PRIMENA MIKROKONTROLERA-MS1PMK

3. deo

2016

Nenad Jovičić

Modovi adresiranja

- Neposredno adresiranje – sastavni deo instrukcije je podatak, tj. konstanta.
- Registarsko 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:

op	Is one of:	
	LDR	Load Register.
	STR	Store Register.
type	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.
cond	Is an optional condition code	
Rt	Specifies the register to load or store.	
Rn	Specifies the register on which the memory address is based.	
Rm	Specifies the register containing a value to be used as the offset.	
LSL #*n*	Is an optional shift, with *n* 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 sadržaja offsetnog 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]
```

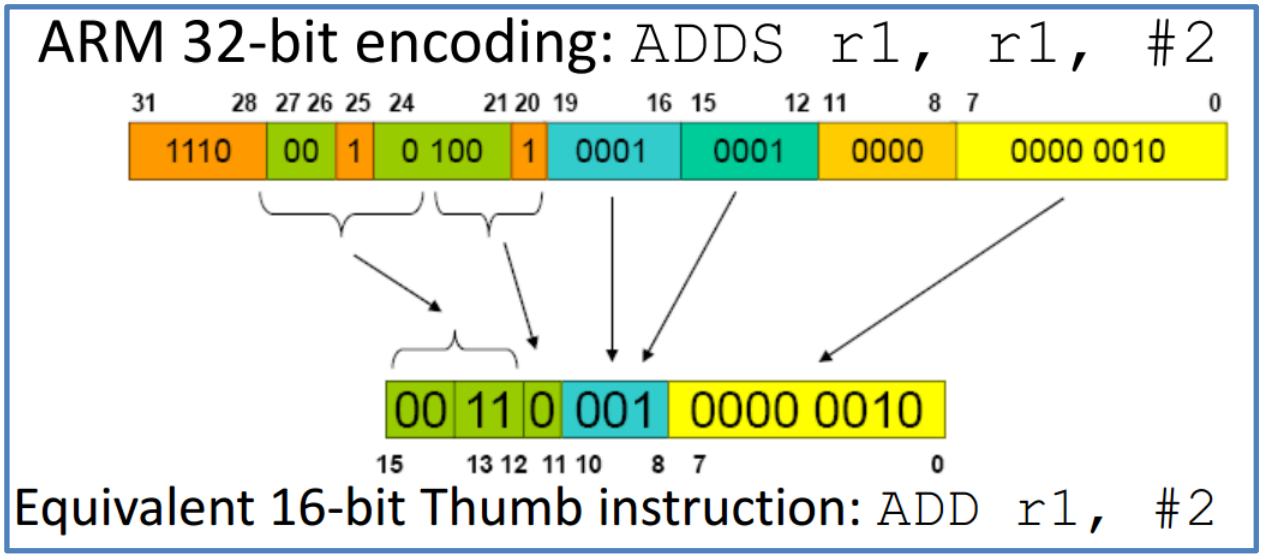
– Post-indexno:
  - Adresa se dobija samo na osnovu baznog registra, ali se nakon instrukcije na sadržaj baznog registra dodaje sadržaj izračunatog offset-a.
  - **[<Rn>], <offset>**

```
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:

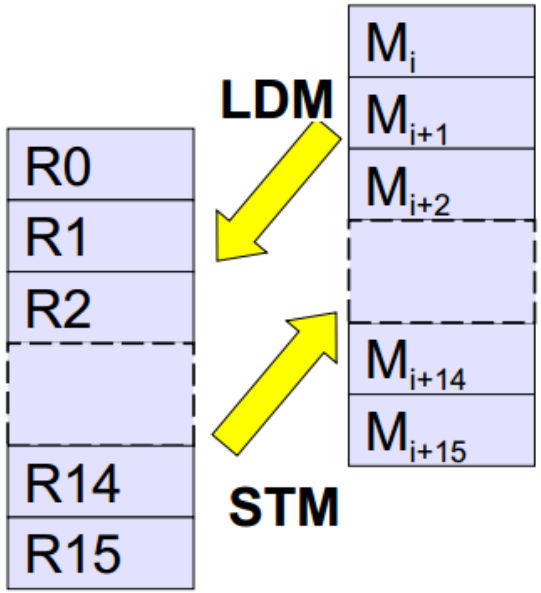
  LDR r0,[pc, offset_to_lit_pool]

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

  DCD 0x55555555

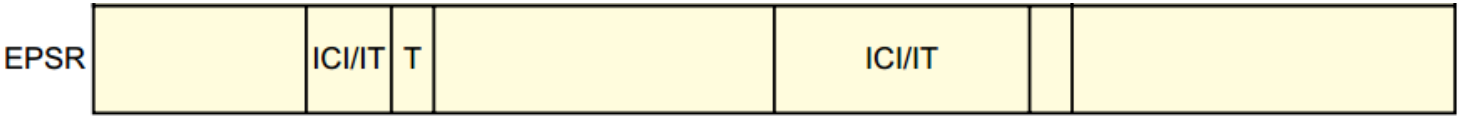
Instrukcije za prenos blokova podataka

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



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

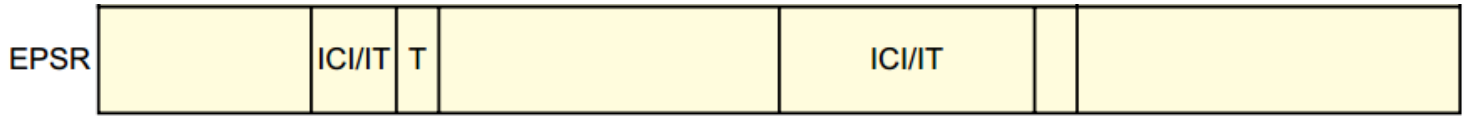
- Većini instrukcija može da se postavi dodatni uslov za izvršavanje koji zavisi od Z,N,V,C flag-ova
- Ova mogućnost je ograničena samo na instrukcije koje se nalaze u okviru takozvanih IT (If-Then blokova)
- Uslovna izvršavanja blokova su prekidiva što se obezbeđuje korišćenjem IT polja u statusnom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


ITTEE  EQ            ; Next 4 instructions are
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.
```

Sintaksa ITx instrukcije

IT{*x*{*y*{*z*}}} *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.

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 posebnom konfiguracionom pinu pri resetu procesora.
- Instukcijskom kodu se uvek pristupa u little-endian formatu.

- Little-Endian : LSB na nižoj adresi

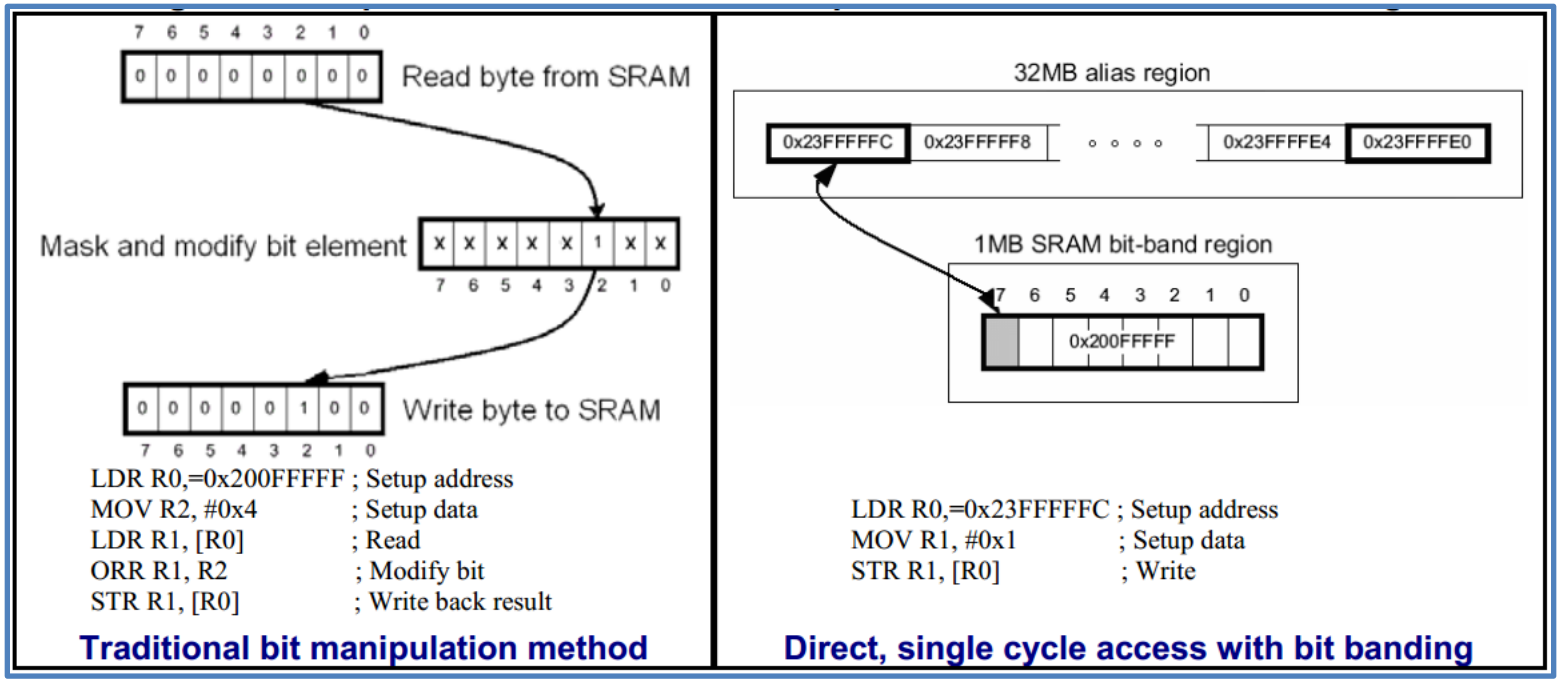
```
                                Memory     Value
                                Offset   (LSB) (MSB)
                                ======   ===========
uint8_t a  = 1;                 0x0000   01 02 FF 00
uint8_t b  = 2;
uint16_t c = 255; // 0x00FF
uint32_t d = 0x12345678;        0x0004   78 56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

