STM32 od registrů po Arduino

Petr Šrámek (www.chiptron.cz, @chiptronCZ)
Cena bastlířů 2018 (www.cenabastliru.cz)

Obsah

znaků: 2556 (+- 10%)

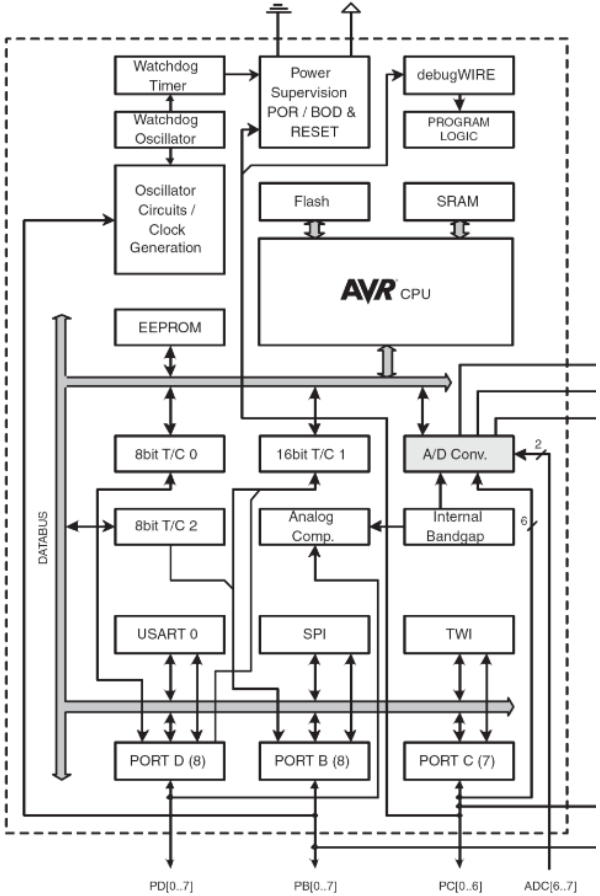
slov: 365 (+- 10%)

