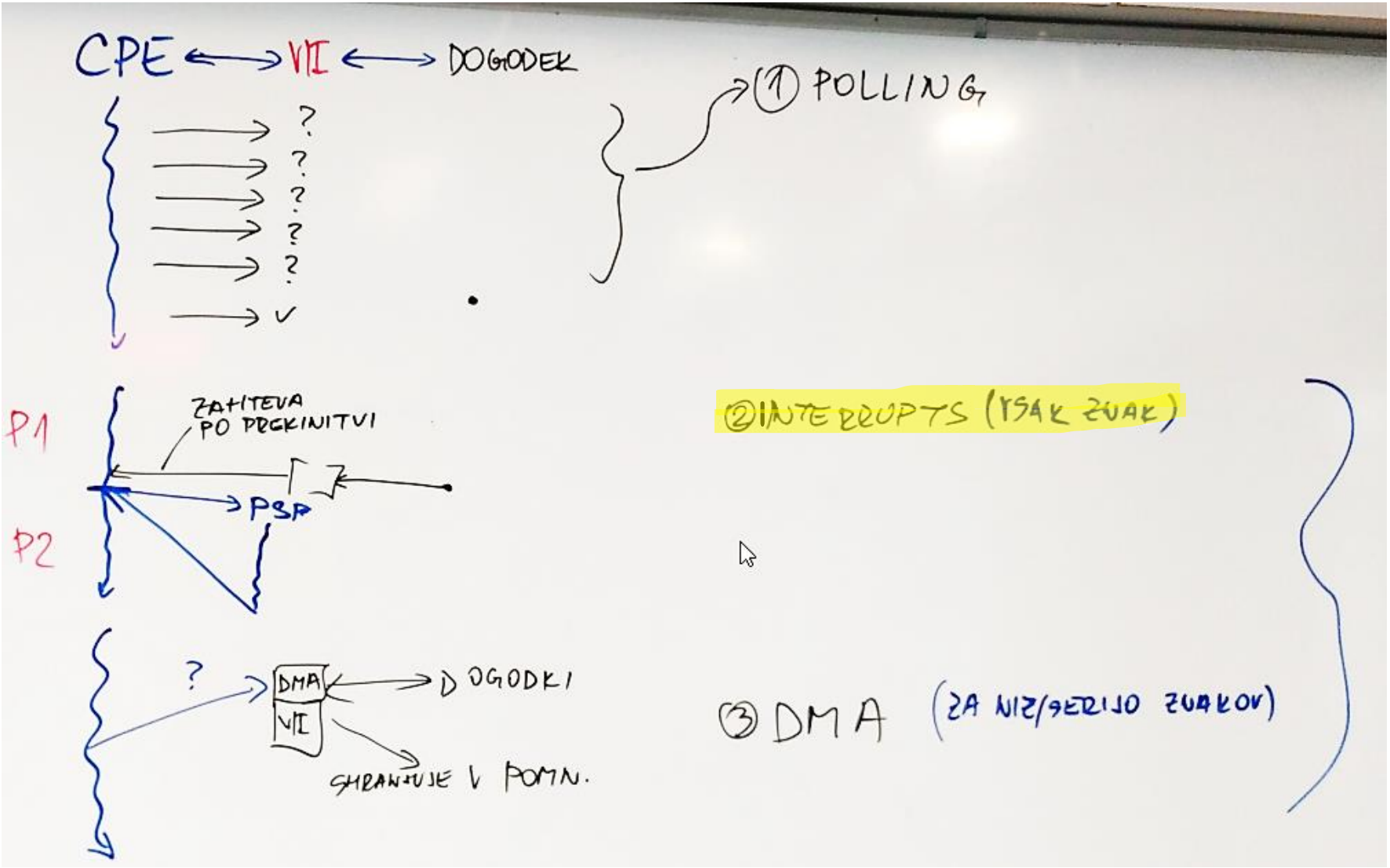
STM32H7

Vhodno / izhodne naprave

Prekinitve
+
SysTick Časovnik

Prekinitve – Zakaj ?

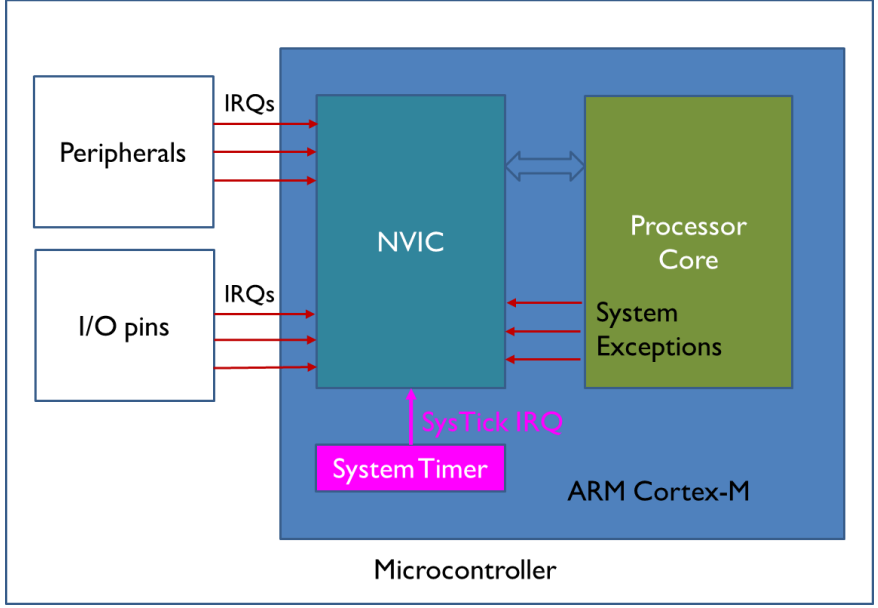


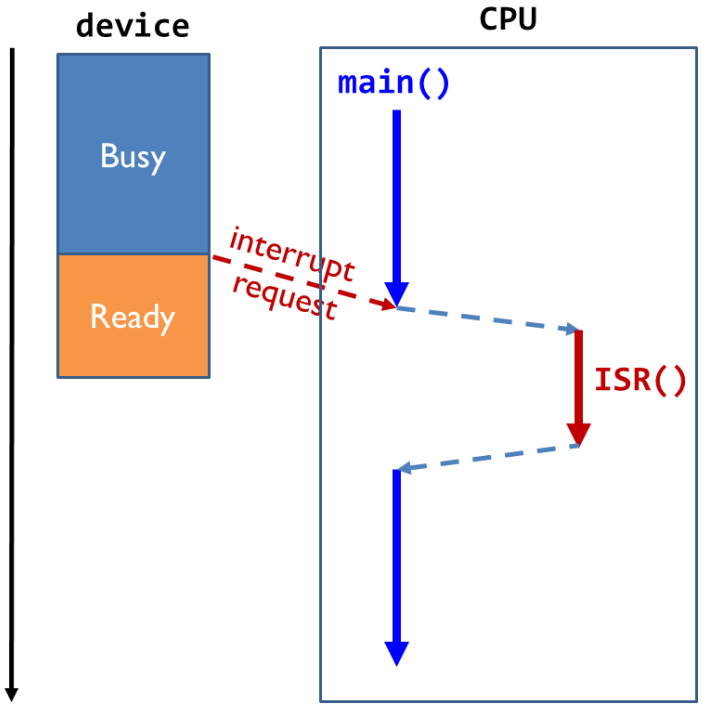
Prekinitve - dogodki

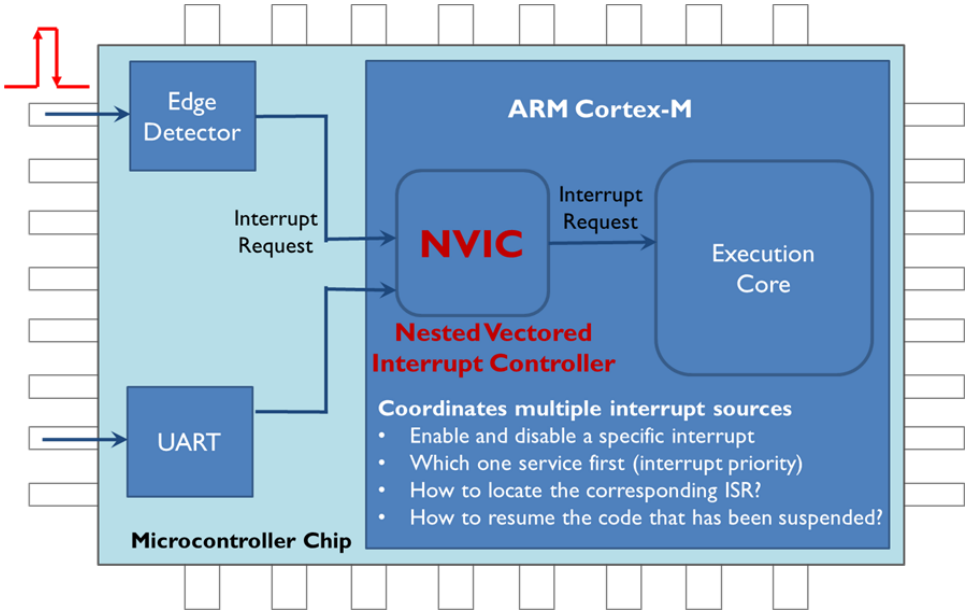
Optimizacija programske opreme za ustrezne reakcije na dogodke v realnem času:

- dogodkovno vodene reakcije
- ciklične prekinitve s časovniki

Prožijo se ustrezni Prekinitveno servisni podprogrami – PSP (ang. ISR)