- Big-Endian : MSB na nižoj adresi

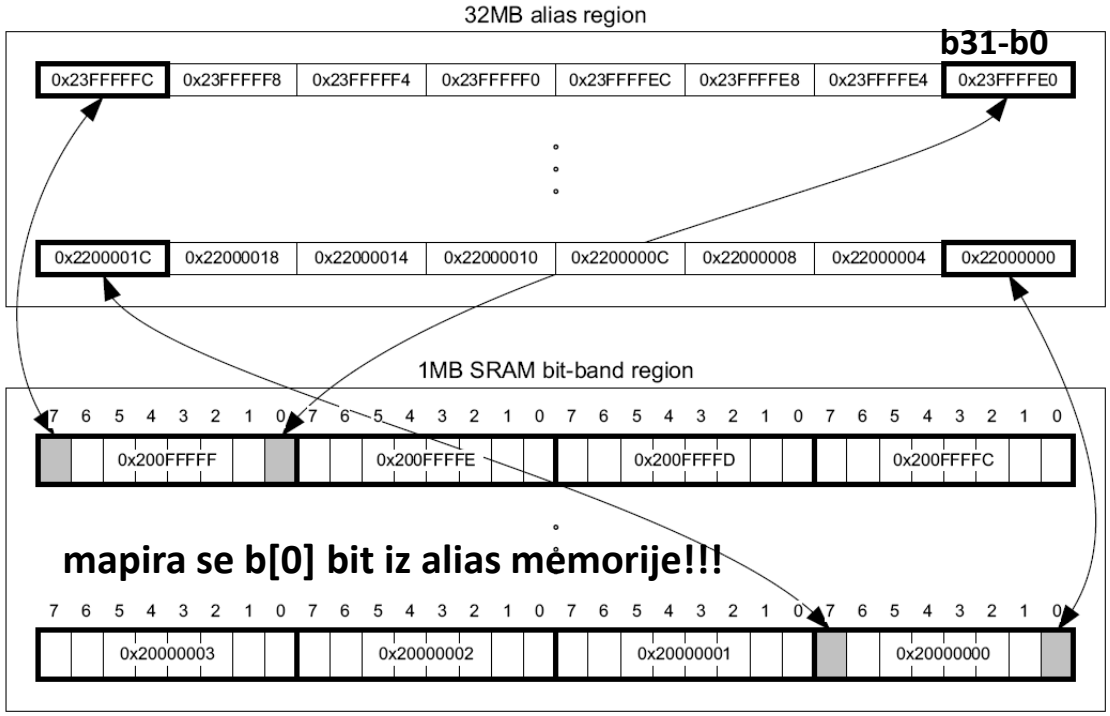
```
                                Memory     Value
                                Offset   (LSB) (MSB)
                                ======   ===========
uint8_t a  = 1;                 0x0000   01 02 00 FF
uint8_t b  = 2;
uint16_t c = 255; // 0x00FF
uint32_t d = 0x12345678;        0x0004   12 34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

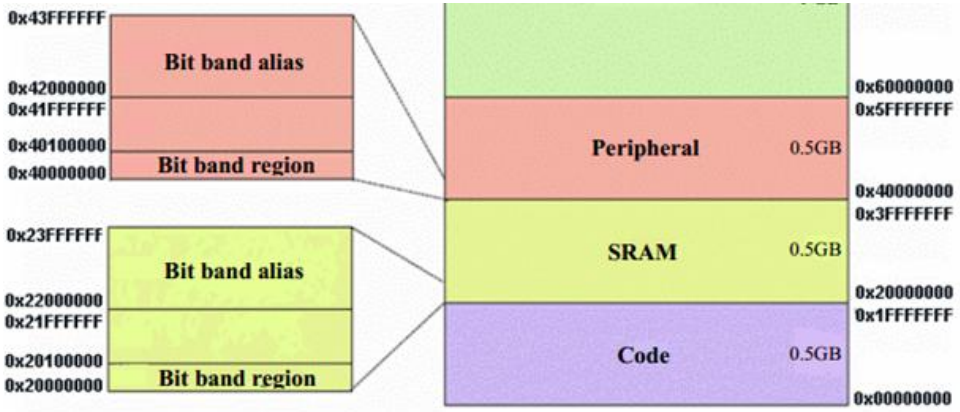
Bit-banding

- 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


32MB alias region
b31-b0
0x23FFFFFC
0x23FFFFF8
0x23FFFFF4
0x23FFFFF0
0x23FFFFEC
0x23FFFFE8
0x23FFFFE4
0x23FFFFE0
0x2200001C
0x22000018
0x22000014
0x22000010
0x2200000C
0x22000008
0x22000004
0x22000000
1MB SRAM bit-band region
7 6 5 4 3 2 1 0
0x200FFFFF
0x200FFFFE
0x200FFFFD
0x200FFFFC
mapira se b[0] bit iz alias mémorije!!!
0x20000003
0x20000002
0x20000001
0x20000000


0x43FFFFFF
Bit band alias
0x42000000
0x41FFFFFF
0x40100000
Bit band region
0x40000000
0x23FFFFFF
Bit band alias
0x22000000
0x21FFFFFF
0x20100000
Bit band region
0x20000000
Peripheral
0.5GB
SRAM
0.5GB
Code
0.5GB
0x60000000
0x5FFFFFFF
0x40000000
0x3FFFFFFF
0x20000000
0x1FFFFFFF
0x00000000

Bit-banding

- Upisivanje sa i bez bit-banding-a

Without bit-band

```
LDR    R0,=0x20000000 ; Setup address
LDR    R1, [R0]       ; Read
ORR.W  R1, #0x4       ; Modify bit
STR    R1, [R0]       ; Write back result
```

With bit-band

```
LDR    R0,=0x22000008 ; Setup add
MOV    R1, #1         ; Setup dat
STR    R1, [R0]       ; Write
```

Atomska operacija

- Očitavanje sa i bez bit-banding-a

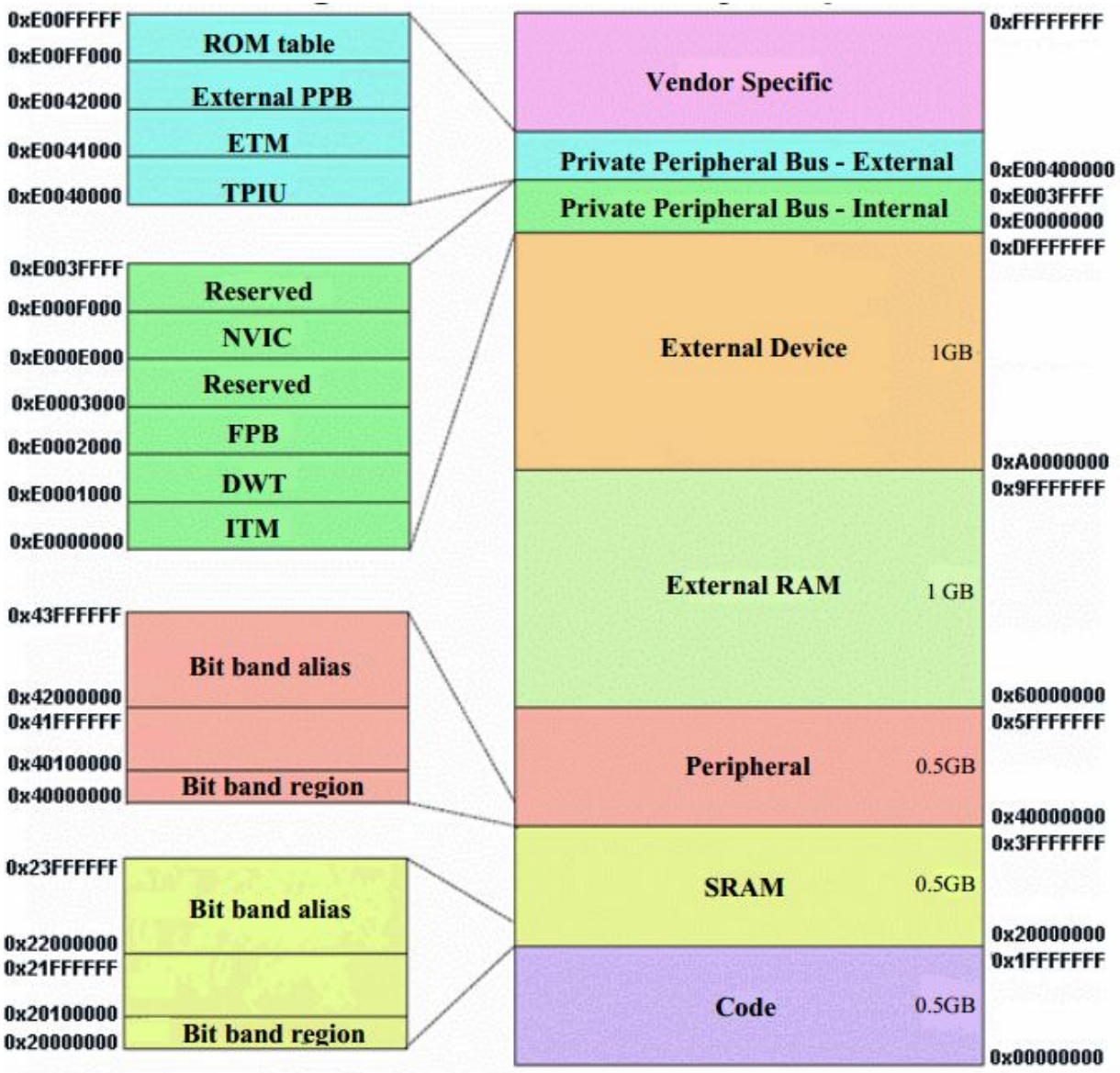
Without bit-band

```
LDR     R0,=0x20000000 ; Setup address
LDR     R1, [R0]       ; Read
UBFX.W  R1,  R1, #2, #1 ; Extract bit[2]
```

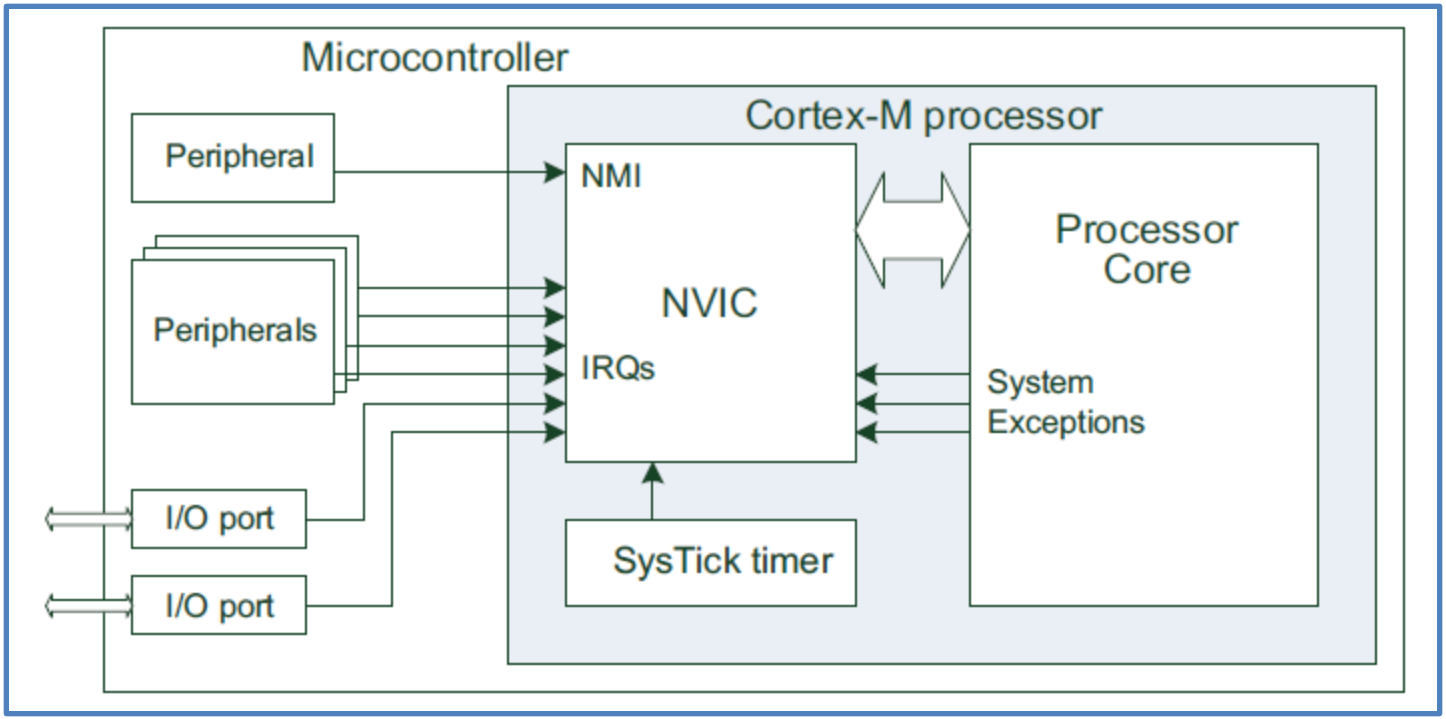
With bit-band