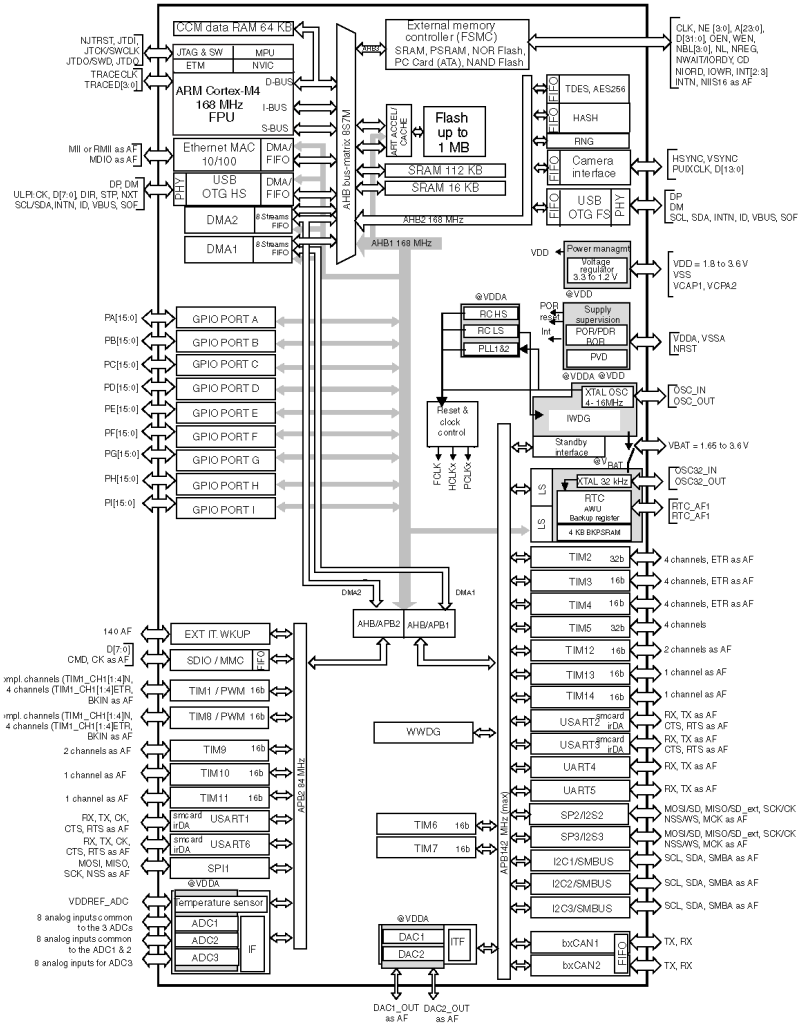
snímků prezentace: 29

obrázků: 33

gifů: 1

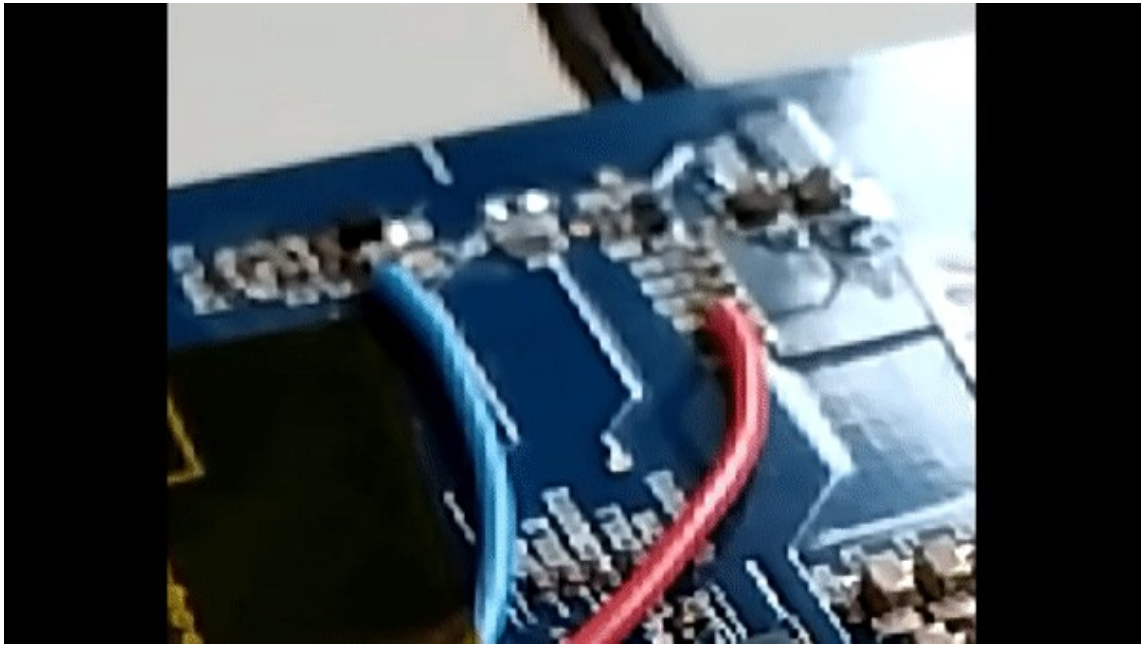
jádro ATmega328 vs. STM32





Rozsvítíme diodu pomocí ...

- vysokého napětí



programujeme pomocí ...

