

PRIMENA MIKROKONTROLERA- MS1PMK

3. deo

2017

Nenad Jovičić

Modovi adresiranja

- Neposredno adresiranje – sastavni deo instrukcije je podatak, tj. konstanta.
- Registarasko adresiranje – podatak je u nekom od registara opšte namene.
- Indirektno registarsko adresiranje – podatak je u memoriji a adresa se nalazi ili proračunava iz odgovarajućeg registra opšte namene.

Load /store arhitektura

- Cortex-M3 je Load/Store arhitektura, što znači da se memoriji pristupa samo preko Load/Store instrukcija, a sve druge operacije se obavljaju samo nad registrima.

LDR	Load Word	STR	Store Word
LDRH	Load Half Word	STRH	Store Half Word
LDRSH	Load Signed Half Word	STRSH	Store Signed Half Word
LDRB	Load Byte	STRB	Store Byte
LDRSB	Load Signed Byte	STRSB	Store Signed Byte

- Nema neposrednog adresiranja memorijske lokacije.

LDRx/STRx instrukcije

- Pristup memoriji je moguć praktično jedino preko ovih instrukcija.
- Sve kasnije instrukcije za obradu podataka rade sa podacima koji su sastavni deo instrukcije (konstante) i podacima koji su već u registrima.

Syntax

op{*type*}{*cond*} *Rt*, [*Rn*, *Rm* {, LSL #*n*}]

where:

<i>op</i>	Is one of: LDR Load Register. STR Store Register.
<i>type</i>	Is one of: B unsigned byte, zero extend to 32 bits on loads. SB signed byte, sign extend to 32 bits (LDR only). H unsigned halfword, zero extend to 32 bits on loads. SH signed halfword, sign extend to 32 bits (LDR only). - omit, for word.
<i>cond</i>	Is an optional condition code
<i>Rt</i>	Specifies the register to load or store.
<i>Rn</i>	Specifies the register on which the memory address is based.
<i>Rm</i>	Specifies the register containing a value to be used as the offset.
LSL # <i>n</i>	Is an optional shift, with <i>n</i> in the range 0 to 3.

Primeri:

```
STR    R0, [R5, R1]      ; Store value of R0 into an address equal to  
                        ; sum of R5 and R1  
LDRSB  R0, [R5, R1, LSL #1] ; Read byte value from an address equal to  
                        ; sum of R5 and two times R1, sign extended it  
                        ; to a word value and put it in R0  
STR    R0, [R1, R2, LSL #2] ; Stores R0 to an address equal to sum of R1  
                        ; and four times R2.
```

LDRx/STRx instrukcije

- Adresa podatka u memoriji se dobija kao kombinacija sadržaja baznog registra i odgovarajućeg offset-a
- Vrednost offset-a se određuje na sledeće načine:
 - Neposredno:
 - Offset je zapisan neposredno u samoj instrukciji. **#10**
 - Registarsko:
 - Offset se nalazi u offset registru **<Rm>**
 - Skalirano registarsko:
 - Offset se dobija tako što se sadržaj offset registra šiftuje za vrednost konstante zapisane u instrukciji **<Rm>, LSL #<shift>**

LDRx/STRx instrukcije

- Neki primeri adresiranja:
 - Offset (klasični offset):
 - Prosto dodavanje konstante na bazni registar
 - [**<Rn>**, **<offset>**]

```
LDRB R0, [R1, #0x3] ; Read a byte value from address R1+0x3, and store the read data in R0.
```

- Offset sa ažuriranjem baznog registra nakon indeksiranja:
 - Adresa se dobija sabiranjem sadržaja baznog registra i offset-a, ali se nakon izvršene operacije sadržaj baznog registra ažurira na novu vrednost
 - [**<Rn>**, **<offset>**]!

```
LDR R0, [R1, #0x8]! ; After the access to memory[R1+0x8], R1 is updated to R1+0x8
```

LDRx/STRx instrukcije

– Pre-index (klasični registarski offset):

- Offset je u registru
- [**<Rn>**, **<offset>**]

```
LDR R3, [R0, R2, LSL #2] ; Read memory[R0+(R2 << 2)] into R3  
STR R5, [R0,R7] ; Write R5 into memory[R0+R7]
```

– Post-indexno:

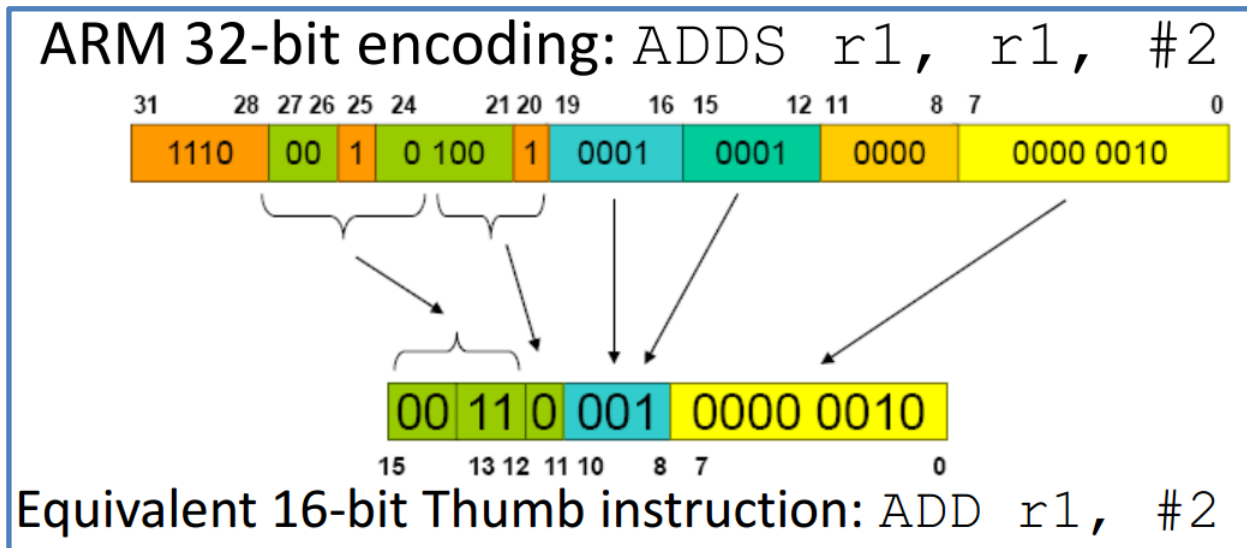
- Adresa se dobija samo na osnovu baznog registra, ali se nakon instrukcije na sadržaj baznog registra dodaje sadržaj izračunatog offset-a.
- [**<Rn>**], **<offset>**

```
LDR R0, [R1], #offset ; Read memory[R1], then R1 updated to R1+offset
```

Konstante

Constant can be any:

- that can be produced by shifting an 8-bit value left by any number of bits within a 32-bit word
- Any constant of the form 0x00XY00XY
- Any constant of the form 0xXY00XY00
- Any constant of the form 0xXYXYXYXY.
- Any 16-bit value



Kako upisati 32-bitnu konstantu u registar?

```
MOVW.W R0,#0x789A ; Set R0 lower half to 0x789A
MOVT.W R0,#0x3456 ; Set R0 upper half to 0x3456. Now
                  ; R0=0x3456789A
```

```
MOVW<c> <Rd>,#<imm16>
```

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	i	1	0	0	1	0	0	imm4				0	imm3			Rd				imm8							

```
d = UInt(Rd); setflags = FALSE; imm32 = ZeroExtend(imm4:i:imm3:imm8, 32);
```

```
MOVT<c> <Rd>,#<imm16>
```

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	i	1	0	1	1	0	0	imm4				0	imm3			Rd				imm8							

```
d = UInt(Rd); imm16 = imm4:i:imm3:imm8;
if d IN {13,15} then UNPREDICTABLE;
```

Još bolji metod!

- Još bolji način je asemblerska sintaksa:

```
LDR r0,=0x55555555
```

- koja generiše:

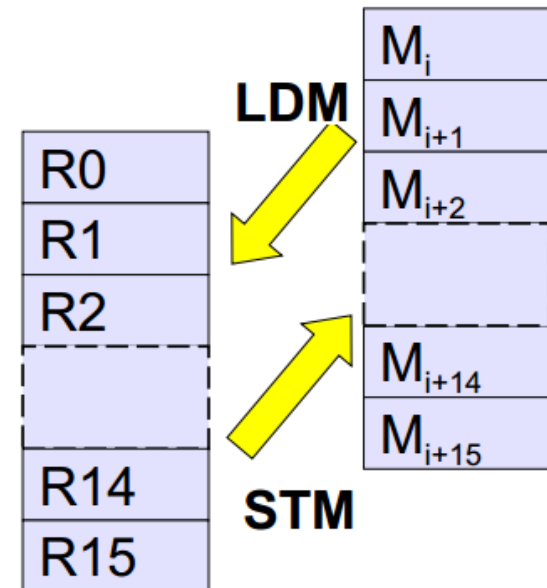
```
LDR r0,[pc, offset_to_lit_pool]
```

- A negde u code memoriji se nalazi oblast lit pool u kojoj se nalazi flashovana konstanta:

```
DCD 0x55555555
```

Instrukcije za prenos blokova podataka

Instruction	Description
Load Multiple, Increment After or Full Descending	<i>LDM, LDMIA, LDMFD</i>
Load Multiple, Decrement Before or Empty Ascending	<i>LDMDB, LDMEA</i>
Pop multiple registers off the stack ^a	<i>POP</i>
Push multiple registers onto the stack ^b	<i>PUSH</i>
Store Multiple, Increment After or Empty Ascending	<i>STM, STMIA, STMEA</i>
Store Multiple, Decrement Before or Full Descending	<i>STMDB, STMFD</i>



subroutine_1

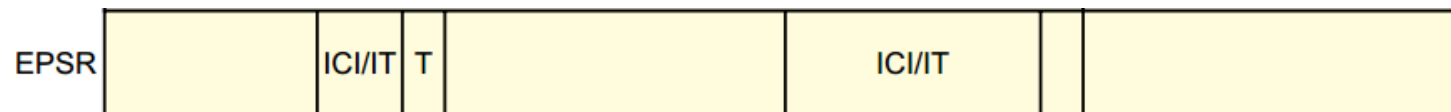
PUSH {R0-R7, R12, R14}; Save registers

... ; Do your processing

POP {R0-R7, R12, R14}; Restore registers

BX R14 ; Return to calling function

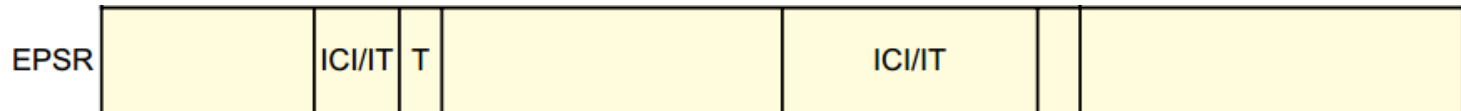
- Instrukcije traju više ciklusa, ali zahvaljujući ICI polju u ESPR statusnom registru mogu da budu prekinute izuzetkom ili prekidom.