```
LDR    R0,=0x22000008 ; Setup address
LDR    R1, [R0]       ; Read
```

Mapa adresnog prostora



NVIC – Nested Vector Interrupt Controller

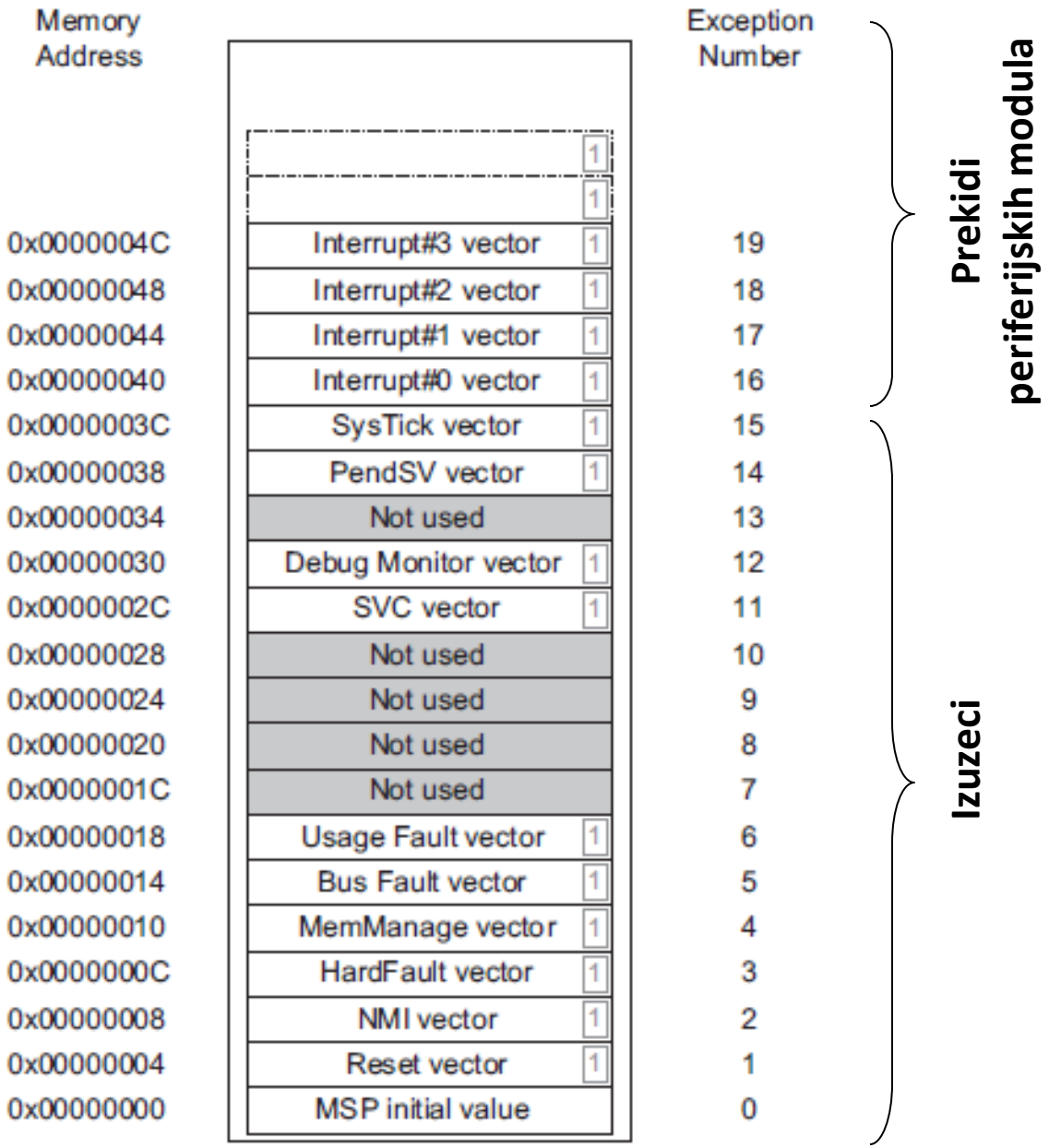


NVIC – Nested Vector Interrupt Controller

- 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

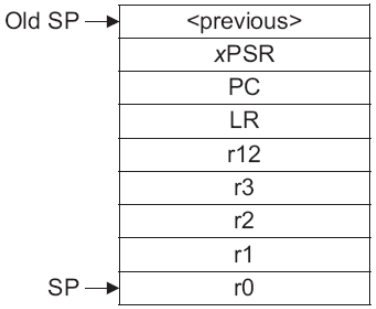
IVT – Interrupt Vector Table

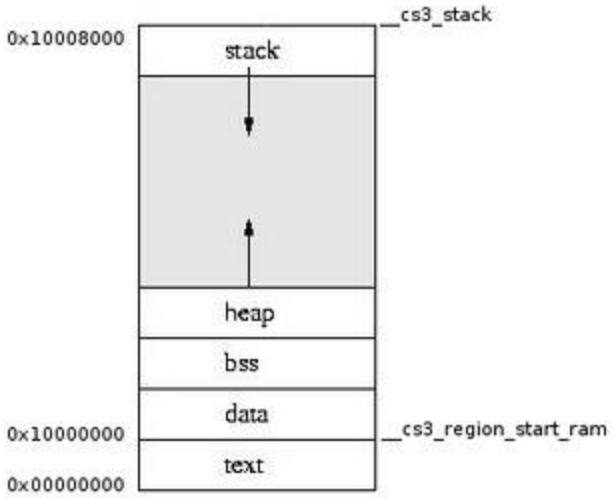
- **Mogućnost remapiranja tabele vektora** preko podešavanja registra ofseta tabele
- Adresa početka tabele poravnata na granicu 1024 bajta.



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

[1]GE bits updates.
[2]Q bit is set when saturation occurs.
[3]Each 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 >= 0x10000 then GE[1:0] = 2'b11 else GE[1:0] = 2'b00 If upper half-word result >= 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 >= 0x100 then GE[0] = 1'b1 else GE[0] = 1'b0 If byte 1 result >= 0x100 then GE[1] = 1'b1 else GE[1] = 1'b0 If byte 2 result >= 0x100 then GE[2] = 1'b1 else GE[2] = 1'b0 If byte 3 result >=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 >= 0x10000 then GE[3:2] = 2'b11 else GE[3:2] = 2'b00
USAX	If lower half-word result >= 0x10000 then GE[1:0] = 2'b11 else GE[1:0] = 2'b00 If upper half-word result >= 0x0 then GE[3:2] = 2'b11 else GE [3:2] = 2'b00

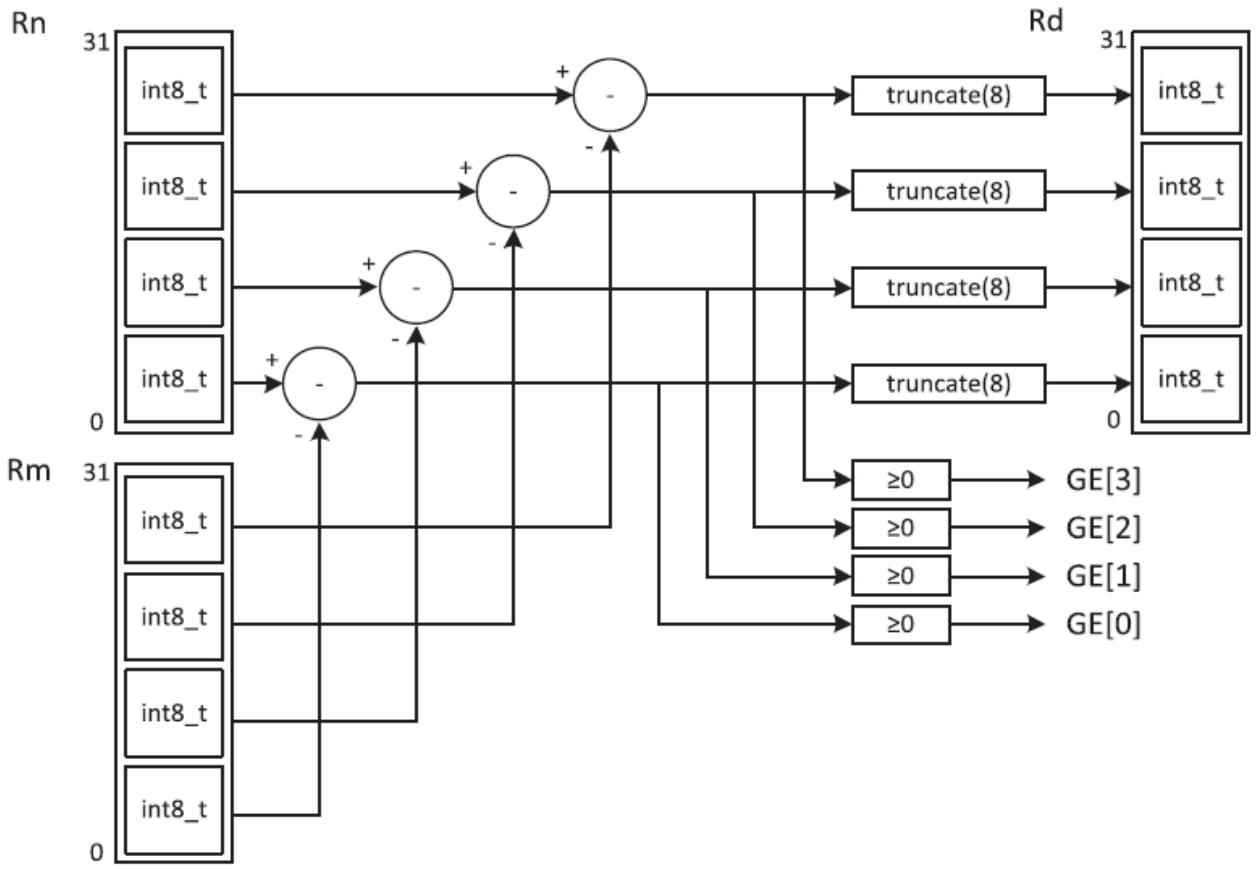
