Primer dela s preprostimi V/I napravami na mikrokrmilniku

ST (STMicroelectronics) je največje evropsko podjetje za načrtovanje in izdelavo polprevodniških čipov (nastalo je z združitvijo podjetij SGS in Thomson).
STM32 je družina mikrokrmilnikov, osnovana na 32-bitni RISC arhitekturi ARM Cortex-M.

Uporabimo razvojno okolje STM32CubeIDE.
Naredimo nov STM32 projekt. Dobimo Target selection, kjer lahko navedemo, kateri mikrokrmilnik (MCU) bomo uporabili (STM32F407VG, ki je na naši ploščici Discovery).
Med ponujenimi opcijami je tudi ploščica (Board) STM32F407G-DISC1 (Discovery), ki jo izberemo.
Navedemo ime projekta in tip (exe in stm32).
Proj1.ioc nam ponudi grafično konfiguracijo. V kategoriji SystemCore imamo že omogočene RCC (clock) in SYS (serial wire debug).
Pri GPIO (General-Purpose Inputs/Outputs) izberemo LED diode PD12 do PD15, ki spadajo v skupino GPIOD (GPIO je razdeljen v več skupin: GPIOA, GPIOB, …)
Vhodno-izhodne naprave so pomnilniško mapirane in jih torej lahko beremo in vanje pišemo podobno kot v glavni pomnilnik – torej z ukazi load in store in ne potrebujemo posebnih ukazov, kot npr. na Intelovih procesorjih.

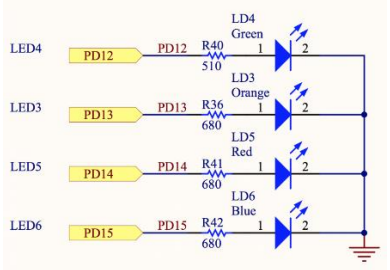
	0x4002 2400 - 0x4002 2FFF	Reserved
	0x4002 2000 - 0x4002 23FF	GPIOI
	0x4002 1C00 - 0x4002 1FFF	GPIOH
	0x4002 1800 - 0x4002 1BFF	GPIOG
	0x4002 1400 - 0x4002 17FF	GPIOF
	0x4002 1000 - 0x4002 13FF	GPIOE
	0X4002 0C00 - 0x4002 0FFF	GPIOD
	0x4002 0800 - 0x4002 0BFF	GPIOC
	0x4002 0400 - 0x4002 07FF	GPIOB
	0x4002 0000 - 0x4002 03FF	GPIOA
	0x4001 5800- 0x4001 FFFF	Reserved

Naslov 0x40020000 je bazni (začetni) naslov GPIO naprav, na naslovu 0x40020c00 pa se začne področje registrov za GPIOD. Prvi register določa način delovanja (mode): po dva bita povesta za vsak GPIO pin, ali naj bo vhod (00), izhod (01), alternativna funkcija (10) ali analogni vhod (11).

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
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
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O direction mode.
00: Input (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

Za LED diode je treba določiti način izhod, torej pari za pine 12 do 15 morajo biti MODER15..12 postavljeni na 01.



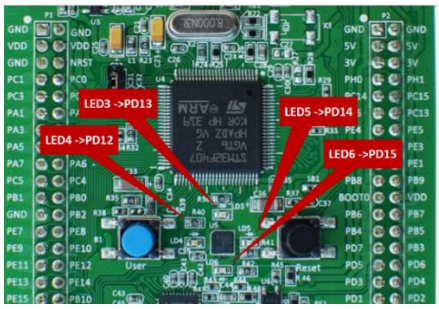
Lahko izberemo še modro tipko (push button), ki mora biti vhod.

main.c:

```
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
  uint32_t odr;

  /* Check the parameters */
  assert_param(IS_GPIO_PIN(GPIO_Pin));

  /* get current Output Data Register value */
  odr = GPIOx->ODR;

  /* Set selected pins that were at low level, and reset ones that were high */
  GPIOx->BSRR = ((odr & GPIO_Pin) << GPIO_NUMBER) | (~odr & GPIO_Pin);
}

  HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin);

int main(void)
{
  int tipka, dly = 1000;
  /* MCU Configuration--------------------------------------------------------*/
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_I2C1_Init();
  MX_I2S3_Init();
  MX_SPI1_Init();
  MX_USB_HOST_Init();

  while (1)
  {
          HAL_GPIO_TogglePin(LD3_GPIO_Port, LD3_Pin);
          HAL_GPIO_TogglePin(LD4_GPIO_Port, LD4_Pin);
          HAL_GPIO_TogglePin(LD5_GPIO_Port, LD5_Pin);
          HAL_GPIO_TogglePin(LD6_GPIO_Port, LD6_Pin);

          tipka = HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin);
          if (tipka)
                  dly = 200;
          else
                  dly = 1000;
          HAL_Delay(dly); // ms

    MX_USB_HOST_Process();

  }
}
```

Ko začnemo s prevajanjem (kar Build, konfiguracija Debug, ikona kladiva), vidimo, da je uporabljen prevajalnik *arm-none-eabi-gcc*, torej gcc za ARM bare-metal (brez operacijskega sistema).