Uslovno izvršavanje

- Većini instrukcija može da se postavi dodatni uslov za izvršavanje koji zavisi od Z,N,V,C flag-ova
- Ova mogućnost je ograničena samo na instrukcije koje se nalaze u okviru takozvanih IT (If-Then blokova)
- Uslovna izvršavanja blokova su prekidiva što se obezbeđuje korišćenjem IT polja u statusnom registru EPSR.

	IT block (each of <x>, <y> and <z> can either be T (true) or E (else))	Examples
Only one conditional instruction	IT <cond> instr1<cond>	IT EQ ADDEQ R0, R0, R1
Two conditional instructions	IT<x> <cond> instr1<cond> instr2<cond or ~(cond)>	ITE GE ADDGE R0, R0, R1 ADDLT R0, R0, R3
Three conditional instructions	IT<x><y> <cond> instr1<cond> instr2<cond or ~(cond)> instr3<cond or ~(cond)>	ITET GT ADDGT R0, R0, R1 ADDLE R0, R0, R3 ADDGT R2, R4, #1
Four conditional instructions	IT<x><y><z> <cond> instr1<cond> instr2<cond or ~(cond)> instr3<cond or ~(cond)> instr4<cond or ~(cond)>	ITETT NE ADDNE R0, R0, R1 ADDEQ R0, R0, R3 ADDNE R2, R4, #1 MOVNE R5, R3



Uslovno izvršavanje - primeri

```
ITTE NE ; Next 3 instructions are conditional
ANDNE R0, R0, R1 ; ANDNE does not update condition flags
ADDSNE R2, R2, #1 ; ADDSNE updates condition flags
MOVEQ R2, R3 ; Conditional move
```

```
CMP R0, #9 ; Convert R0 hex value (0 to 15) into ASCII
; ('0'-'9', 'A'-'F')
ITE GT ; Next 2 instructions are conditional
ADDGT R1, R0, #55 ; Convert 0xA -> 'A'
ADDLE R1, R0, #48 ; Convert 0x0 -> '0'
```

```
IT GT ; IT block with only one
ADDGT R1, R1, #1 ; Increment R1 conditionally
```

```
ITTEE EQ ; Next 4 instructions are conditional
MOVEQ R0, R1 ; Conditional move
ADDEQ R2, R2, #10 ; Conditional add
ANDNE R3, R3, #1 ; Conditional AND
BNE.W dloop ; Branch instruction can only be used in the last
; instruction of an IT block
```

```
IT NE ; Next instruction is conditional
ADD R0, R0, R1 ; Syntax error: no condition code used in IT block.
```

$IT\{x\{y\{z\}\}\} \textit{cond}$

where:

x specifies the condition switch for the second instruction in the IT block.
y specifies the condition switch for the third instruction in the IT block.
z specifies the condition switch for the fourth instruction in the IT block.
cond specifies the condition for the first instruction in the IT block.

The condition switch for the second, third and fourth instruction in the IT block can be either:

T Then. Applies the condition *cond* to the instruction.
E Else. Applies the inverse condition of *cond* to the instruction.

Sintaksa ITx instrukcije

Veza uslova i flag-ova

Table 5.35 Suffixes for Conditional Branches and Conditional Execution

Suffix	Branch Condition	Flags (APSR)
EQ	Equal	Z flag is set
NE	Not equal	Z flag is cleared
CS/HS	Carry set / unsigned higher or same	C flag is set
CC/LO	Carry clear / unsigned lower	C flag is cleared
MI	Minus / negative	N flag is set (minus)
PL	Plus / positive or zero	N flag is cleared
VS	Overflow	V flag is set
VC	No overflow	V flag is cleared
HI	Unsigned higher	C flag is set and Z is cleared
LS	Unsigned lower or same	C flag is cleared or Z is set
GE	Signed greater than or equal	N flag is set and V flag is set, or N flag is cleared and V flag is cleared (N == V)
LT	Signed less than	N flag is set and V flag is cleared, or N flag is cleared and V flag is set (N != V)
GT	Signed greater then	Z flag is cleared, and either both N flag and V flag are set, or both N flag and V flag are cleared (Z == 0 and N == V)
LE	Signed less than or equal	Z flag is set, or either N flag set with V flag cleared, or N flag cleared and V flag set (Z == 1 or N != V)

Format memorije

- CPU vidi memoriju kao linearno mapirani skup bajtova numerisan od lokacije 0.
- Podacima u memoriji (u word ili half-word formatu) procesor pristupa u little-endian ili big-endian formatu što se podešava nivoom signala na posebnoj konfiguracionom pinu pri resetu procesora.
- Instukcijskom kodu se uvek pristupa u little-endian formatu.

- Little-Endian : LSB na nižoj adresi

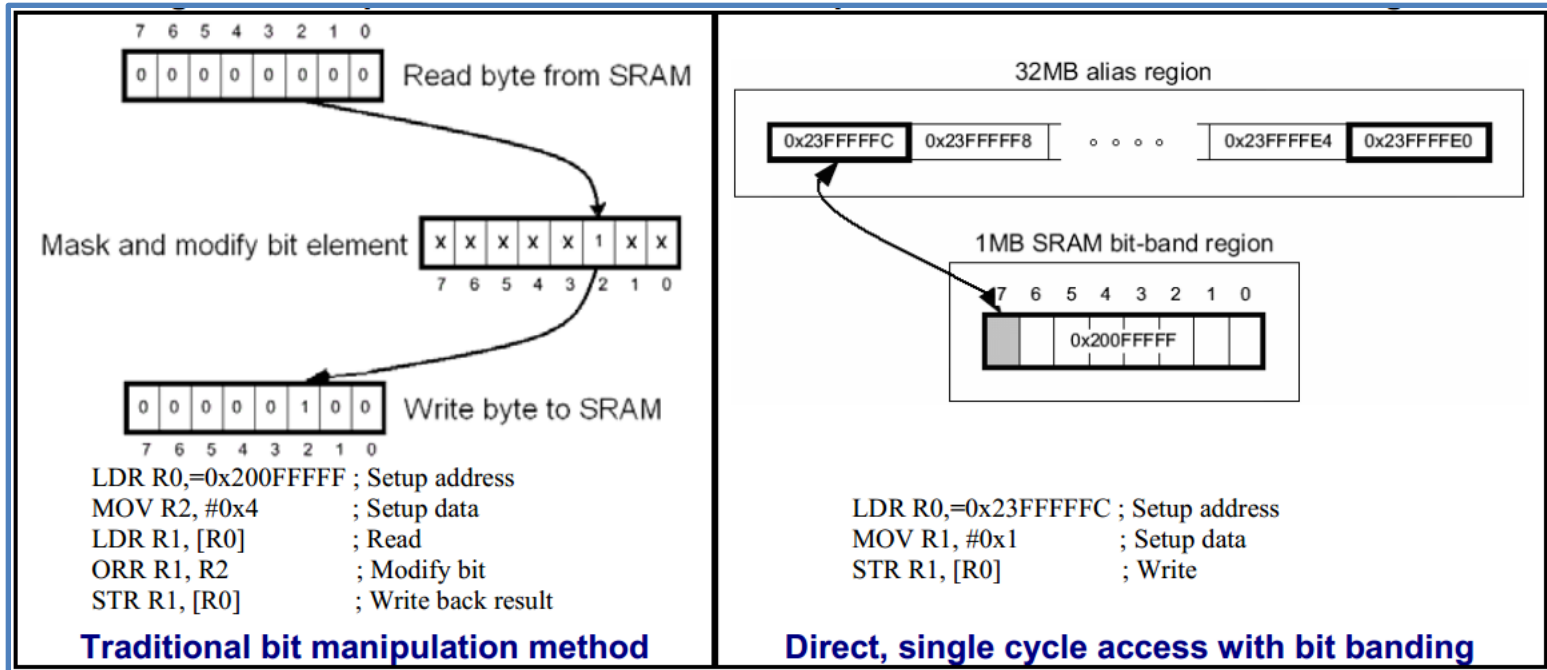
	Memory Offset	Value (LSB) (MSB)
	===== 0x0000	===== 01 02 FF 00
<code>uint8_t a = 1;</code>		
<code>uint8_t b = 2;</code>		
<code>uint16_t c = 255; // 0x00FF</code>		
<code>uint32_t d = 0x12345678;</code>	0x0004	78 56 34 12