SIMD

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

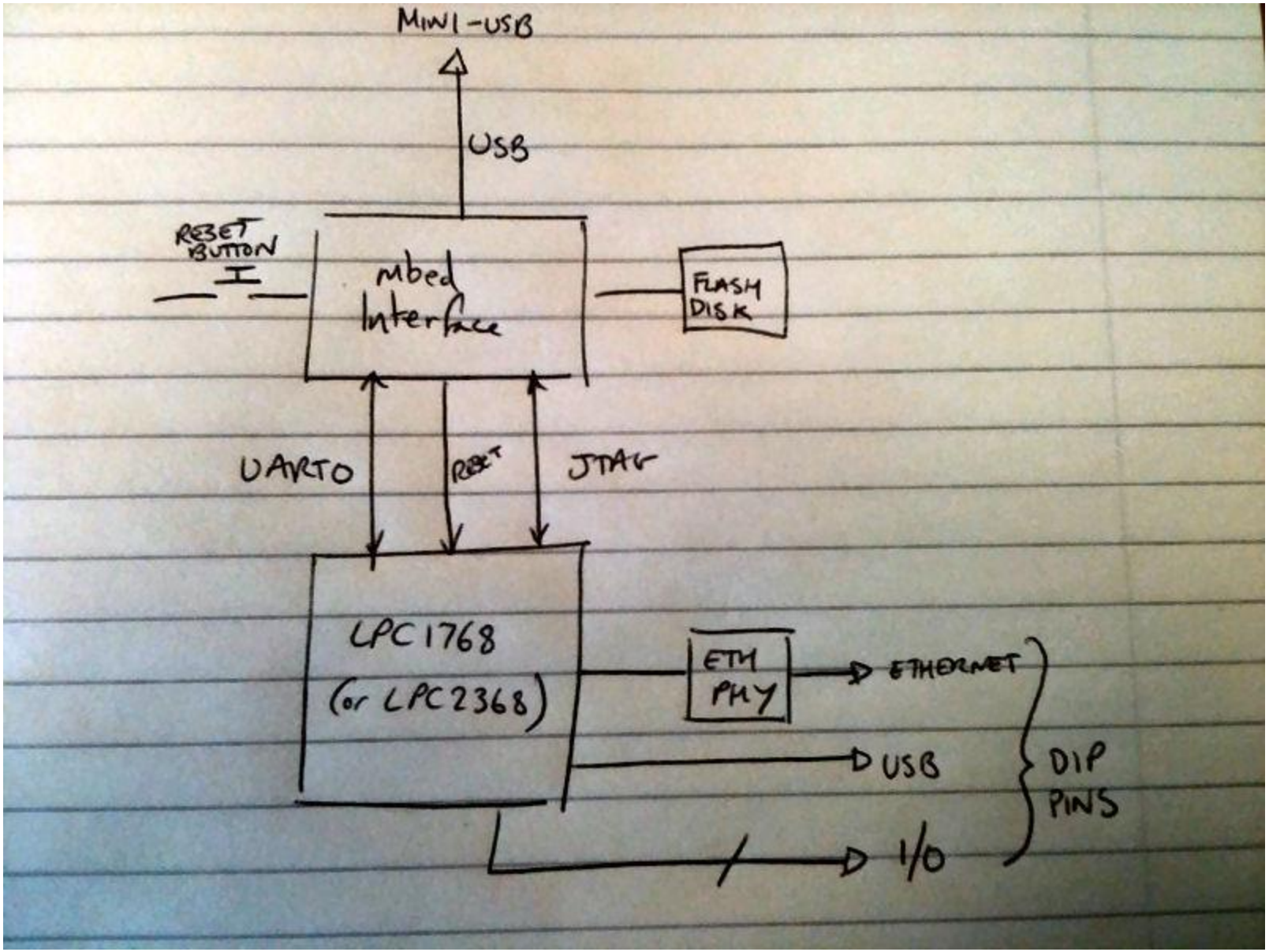
SSUB8 {<Rd>,} <Rn>, <Rm>



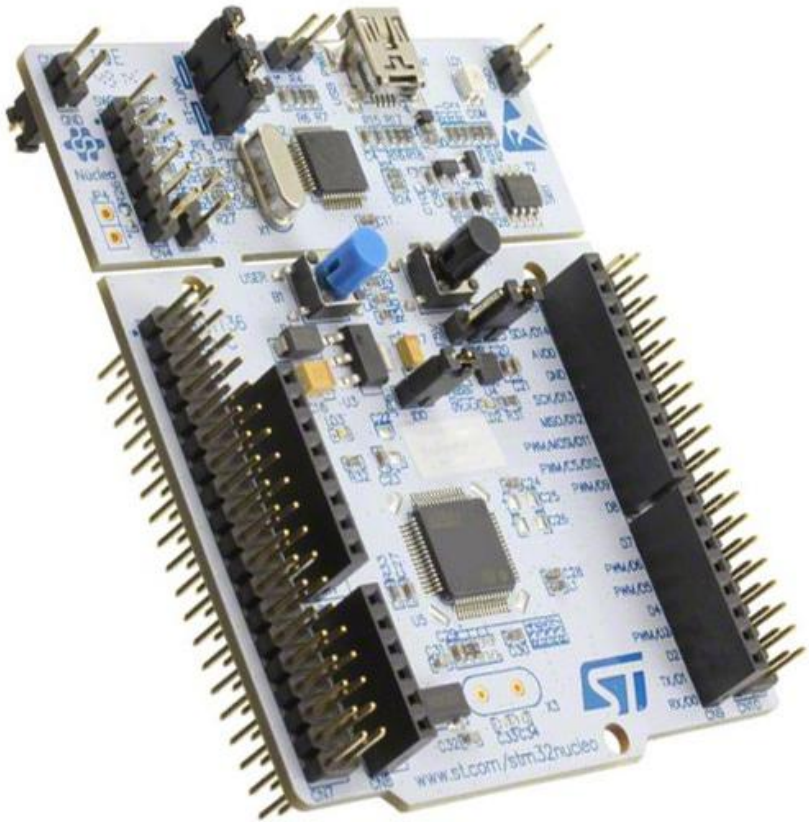
MBED platoforme

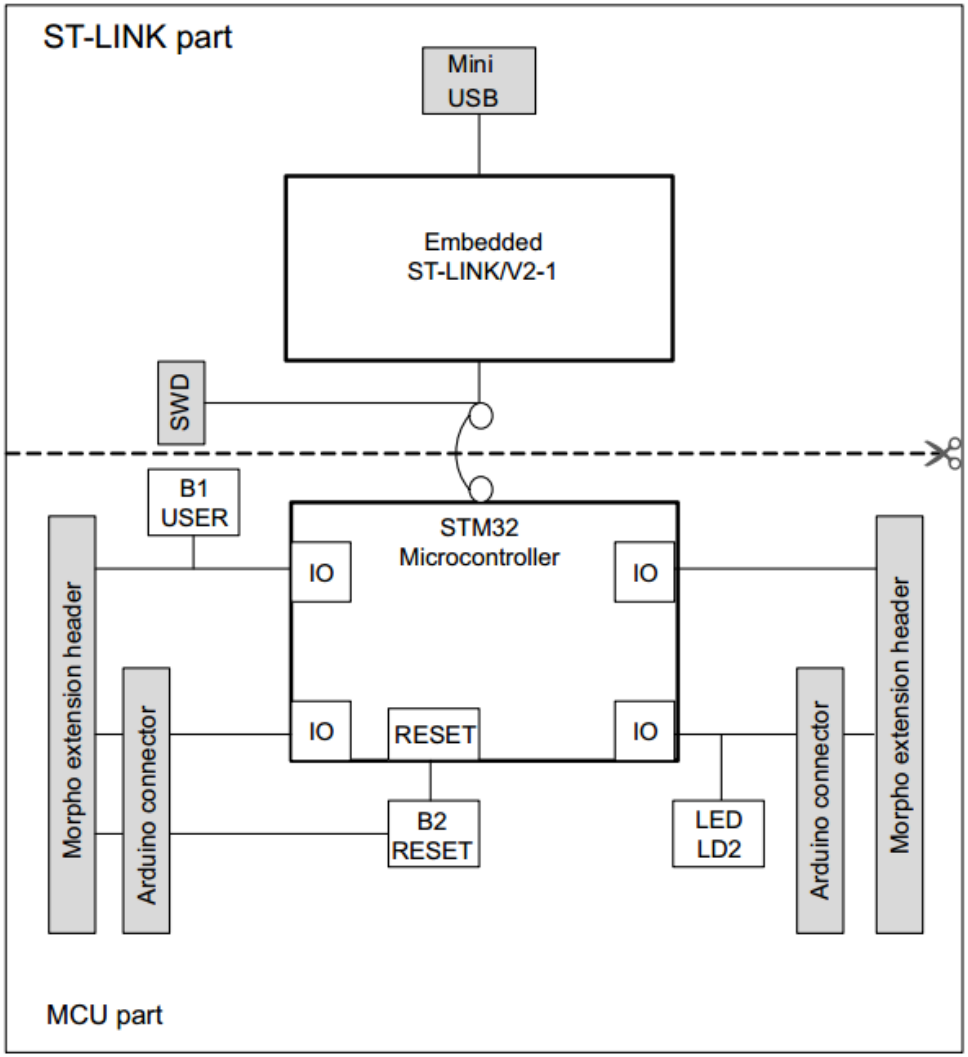
- Svakim danom broj MBED-enabled platformi raste

MBED -arhitektura

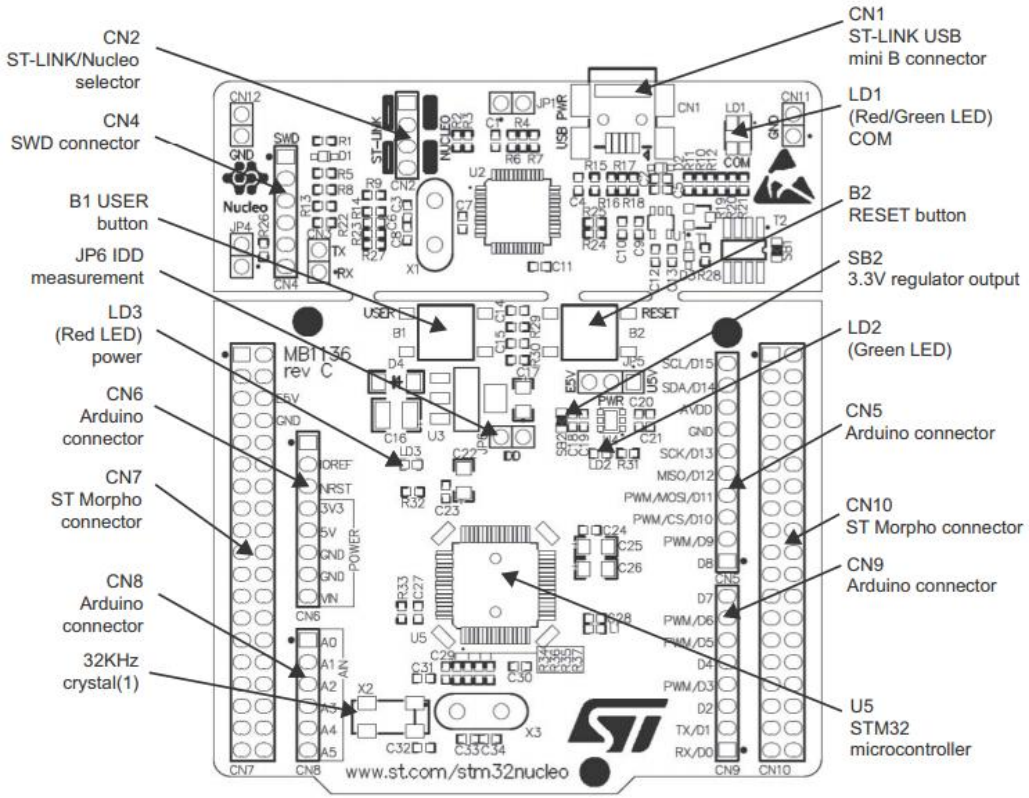


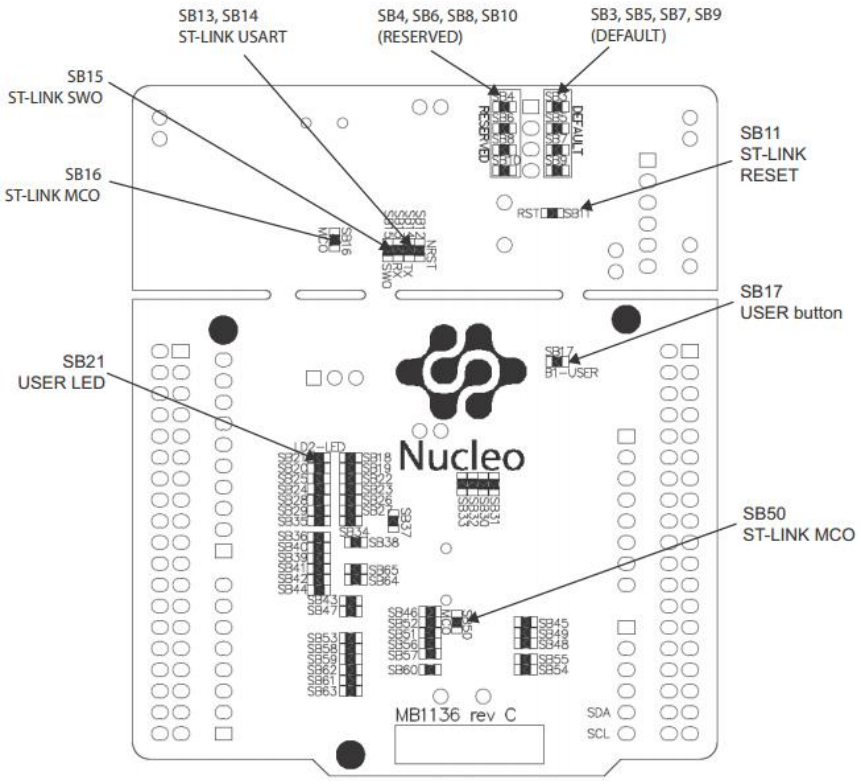
MBED - NUCLEO



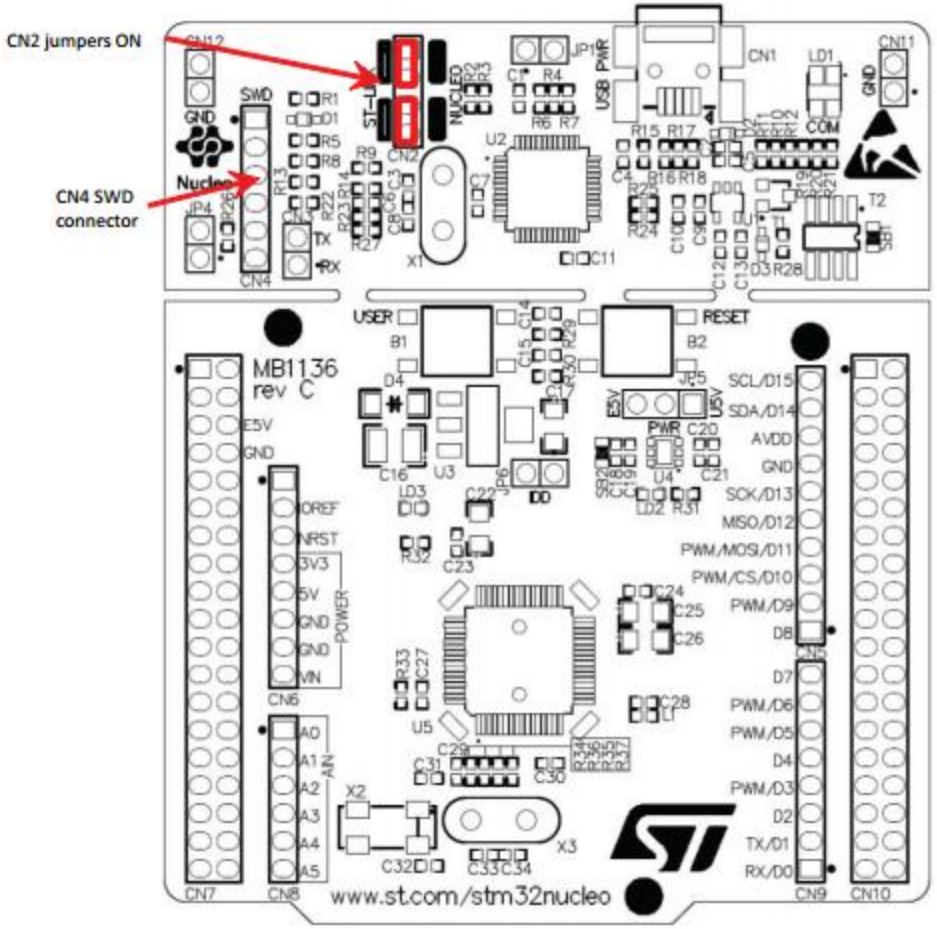

ST-LINK part
Mini USB
Embedded ST-LINK/V2-1
SWD
B1 USER
STM32 Microcontroller
IO
IO
IO
RESET
IO
B2 RESET
LED LD2
Morpho extension header
Arduino connector
Arduino connector
Morpho extension header
MCU part

Sta ima na plocici





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 µA/MHz run mode
- 4 µs 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)

- 3× 12-bit ADC 5 Msps, up to 16-bit with hardware oversampling, 200 µA/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 SAIs (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

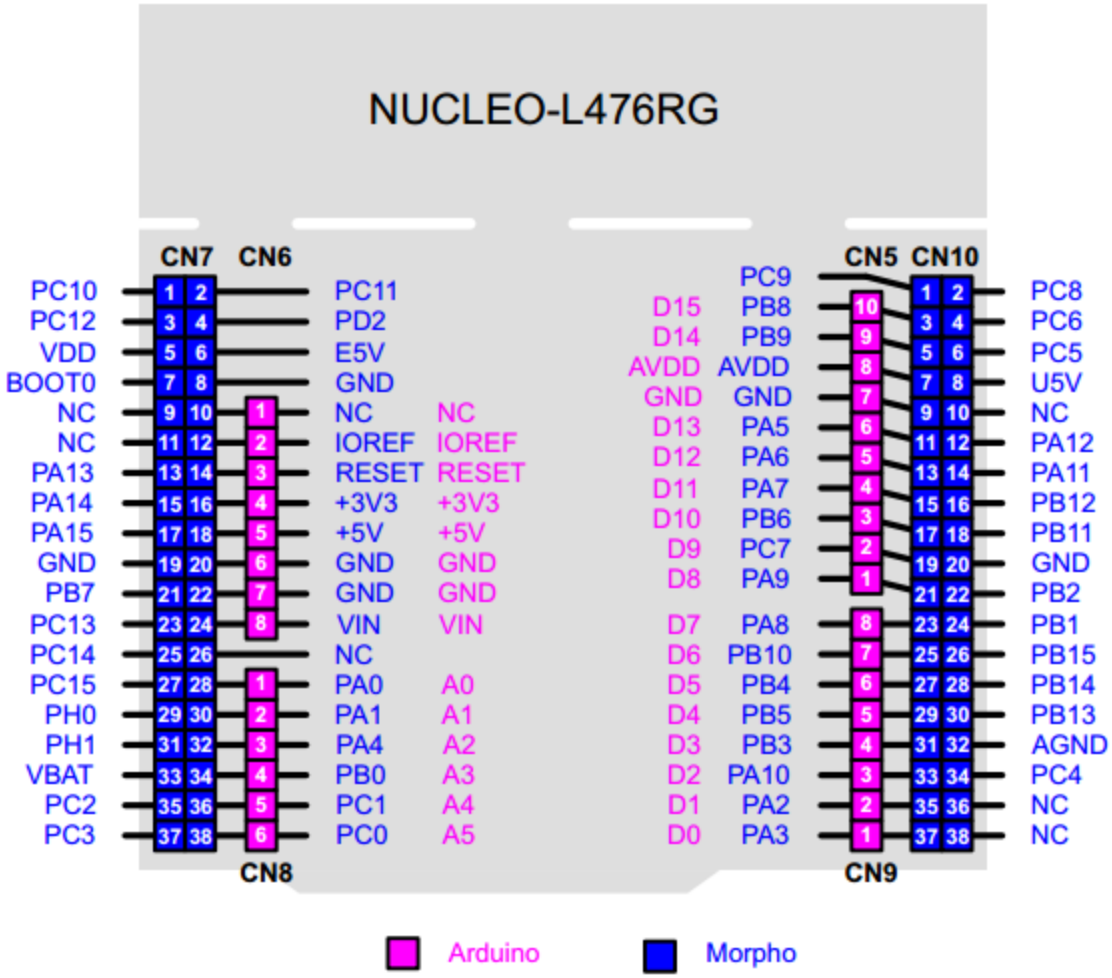
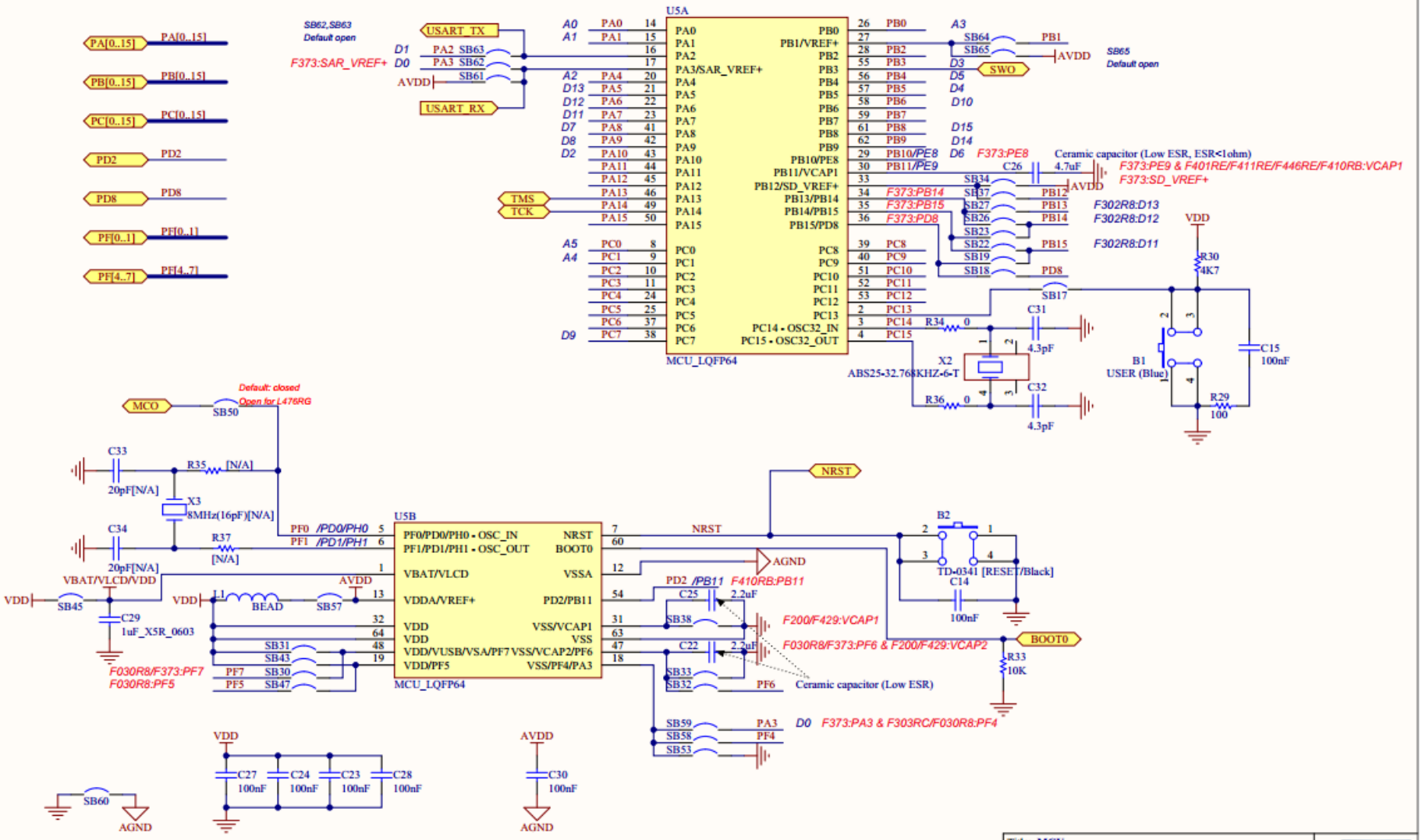
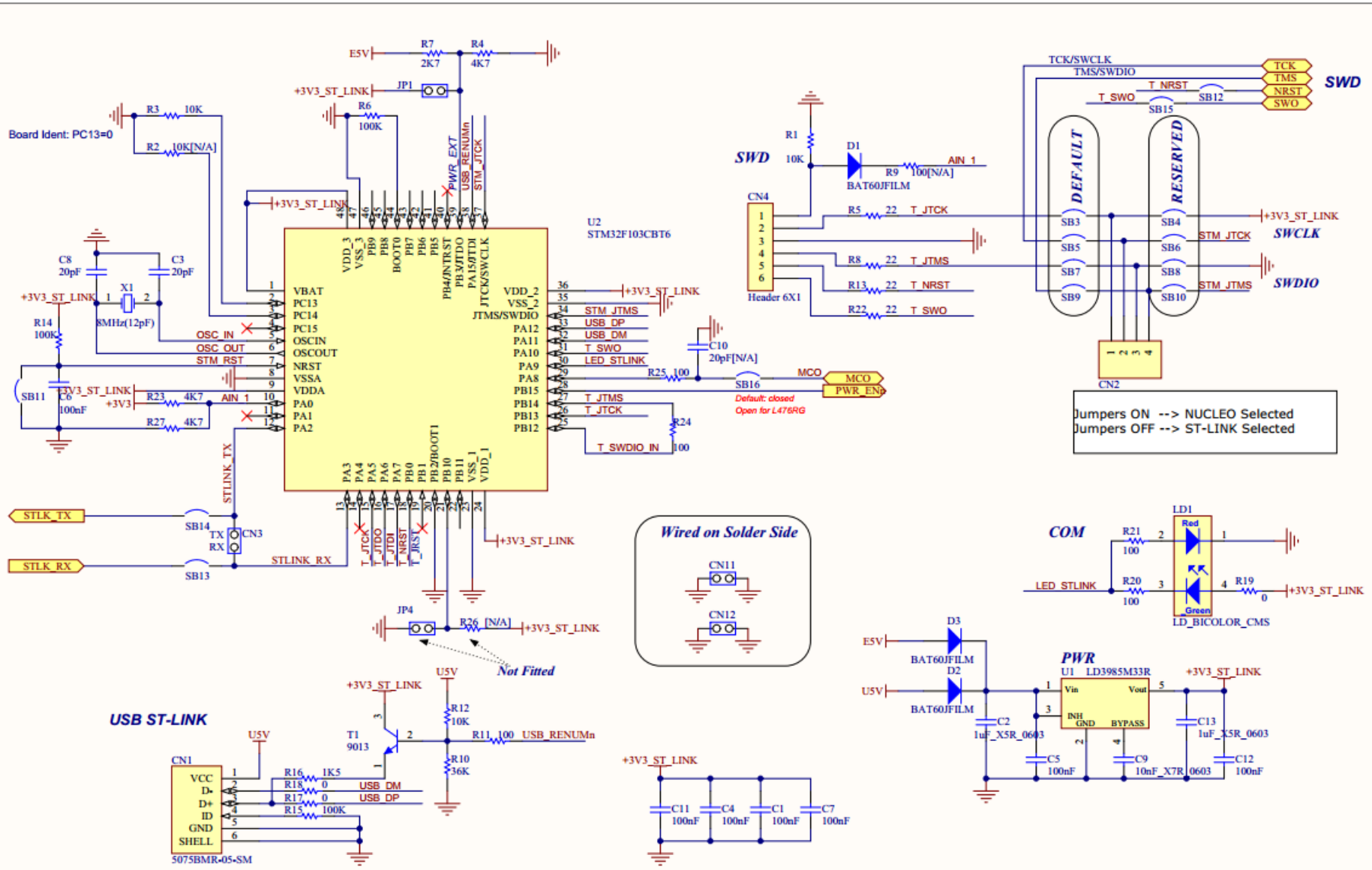


Table 29. STMicroelectronics Morpho connector on NUCLEO-L476RG

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

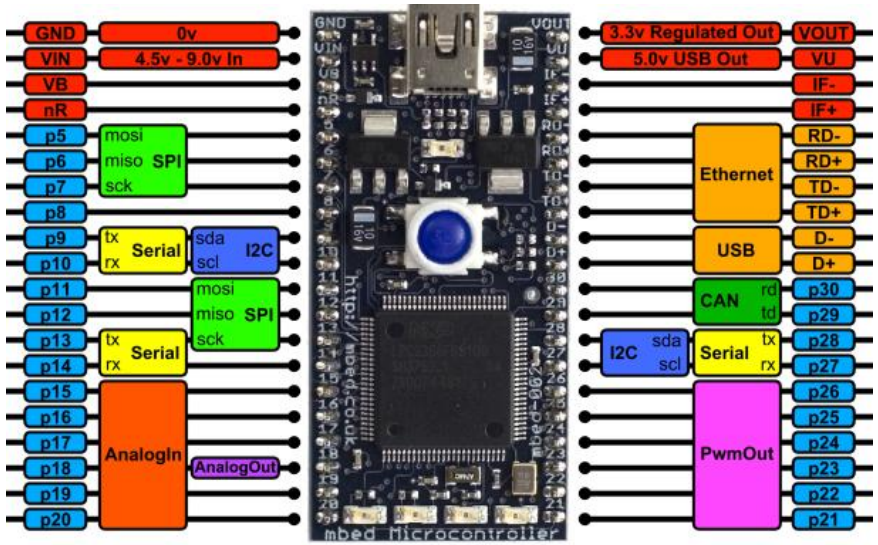

PA[0..15]
PB[0..15]
PC[0..15]
PD2
PD8
PF[0..1]
PF[4..7]