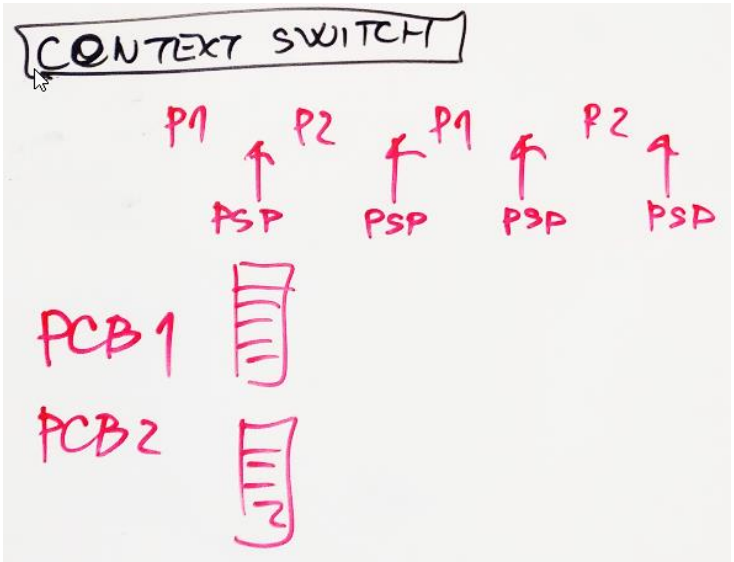
Veliko virov (naprav) povezanik na krmilnik (NVIC), ki naprej sporoča zahteve na CPE

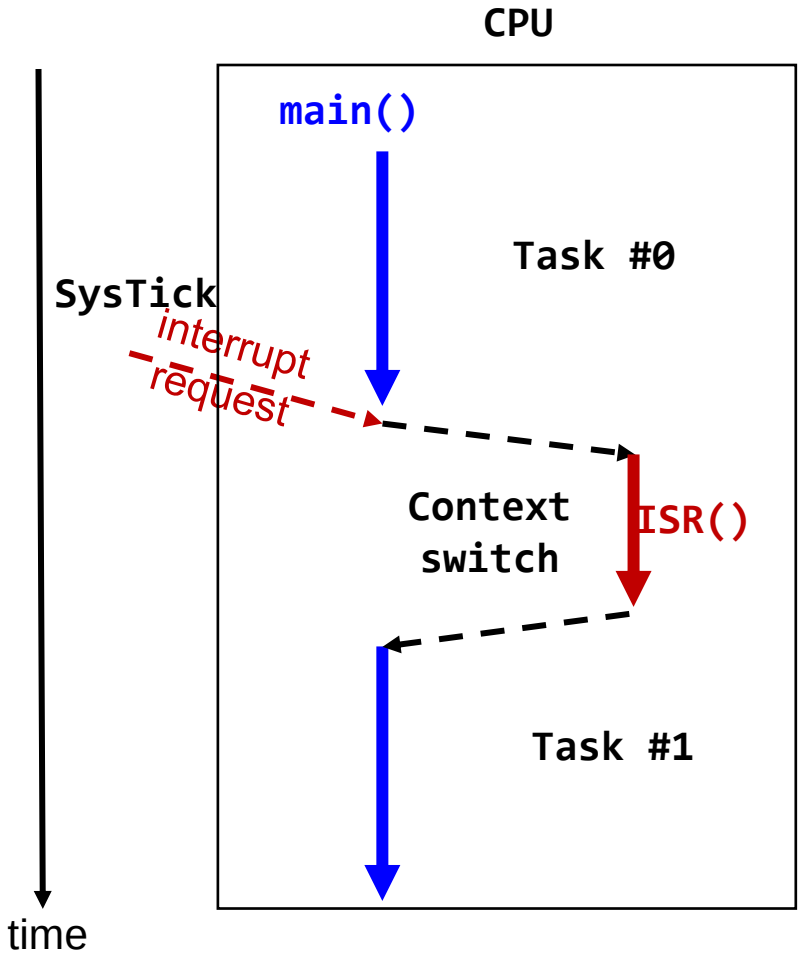
Prekinitve - periodične

Izvedba periodičnih opravil, kot je preklop med neodvisnimi opravili:

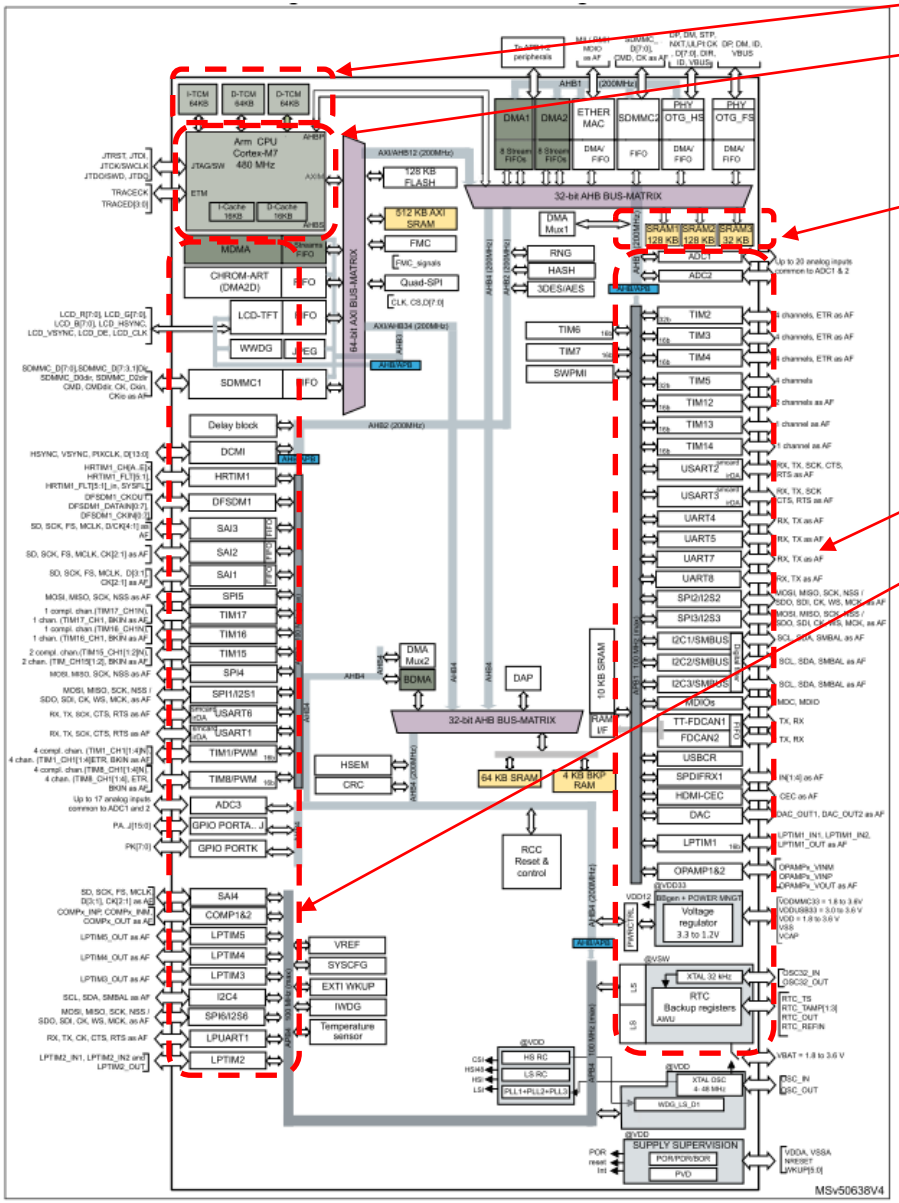
- Shranimo stanje trenutnega opravila
- Vzpostavimo stanje naslednjega opravila
- Izvajamo do naslednje prekinitve

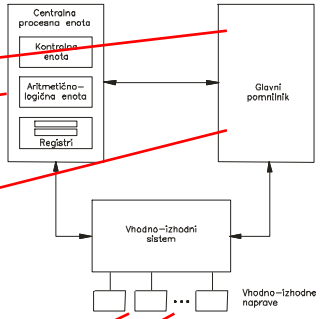
Periodično se prožijo ustrezni Prekinitveno servisni podprogrami – PSP (ang. ISR) – npr. SysTick_Handler





STM32H750XB





Delo na STM32H7 razvojnem sistemu

Priključitev :

- **Mikro USB** priklop na **daljši stranici (nad LCD, srednji !!!)**

Poseben začetni projekt (github) in info za *STM32H7* (e-učilnica):

- ***dodajanje vsebine (Main.s):***

CubeIDEWorkspace - stm32h7-asm/Core/Src/Main.s - STM32CubeIDE

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer

- CubeIDE_Workspace
  - stm32f4-asm-qemu
- Delo
  - ARM9Template
  - stm32f4-asm (in STM32AsmTemplate)
  - ARM9Template.zip
- Node_V4 (in node_v4)
- Sluzba
  - CAN_IEX_Module
  - ORLab-STM32H7
    - stm32h7-asm
      - Binaries
      - Includes
      - Core
        - Src
          - Main.s
        - Startup
          - startup_stm32h750xbhx.s
      - Debug
      - out
      - makefile
      - README.md
      - STM32H750X.svd
      - STM32H750XBHX_FLASH.ld
      - STM32H750XBHX_RAM.ld
    - README.md
  - RALab-STM32H7
    - stm32h7-asm_RA_LED
    - README.md
  - STM32_USB_Key_AdvDebug
  - STM32_USB_Key_FreeRTOS_AdvDebug
  - STM32CubeIDE_Adv_Debug
  - STM32F4_Discovery_VIN_Projects