- Big-Endian : MSB na nižoj adresi

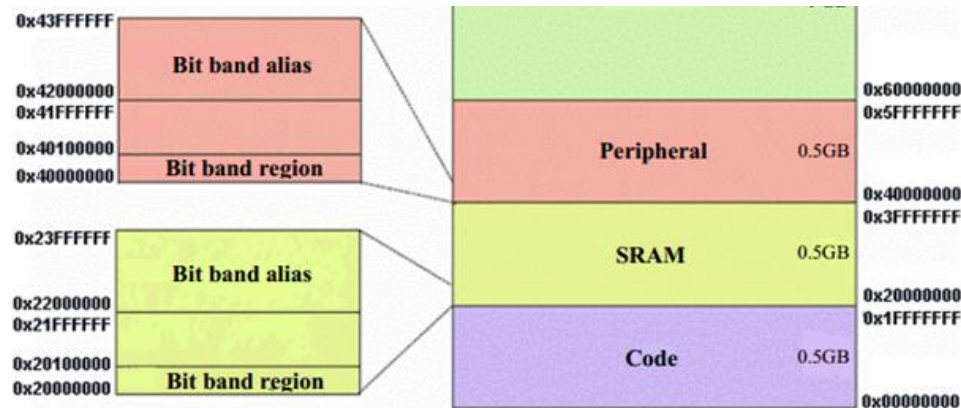
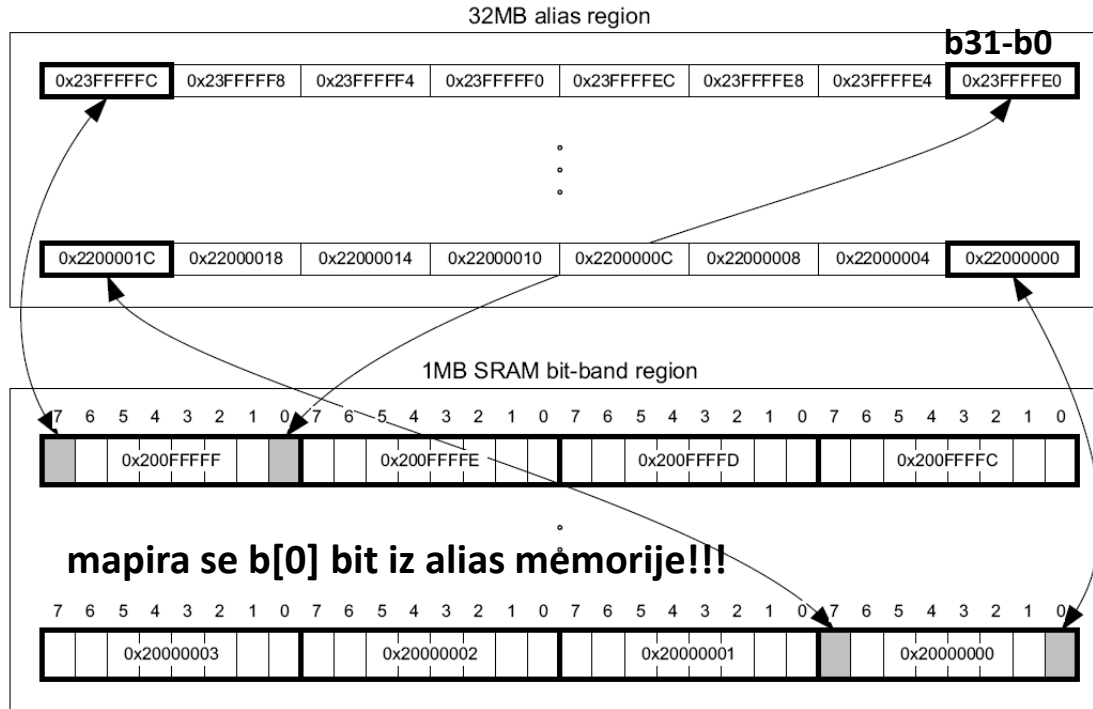
	Memory Offset	Value (LSB) (MSB)
	===== 0x0000	===== 01 02 00 FF
<code>uint8_t a = 1;</code>		
<code>uint8_t b = 2;</code>		
<code>uint16_t c = 255; // 0x00FF</code>		
<code>uint32_t d = 0x12345678;</code>	0x0004	12 34 56 78

Bit-banding

- Rešavanje problema tipičnog za Load/Store arhitekturu
- Kako obezbediti atomski (neprekidan) pristup nekoj memorijskoj lokaciji i izmenu jednog bita?
- Uobičajena sekvenca: `disable_int`→`load`→`modify`→`store`→`enable_int`.
- Bit-banding pristup je posredan pristup jednobitnoj lokaciji mapiranoj preko 32-bitne lokacije u alias memorijskom regionu.



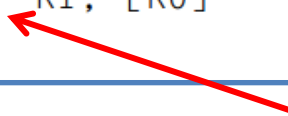
Bit-banding



Bit-banding

- Upisivanje sa i bez bit-banding-a

Without bit-band	With bit-band
LDR R0, =0x20000000 ; Setup address	LDR R0, =0x22000008 ; Setup add
LDR R1, [R0] ; Read	MOV R1, #1 ; Setup dat
ORR.W R1, #0x4 ; Modify bit	STR R1, [R0] ; Write
STR R1, [R0] ; Write back result	

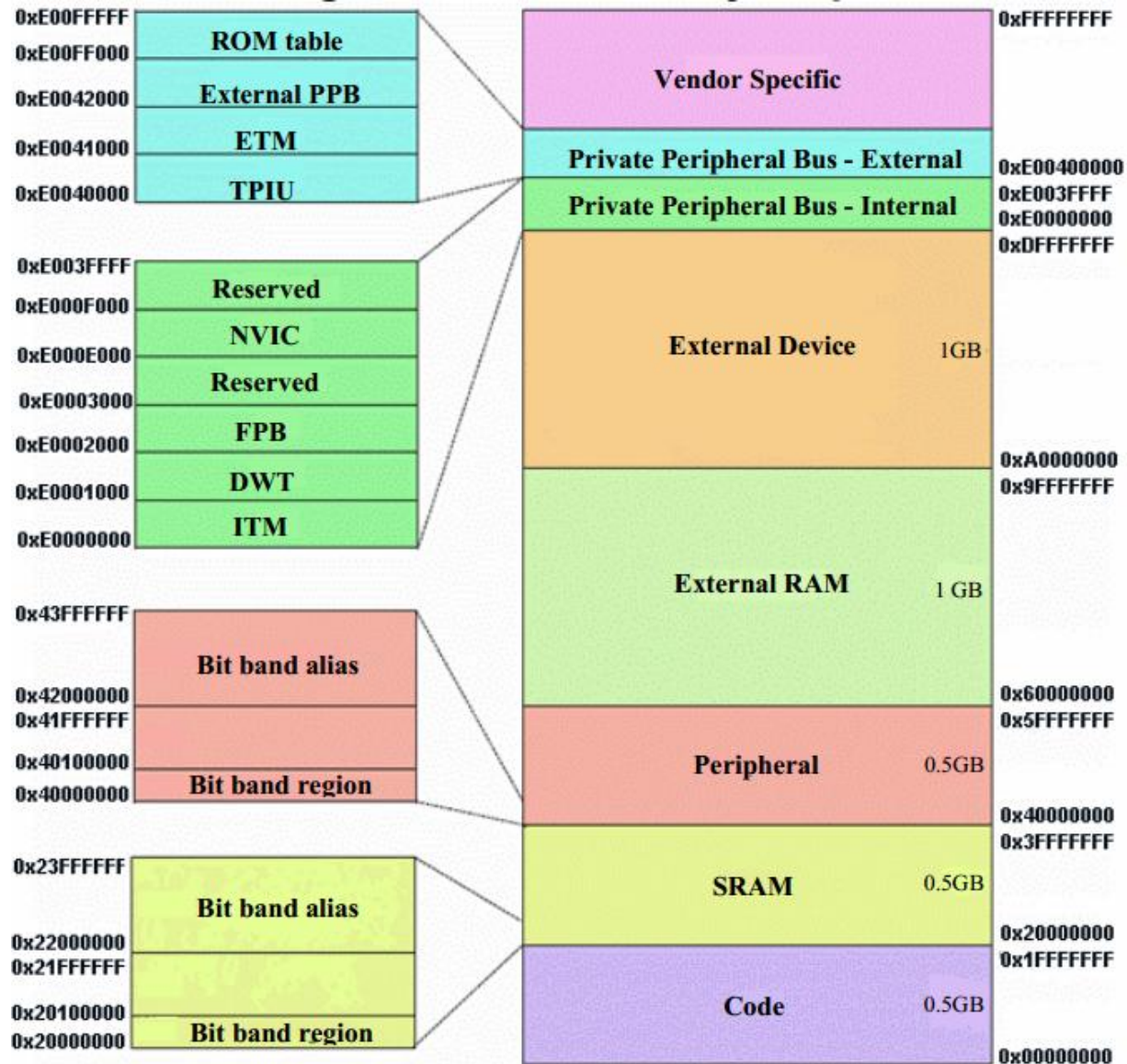


Atomska operacija

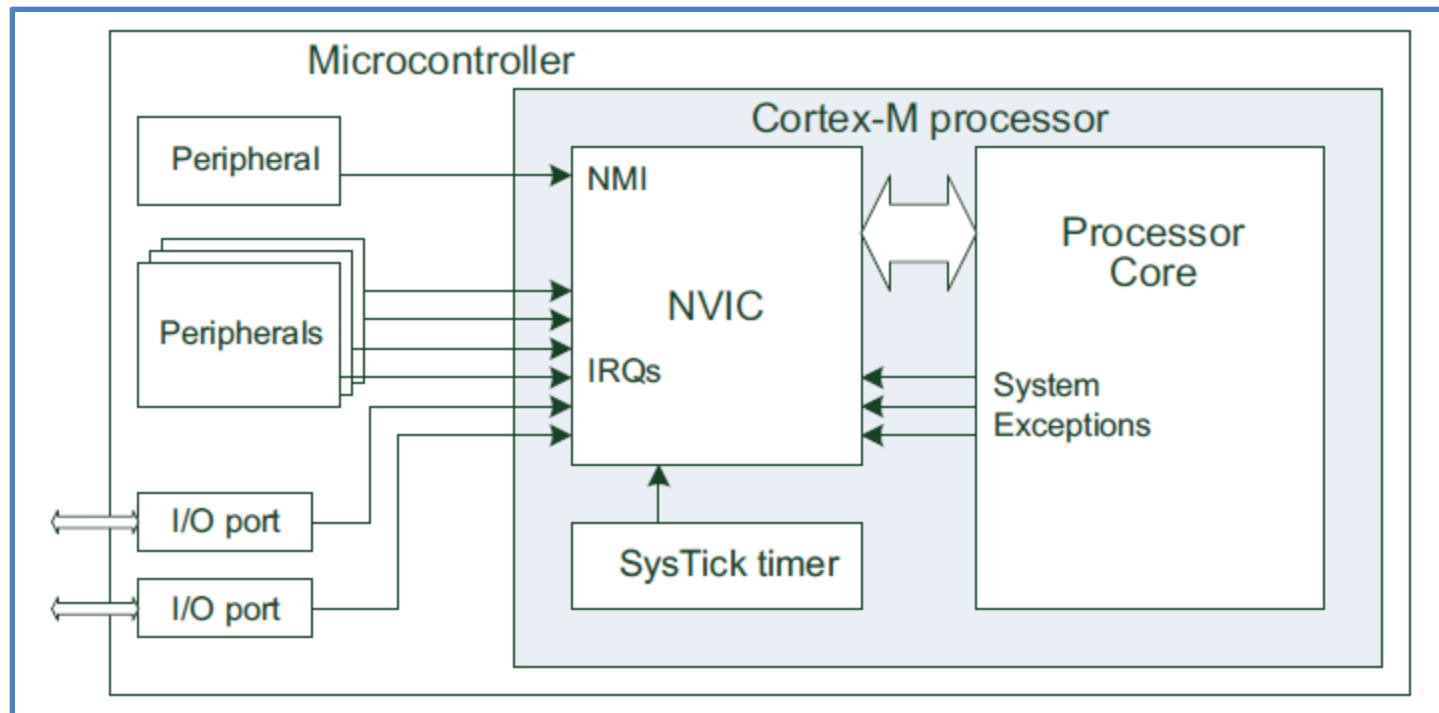
- Očitavanje sa i bez bit-banding-a

Without bit-band	With bit-band
LDR R0, =0x20000000 ; Setup address	LDR R0, =0x22000008 ; Setup address
LDR R1, [R0] ; Read	LDR R1, [R0] ; Read
UBFX.W R1, R1, #2, #1 ; Extract bit[2]	

