of application 1, sets MSP at the end of RAM, sets SCB->VTOR based on variable from the linker script, blinks a LED

```
#include "stm32f411xe.h"
#include "app1vector.h"

void time_init()
{
   RCC->APB2ENR   |= RCC_APB2ENR_TIM9EN;     // enable clock to TIM9 peripheral
   RCC->APB2LPENR |= RCC_APB2LPENR_TIM9LPEN; // enable clock to TIM9 peripheral
also in low power mode
   RCC->APB2RSTR  |= RCC_APB2RSTR_TIM9RST;   // reset TIM9 interface
   RCC->APB2RSTR  &= ~RCC_APB2RSTR_TIM9RST;  // reset TIM9 interface
   TIM9->PSC  = 999;
   TIM9->ARR  = 7999;
   NVIC->IP[TIM1_BRK_TIM9_IRQn] = 0x70; // interrupt priority
   NVIC->ISER[TIM1_BRK_TIM9_IRQn>>5] = 1<<(TIM1_BRK_TIM9_IRQn&0x1F);  // set
enable IRQ

   // enable update (overflow) interrupt
   TIM9->DIER |= TIM_DIER_UIE;

   // Enable timer counting
   TIM9->CR1 = TIM_CR1_CEN;
}

void delay(int time)
{
```

```
    for (int i=0; i<=time; i++);
}

__attribute__((long_call,noreturn,section(".textMain"))) void main()
{
    __set_MSP(0x20020000);
    extern unsigned int __vtorAddr__;
    SCB->VTOR = (unsigned int)&__vtorAddr__;
    RCC->AHB1ENR  |= RCC_AHB1ENR_GPIOBEN | RCC_AHB1ENR_GPIOAEN;
    time_init();
    GPIOA->MODER &= 0xFFFCFFFF;
    GPIOA->MODER |= 0x00010000; //configure PA8 as output - corresponding MODER bits
set to 01
    GPIOA->ODR |= (1 << 8);
    __enable_irq();                    //OTYPER, PUPDR, OSPEEDR are default
    while(1)
    {
        //GPIOA->ODR ^= (1 << 8);
        //delay(200000);
    }
}

void TIM1_BRK_TIM9_IRQHandler ()
{
    if(TIM9->SR & TIM_SR_UIF) { // overflow
        GPIOA->ODR ^= (1 << 8);
    }
    TIM9->SR = 0;   //clear interrupt flag
}
```

app1_vector.h - header of .c file with vector table and weak ISRs. It only has typedef of pointer to function and external variable of the end of stack from linker script

```
#include "stm32f4xx.h"

// Stack start variable, needed in the vector table.
extern unsigned int _estack;
// Typedef for the vector table entries.
typedef void (* const pHandler)(void);
```