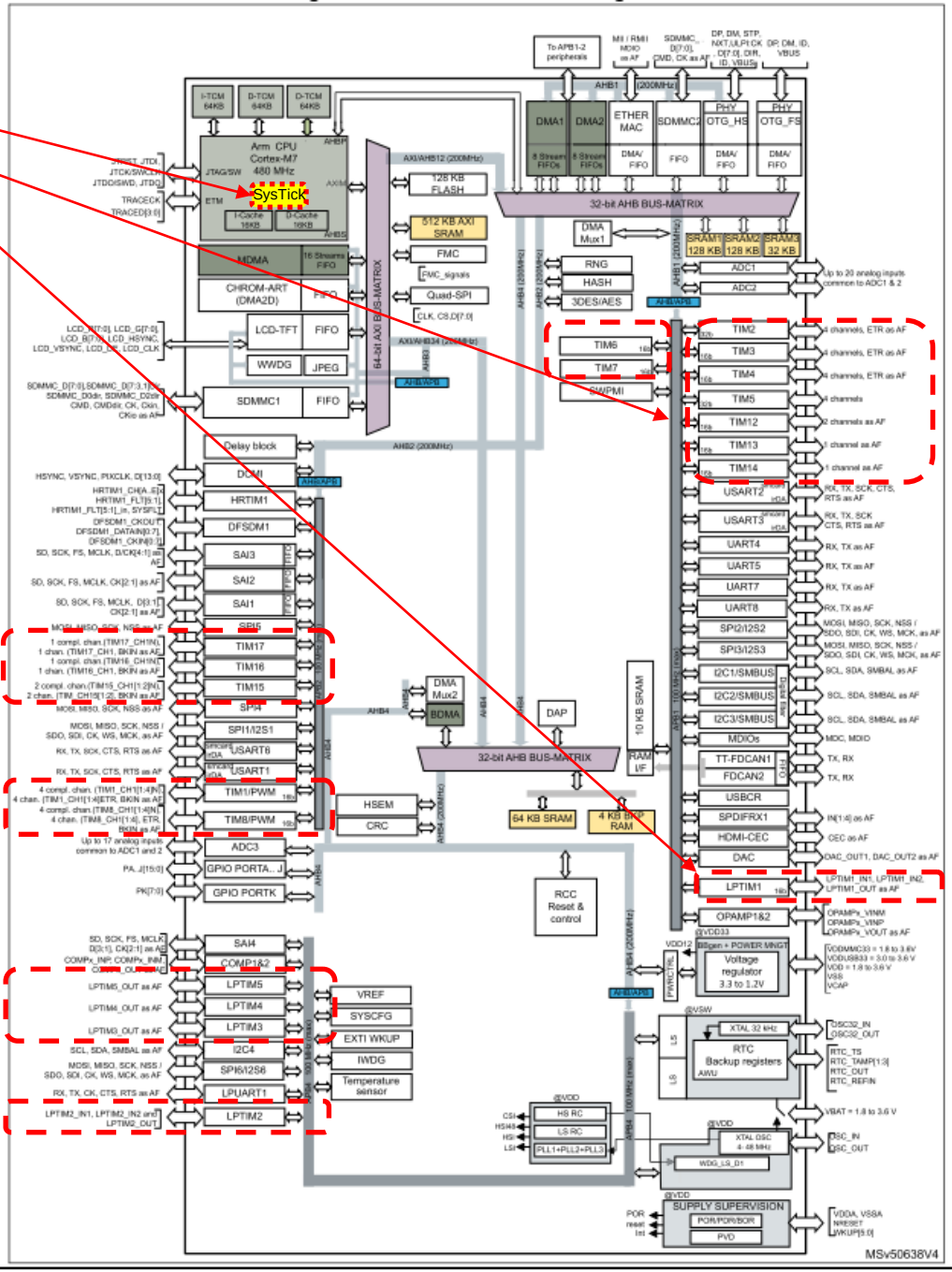
Main.s × startup_stm32h750xbhx.s

```
12
13 ////////////////////////////////////////////////////////////////////////////////
14 // Definitions
15 ////////////////////////////////////////////////////////////////////////////////
16 // Definitions section. Define all the registers and
17 // constants here for code readability.
18
19 // Constants
20
21
22 // Start of data section
23         .data
24
25         .align
26
27 STEV1:  .word   0x10    // 32-bitna spr.
28 STEV2:  .word   0x40    // 32-bitna spr.
29 VSOTA:  .word   0       // 32-bitna spr.
30
31
32 // Start of text section
33         .text
34
35         .type  main, %function
36         .global main
37
38         .align
39 main:
40         ldr r0, =STEV1   // Naslov od STEV1 -> r0
41         ldr r1, [r0]     // Vsebina iz naslova v r0 -> r1
42
43         ldr r0, =STEV2   // Naslov od STEV1 -> r0
44         ldr r2, [r0]     // Vsebina iz naslova v r0 -> r2
45
46         add r3,r1,r2     // r1 + r2 -> r3
47
48         ldr r0, =VSOTA   // Naslov od STEV1 -> r0
49         str r3,[r0]      // iz registra r3 -> na naslov v r0
50
51 __end:  b    __end
52
```

--------- Razvojni sistem STM32H750-DK ----------

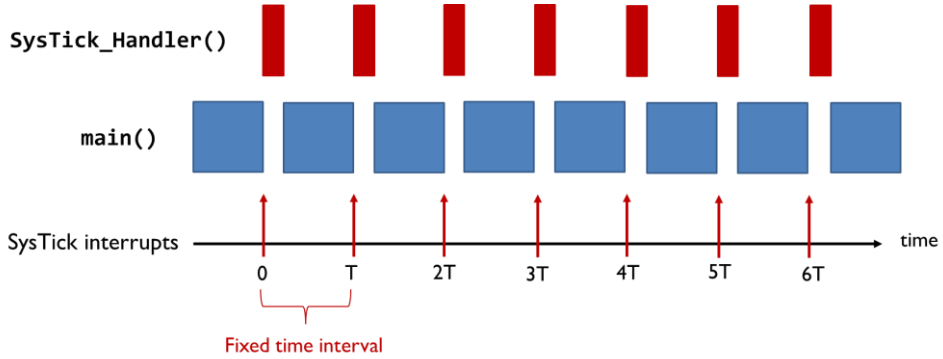
- STM32H750B-DK Discovery kit with STM32H750XB MCU
- ORLab-STM32H7 - GitHub repozitorij
- User Manual Discovery kit stm32h750xb Uploaded 11/11/22, 10.15
- DataSheet_stm32h750xb Uploaded 11/11/22, 10.16
- Reference Manual rm0433-stm32h750xb Uploaded 11/11/22, 10.17
- Programming_Manual_pm0253-stm32h750xb Uploaded 11/11/22, 10.17
- Errata_es0396-stm32h750xb Uploaded 11/11/22, 10.19

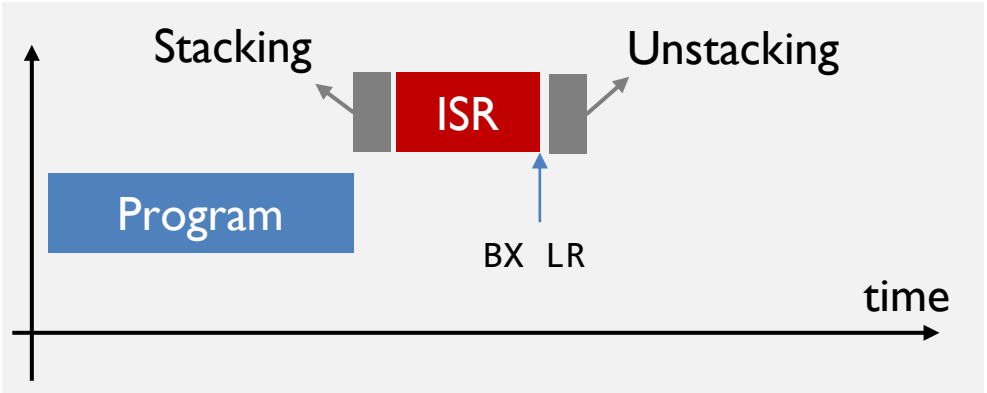
Časovniki

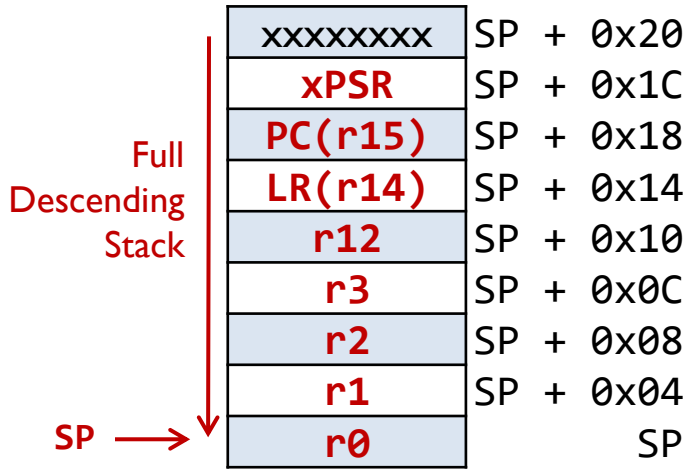

SysTick
Arm CPU Cortex-M7 480 MHz
32-bit AHB BUS-MATRIX
64-bit AXI BUS-MATRIX
TIM1/PWM
TIM8/PWM
TIM2
TIM3
TIM4
TIM5
TIM12
TIM13
TIM14
TIM6
TIM7
TIM15
TIM16
TIM17
LPTIM1
LPTIM2
LPTIM3
LPTIM4
LPTIM5
MSv50638V4

Prekinitve – STM32

Izvedba periodičnih opravil, kot je preklop med neodvisnimi opravili:







Vira: Reference & Programming manuals

life.augmented

PM0253
Programming manual
STM32F7 Series and STM32H7 Series Cortex®-M7 processor programming manual

4 Cortex-M7 peripherals

4.1 About the Cortex-M7 peripherals

The address map of the *Private peripheral bus* (PPB)

4.5 SysTick timer (STK)

0xE000E100-0xE000E4EF	Nested Vectored Interrupt Controller	Table 40 on page 184
0xE000ED00-0xE000ED3F	System control block	Table 50 on page 192
0xE000ED78- 0xE000ED84	Processor features	Table 77 on page 217
0xE000ED90-0xE000EDB8	Memory Protection Unit	Table 84 on page 222
0xE000EF00-0xE000EF03	Nested Vectored Interrupt Controller	Table 40 on page 184
0xE000EF30-0xE000EF44	Floating-Point Unit	Table 94 on page 233
0xE000EF50-0xE000EF78	Cache maintenance operations	Table 100 on page 240
0xE000EF90-0xE000EFA8	Access control	Table 104 on page 245

Table 71. System timer registers summary

Address	Name	Type	Required privilege	Reset value	Description
0xE000E010	SYST_CSR	RW	Privileged	0x00000004	*SysTick control and status register*
0xE000E014	SYST_RVR	RW	Privileged	UNKNOWN	*SysTick reload value register*
0xE000E018	SYST_CVR	RW	Privileged	UNKNOWN	*SysTick current value register*
0xE000E01C	SYST_CALIB	RO	Privileged	0xC0000000	*SysTick calibration value register*

life.augmented

RM0433
Reference manual
STM32H742, STM32H743/753 and STM32H750 Value line advanced Arm®-based 32-bit MCUs

SysTick časovnik + prek. – stanje , nastavitve

Bazni naslov za registre SysTick je 0xE000E010

Ponovitev OR-LAB 9: SysTick+prek.