Poženemo še Debug (ikona hrošča). Če se program ustavi pri main, izberemo Resume, da nadaljuje. Po potrebi lahko postavljamo breakpointe in potem program teče do tja. Nato lahko gremo po korakih (Step into, Step over, itd).
V primeru našega programa bodo na ploščici 4 LED diode spreminjale stanje vsako sekundo, če pa pritisnemo modro tipko, pa na vsakih 200 ms.
Na koncu končamo razhroščevanje s Terminate (oz. rdeč kvadrat).



Lahko pa postavimo breakpoint na

```
HAL_GPIO_TogglePin(LD3_GPIO_Port, LD3_Pin);
```

in pogledamo disassembly:

```
080004f1:   bl      0x800744c <MX_USB_HOST_Init>
 55             HAL_GPIO_TogglePin(LD3_GPIO_Port, LD3_Pin);
080004f4:   mov.w   r1, #8192       ; 0x2000
080004f8:   ldr     r0, [pc, #76]   ; (0x8000548 <main+124>)
080004fa:   bl      0x80013fa <HAL_GPIO_TogglePin>
 56             HAL_GPIO_TogglePin(LD4_GPIO_Port, LD4_Pin);
080004fe:   mov.w   r1, #4096       ; 0x1000
08000502:   ldr     r0, [pc, #68]   ; (0x8000548 <main+124>)
08000504:   bl      0x80013fa <HAL_GPIO_TogglePin>
 57             HAL_GPIO_TogglePin(LD5_GPIO_Port, LD5_Pin);
08000508:   mov.w   r1, #16384      ; 0x4000
0800050c:   ldr     r0, [pc, #56]   ; (0x8000548 <main+124>)
0800050e:   bl      0x80013fa <HAL_GPIO_TogglePin>
 58             HAL_GPIO_TogglePin(LD6_GPIO_Port, LD6_Pin);
08000512:   mov.w   r1, #32768      ; 0x8000
08000516:   ldr     r0, [pc, #48]   ; (0x8000548 <main+124>)
08000518:   bl      0x80013fa <HAL_GPIO_TogglePin>
 60             tipka = HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin);
0800051c:   movs    r1, #1
0800051e:   ldr     r0, [pc, #44]   ; (0x800054c <main+128>)
08000520:   bl      0x8001398 <HAL_GPIO_ReadPin>
08000524:   mov     r3, r0
08000526:   str     r3, [r7, #0]
 61             if (tipka)
08000528:   ldr     r3, [r7, #0]
0800052a:   cmp     r3, #0
0800052c:   beq.n   0x8000534 <main+104>
 62                 dly = 200;
0800052e:   movs    r3, #200        ; 0xc8
08000530:   str     r3, [r7, #4]
08000532:   b.n     0x800053a <main+110>
 64                 dly = 1000;
08000534:   mov.w   r3, #1000       ; 0x3e8
08000538:   str     r3, [r7, #4]
 65             HAL_Delay(dly); // ms
0800053a:   ldr     r3, [r7, #4]
0800053c:   mov     r0, r3
0800053e:   bl      0x8000df4 <HAL_Delay>
 67           MX_USB_HOST_Process();
```

Seveda je koda v zbirnem jeziku za ARM, vidimo pa podobnosti s kodo za RISC-V.
Pri TogglePin je skok (bl) na funkcijo, ki je v pomnilniku malo naprej.
Vidimo, kako se sp zmanjša za 20, nato pa se frame pointer (r7), ki se prej shrani na sklad, postavi na njegovo vrednost.

```
434       {
          HAL_GPIO_TogglePin:
080013fa:   push    {r7}
080013fc:   sub     sp, #20
080013fe:   add     r7, sp, #0
08001400:   str     r0, [r7, #4]
08001402:   mov     r3, r1
08001404:   strh    r3, [r7, #2]
441         odr = GPIOx->ODR;
08001406:   ldr     r3, [r7, #4]
08001408:   ldr     r3, [r3, #20]
0800140a:   str     r3, [r7, #12]
444         GPIOx->BSRR = ((odr & GPIO_Pin) << GPIO_NUMBER) | (~odr & GPIO_Pin);
0800140c:   ldrh    r2, [r7, #2]
0800140e:   ldr     r3, [r7, #12]
08001410:   ands    r3, r2
08001412:   lsls    r2, r3, #16
08001414:   ldr     r3, [r7, #12]
08001416:   mvns    r1, r3
08001418:   ldrh    r3, [r7, #2]
0800141a:   ands    r3, r1
0800141c:   orrs    r2, r3
0800141e:   ldr     r3, [r7, #4]
08001420:   str     r2, [r3, #24]
445       }
08001422:   nop
08001424:   adds    r7, #20
08001426:   mov     sp, r7
08001428:   ldr.w   r7, [sp], #4
0800142c:   bx      lr
          HAL_HCD_Init:
```