Mapa adresnog prostora



NVIC – Nested Vector Interrupt Controller



NVIC – Nested Vector Interrupt Controller

- NVIC podržava do 240 prekida sa 256 nivoa prioriteta koji se mogu dinamički menjati.
- NVIC implementira mehanizme za malo vreme kašnjenja servisiranja prekida.
- Puni pristup registrima NVIC je moguć iz privilegovanog moda izvršavanja u bilo kom formatu (bajt, polureč, reč).
- Registri NVIC su sistemski registri u little endian formatu zapisa koji sadrže informacije o broju prekidnih linija, podešavanje sistemskog intervala, dozvoli prekida, baferisanju prekida (pending, pre-empted), prioritetima prekida, informacije o broju i verziji CPU, procesiranju izuzetaka (prioritetima, dozvoli), adresi početka VT, kontroli stanja CPU sa smanjenom potrošnjom.
- Omogućeno gneždenje prekida

IVT – Interrupt Vector Table

- **Mogućnost remapiranja tabele vektora preko podešavanja registra ofseta tabele**
- **Adresa početka tabele poravnata na granicu 1024 bajta.**

Memory Address		Exception Number
		1
		1
0x0000004C	Interrupt#3 vector	19
0x00000048	Interrupt#2 vector	18
0x00000044	Interrupt#1 vector	17
0x00000040	Interrupt#0 vector	16
0x0000003C	SysTick vector	15
0x00000038	PendSV vector	14
0x00000034	Not used	13
0x00000030	Debug Monitor vector	12
0x0000002C	SVC vector	11
0x00000028	Not used	10
0x00000024	Not used	9
0x00000020	Not used	8
0x0000001C	Not used	7
0x00000018	Usage Fault vector	6
0x00000014	Bus Fault vector	5
0x00000010	MemManage vector	4
0x0000000C	HardFault vector	3
0x00000008	NMI vector	2
0x00000004	Reset vector	1
0x00000000	MSP initial value	0

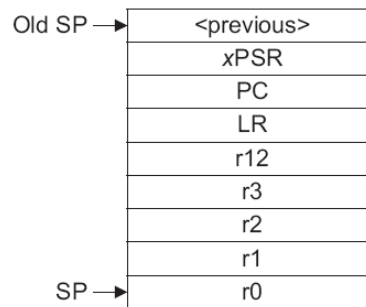
Prekidi
periferijskih modula

Izuzeci

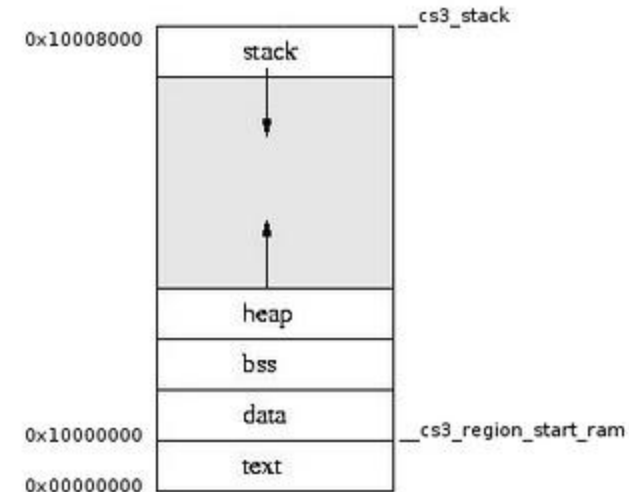
Zamena konteksta

- Zamena konteksta je karakteristična za promenu toka izvršavanja programskog koda kao posledice pojave izuzetka ili promene aktivnog taska (procesa, tj. thread-a) na sistemu sa RTOS.
- Kod Cortex-M3 procesora se pri zameni konteksta na steku pamte sledeći registri:

- Programski brojač PC
- Statusni registar procesora xPSR
- r0-r3
- r12
- Link registar LR



- Nakon završetka operacije smeštanja podataka na stek SP se dekrementira za 8 reči.



MPU

- MPU je komponenta koja omogućava zaštitu pristupa memoriji kroz podršku za zaštitu regiona, preklapanje zaštićenih regiona, **dozvolu pristupa** i prosleđivanje memorijskih parametara sistemu.
- U slučaju grešaka u pristupu generiše se **Memory_Management** izuzetak čiji je prioritet programabilan.
- Pomoću MPU jedinice moguće je implementirati **privilegije pristupa**, razdvojiti procese i implementirati pravila pristupa.
- Registri MPU definišu broj podržanih regiona, baznu adresu, veličinu, prava pristupa...

MPU

Table 9-6 MPU Region Attribute and Size Register bit assignments

Bits	Field	Function
[31:29]	-	Reserved.
[28]	XN	Instruction access disable bit: 1 = disable instruction fetches 0 = enable instruction fetches.
[27]	-	Reserved.
[26:24]	AP	Data access permission field:

Value	Privileged permissions	User permissions
b000	No access	No access
b001	Read/write	No access
b010	Read/write	Read-only
b011	Read/write	Read/write
b100	Reserved	Reserved
b101	Read-only	No access
b110	Read-only	Read-only
b111	Read-only.	Read-only.

- Kod Cortex-M3 MPU podržava do 8 regiona koji mogu biti podeljeni na po 8 podregiona.

Neprivilegovani mod izvršavanja

Privilegovani mod izvršavanja

Opcije podešavanja prava pristupa

SIMD

GE[3:0]

Greater-Than or Equal flags for each byte lane (ARMv7E-M only; not available in ARMv6-M or Cortex[®]-M3).

Table 5.49 SIMD Instructions

Prefix Operation (see next table)	S ¹ Signed	Q ² Signed Saturating	SH ³ Signed Halving	U ¹ Unsigned	UQ ² Unsigned Saturating	UH ³ Unsigned Halving
ADD8	SADD8	QADD8	SHADD8	UADD8	UQADD8	UHADD8
SUB8	SSUB8	QSUB8	SHSUB8	USUB8	UQSUB8	UHSUB8
ADD16	SADD16	QADD16	SHADD16	UADD16	UQADD16	UHADD16
SUB16	SSUB16	QSUB16	SHSUB16	USUB16	UQSUB16	UHSUB16
ASX	SASX	QASX	SHASX	UASX	UQASX	UHASX
SAX	SSAX	QSAX	SHSAX	USAX	UQSAX	UHSAX

¹GE bits updates.

²Q bit is set when saturation occurs.

³Each data in the SIMD operation result is divided by 2 in Signed Halving (SH) and Unsigned Halving (UH) operations.

SIMD pravila postavljanja GE flagova

SIMD Operation	Results
SADD16, SSUB16, USUB16, SASX, SSAX	If lower half-word result ≥ 0 then $GE[1:0] = 2'b11$ else $GE[1:0] = 2'b00$ If upper half-word result ≥ 0 then $GE[3:2] = 2'b11$ else $GE[3:2] = 2'b00$
UADD16	If lower half-word result $\geq 0x10000$ then $GE[1:0] = 2'b11$ else $GE[1:0] = 2'b00$ If upper half-word result $\geq 0x10000$ then $GE[3:2] = 2'b11$ else $GE[3:2] = 2'b00$
SADD8, SSUB8, USUB8	If byte 0 result ≥ 0 then $GE[0] = 1'b1$ else $GE[0] = 1'b0$ If byte 1 result ≥ 0 then $GE[1] = 1'b1$ else $GE[1] = 1'b0$ If byte 2 result ≥ 0 then $GE[2] = 1'b1$ else $GE[2] = 1'b0$ If byte 3 result ≥ 0 then $GE[3] = 1'b1$ else $GE[3] = 1'b0$
UADD8	If byte 0 result $\geq 0x100$ then $GE[0] = 1'b1$ else $GE[0] = 1'b0$ If byte 1 result $\geq 0x100$ then $GE[1] = 1'b1$ else $GE[1] = 1'b0$ If byte 2 result $\geq 0x100$ then $GE[2] = 1'b1$ else $GE[2] = 1'b0$ If byte 3 result $\geq 0x100$ then $GE[3] = 1'b1$ else $GE[3] = 1'b0$
UASX	If lower half-word result ≥ 0 then $GE[1:0] = 2'b11$ else $GE[1:0] = 2'b00$ If upper half-word result $\geq 0x10000$ then $GE[3:2] = 2'b11$ else $GE[3:2] = 2'b00$
USAX	If lower half-word result $\geq 0x10000$ then $GE[1:0] = 2'b11$ else $GE[1:0] = 2'b00$ If upper half-word result $\geq 0x0$ then $GE[3:2] = 2'b11$ else $GE[3:2] = 2'b00$

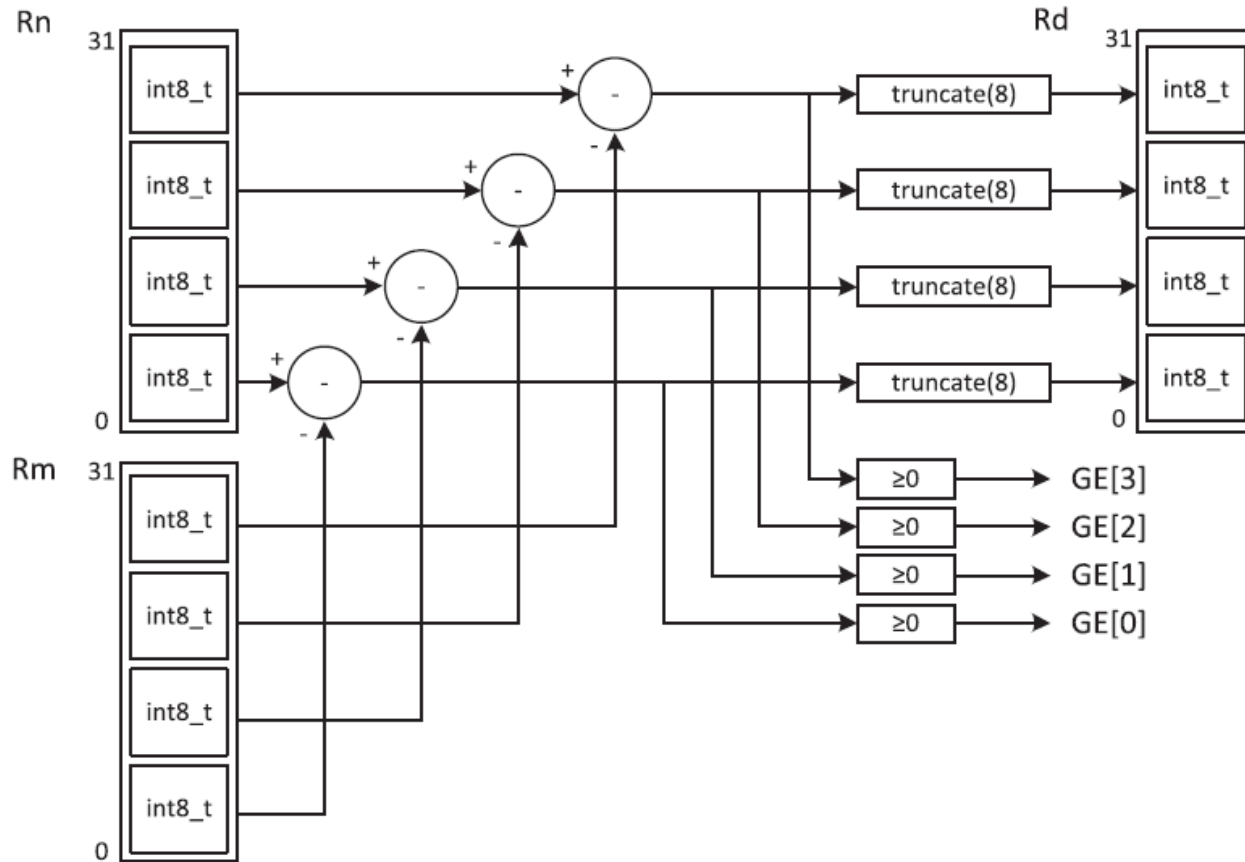
SIMD

SIMD and saturating instructions:

Mnemonic	Operands	Brief Description	Flags	Figure
SADD8	{Rd,} Rn, Rm	Signed Add 8	GE [3:0]	B.13
SADD16	{Rd,} Rn, Rm	Signed Add 16	GE [3:0]	B.14
SSUB8	{Rd,} Rn, Rm	Signed Subtract 8	GE [3:0]	B.17
SSUB16	{Rd,} Rn, Rm	Signed Subtract 16	GE [3:0]	B.18
SASX	{Rd,} Rn, Rm	Signed Add and Subtract with Exchange	GE [3:0]	B.21
SSAX	{Rd,} Rn, Rm	Signed Subtract and Add with Exchange	GE [3:0]	B.22

SIMD

SSUB8 {<Rd>}, <Rn>, <Rm>

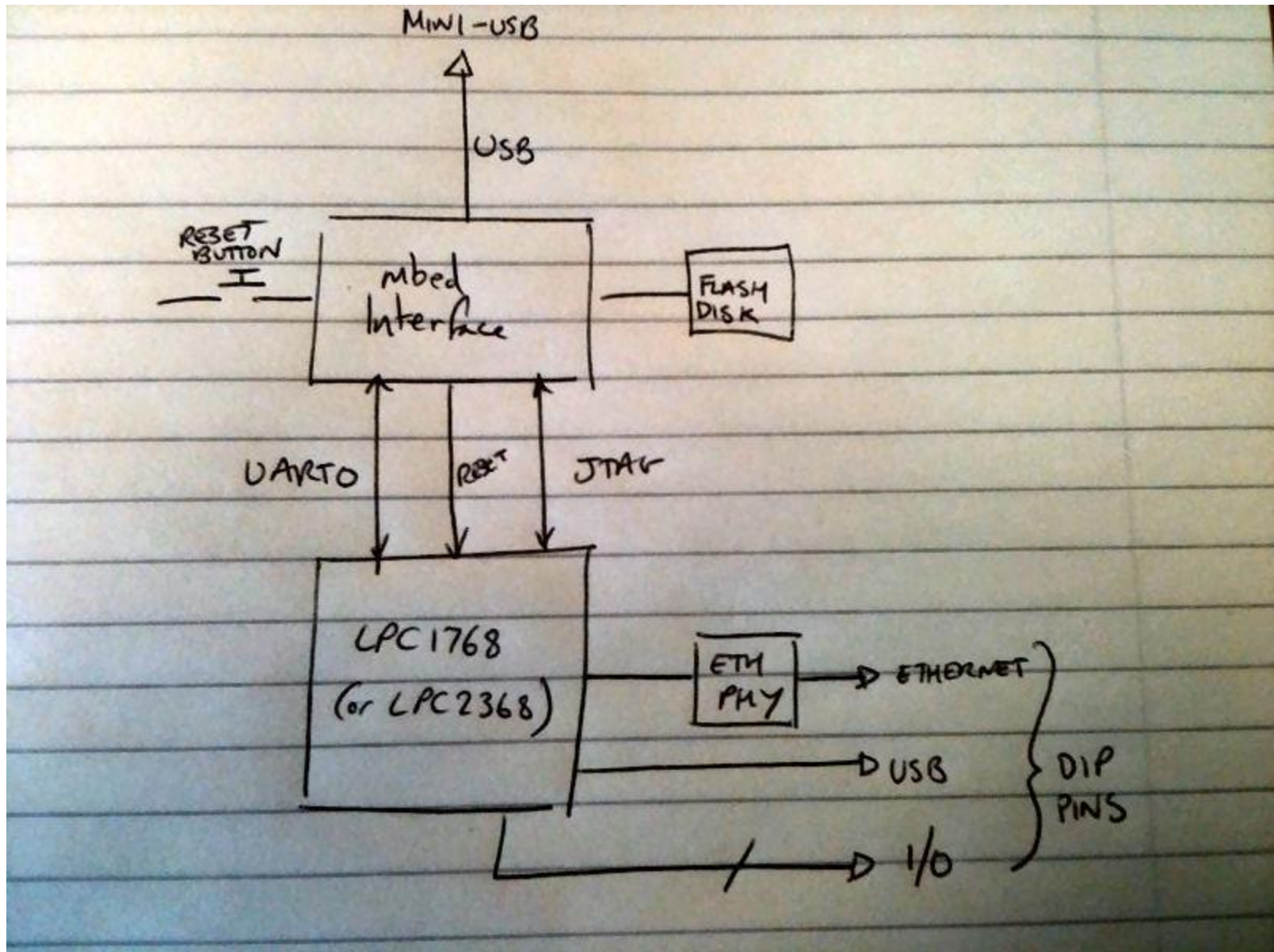


MBED platforme

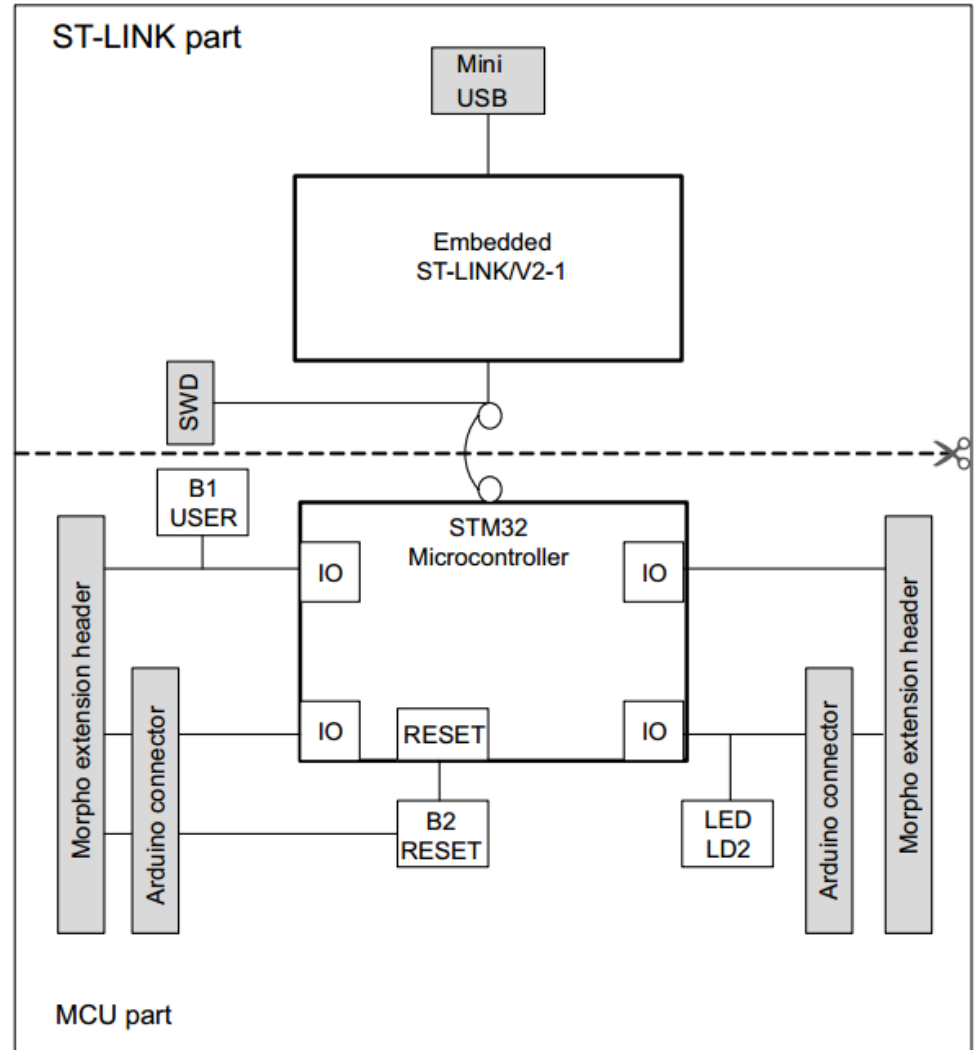
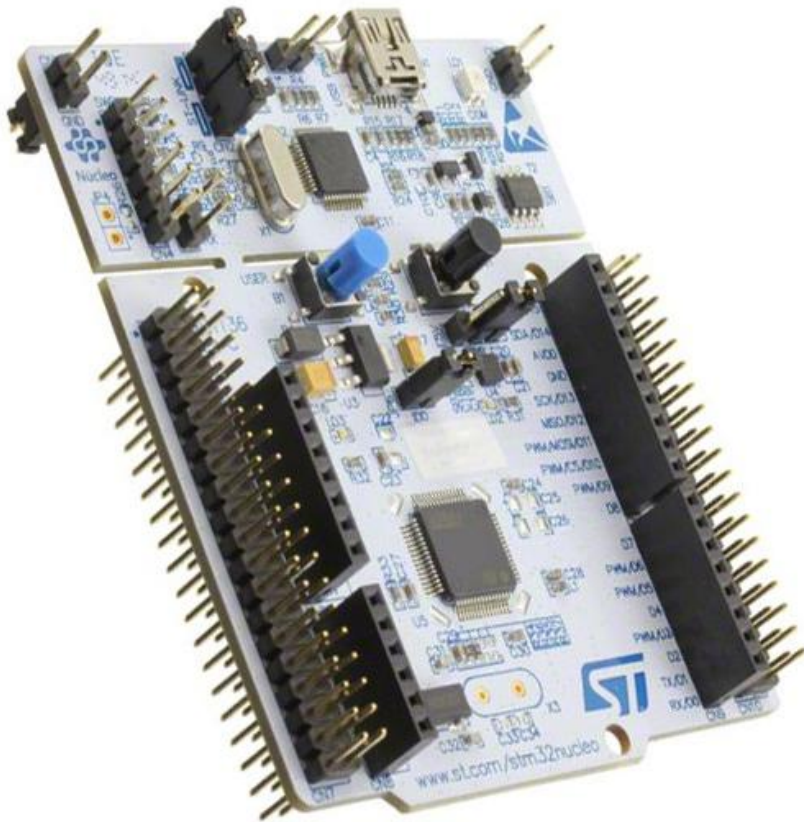
- Svakim danom broj MBED-enabled platformi raste