Figure 40. SysTick SYST_CSR bit assignments

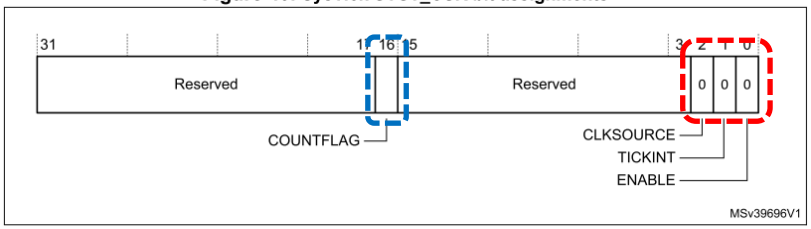


Figure 41. SYST_RVR bit assignments

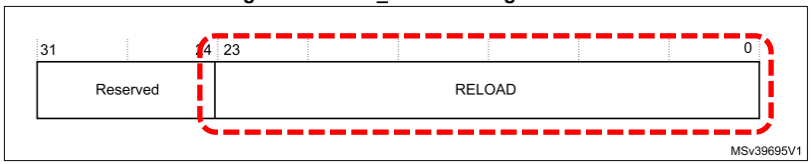
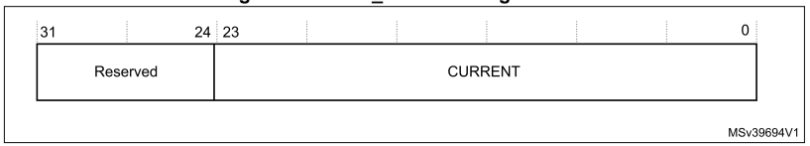


Figure 42. SYST_CVR bit assignments:



Osnovni registri za delovanje SysTick časovnika:

SYST_CSR : vklop časovnika

CLKSOURCE=1, TICKINT=1, ENABLE=1

COUNTERFLAG=1, ko prešteje do 0 (postavi na SYST_RVR in nadaljuje)

SYST_RVR : zač. vrednost štetja (šteje proti 0)

SYST_RVR = število period

SYST_CVR : trenutna vrednost števca

SYST_CVR = nekje med SYST_RVR in 0

SysTick časovnik *(Registri za nastavitve delovanja)*

Ponovitev OR-LAB 9: SysTick+prek.

Figure 40. SysTick SYST_CSR bit assignments

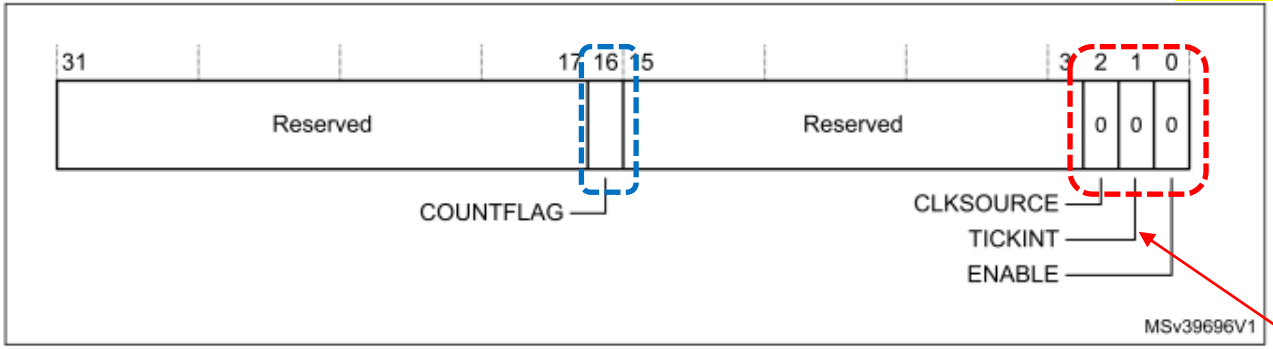


Table 72. SysTick SYST_CSR bit assignments

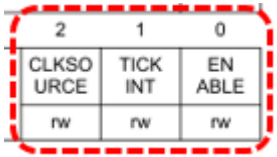
Bits	Name	Function
[31:17]	-	Reserved.
[16]	COUNTFLAG	Returns 1 if timer counted to 0 since last time this was read.
[15:3]	-	Reserved.
[2]	CLKSOURCE	Indicates the clock source: – 0: External clock. – 1: Processor clock.
[1]	TICKINT	Enables SysTick exception request: 0: Counting down to zero does not assert the SysTick exception request. 1: Counting down to zero asserts the SysTick exception request. Software can use COUNTFLAG to determine if SysTick has ever counted to zero.
[0]	ENABLE	Enables the counter: 0: Counter disabled. 1: Counter enabled.

Vklop prekinitve

SysTick Časovnik – krmiljenje

Potrebni koraki za krmiljenje časovnika SysTick:

1. **SYST_RVR (Reload Value Register): Value** SYSTICK_RELOAD_1MS
2. **SYST_CVR (Current Value Register): 0, reset to zero**
3. **SYST_CSR (Control/Status Register): 0b111 : Proc. Clock, TickInt, Enable**
   -> Start SysTick



4. **Delovanje:**
   Proženje SysTick_Handler vsako 1 ms

SysTick_Handler :

```
.global SysTick_Handler

.section  .text.SysTick_Handler,"ax",%progbits
.type  SysTick_Handler, %function


SysTick_Handler:
    push {r3, r4, r5, lr}
    …
RET: pop {r3, r4, r5, pc}
```

Prekinitveni vektorji

```
// Start of text section
.section .text
///////////////////////////////////////////////////////////////////////////////
// Vectors
///////////////////////////////////////////////////////////////////////////////
// Vector table start
// Add all other processor specific exceptions/interrupts in order here
.long    __StackTop                  // Top of the stack. from linker script
.long    _start +1                   // reset location, +1 for thumb mode
  .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

  /* 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 */
```

```
.global SysTick_Handler
.section .text.SysTick_Handler,"ax",%progbits
.type SysTick_Handler, %function


SysTick_Handler:

            push {r3, r4, r5, r6, lr}
            …
```

Prekinitve *(Register za nastavitve vektorske tabele v RAM pomnilniku)*

4.3.4 Vector table offset register

The VTOR indicates the offset of the vector table base address from memory address 0x00000000. See the register summary in *Table 50 on page 192* for its attributes. The bit assignments are:

Figure 27. VTOR bit assignments

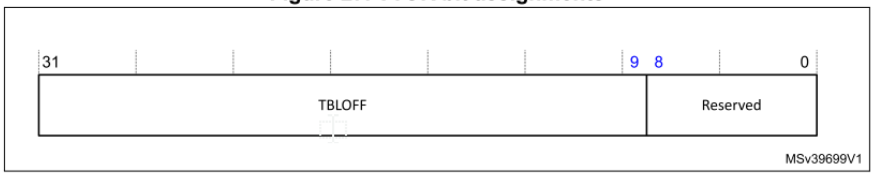


Table 54. VTOR bit assignments

Bits	Name	Function
[31:9]	TBLOFF	Vector table base offset field. It contains bits [29:7] of the offset of the table base from the bottom of the memory map.
[8:0]	-	Reserved.

When setting TBLOFF, the user must align the offset to the number of exception entries in the vector table.

The table alignment requirements mean that bits [8:0] of the table offset are always zero.

4.3 System control block

The *System Control Block* (SCB) provides system implementation information, and system control. This includes configuration, control, and reporting of the system exceptions. The system control block registers are:

Table 50. Summary of the system control block registers

Address	Name	Type	Required privilege	Reset value	Description
0xE000E008	ACTLR	RW	Privileged	0x00000000	*Auxiliary control register on page 193*
0xE000ED00	CPUID	RO	Privileged	0x410FC270	*CPUID base register on page 194*
0xE000ED04	ICSR	RW(1)	Privileged	0x00000000	*Interrupt control and state register on page 194*
0xE000ED08	VTOR	RW	Privileged	Unknown	*Vector table offset register on page 197*
0xE000ED0C	AIRCR	RW(1)	Privileged	0xFA050000	*Application interrupt and reset control register on page 197*

Ponovitev OR-LAB 9: SysTick+prek.

```
.equ VTOR,0xE000ED08

RELLOC_VECTBL:
push {r0, r1, lr}
ldr r1, =VTOR // Set Vector table addr. to 0x24000000
ldr r0, =0x24000000
str r0, [r1]
pop {r0, r1, pc}
```

Dodamo samo, če poganjamo kodo iz RAM pomnilnika !!!
Sicer je naslov vektorske tabele 0x08000000 (Flash)

Register	Address	Value
Control		
ACTLR	0xe000e008	0x1000
ICSR	0xe000ed04	0x0
VTOR	0xe000ed08	0x8000000
TBLOFF	[7:25]	0x100000
TBLBASE	[29:1]	0x0

Register	Address	Value
Control		
ACTLR	0xe000e008	0x1000
ICSR	0xe000ed04	0x0
VTOR	0xe000ed08	0x24000000
TBLOFF	[7:25]	0x480000
TBLBASE	[29:1]	0x1

SysTick Časovnik s prekinitvami – Mini RTOS

Začetna koda:

```
main:
#ifdef RAM_LinkScript
// Relocating Vector table to RAM (only for RAM Linker Script)
   bl RELLOC_VECTBL
#endif

   bl INIT_GPIOs
   bl INIT_USART3 // Priprava USART3 naprave
   bl ITM_Init

// MiniRTOS_Init
// Set new User SP that will be used for user program (MSP is reset default)
   ldr r0,=USERSTACK_End-4
   msr psp, r0

// Set PSP as SP when stacking/unstacking user processes
// Access special registers
   mrs r0, CONTROL // Read CONTROL into R0
   orr r0, R0, #0x2 // Set SPSEL (PSP to be used in Thread mode) to select PSP
   msr CONTROL, r0 // Write R0 into CONTROL
   isb // Instruction Synchronization Barrier - needed

   CPSID I // Disable interrupts

   bl INIT_TC_PSP // Initialize SysTick periodic interrupt (every msec)

CPSIE I // Enable interrupts (context switches will happen)

b TASK0_Start // Go into first task (it will switch with TASK1 every msec)

__end: b __end
```