Spodaj vidimo, kako je implementirana zakasnitev HAL_Delay():

```
08000df2:   movs    r0, #0
          HAL_Delay:
08000df4:   push    {r7, lr}
08000df6:   sub     sp, #16
08000df8:   add     r7, sp, #0
08000dfa:   str     r0, [r7, #4]
08000dfc:   bl      0x8000ddc <HAL_GetTick>
08000e00:   str     r0, [r7, #8]
08000e02:   ldr     r3, [r7, #4]
08000e04:   str     r3, [r7, #12]
08000e06:   ldr     r3, [r7, #12]
08000e08:   cmp.w   r3, #4294967295
08000e0c:   beq.n   0x8000e1a <HAL_Delay+38>
08000e0e:   ldr     r3, [pc, #40]   ; (0x8000e38 <HAL_Delay+68>)
08000e10:   ldrb    r3, [r3, #0]
08000e12:   mov     r2, r3
397           wait += (uint32_t)(uwTickFreq);
08000e14:   ldr     r3, [r7, #12]
08000e16:   add     r3, r2
08000e18:   str     r3, [r7, #12]
400         while((HAL_GetTick() - tickstart) < wait)
08000e1a:   nop
08000e1c:   bl      0x8000ddc <HAL_GetTick>
08000e20:   mov     r2, r0
08000e22:   ldr     r3, [r7, #8]
08000e24:   subs    r3, r2, r3
08000e26:   ldr     r2, [r7, #12]
08000e28:   cmp     r2, r3
08000e2a:   bhi.n   0x8000e1c <HAL_Delay+40>
403       }
08000e2c:   nop
08000e2e:   nop
08000e30:   adds    r7, #16
08000e32:   mov     sp, r7
08000e34:   pop     {r7, pc}
08000e36:   nop
08000e38:   movs    r0, r1
08000e3a:   movs    r0, #0
          NVIC_SetPriorityGrouping:
```

```
62              if (tipka)
08000528:   ldr     r3, [r7, #0]
0800052a:   cmp     r3, #0
0800052c:   beq.n   0x8000534 <main+104>
63                dly = 200;
0800052e:   movs    r3, #200        ; 0xc8
08000530:   str     r3, [r7, #4]
08000532:   b.n     0x800053a <main+110>
65                dly = 1000;
08000534:   mov.w   r3, #1000       ; 0x3e8
08000538:   str     r3, [r7, #4]
```

Lahko pa za to kodo zapišemo C-funkcijo in jo damo v Compiler Explorer (https://godbolt.org/, CE) in izberemo ARM gcc (lahko tudi za Linux)

```
int set_delay(int tipka)
{
        if (tipka)
                return 200;
        else
                return 1000;
}
```

Dobimo tole:

```
set_delay:
        push    {r7}
        sub     sp, sp, #12
        add     r7, sp, #0
        str     r0, [r7, #4]
        ldr     r3, [r7, #4]
        cmp     r3, #0
        beq     .L2
        movs    r3, #200
        b       .L3
.L2:
        mov     r3, #1000
.L3:
        mov     r0, r3
        adds    r7, r7, #12
        mov     sp, r7
        ldr     r7, [sp], #4
        bx      lr
```

Naredimo novo datoteko v Cube, s končnico .s (npr. set_delay.s). Vanjo zapišemo

```
.syntax unified
.global set_delay
```

nato pa dodamo gornjo zbirno kodo od CE.

Rabimo še header set_delay.h, v katerem je deklaracija funkcije:

```
#ifndef SRC_SET_DELAY_H_
#define SRC_SET_DELAY_H_

extern int set_delay(int tipka);

#endif
```

Ta header vključimo v main.c:

```
#include "set_delay.h"
```

Ko spet poženemo razhroščevalnik, vidimo, da program deluje enako kot prej. Torej načelno lahko vedno vključimo program ali funkcijo, napisano v zbirnem jeziku, v C-program.