<p>EA LPC4088 QuickStart Board</p> <ul style="list-style-type: none"> • Cortex-M4, 120MHz • 512KB Flash, 96KB SRAM 	<p>DipCortex M0</p> <ul style="list-style-type: none"> • Cortex-M0, 50MHz • 32KB Flash, 8KB RAM 	<p>DipCortex M3</p> <ul style="list-style-type: none"> • Cortex-M3, 72MHz • 64KB Flash, 12KB RAM 	<p>BlueBoard-LPC1114</p> <ul style="list-style-type: none"> • Cortex-M0, 48MHz • 32KB Flash, 8KB RAM
<p>WiFi DipCortex</p> <ul style="list-style-type: none"> • Cortex-M3, 72MHz • 64KB Flash, 12KB RAM 	<p>Seeeduino-Arch</p> <ul style="list-style-type: none"> • Cortex-M0, 48MHz • 32KB Flash, 8KB RAM 	<p>LPC1114FN28</p> <ul style="list-style-type: none"> • Cortex-M0, 50MHz • 32KB Flash, 4KB RAM 	<p>u-blox GD27</p> <ul style="list-style-type: none"> • Cortex-M3, 96MHz • 512KB Flash, 32KB RAM • Onboard cellular module
<p>EA LPC1114U35 QuickStart Board</p> <ul style="list-style-type: none"> • Cortex M0, 48MHz • 64KB Flash, 10KB RAM 	<p>ST Nucleo F103RB</p> <ul style="list-style-type: none"> • STM32F103RBT6 mcu • Cortex-M3 72MHz • 128-KB Flash, 20-KB SRAM 	<p>FRDM-KL46Z</p> <ul style="list-style-type: none"> • Cortex-M0-, 48MHz • 256KB Flash, 32KB RAM • USB OTG 	<p>Seeeduino-Arch-Pro</p> <ul style="list-style-type: none"> • Cortex-M3, 96MHz • 512KB Flash, 32KB RAM
<p>ST Nucleo L152RE</p> <ul style="list-style-type: none"> • STM32L152RET6 mcu • Cortex-M3 32MHz • 512-KB Flash, 80-KB SRAM 	<p>ST Nucleo F401RE</p> <ul style="list-style-type: none"> • STM32F401RET6 mcu • Cortex-M4 84MHz • 512-KB Flash, 96-KB SRAM 	<p>ST Nucleo F030R8</p> <ul style="list-style-type: none"> • STM32F030R8T6 mcu • Cortex-M0 48MHz • 64-KB Flash, 8-KB SRAM 	<p>Nordic nRF51822</p> <ul style="list-style-type: none"> • Bluetooth v4.1 • Cortex-M0, 16MHz • 128KB Flash, 16KB RAM

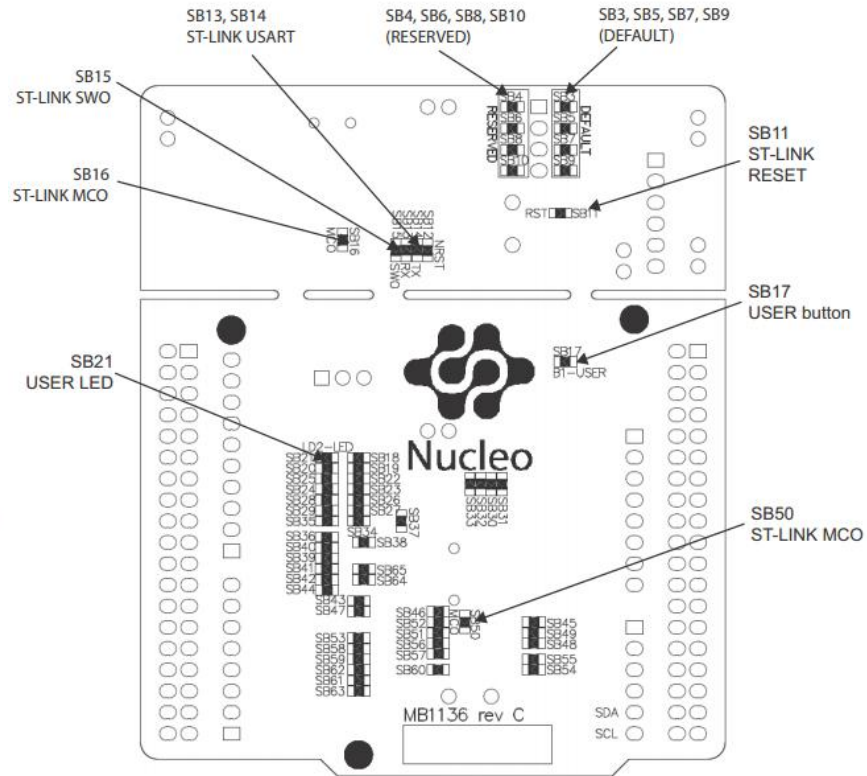
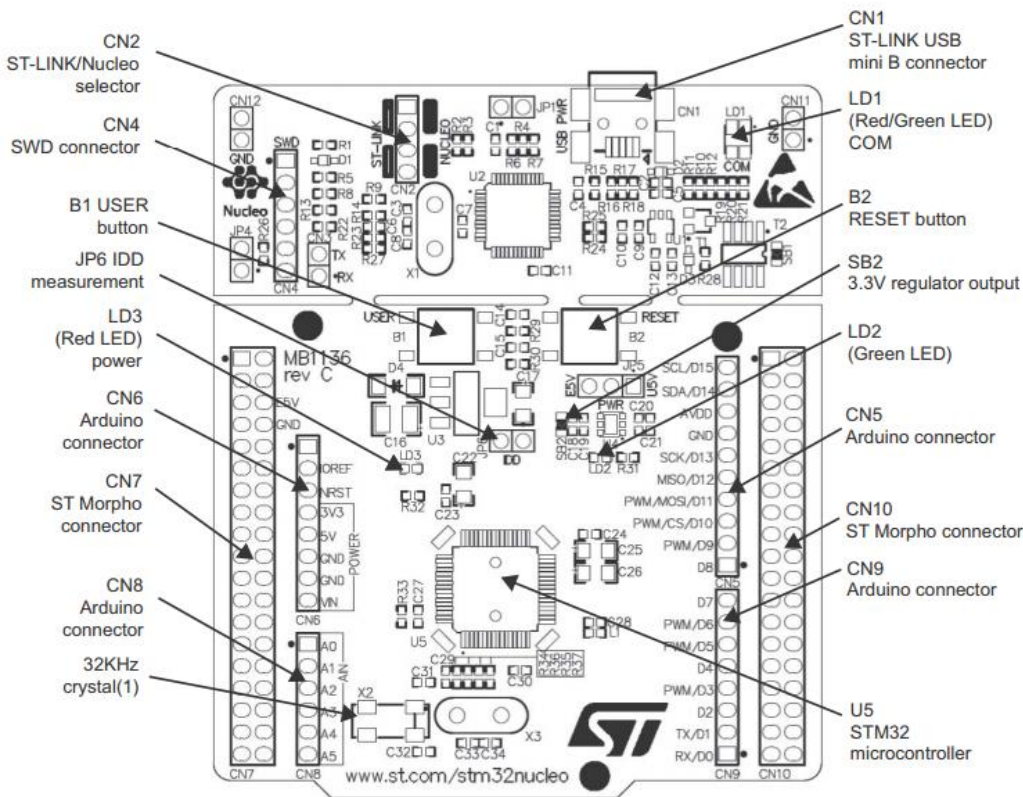
MBED -arhitektura



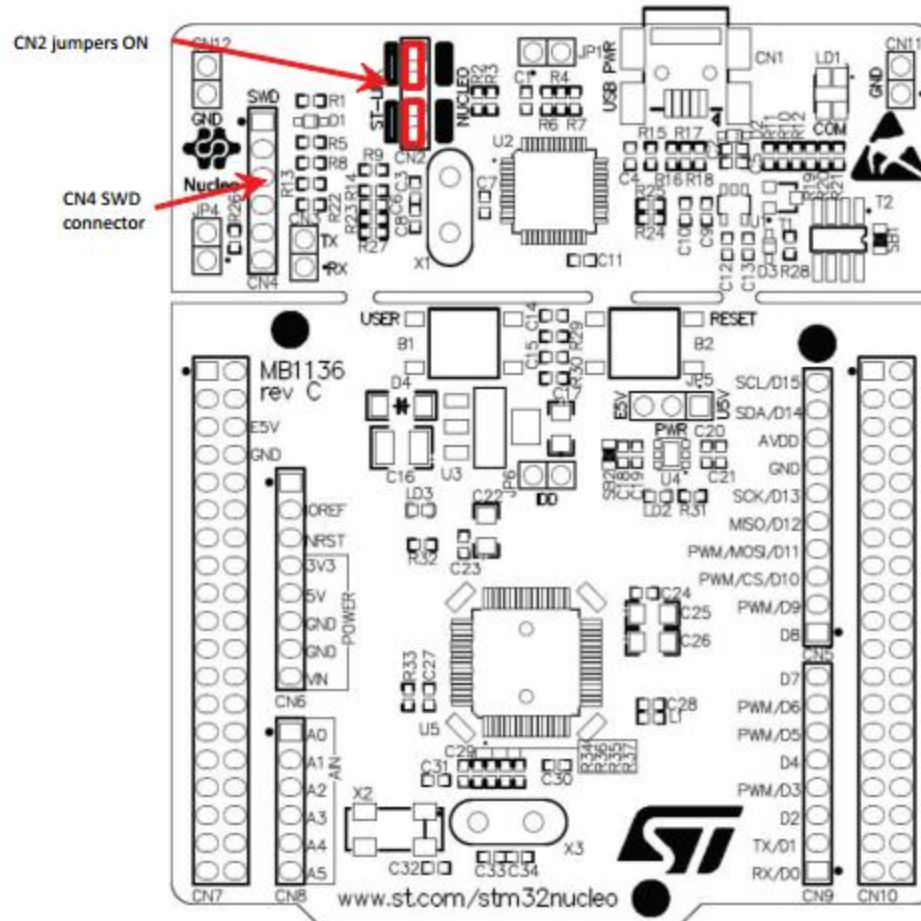
MBED - NUCLEO



Sta ima na pločici



Programiranje



STM32L476

Ultra-low-power with FlexPowerControl

- 1.71 V to 3.6 V power supply
- 300 nA in VBAT mode: supply for RTC and 32x32-bit backup registers
- 100 μ A/MHz run mode
- 4 μ s wakeup from Stop mode
- Core: ARM[®] 32-bit Cortex[®]-M4 CPU with FPU, Adaptive real-time accelerator (ART Accelerator™) allowing 0-wait-state execution from Flash memory, frequency up to 80 MHz, MPU, 100DMIPS/1.25DMIPS/MHz (Dhrystone 2.1), and DSP instructions