SysTick MiniRTOS – PSP -> Context Switch

SysTick_Handler :

```
SysTick_Handler:
/// STEP 1 - SAVE THE CURRENT TASK CONTEXT
/// Disable interrupts
CPSID I

/// Push remaining registers R4 to R11
// push {r4-r11}
mrs r3, psp // move PSP to r3
stmfd r3!, {r4-r11} // Push onto a FD Stack

ldr r0,=ACT_TASKID // Check current Active task ID
ldr r1,[r0]
cmp r1,#0
bne CONT1

// TASK 0 current
mov r2,#1 // Task1 is new
str r2,[r0]

ldr r0,=StackPointer0 // Store current PSP
str r3,[r0]

ldr r0,=PCB1
ldr r1,=StackPointer1
b STEP2

…
```

```
…

// TASK 1 current
CONT1:
mov r2,#0 // Task0 is new
str r2,[r0]

ldr r0,=StackPointer1 // Store current PSP
str r3,[r0]

ldr r0,=PCB0
ldr r1,=StackPointer0

/// STEP 2: LOAD THE NEW PSP and TASK CONTEXT FROM ITS
STACK TO THE CPU REGISTERS.
STEP2:
ldr r3,[r1] // Read new PSP into r3

/// Pop registers R4-R11
ldmfd r3!, {r4-r11} // Load from a FD Stack

msr psp, r3 // move r3 to PSP for upcoming task

/// Enable interrupts
CPSIE I

BX LR
```

SysTick MiniRTOS – Task0, Task1

```
TASK0_Start:

      bl LED1_ON
      mov r0,#0
      mov r1,#'1'
      bl ITM_Send

      mov r0,#'0'
      bl SEND_UART

      mov r0,#500
      bl DELAY // Zakasnitev SW Delay: r0 x 1msec

      bl LED1_OFF
      mov r0,#0
      mov r1,#'0'
      bl ITM_Send

      mov r0,#500
      bl DELAY // Zakasnitev SW Delay: r0 x 1msec

      b TASK0_Start
```

```
TASK1_Start:

      bl LED2_OFF
      mov r0,#1
      mov r1,#'+'
      bl ITM_Send

      mov r0,#'1'
      bl SEND_UART

      mov r0,#500
      bl DELAY // Zakasnitev SW Delay: r0 x 1msec

      bl LED2_ON
      mov r0,#1
      mov r1,#'-'
      bl ITM_Send

      mov r0,#500
      bl DELAY // Zakasnitev SW Delay: r0 x 1msec

      b TASK1_Start
```

```
.equ STACKSIZE , 240 // + 16 registers of PCB

USERSTACK_Start: .space (STACKSIZE+16) * 4
USERSTACK_End:

StackPointer0: .word PCB0 // PSP for Task0
StackPointer1: .word PCB1 // PSP for Task1

ACT_TASKID: .word 0 // current process ID (0,1,...)
```

```
STACK1_Start: .space STACKSIZE * 4
// PCB: ISR saves: R4-R11 (8regs), CPU saves: R0-R3, R12, LR, PC, PSR
PCB1: .word 4,5,6,7,8,9,10,11, 0,1,2,3, 12, 0xFFFFFFFF, TASK1_Start, 0x01000000
STACK1_End: // was PSP Thumb mode
```

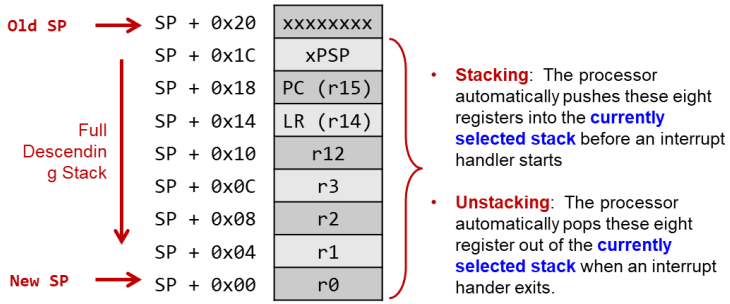
SysTick MiniRTOS – Task0, Task1

```
TASK0_Start:

    …

    b TASK0_Start
```

```
TASK1_Start:

    …

    b TASK1_Start
```

```
.equ STACKSIZE , 240 // + 16 registers of PCB

USERSTACK_Start: .space (STACKSIZE+16) * 4
USERSTACK_End:

StackPointer0: .word PCB0 // PSP for Task0

ACT_TASKID: .word 0 // current process ID (0,1,...)
```

```
STACK1_Start: .space STACKSIZE * 4
// PCB: ISR saves: R4-R11 (8regs), CPU saves: R0-R3,
R12, LR, PC, PSR
PCB1: .word 4,5,6,7,8,9,10,11, 0,1,2,3, 12,
0xFFFFFFFF, TASK1_Start, 0x01000000
STACK1_End: // was PSP Thumb mode

StackPointer1: .word PCB1 // PSP for Task1
```

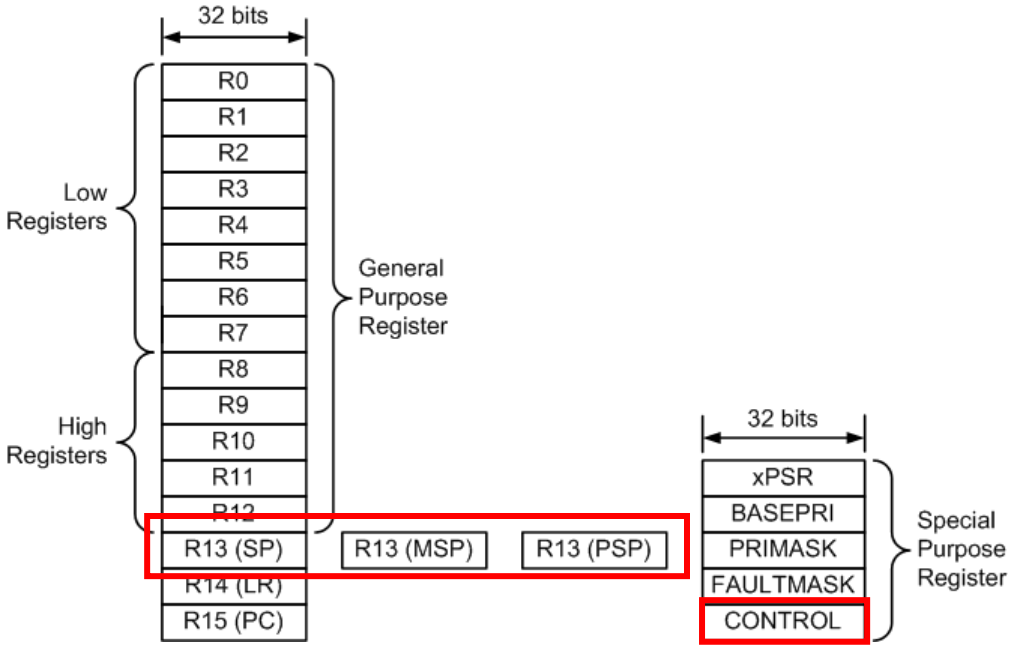


r4-r11: SysTick Handler dodatno shrani na sklad

```
mrs r3, psp // move PSP to r3
stmfd r3!, {r4-r11} // Push onto a FD Stack
```

Kazalca na sklad sta shranjena posebej:
StackPointer0, StackPointer1

Stack Pointer: MSP vs PSP



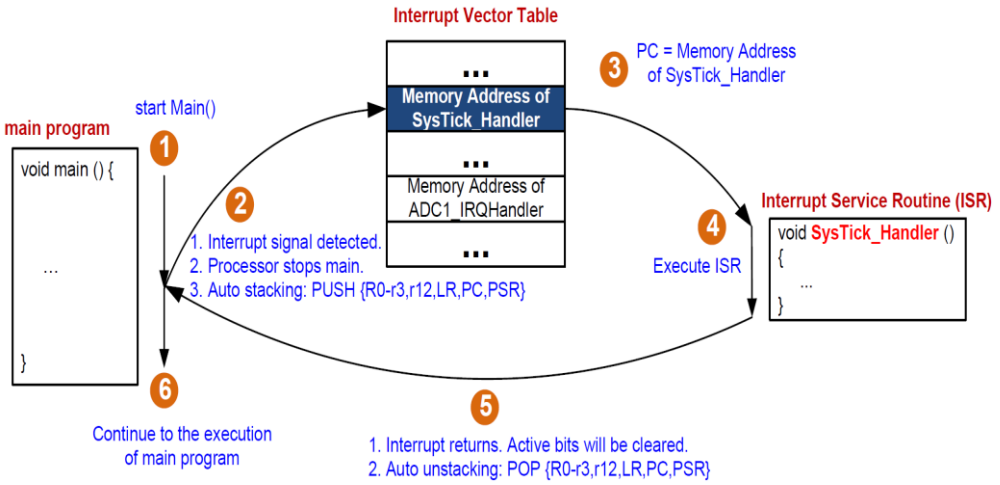
- **MSP**: Main Stack Pointer (selected at reset)
- **PSP**: Process Stack Pointer
- In assembly, both `MSP` and `PSP` are called `R13` or `SP`

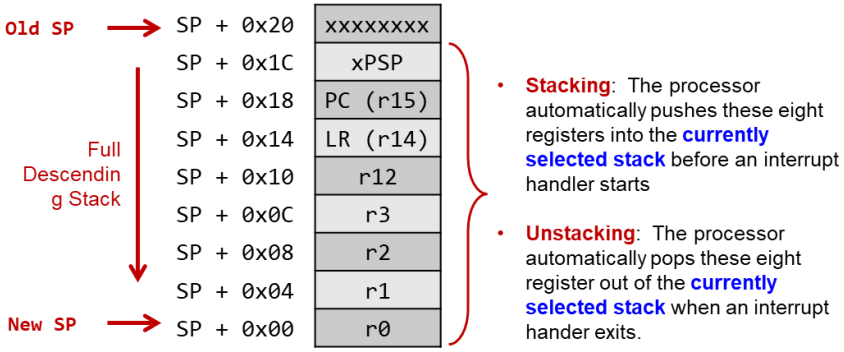
For interrupts, which stack does auto stacking/unstacking use?