app1_vector.c - contains prototypes of ISRs as well as the vector table iitself

```
#include "stm32f4xx.h"
#include "app1vector.h"


//to do: change comments
/** Default exception/interrupt handler */
void __attribute__ ((section(".after_vectors"), noreturn)) __Default_Handler(void)
{
  while (1);
}

// Reset handler
void __attribute__ ((section(".after_vectors"), noreturn)) Reset_Handler (void);

/** Non-maskable interrupt (RCC clock security system) */
```

```
void NMI_Handler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** All class of fault */
void HardFault_Handler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** Memory management */
void MemManage_Handler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** Pre-fetch fault, memory access fault */
void BusFault_Handler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** Undefined instruction or illegal state */
void UsageFault_Handler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** System service call via SWI instruction */
void SVC_Handler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** Debug monitor */
void DebugMon_Handler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** Pendable request for system service */
void PendSV_Handler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** System tick timer */
void SysTick_Handler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** Window watchdog interrupt */
void WWDG_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** PVD through EXTI line detection interrupt */
void PVD_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** Tamper interrupt */
void TAMP_STAMP_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** RTC global interrupt */
void RTC_WKUP_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** Flash global interrupt */
void FLASH_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** RCC global interrupt */
void RCC_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** EXTI Line0 interrupt */
void EXTI0_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** EXTI Line1 interrupt */
void EXTI1_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** EXTI Line2 interrupt */
void EXTI2_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** EXTI Line3 interrupt */
void EXTI3_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));
```

```
/** EXTI Line4 interrupt */
void EXTI4_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** DMA1 Channel1 global interrupt */
void DMA1_Channel0_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** DMA1 Channel2 global interrupt */
void DMA1_Channel1_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** DMA1 Channel3 global interrupt */
void DMA1_Channel2_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** DMA1 Channel4 global interrupt */
void DMA1_Channel3_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** DMA1 Channel5 global interrupt */
void DMA1_Channel4_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** DMA1 Channel6 global interrupt */
void DMA1_Channel5_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** DMA1 Channel7 global interrupt */
void DMA1_Channel6_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** ADC1 and ADC2 global interrupt */
void ADC_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** EXTI Line[9:5] interrupts */
void EXTI9_5_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** TIM1 break interrupt */
void TIM1_BRK_TIM9_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** TIM1 update interrupt */
void TIM1_UP_TIM10_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** TIM1 trigger and commutation interrupts */
void TIM1_TRG_COM_TIM11_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** TIM1 capture compare interrupt */
void TIM1_CC_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** TIM2 global interrupt */
void TIM2_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** TIM3 global interrupt */
void TIM3_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));
```

```
/** TIM4 global interrupt */
void TIM4_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** I2C1 event interrupt */
void I2C1_EV_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** I2C1 error interrupt */
void I2C1_ER_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** I2C2 event interrupt */
void I2C2_EV_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** I2C2 error interrupt */
void I2C2_ER_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** SPI1 global interrupt */
void SPI1_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** SPI2 global interrupt */
void SPI2_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** USART1 global interrupt */
void USART1_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** USART2 global interrupt */
void USART2_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** EXTI Line[15:10] interrupts */
void EXTI15_10_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** RTC alarm through EXTI line interrupt */
void RTC_Alarm_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** USB wakeup from suspend through EXTI line interrupt */
void OTG_FS_WKUP_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** TIM8 break interrupt */
void DMA1_Stream7_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** TIM8 update interrupt */
void SDIO_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** TIM8 trigger and commutation interrupts */
void TIM5_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** TIM8 capture compare interrupt */
void SPI3_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** ADC3 global interrupt */
void DMA2_Stream0_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** FSMC global interrupt */
```

```
void DMA2_Stream1_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** SDIO global interrupt */
void DMA2_Stream2_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** TIM5 global interrupt */
void DMA2_Stream3_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** SPI3 global interrupt */
void DMA2_Stream4_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** UART4 global interrupt */
void OTG_FS_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** UART5 global interrupt */
void DMA2_Stream5_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** TIM6 global interrupt */
void DMA2_Stream6_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** TIM7 global interrupt */
void DMA2_Stream7_IRQHandler(void) __attribute__ ((interrupt, weak,
alias("__Default_Handler")));

/** DMA2 Channel1 global interrupt */
void USART6_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** DMA2 Channel2 global interrupt */
void I2C3_EV_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** DMA2 Channel3 global interrupt */