Memories

- Up to 1 MB Flash, 2 banks read-while-write, proprietary code readout protection
- Up to 128 KB of SRAM including 32 KB with hardware parity check

Rich analog peripherals (independent supply)

- 3x 12-bit ADC 5 Msps, up to 16-bit with hardware oversampling, 200 μ A/Msps
- 2x 12-bit DAC, low-power sample and hold
- 2x operational amplifiers with built-in PGA
- 2x ultra-low-power comparators
- 18x communication interfaces
- USB OTG 2.0 full-speed, LPM and BCD
- 2x SAls (serial audio interface)
- 3x I2C FM+(1 Mbit/s), SMBus/PMBus
- 6x USARTs (ISO 7816, LIN, IrDA, modem)
- 3x SPIs (4x SPIs with the Quad SPI)
- CAN (2.0B Active) and SDMMC interface
- SWPMI single wire protocol master I/F
- 14-channel DMA controller
- True random number generator
- CRC calculation unit, 96-bit unique ID
- Development support: serial wire debug (SWD), JTAG, Embedded Trace Macrocell™

NUCLEO pinovi

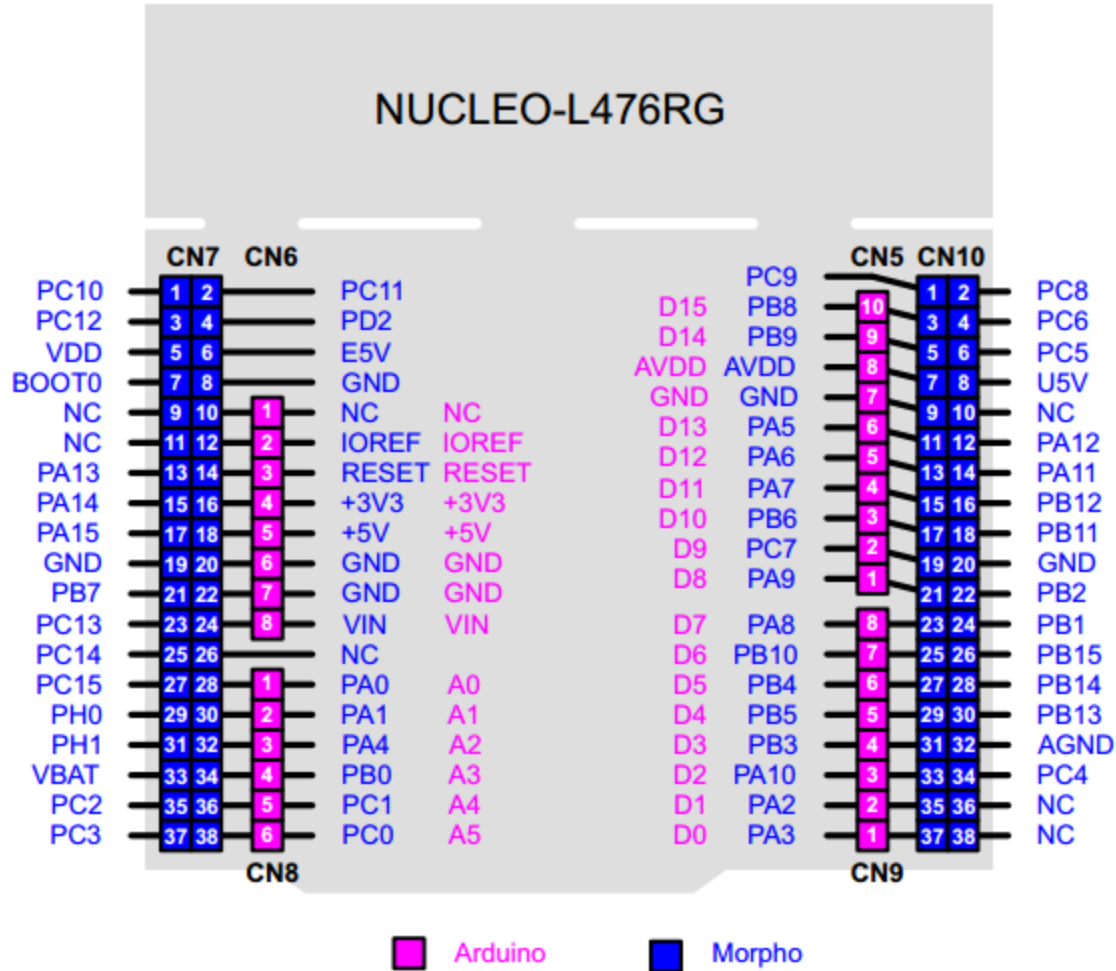
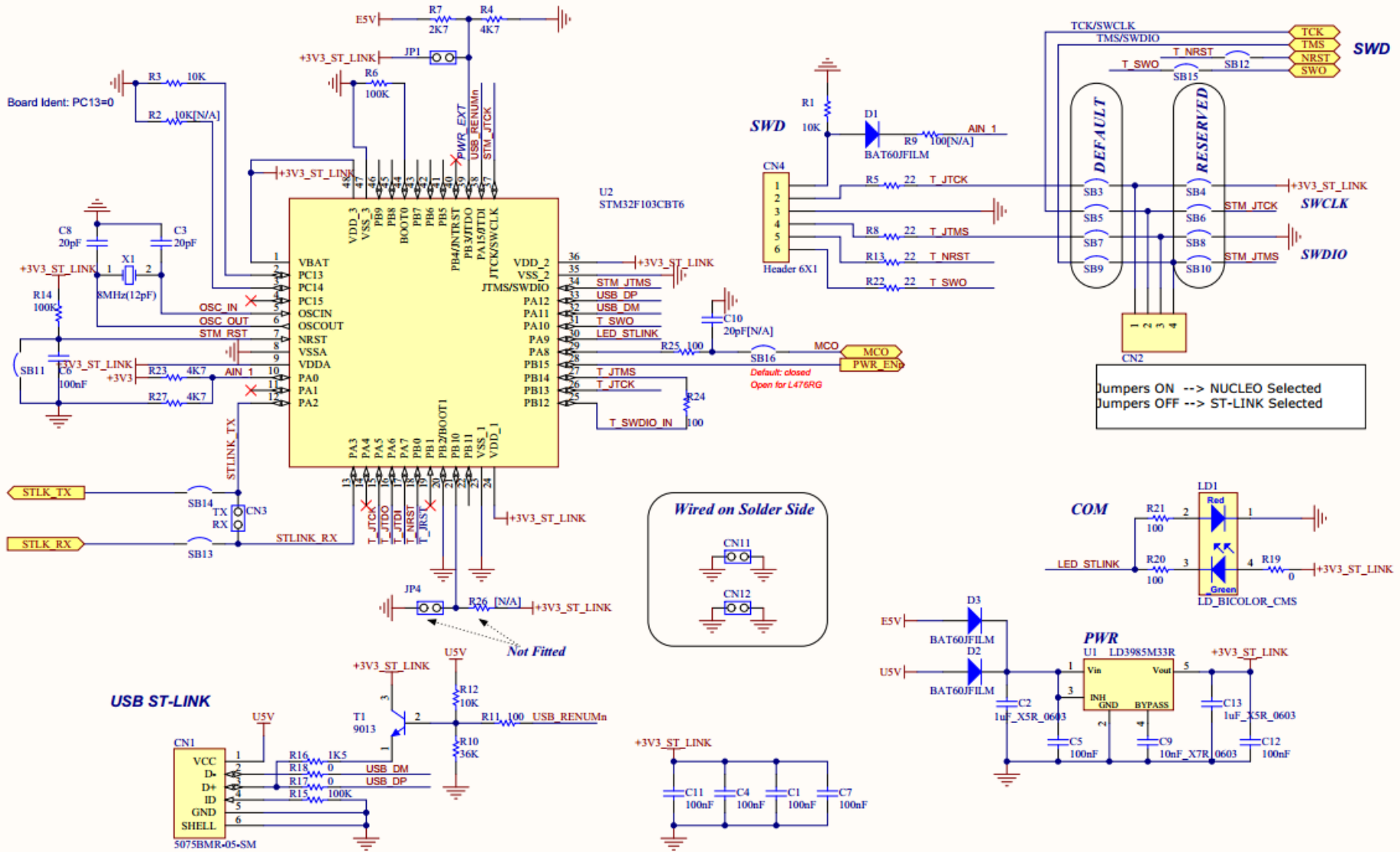
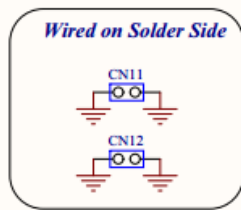


Table 29. STMicroelectronics Morpho connector on NUCLEO-L476RG

CN7 odd pins		CN7 even pins		CN10 odd pins		CN10 even pins	
Pin No.	Name	Name	Pin No.	Pin No.	Name	Name	Pin No.
1	PC10	PC11	2	1	PC9	PC8	2
3	PC12	PD2	4	3	PB8	PC6	4
5	VDD	E5V	6	5	PB9	PC5	6
7	BOOT0 ⁽¹⁾	GND	8	7	AVDD	U5V ⁽²⁾	8
9	-	-	10	9	GND	-	10
11	-	IOREF	12	11	PA5	PA12	12
13	PA13 ⁽³⁾	RESET	14	13	PA6	PA11	14
15	PA14 ⁽³⁾	+3V3	16	15	PA7	PB12	16
17	PA15	+5V	18	17	PB6	PB11	18
19	GND	GND	20	19	PC7	GND	20
21	PB7	GND	22	21	PA9	PB2	22
23	PC13	VIN	24	23	PA8	PB1	24
25	PC14	-	26	25	PB10	PB15	26
27	PC15	PA0	28	27	PB4	PB14	28
29	PH0	PA1	30	29	PB5	PB13	30
31	PH1	PA4	32	31	PB3	AGND	32
33	VBAT	PB0	34	33	PA10	PC4	34
35	PC2	PC1 or PB9 ⁽⁴⁾	36	35	PA2	-	36
37	PC3	PC0 or PB8 ⁽⁴⁾	38	37	PA3	-	38