SB62,SB63
Default open
USART_TX
USART_RX
F373:SAR_VREF+
AVDD
TMS
TCK
U5A
MCU_LQFP64
SB65
Default open
SWO
Ceramic capacitor (Low ESR, ESR<1ohm)
4.7uF
F373:PE9 & F401RE/F411RE/F446RE/F410RB:VCAP1
F373:SD_VREF+
F302R8:D13
F302R8:D12
F302R8:D11
R30
4K7
C31
4.3pF
X2
ABS25-32.768KHZ-6-T
C32
4.3pF
R34
R36
B1
USER (Blue)
C15
100nF
R29
100
MCO
SB50
Default: closed
Open for L476RG
C33
20pF[N/A]
R35
[N/A]
X3
8MHz(16pF)[N/A]
C34
20pF[N/A]
R37
[N/A]
U5B
MCU_LQFP64
NRST
B2
TD-0341 [RESET/Black]
C14
100nF
BOOT0
R33
10K
VBAT/VLCD/VDD
SB45
C29
1uF_X5R_0603
BEAD
SB57
C25
2.2uF
C22
2.2uF
F200/F429:VCAP1
F030R8/F373:PF6 & F200/F429:VCAP2
Ceramic capacitor (Low ESR)
F030R8/F373:PF7
F030R8:PF5
SB60
AGND
C27
100nF
C24
100nF
C23
100nF
C28
100nF
C30
100nF
D0
F373:PA3 & F303RC/F030R8:PF4
Title: MCU
Project: NUCLEO-XXXXRX
Size: A4
Reference: MB1136
Revision: C.3
Date: 5/11/2015
Sheet: 2 of 4
life.augmented

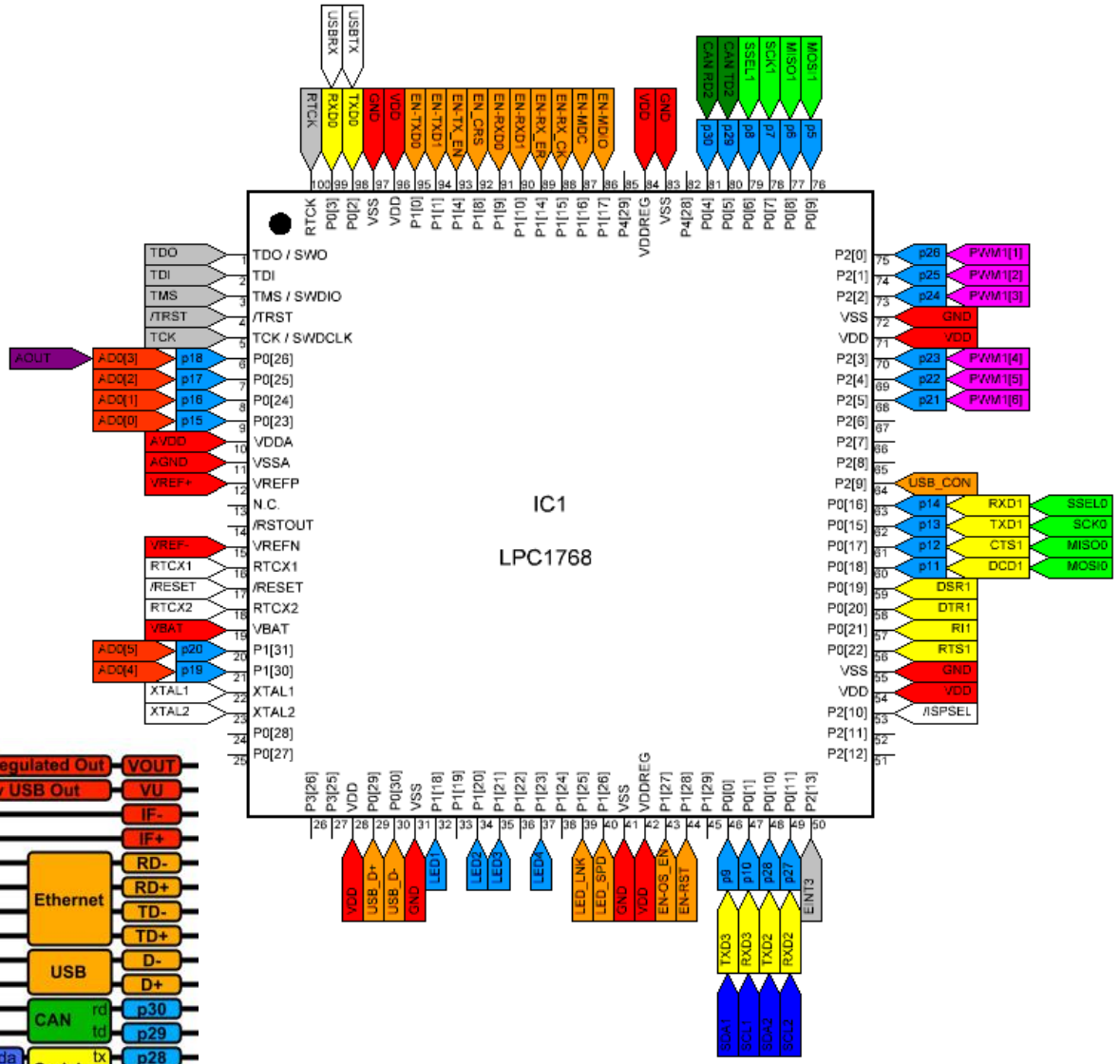


Title: STLINK/V2-1		
Project: NUCLEO-XXXXRX		
Size: A4	Reference: MB1136	Revision: C.3
Date: 12/9/2014		Sheet: 3 of 4

life.augmented

MBED i LPC1768





MBED i LPC1768

- Tabela prikazuje veze LPC1768 sa pinovima MBED-a.

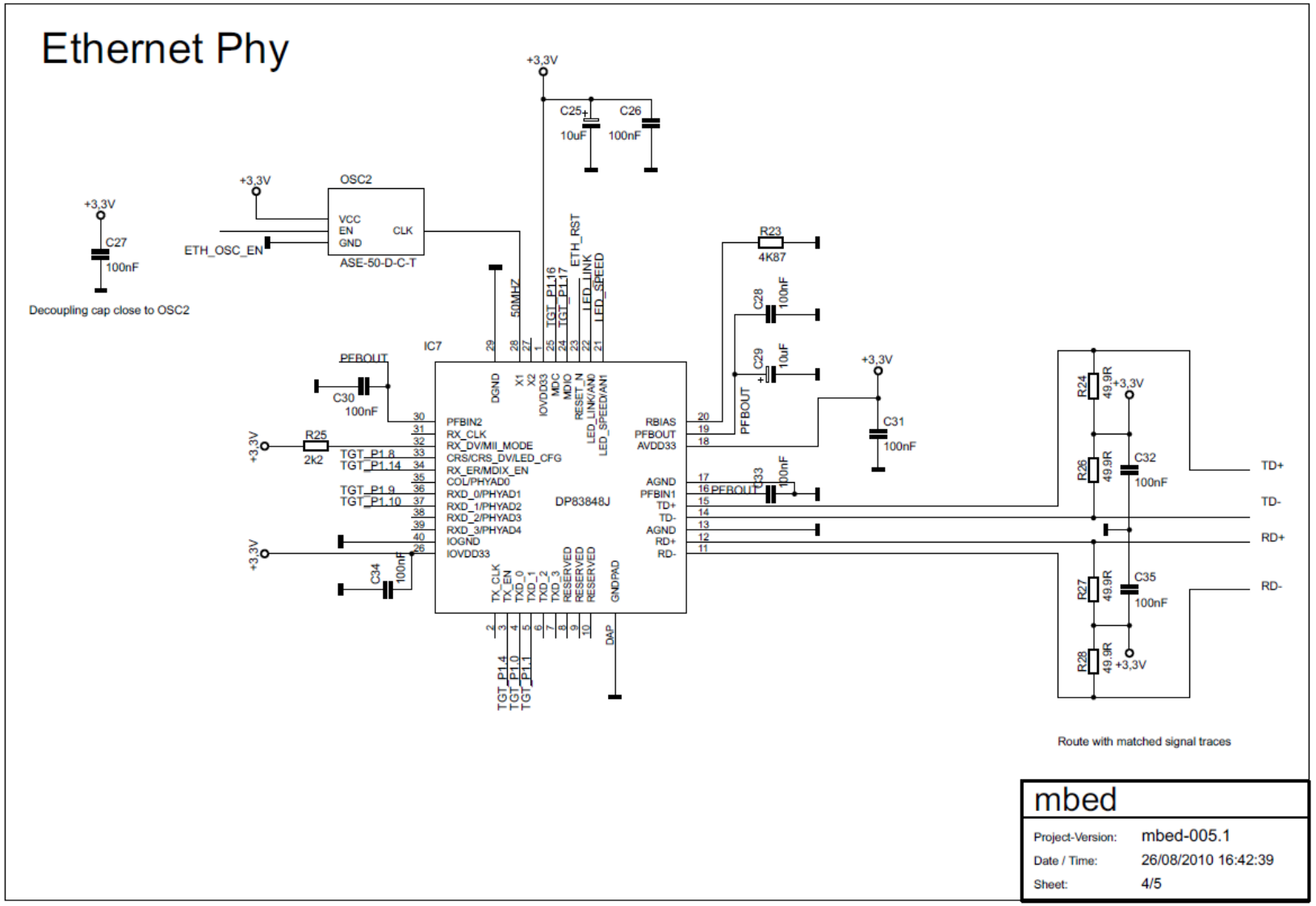
LPC1768 Pin functions

Pin	mbed Pin	Func 1	Func 2	Func 3	Func 4	Pin	mbed Pin	Func 1	Func 2	Func 3	Func 4
1		TDO	SWO			51		P2.12	EINT2N	I2STX_WS	