void I2C3_ER_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** DMA2 Channel4 and DMA2 Channel5 global interrupts */
void FPU_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** DMA2 Channel4 and DMA2 Channel5 global interrupts */
void SPI4_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));

/** DMA2 Channel4 and DMA2 Channel5 global interrupts */
void SPI5_IRQHandler(void) __attribute__ ((interrupt, weak, alias("__Default_Handler")));


/** STM32F103 Vector Table */
__attribute__ ((section(".vectors"), used, aligned(256))) pHandler vectors[] =
{
   (pHandler) &_estack,             // The initial stack pointer
   Reset_Handler,                  // The reset handler
   NMI_Handler,                    // The NMI handler
   HardFault_Handler,              // The hard fault handler
   MemManage_Handler,                  // The MPU fault handler
   BusFault_Handler,               // The bus fault handler
```

```
  UsageFault_Handler,               // The usage fault handler
  0,                        // Reserved
  0,                        // Reserved
  0,                        // Reserved
  0,                        // Reserved
  SVC_Handler,                  // SVCall handler
  DebugMon_Handler,             // Debug monitor handler
  0,                        // Reserved
  PendSV_Handler,               // The PendSV handler
  SysTick_Handler,              // The SysTick handler
  // -------------------------------------------------------------------
  WWDG_IRQHandler,              // Window watchdog interrupt
  PVD_IRQHandler,               // PVD through EXTI line detection interrupt
  TAMP_STAMP_IRQHandler,            // Tamper interrupt
  RTC_WKUP_IRQHandler,              // RTC global interrupt
  FLASH_IRQHandler,             // Flash global interrupt
  RCC_IRQHandler,               // RCC global interrupt
  EXTI0_IRQHandler,             // EXTI Line0 interrupt
  EXTI1_IRQHandler,             // EXTI Line1 interrupt
  EXTI2_IRQHandler,             // EXTI Line2 interrupt
  EXTI3_IRQHandler,             // EXTI Line3 interrupt
  EXTI4_IRQHandler,             // EXTI Line4 interrupt
  DMA1_Channel0_IRQHandler,         // DMA1 Channel1 global interrupt
  DMA1_Channel1_IRQHandler,         // DMA1 Channel2 global interrupt
  DMA1_Channel2_IRQHandler,         // DMA1 Channel3 global interrupt
  DMA1_Channel3_IRQHandler,         // DMA1 Channel4 global interrupt
  DMA1_Channel4_IRQHandler,         // DMA1 Channel5 global interrupt
  DMA1_Channel5_IRQHandler,         // DMA1 Channel6 global interrupt
  DMA1_Channel6_IRQHandler,         // DMA1 Channel7 global interrupt
  ADC_IRQHandler,               // ADC global interrupt
  0,                        // Reserved
  0,
  0,
  0,
  EXTI9_5_IRQHandler,               /* External Line[9:5]s      */
  TIM1_BRK_TIM9_IRQHandler,         /* TIM1 Break and TIM9      */
  TIM1_UP_TIM10_IRQHandler,         /* TIM1 Update and TIM10    */
  TIM1_TRG_COM_TIM11_IRQHandler,    /* TIM1 Trigger and Commutation and TIM11
*/
  TIM1_CC_IRQHandler,               /* TIM1 Capture Compare     */
  TIM2_IRQHandler,              /* TIM2                     */
  TIM3_IRQHandler,              /* TIM3                     */
  TIM4_IRQHandler,              /* TIM4                     */
  I2C1_EV_IRQHandler,               /* I2C1 Event               */
  I2C1_ER_IRQHandler,               /* I2C1 Error               */
  I2C2_EV_IRQHandler,               /* I2C2 Event               */
  I2C2_ER_IRQHandler,               /* I2C2 Error               */
  SPI1_IRQHandler,              /* SPI1                     */
  SPI2_IRQHandler,              /* SPI2                     */
  USART1_IRQHandler,                /* USART1                   */
  USART2_IRQHandler,                /* USART2                   */
  0,                        /* Reserved                 */
  EXTI15_10_IRQHandler,             /* External Line[15:10]s    */
  RTC_Alarm_IRQHandler,             /* RTC Alarm (A and B) through EXTI Line */

  OTG_FS_WKUP_IRQHandler,           /* USB OTG FS Wakeup through EXTI line
*/
  0,                        /* Reserved                 */
```

```
    0,                      /* Reserved                 */
    0,                      /* Reserved                 */
    0,                      /* Reserved                 */
    DMA1_Stream7_IRQHandler,        /* DMA1 Stream7             */
    0,                      /* Reserved                 */
    SDIO_IRQHandler,            /* SDIO                 */
    TIM5_IRQHandler,            /* TIM5                 */
    SPI3_IRQHandler,            /* SPI3                 */
    0,                      /* Reserved                 */
    0,                      /* Reserved                 */
    0,                      /* Reserved                 */
    0,                      /* Reserved                 */
    DMA2_Stream0_IRQHandler,        /* DMA2 Stream 0            */
    DMA2_Stream1_IRQHandler,        /* DMA2 Stream 1            */
    DMA2_Stream2_IRQHandler,        /* DMA2 Stream 2            */
    DMA2_Stream3_IRQHandler,        /* DMA2 Stream 3            */
    DMA2_Stream4_IRQHandler,        /* DMA2 Stream 4            */
    0,                      /* Reserved                 */
    0,                      /* Reserved                 */
    0,                      /* Reserved                 */
    0,                      /* Reserved                 */
    0,                      /* Reserved                 */
    0,                      /* Reserved                 */
    OTG_FS_IRQHandler,          /* USB OTG FS               */
    DMA2_Stream5_IRQHandler,        /* DMA2 Stream 5            */
    DMA2_Stream6_IRQHandler,        /* DMA2 Stream 6            */
    DMA2_Stream7_IRQHandler,        /* DMA2 Stream 7            */
    USART6_IRQHandler,          /* USART6                   */
    I2C3_EV_IRQHandler,         /* I2C3 event               */
    I2C3_ER_IRQHandler,         /* I2C3 error               */
    0,                      /* Reserved                 */
    0,                      /* Reserved                 */
    0,                      /* Reserved                 */
    0,                      /* Reserved                 */
    0,                      /* Reserved                 */
    0,                      /* Reserved                 */
    0,                      /* Reserved                 */
    FPU_IRQHandler,             /* FPU                  */
    0,                      /* Reserved                 */
    0,                      /* Reserved                 */
    SPI4_IRQHandler,            /* SPI4                 */
    SPI5_IRQHandler,            /* SPI5                 */
};

/** Reset handler */
void __attribute__ ((section(".after_vectors"), noreturn)) Reset_Handler(void)
{
  //_start();

  while(1);
}
```