Jumpers ON --> NUCLEO Selected
 Jumpers OFF --> ST-LINK Selected

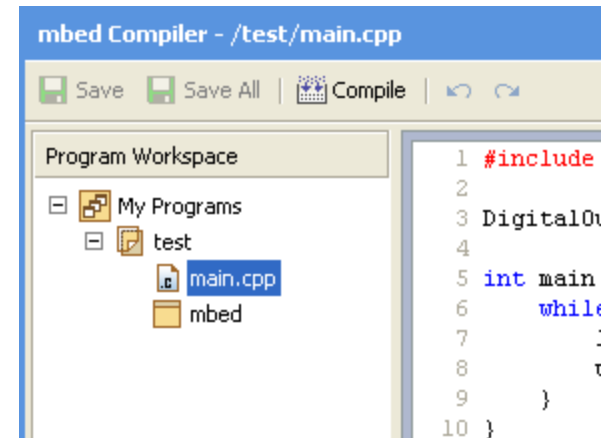
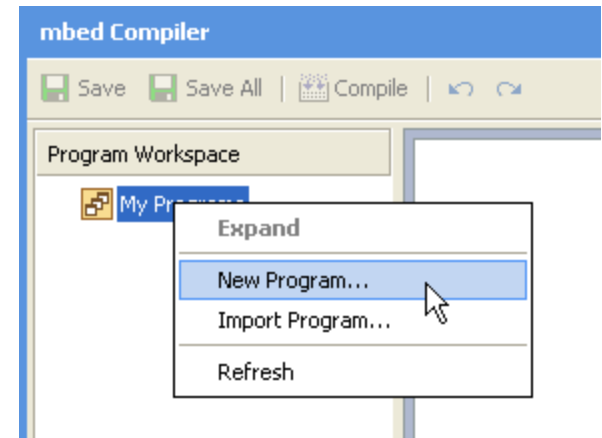


Title: STLINK/V2-1		
Project: NUCLEO-XXXXRX		
Size: A4	Reference: MB1136	Revision: C.3
Date: 12/9/2014	Sheet: 3 of 4	



Hello World

- Postojeće programe i biblioteke je moguće uvesti u kompajler korišćenjem Import funkcije.
- Osim toga, prilikom kreiranja novog programa automatski će biti kreiran main.cpp fajl sa spektakularnom funkcijom treperenja LED diode. To je praktično Hello World.



C++

naziv tipa (klasa)

myled objekat tipa DigitalOut, poziv konstruktora

pin1_18 (definisano u PinNames.h)

```
#include "mbed.h"
```

```
DigitalOut myled(LED1);
```

```
int main() {
```

```
    while(1) {
```

```
        myled = !myled;
```

```
        wait(0.25);
```

```
    }
```

```
}
```

Upotreba DigitalOut biblioteke

Nema adresiranja porta

Nema bit operacija

Nema pristupa pristupa direction registru

operator=

```
DigitalOut& operator= (DigitalOut& rhs) {
```

```
    write(rhs.read());
```

```
    return *this;
```

```
}
```

Pozivi funkcija DigitalOut biblioteke

Mogućnosti poziva funkcija u formi složenih

logičkih i aritmetičkih konstrukcija

Debugovanje

<http://mbed.org/handbook/Debugging>

- Run-time greske – greške koje se ne vide u toku prevođenja. Nastaju kada loše napisan program pokušava da radi nedozvoljene radnje ili pokušava da pristupa nedozvoljenim resursima. Na sledećem primeru se pokušava definisanje pwm izlaza na portu p20 što ne nedozvoljeno.

```
#include "mbed.h"
```

```
PwmOut led(p20);
```

```
int main() {
```

```
    while(1) {
```

```
        for(float p = 0.0f; p < 1.0f; p += 0.1f) {
```

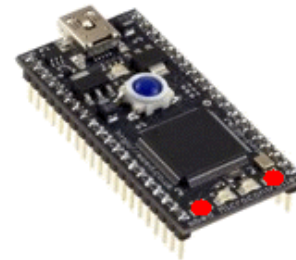
```
            led = p;
```

```
            wait(0.1);
```

```
        }
```

```
    }
```

```
}
```



Efekat koji će se javiti je specifična signalizacija LE dioda.

Debugovanje funkcionalnosti – Online Compiler

- U slučaju korišćenja ONLINE kompjalera, MBED nema mogućnost prave podrške za real-time debugging kao što to recimo ima MSP430 korišćenjem FET debugger-a, i to je jedna od većih mana MBED-a.
- Nemoguće je zaustaviti izvršavanje programa u nekom trenutku niti postaviti breakpoint. Nemoguće je i očitati proizvoljan sadržaj memorije niti registara mikrokontrolera.
- U nedostatku boljeg koriste se dva odavno poznata načina debugovanja.
- Jedan je korišćenje neke od 4 LE diode na samom MBED-u. Ideja je da se različita stanja korisničkog programa signaliziraju različitim stanjima uključenosti dioda.
- Drugi način je korišćenje sistemskog serijskog interfejsa koji je sastavni deo USB veze PC-ja i MBED-a.
- Standardna MBED biblioteka poseduje klasu serial pomoću koje se može konfigurisati i pristupati USB serijskom ali i ostalim serijskim portovima MBED-a.

“Pravo” debugovanje - STM32 – Nucleo

The screenshot displays the µVision IDE interface for a project named "mbed NUCLEO_F401RE". The main window shows the source code for "stm32f4xx_hal_gpio_ex.h" with the following content:

```
1 #include "mbed.h"
2
3 DigitalOut myled(LED1);
4
5 GPIO_InitTypeDef GPIO_InitStructure;
```

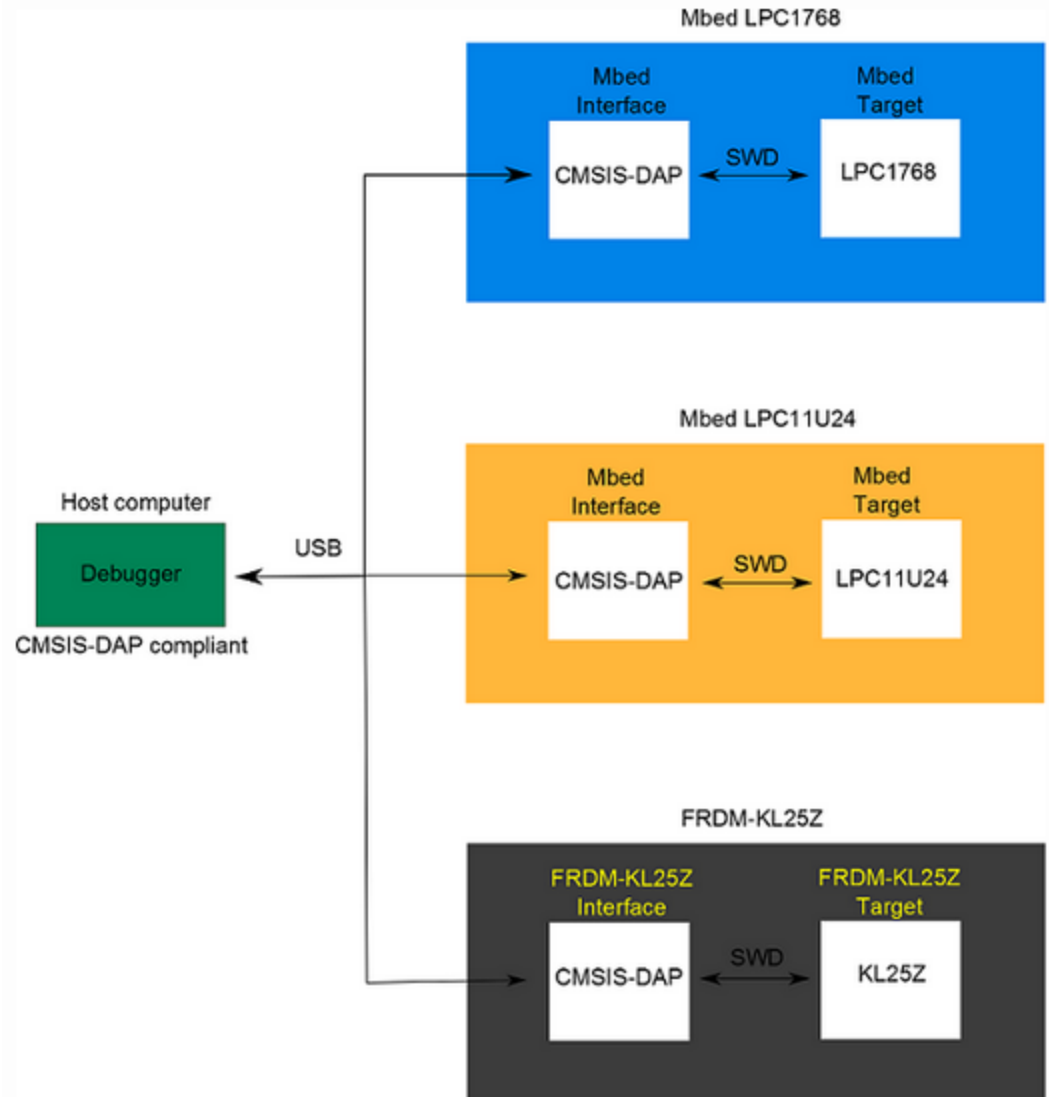
The "Options for Target 'mbed NUCLEO_F401RE'" dialog box is open, with the "Debug" tab selected. The "Use:" dropdown menu is set to "ST-Link Debugger", which is highlighted with a red box. Other settings include:

- Use Simulator
- Limit Speed to Real-Time
- Load Application at Startup
- Run to main()
- Breakpoints
- Toolbox
- Watch Windows & Performance Analyzer
- Watch Windows
- Memory Display
- System Viewer

On the right side of the image, two photographs of the STM32 Nucleo board are shown. The top one is a smaller view with a red box highlighting the top section, and the bottom one is a larger view with a blue box highlighting the bottom section.

“Pravo” debugovanje - LPC

- Od skoro je data podrška za pravo debugovanje kroz korišćenje CMSIS- DAP (debugg access port).
- To je moguće samo ukoliko se koristi offline kompajler koji podržava pristup (KEIL) i ukoliko je moguće firmware MBED interfejsa unaprediti za tu svrhu.



Eksportovanje prema eksternom debugger-u