2		TDI				52		P2.11	EINT1N	I2STX_CLK	
3		TMS	SWDIO			53		P2.10	EINT0N	NMI	
4		nTRST				54		VDD			
5		TCK	SWDCLK			55		VSS			
6	p18	P0.26	AD0.3	AOUT	RXD3	56		P0.22	RTS1	CAN_TX1	
7	p17	P0.25	AD0.2	I2SRX_SDA	TXD3	57		P0.21	RI1	CAN_RX1	
8	p16	P0.24	AD0.1	I2SRX_WS	CAP3.1	58		P0.20	DTR1	SCL1	
9	p15	P0.23	AD0.0	I2SRX_CLK	CAP3.0	59		P0.19	DSR1	MCICLK	SDA1
10		VDDA				60	p11	P0.18	DCD1	MOSI0	MOSI
11		VSSA				61	p12	P0.17	CTS1	MISO0	MISO
12		VREFP				62	p13	P0.15	TXD1	SCK0	SCK
13		N.C.				63	p14	P0.16	RXD1	SSEL0	SSEL
14		nRSTOUT				64		P2.9	USB_CONNECT	RXD2	
15		VREFN				65		P2.8	CAN_TX2	TXD2	
16		RTCX1				66		P2.7	CAN_RX2	RTS1	
17		nRESET				67		P2.6	PCAP1.0	RI1	TRACECLK
18		RTCX2				68	p21	P2.5	PWM1.6	DTR1	TRACEDATA0
19		VBAT				69	p22	P2.4	PWM1.5	DSR1	TRACEDATA1
20	p20	P1.31	SCK1	AD0.5		70	p23	P2.3	PWM1.4	DCD1	TRACEDATA2
21	p19	P1.30	VBUS	AD0.4		71		VDD			
22		XTAL1				72		VSS			
23		XTAL2				73	p24	P2.2	PWM1.3	CTS1	TRACEDATA3
24		P0.28	SCL0	USB_SCL		74	p25	P2.1	PWM1.2	RXD1	
25		P0.27	SDA0	USB_SDA		75	p26	P2.0	PWM1.1	TXD1	TRACECLK

Pin	mbed Pin	Func 1	Func 2	Func 3	Func 4	Pin	mbed Pin	Func 1	Func 2	Func 3	Func 4
26		P3.26	STCLK	MAT0.1	PWM1.3	76	p5	P0.9	I2STX_SDA	MOSI1	MAT2.3
27		P3.25	MAT0.0	PWM1.2		77	p6	P0.8	I2STX_WS	MISO1	MAT2.2
28		VDD				78	p7	P0.7	I2STX_CLK	SCK1	MAT3.1
29	USBD+	P0.29	USB_D+			79	p8	P0.6	I2SRX_SDA	SSEL1	MAT2.0