app1_linker.ld - puts the application code in FLASH but links it for execution in RAM

```
/* Highest address of the user mode stack */
_estack = 0x20020000;    /* end of RAM */
/* Generate a link error if heap and stack don't fit into RAM */
_Min_Heap_Size = 0x200;      /* required amount of heap  */
```

```
_Min_Stack_Size = 0x400; /* required amount of stack */

/* Specify the memory areas */
MEMORY
{
RAMAPP (xrw)              : ORIGIN = 0x20001000, LENGTH = 124K
FLASH_APP1 (rx)           : ORIGIN = 0x8010000, LENGTH = 64K
}

/* Define output sections */
SECTIONS
{
  .text : ALIGN(4)
  {
    . = ALIGN(4);
    PROVIDE(__textdata1__ = LOADADDR(.code1));
    PROVIDE(__scode1__ = .);
    KEEP(*(.textMain))
    KEEP(*(.text))
    KEEP(*(.text*))
    *(.glue_7)         /* glue arm to thumb code */
    *(.glue_7t)        /* glue thumb to arm code */
    *(.eh_frame)

    KEEP (*(.init))
    KEEP (*(.fini))
    . = ALIGN(4);
    PROVIDE(__ecode1__ = .);
  } > RAMAPP AT> FLASH_APP1

  . = ALIGN(8);
  .isr_vector : ALIGN(8)
  {
    . = ALIGN(8);
    PROVIDE(__vtorAddr__ = .);
    KEEP(*(.vectors)) /* Startup code */
    *(.after_vectors .after_vectors.*)
    . = ALIGN(8);
  } > RAMAPP AT> FLASH_APP1

  .ARM.extab : { *(.ARM.extab* .gnu.linkonce.armextab.*) } > RAMAPP AT> FLASH_APP1
  .ARM : {
    __exidx_start = .;
    *(.ARM.exidx*)
    __exidx_end = .;
  } > RAMAPP AT> FLASH_APP1

  /* Initialized data sections goes into RAM, load LMA copy after code */
  .data :
  {
    . = ALIGN(4);
    _sdata = .;        /* create a global symbol at data start */
    *(.data)           /* .data sections */
    *(.data*)          /* .data* sections */

    . = ALIGN(4);
    _edata = .;        /* define a global symbol at data end */
```

```
  } >RAMAPP AT> FLASH_APP1

  /* Uninitialized data section */
  . = ALIGN(4);
  .bss :
  {
   /* This is used by the startup in order to initialize the .bss secion */
   _sbss = .;         /* define a global symbol at bss start */
   __bss_start__ = _sbss;
   *(.bss)
   *(.bss*)
   *(COMMON)

   . = ALIGN(4);
   PROVIDE(_ebss = .);         /* define a global symbol at bss end */
   PROVIDE(__bss_end__ = _ebss);
  } >RAMAPP


  /* User_heap_stack section, used to check that there is enough RAM left */
  ._user_heap_stack :
  {
   . = ALIGN(8);
   PROVIDE ( end = . );
   PROVIDE ( _end = . );
   . = . + _Min_Heap_Size;
   . = . + _Min_Stack_Size;
   . = ALIGN(8);
  } >RAMAPP


  /* Constant data goes into FLASH */
  .rodata :
  {
   . = ALIGN(4);
   *(.rodata)         /* .rodata sections (constants, strings, etc.) */
   *(.rodata*)        /* .rodata* sections (constants, strings, etc.) */
   . = ALIGN(4);
  } >FLASH_APP1

  /* Remove information from the standard libraries */
  /DISCARD/ :
  {
   libc.a ( * )
   libm.a ( * )
   libgcc.a ( * )
  }

   .ARM.attributes 0 : { *(.ARM.attributes) }
 }
```