Depends on

- processor mode: thread vs handler
- setting in the control register

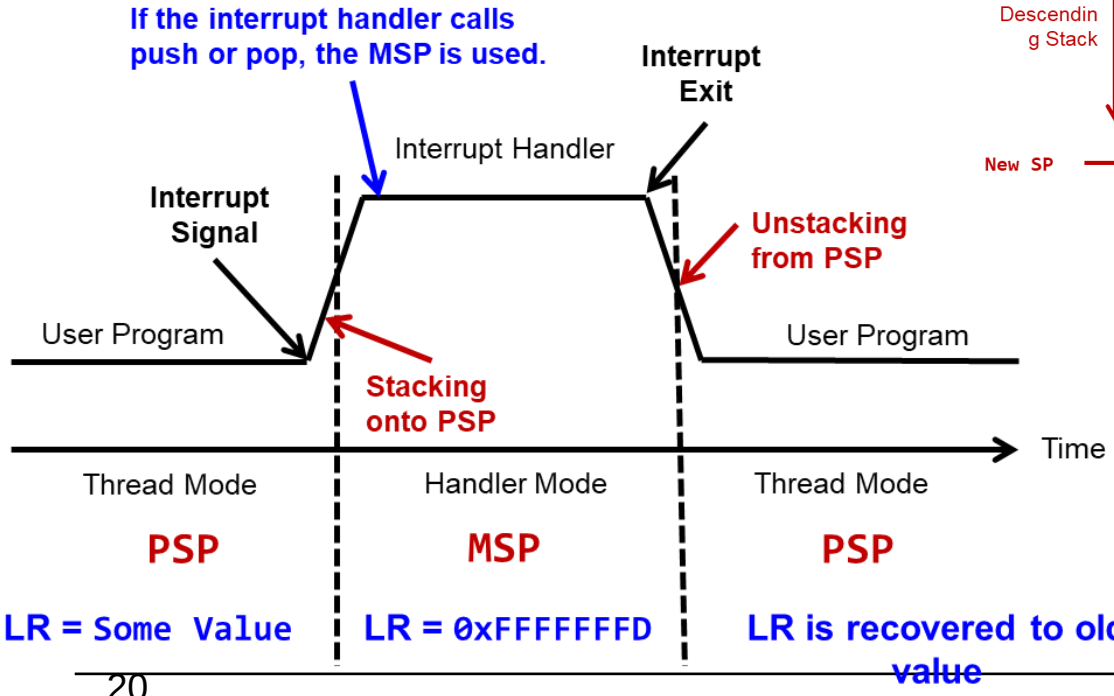
SysTick Interrupt



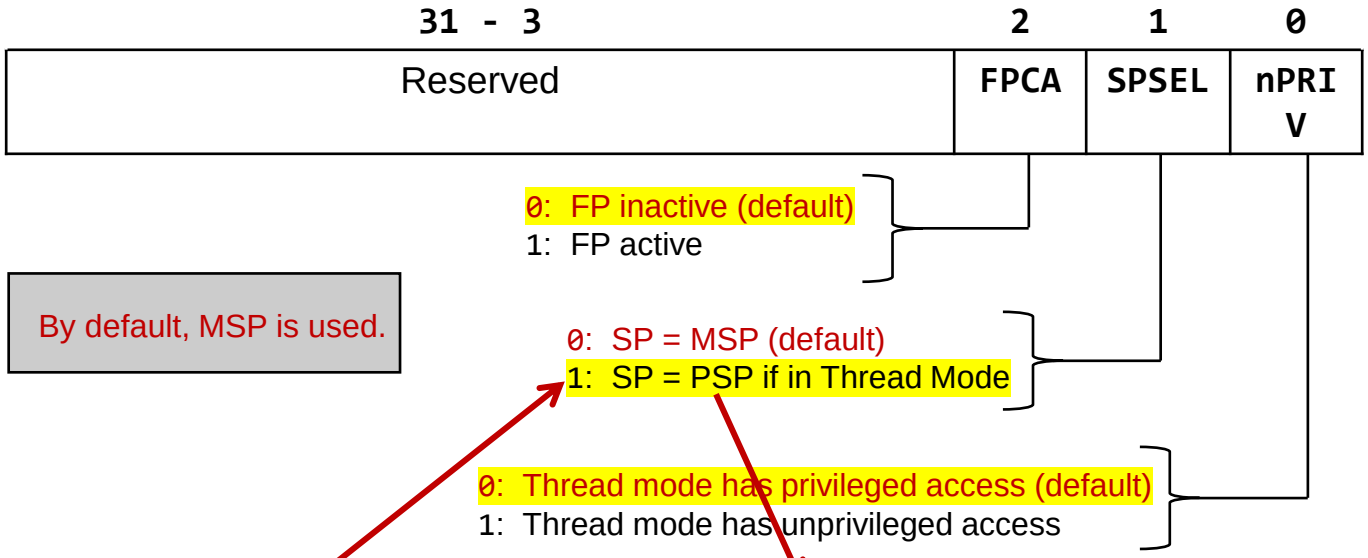


- **Stacking**: The processor automatically pushes these eight registers into the **currently selected stack** before an interrupt handler starts
- **Unstacking**: The processor automatically pops these eight register out of the **currently selected stack** when an interrupt hander exits.

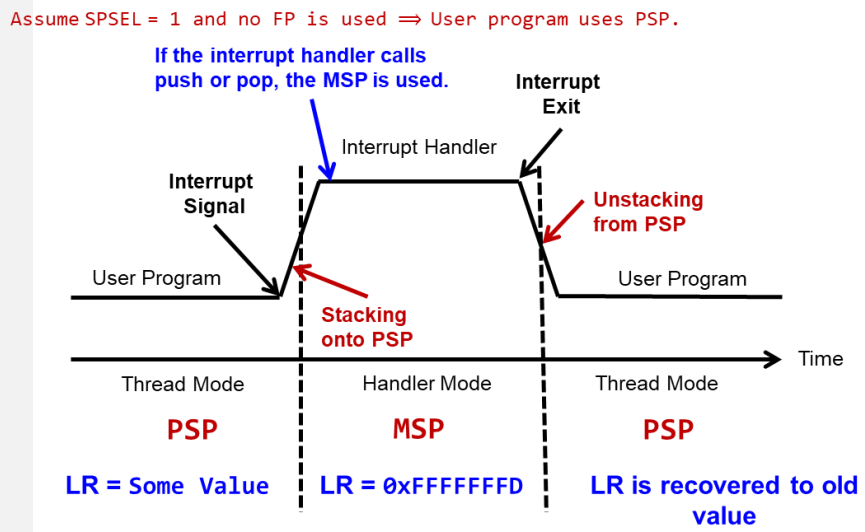
Assume SPSEL = 1 and no FP is used ⟹ User program uses PSP.



Control Register




```
// MiniRTOS_Init


// Set new User SP that will be used for user program (MSP
is reset default)
ldr r0,=USERSTACK_End-4
msr psp, r0


// Set PSP as SP when stacking/unstacking user processes
// Access special registers
mrs r0, CONTROL // Read CONTROL into R0
orr r0, R0, #0x2 // Set SPSEL (PSP to be used in Thread
mode) to select PSP
msr CONTROL, r0 // Write R0 into CONTROL
isb // Instruction Synchronization Barrier - needed
```



CONTROL register

2.1.1 Processor mode and privilege levels for software execution

The processor *modes* are:

Thread mode	Executes application software. The processor enters Thread mode when it comes out of reset.
Handler mode	Handles exceptions. The processor returns to Thread mode when it has finished all exception processing.

CONTROL register

The CONTROL register controls the stack used and the privilege level for software execution when the processor is in Thread mode and indicates whether the FPU state is active. See the register summary in *Table 10 on page 27* for its attributes. The bit assignments are:

Figure 7. Control bit assignments

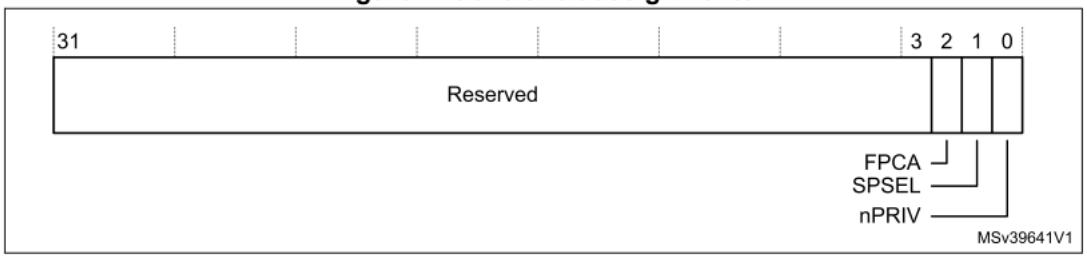


Table 10. Control register bit assignments (continued)

Bits	Name	Function
[1]	SPSEL	Defines the currently active stack pointer: 0 = MSP is the current stack pointer. 1 = PSP is the current stack pointer. In Handler mode this bit reads as zero and ignores writes. The Cortex®-M7 processor updates this bit automatically on exception return.
[0]	nPRIV	Defines the Thread mode privilege level: 0 = Privileged. 1 = Unprivileged.

ITM Trace - Init

ITM trace enable register (M7_ITM_TER)

Address offset: 0xE00

Reset value: 0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
STIMENA[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STIMENA[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **STIMENA[31:0]**: Stimulus port enable

Each bit n (0:31) enables the stimulus port associated with the M7_ITM_STIMn register.