30	USBD-	P0.30	USB_D-			80	p29	P0.5	I2SRX_WS	CAN_TX2	CAP2.1
31		VSS				81	p30	P0.4	I2SRX_CLK	CAN_RX2	CAP2.0
32	LED1	P1.18	USB_UP	PWM1.1	CAP1.0	82		P4.28	RX_MCLK	MAT2.0	TXD3
33		P1.19	MC0A	USB_PPWR	CAP1.1	83		VSS			
34	LED2	P1.20	MCFB0	PWM1.2	SCK0	84		VDD			
35	LED3	P1.21	MCABORT	PWM1.3	SSEL0	85		P4.29	TX_MCLK	MAT2.1	RXD3
36		P1.22	MC0B	USB_PWRD	MAT1.0	86		P1.17	ENET-MDIO		
37	LED4	P1.23	MCFB1	PWM1.4	MISO0	87		P1.16	ENET-MDC		
38		P1.24	MCFB2	PWM1.5	MOSI0	88		P1.15	ENET-RX_CLK		
39		P1.25	MC1A	MAT1.1		89		P1.14	ENET-RC_ER		
40		P1.26	MC1B	PWM1.6	CAP0.0	90		P1.10	ENET_RXD1		
41		VSS				91		P1.9	ENET_RXD0		
42		VDD				92		P1.8	ENET_CRS		
43		P1.27	CLKOUT	USB_OVRCR	CAP0.1	93		P1.4	ENET-TX_EN		
44		P1.28	MC2A	PCAP1.0	MAT0.0	94		P1.1	ENET-TXD1		
45		P1.29	MC2B	PCAP1.1	MAT0.1	95		P1.0	ENET-TXD0		
46	p9	P0.0	CAN_RX1	TXD3	SDA1	96		VDD			
47	p10	P0.1	CAN_TX1	RXD3	SCL1	97		VSS			
48	p28	P0.10	TXD2	SDA2	MAT3.0	98		P0.2	TXD0	AD0.7	
49	p27	P0.11	RXD2	SCL2	MAT3.1	99		P0.3	RXD0	AD0.6	
50		P2.13	EINT3N	I2STX_SDA		100		RTCK			

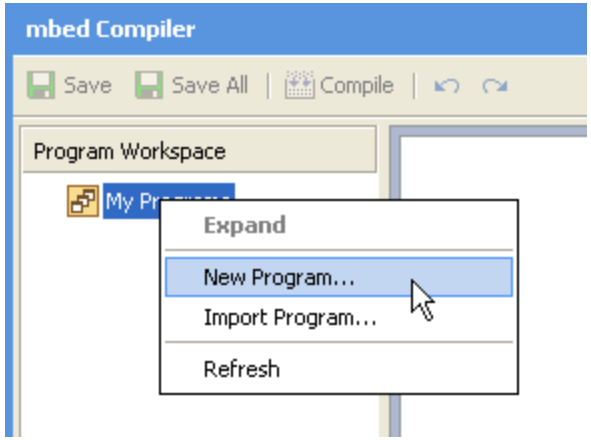
Interruptable

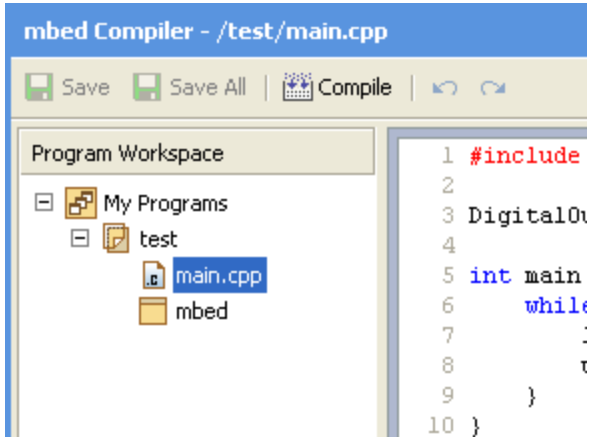
Ethernet chipset



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

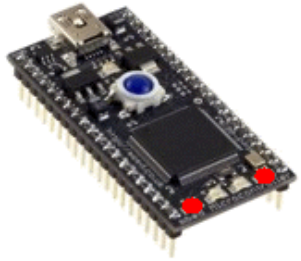
Debagovanje

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);
      }
   }
}
```