The files for app2 are almost the same. The only differences are:

- flash location of the code in the linker script
  - FLASH_APP2 (rx)   : ORIGIN = 0x8020000, LENGTH = 64K
- prescaler value in the set up of the timer interrupt

- TIM9->ARR = 15999;

Using GDB for the code debugging I am able to see that after the jump to RAM, the execution of app1/2 works well and gets to the infinite while() loop in main(). If I keep steping through the code nothing happens - it's stuck in the while() loop and the ISR never gets called. I've even tried to put some code inside while() to be able to see what's going on:

```
while(1)
{
   volatile int counter=1;
   counter=TIM9->CNT;
   counter=TIM9->SR;
   counter=TIM9->DIER;
}
```

Using that I am able to see that the CNT value in fact rises and overflows. The SR register value is constantly 0x00000007 which means that there timer update event occured. The DIER is 0x00000001 (as I set it). These values means that the ISR should fire up but it doesn't when I step through the code, but when I tell GDB to Continue and then Pause, I can see some strange behaviour:

I have no idea where these registers values come from and why the PC ends up at 0x00000100. I can only tell from xPSR register that its a HardFault.

What's more funny, when I don't align the vector table to 2^8 bytes (deleting attribute(align(256)) from vector table definiction) I get more understandable behaviour - the program ends up in Default Handler with IPSR value of 28, which problably means that the Timer 9 interrupt fired up, but due to the lack of alignment, the adressess in the vector table were messed up, so the program ended up in the DefaultHandler instead of the Timer 9 ISR.

As I previously said - using objdump I am able to see that the vector table ends up at 0x20001600 memory location (with alignment) and this value is in fact set in SCB-VTOR. I can also confirm that the Timer 9 entry in the vector table correctly points to my implementation of TIM1_BRK_TIM9_IRQHandler (with LSB set for thumb instruction).

I'm running out of ideas what could be wrong and I completely don't understand why I'm getting more reasonable results when not aligning the vector table.

I know that it's a long post but perhaps someone here would be able to help me. I would be very grateful.