0: Port disabled

1: Port enabled

ITM trace control register (M7_ITM_TCR)

Address offset: 0xE80

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	BUSY	TRACEBUSID[6:0]						
								rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	GTSFREQ[1:0]		TSPRESCALE [1:0]		Res.	Res.	Res.	SWOE NA	TXENA	SYNC ENA	TSENA	ITM ENA
				rw	rw	rw					r	rw	rw	rw	rw

Bit 0 **ITMENA**: ITM enable

0: Disabled

1: Enabled

ITM stimulus register x (M7_ITM_STIMx)

Address offset: 0x000 + x * 0x4 (x = 0 to 31)

Reset value: Undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
STIMULUS[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STIMULUS[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **STIMULUS[31:0]**: Software event packet / FIFOREADY

Write data is output on the trace bus as a software event packet. When reading, bit 0 is a FIFOREADY indicator:

0: Stimulus port buffer is full (or port is disabled)

1: Stimulus port can accept new write data

```
ITM_Init:
push {r0, r1, lr}

ldr r0, =M7_ITM_BASE // Base address
mov r1, #0b11
str r1,[r0,#M7_ITM_TER]

mov r1, #1
str r1,[r0,#M7_ITM_TCR]

pop {r0, r1, pc}
```

ITM Trace - Send

```
ITM_Send:
  push {r2, lr}

  ldr r2, =M7_ITM_BASE // Base address

  cmp r0,#0 // Channel 0 ?
  bne Ch1

  str r1,[r2] // Channel 0
  b contx

Ch1:
  str r1,[r2,#4] // Channel 1

contx:
  pop {r2, pc}
```

ITM stimulus register x (M7_ITM_STIMx)

Address offset: 0x000 + x * 0x4 (x = 0 to 31)

Reset value: Undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
STIMULUS[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STIMULUS[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **STIMULUS[31:0]**: Software event packet / FIFOREADY
Write data is output on the trace bus as a software event packet. When reading, bit 0 is a FIFOREADY indicator:

0: Stimulus port buffer is full (or port is disabled)
1: Stimulus port can accept new write data

ITM Registers

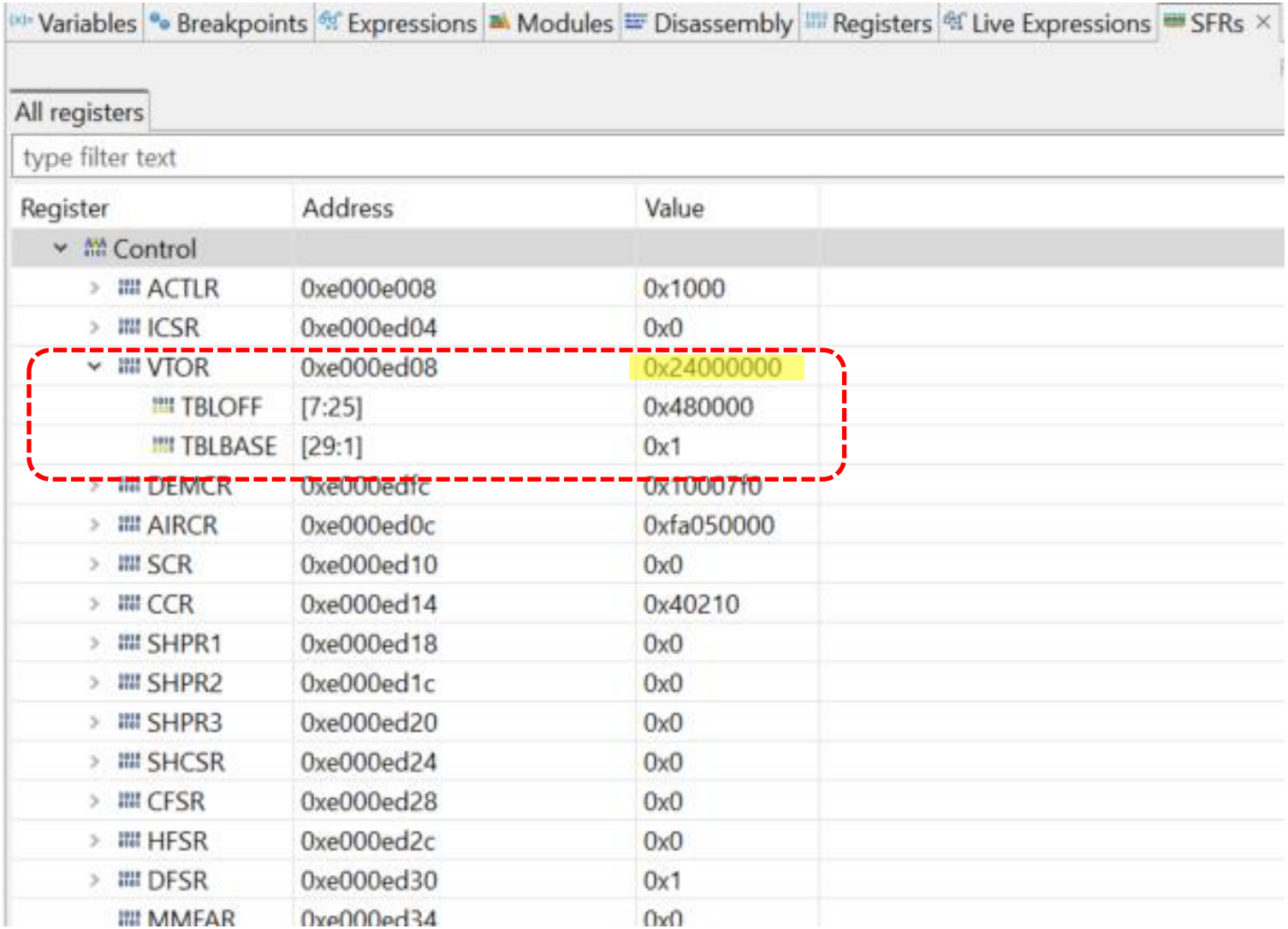
Cortex-M7 ITM register map and reset values

The ITM registers are located at address range 0xE0000000 to 0xE0000FFC, on the AHBD.

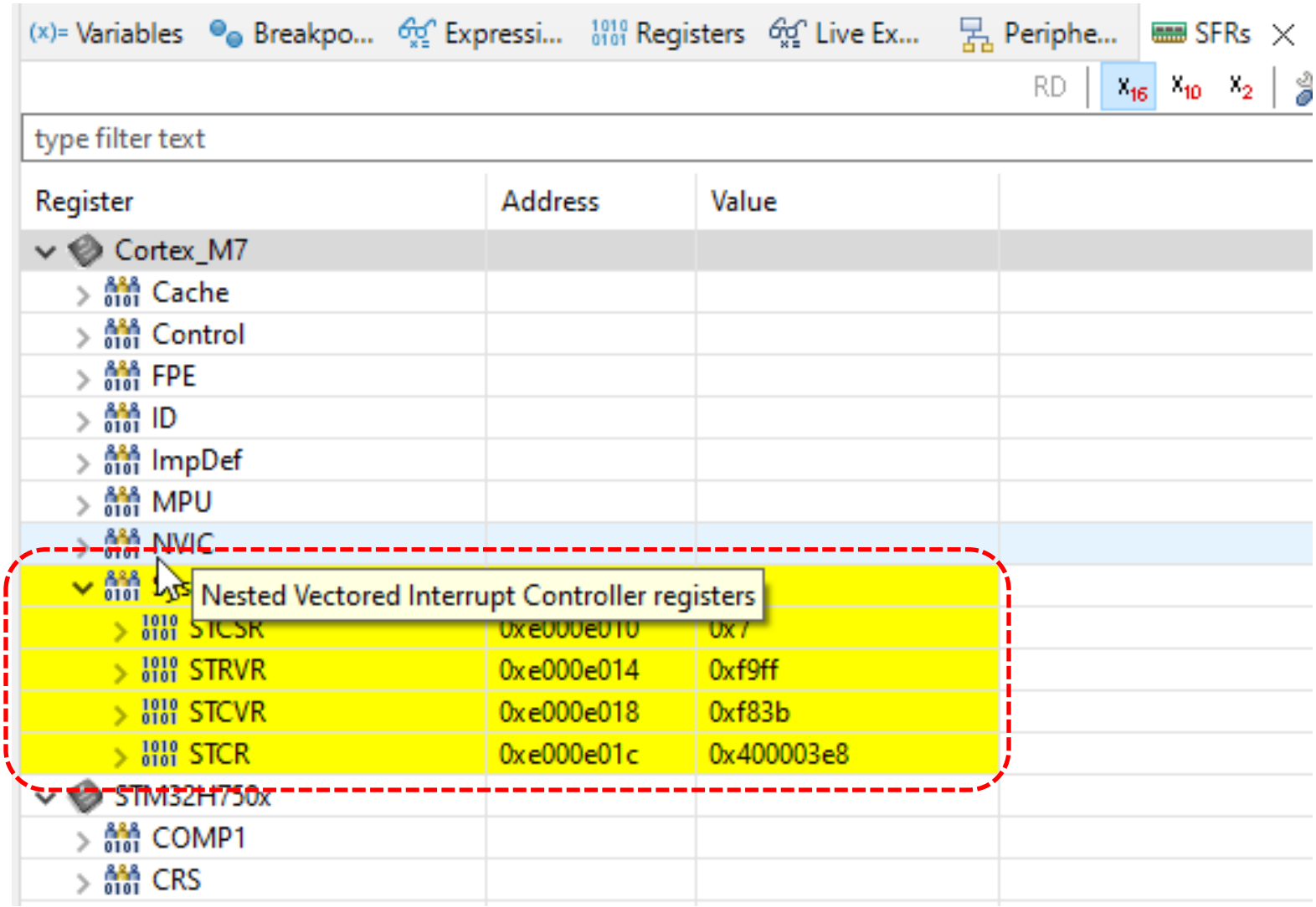
Table 609. Cortex-M7 ITM register map and reset values

Offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x000 to 0x07C	M7_ITM_STIM0-31	STIMULUS[31:0]																															
	Reset value	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
0x080 to 0xDFC	Reserved	Reserved																															
0xE00	M7_ITM_TER	STIMENA[31:0]																															
	Reset value	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
0xE04 to 0xE3C	Reserved	Reserved																															
0x0E40	M7_ITM_TPR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	PRIVMASK[3:0]			
	Reset value																													0	0	0	0
0xE44 to 0xE7C	Reserved	Reserved																															
0xE80	M7_ITM_TCR	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	BUSY	TRACEBUSID[6:0]							Res.	Res.	Res.	Res.	GTSFREQ[1:0]		TSPRESCALE[1:0]		Res.	Res.	Res.	SWOENA	TXENA	SYNCENA	TSENA	ITMENA
	Reset value									0	0	0	0	0	0	0	0					0	0	0	0				0	0	0	0	0

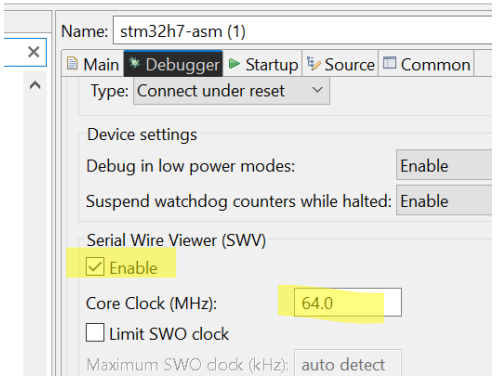
CubeIDE – SFR okno

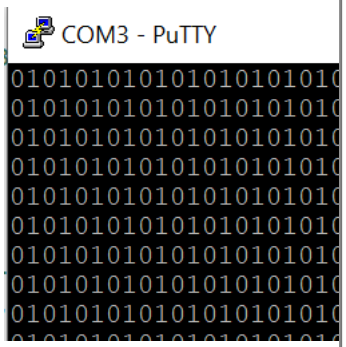


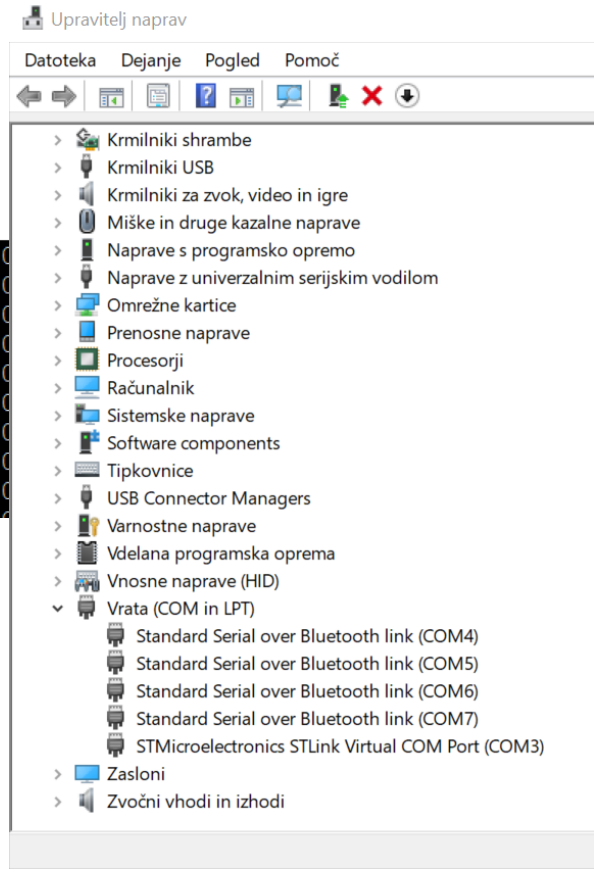
CubeIDE – SFR okno

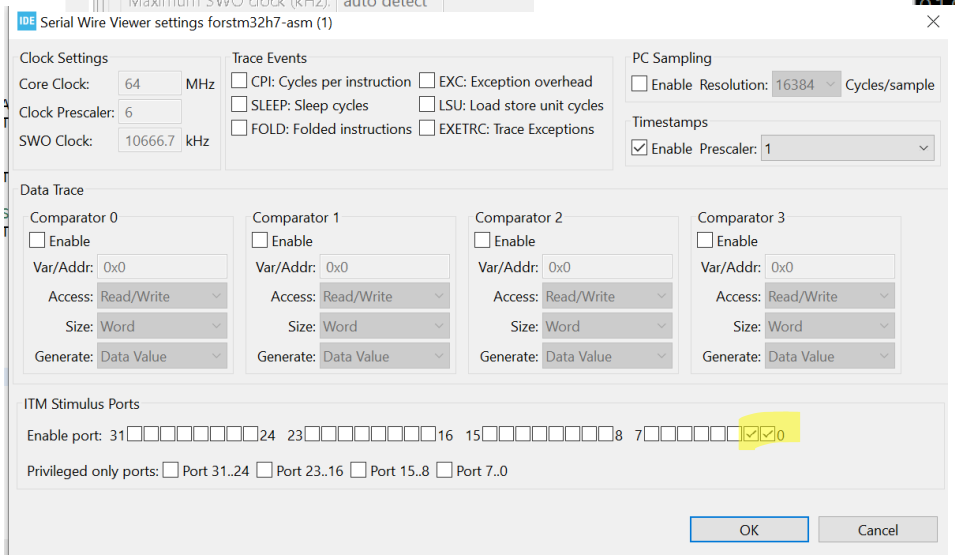


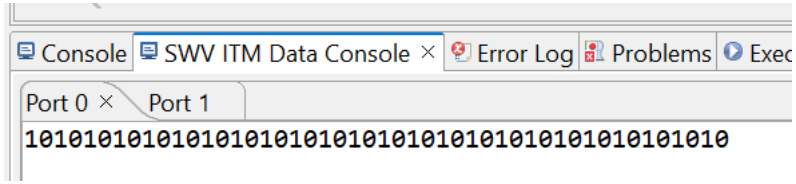
CubeIDE – Sledenje delovanja

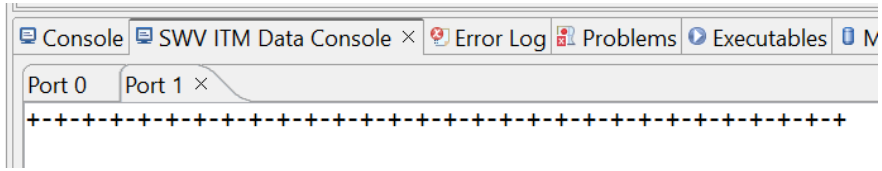




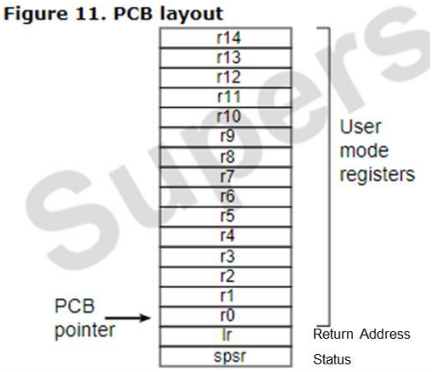








Prekinitve – Izvedba več procesov



start.s:

Dodana koda :

Initialization: …

```
        adr r12,PCB0+8      @ first process is PCB0
```

R12 = PCB pointer

Variables: …

PCB0, PCB1

```
ACT_PROC: .word 0           @ current process ID
PCB0:     .word 0, 0,               0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0  @ SPSR,lr, r0-11
PCB1:     .word 0b10000, PROC_1,    0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0,0, 0  @ SPSR,lr, r0-11
CNT_PROC: .word 0           @ counter for process 1
```

SPSR ..SR procesa
lr .. PC procesa

Process ID=1: …

```
PROC_1:     ldr r0, CNT_PROC
            add r0, r0, #1
            str r0, CNT_PROC
            b PROC_1
```

Dodatni proces ID=1:
- Povečevanje CNT_PROC

(vzel bo pribl. polovico CPE časa)

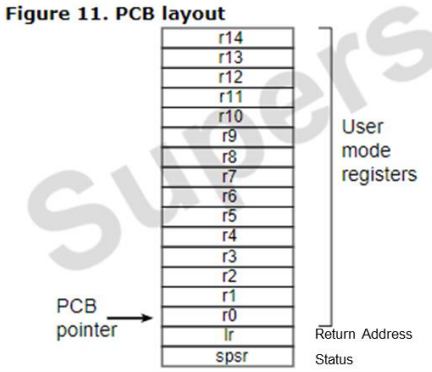
`0b10000` .. Zač. SR procesa (UserMode)

`PROC_1` .. Zač. naslov procesa

Osnovni proces ID=0,
enak kot prej:
- Utripanje LED diode

(vzel bo pribl. polovico CPE časa)

Prekinitve – Izvedba več procesov



start.s:

```
PSP: …
IRQ17_Hand:
 sub lr,lr,#4
  stm     r12,{r0-r11}^        @ ^ -> Dump user mode registers above R13
  mrs     r0, SPSR             @ Pick up the user status register  (saved into SPSR)
  stmdb   r12, {r0, lr}        @ and dump with return address below sp.

          /* servisiraj zahtevo ... */

IRQ17_Exit:

  ldr  r12,ACT_PROC            @ change ID to next process
  eors r12, r12, #1
  str r12,ACT_PROC

  adreq r12,PCB0+8             @ New proc 0: set r12 to PCB0
  adrne r12,PCB1+8             @ New proc 1: set r12 to PCB1

  ldmdb r12, {r0, lr}          @ Pick up status and return address.
  msr   SPSR_cxsf, r0          @ Restore the status.
  ldm   r12, {r0 - r11}^       @ Get the rest of the user registers,
                                    ^ brez pc pomeni user registre

  stmfd r13!, {lr}             /* na sklad shrani povratni naslov.    */
  ldmfd r13!, {pc}^            /* vrni se na naslednji proces - prekinjeni program,
                                   ^ s pc -> pred tem obnovi CPSR iz SPSR_irq*/
                                      pred tem obnovi CPSR iz SPSR_irq*/
```

Shrani registre prekinjenega procesa

PCB0, PCB1

ID: 0 ali 1

R12 -> novi PCB

Povrni registre novega procesa

Vrnitev