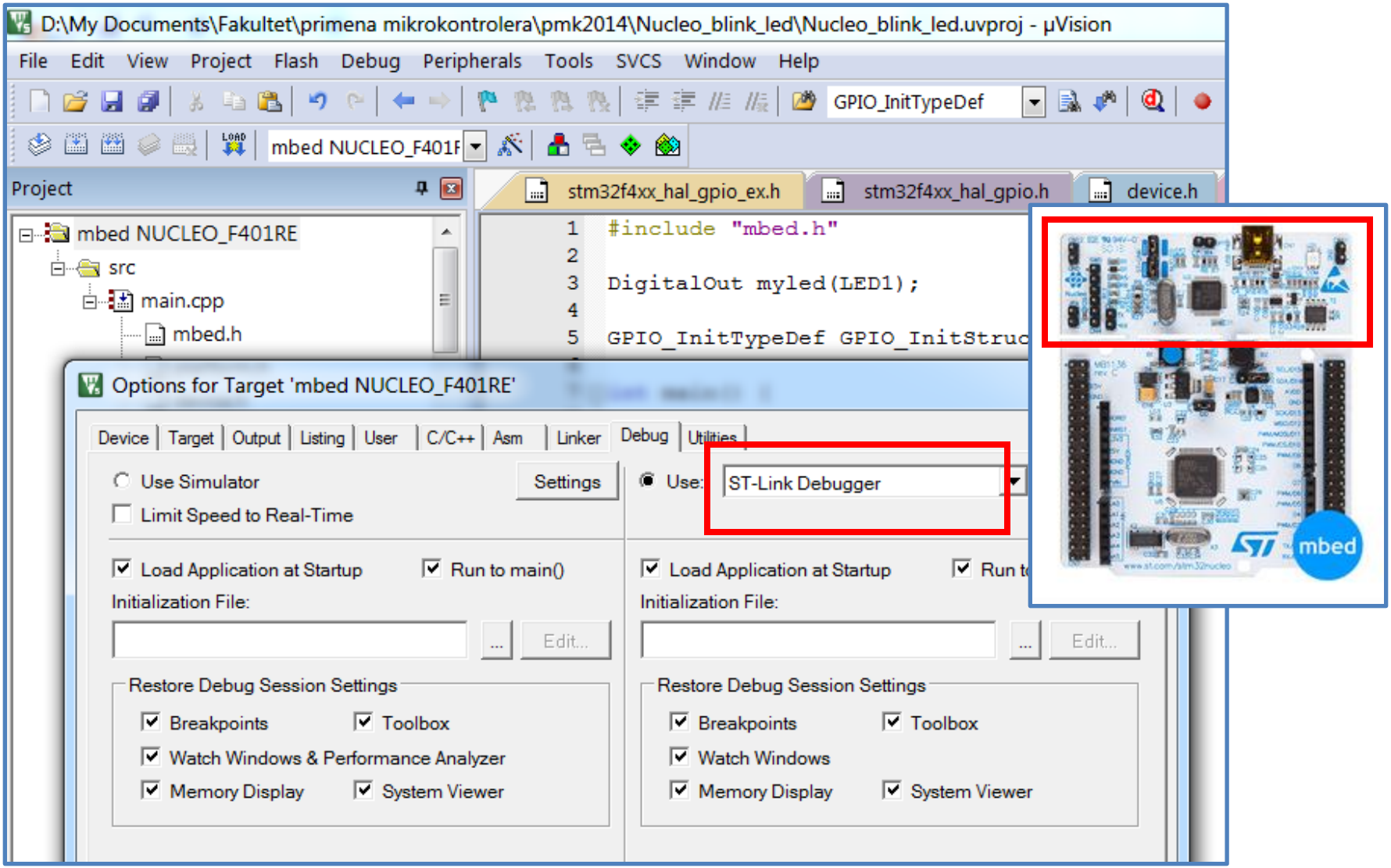


Efekat koji će se javiti je specifična signalizacija LE dioda.

Debagovanje funkcionalnosti – Online Compiler

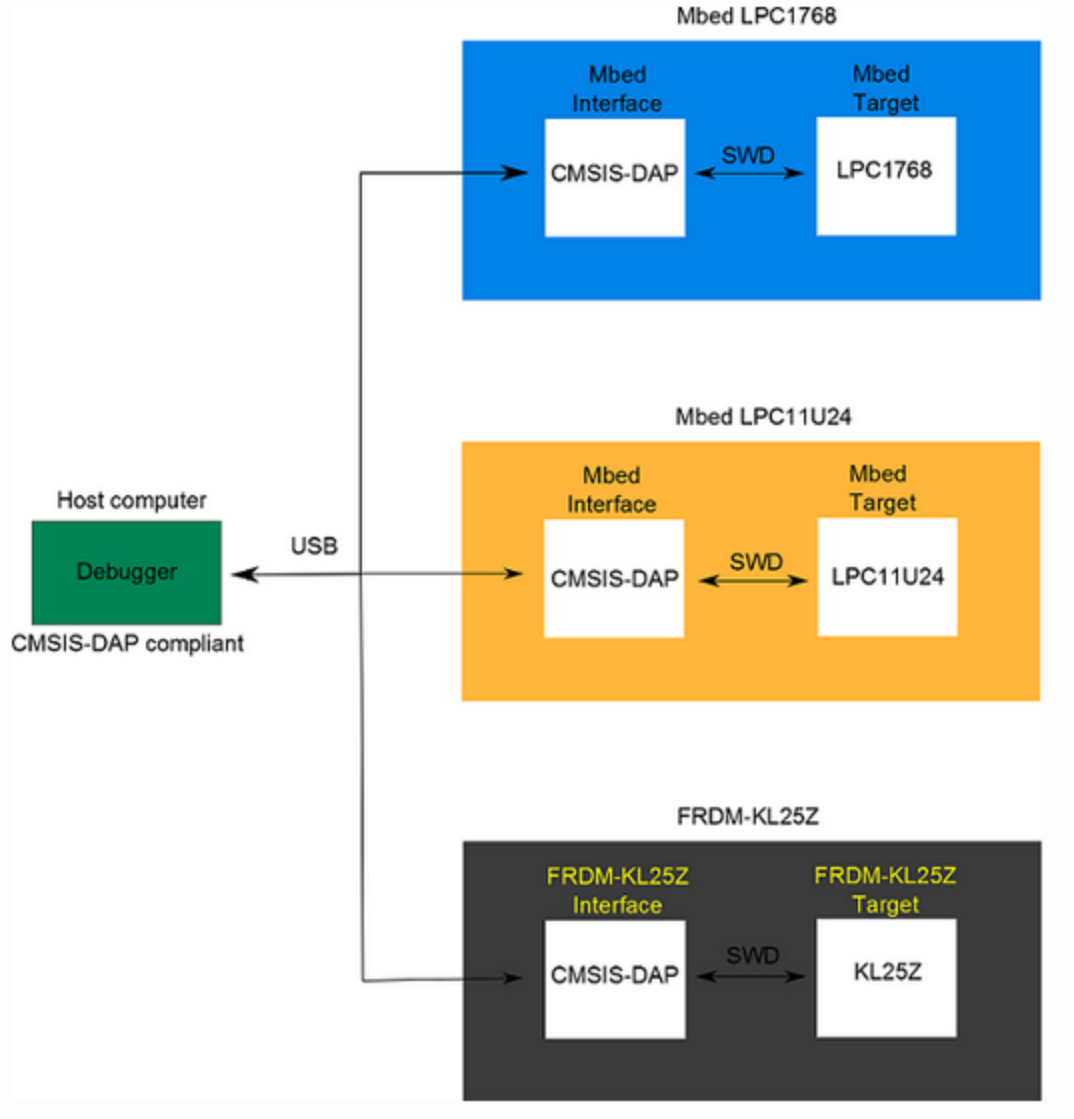
- U slučaju korišćenja ONLINE kompjalera, MBED nema mogućnost prave podrške za real-time debugging kao što to recimo ima MSP430 korišćenjem FET debugger-a, i to je jedna od većih mana MBED-a.
- Nemoguće je zaustaviti izvršavanje programa u nekom trenutku niti postaviti breakpoint. Nemoguće je i očitati proizvoljan sadržaj memorije niti registara mikrokontrolera.
- U nedostatku boljeg koriste se dva odavno poznata načina debuggovanja.
- Jedan je korišćenje neke od 4 LE diode na samom MBED-u. Ideja je da se različita stanja korisničkog porgrama 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” debagovanje - STM32 – Nucleo



“Pravo” debagovanje - LPC

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



Eksportovanje prema eksternom debugger-u