- assembler
- přímý přístup do registrů
- SPL (Standard Peripheral Library)
- HAL (Hardware Abstraction Layer)
- LL API (Low Layer API)
- Arduino (Wiring)

Assembler

```
    .syntax unified
    .cpu cortex-m4
    .thumb

    // configuration values for STM32F4x
    // see: http://www.st.com/web/en/resource/tech

#define AHB1PERIPH_BASE 0x40020000
#define RCC_BASE        AHB1PERIPH_BASE + 0x3800
#define RCC_AHB1_ENR    RCC_BASE + 0x30
#define GPIOD_BASE      AHB1PERIPH_BASE + 0x0c00
#define GPIOD_MODER     GPIOD_BASE
#define GPIOD_ODR       GPIOD_BASE + 0x14
#define GPIOD_BIT       8
#define LED_PIN         15
#define LED_MODER       1 << (2 * LED_PIN)
#define LED_BIT         1 << LED_PIN
#define DELAY           0x80000
```

Assembler

```
    .syntax unified
    .cpu cortex-m4
    .thumb

    // configuration values
    // see: http://www.st.c

#define AHB1PERIPH_BASE 0x4
#define RCC_BASE        AHB
#define RCC_AHB1_ENR    RCC
#define GPIOD_BASE      AHB
#define GPIOD_MODER     GPI
#define GPIOD_ODR       GPI
#define GPIOD_BIT       8
#define LED_PIN         15
#define LED_MODER       1 <
#define LED_BIT         1 <
#define DELAY           0x8
```

```
    .section  .text
    .weak     _start
    .type     _start, %function
_start:
    ldr     r0, =RCC_AHB1_ENR   // get contents of AHB1 bus cloc
    ldr     r1, [r0]
    orr     r1, GPIOD_BIT       // set bit to enable clock for GI
    str     r1, [r0]
    ldr     r0, =GPIOD_MODER    // configure LED pin as outout
    ldr     r1, =LED_MODER
    str     r1, [r0]
    ldr     r0, =GPIOD_ODR      // load address of GPIOD output d
.blink:
    movw    r1, LED_BIT
    str     r1, [r0]            // setting LED bit in GPIOD_ODR
    bl      .delay              // jump to delay function below
    eors    r1, r1              // clear r1 register
    str     r1, [r0]            // turn LED off
    bl      .delay              // wait again
    b       .blink              // infinite loop
```

Assembler

```
    .syntax unified
    .cpu cortex-m4
    .thumb

    // configuration values
    // see: http://www.st.c

#define AHB1PERIPH_BASE 0x4
#define RCC_BASE        AHB
#define RCC_AHB1_ENR    RCC
#define GPIOD_BASE      AHB
#define GPIOD_MODER     GPI
#define GPIOD_ODR       GPI
#define GPIOD_BIT       8
#define LED_PIN         15
#define LED_MODER       1 <
#define LED_BIT         1 <
#define DELAY           0x8
```

```
    .section  .text
    .weak     _start
    .type     _start, %function
_start:
    ldr     r0, =RCC_AHB1_ENR   //
    ldr     r1, [r0]
    orr     r1, GPIOD_BIT       //
    str     r1, [r0]
    ldr     r0, =GPIOD_MODER    //
    ldr     r1, =LED_MODER
    str     r1, [r0]
    ldr     r0, =GPIOD_ODR      //
.blink:
    movw    r1, LED_BIT
    str     r1, [r0]            //
    bl      .delay              //
    eors    r1, r1              //
    str     r1, [r0]            // turn LED off
    bl      .delay              // wait again
    b       .blink              // infinite loop
```

```
.delay:
    ldr     r2, =DELAY          // load DELAY value
1:  subs    r2, r2, #1          // count down to 0 = wait
    bne     1b
    bx      lr                  // return to caller

    // partial, but mandatory ARM Cortex M interrupt table
    // here we only specify the reset handler, which also
    .section .isr_vector_table, "a", %progbits
    .type    isr_vectors, %object
isr_vectors:
    .word   _estack             // stack start address (de
    .word   _start              // address of the reset ha
    .size   isr_vectors, . - isr_vectors
```

https://github.com/istepaniuk/stm32-templates/blob/master/blink-asm/src/blink.s

Přímý přístup do registrů

Kde to najdu?

datasheet

reference manual

-> st.com

```c
int main(void)
{
    //----LED----//
    RCC->AHB1RSTR |= RCC_AHB1RSTR_GPIOIRST; //reset GPIOI
    RCC->AHB1RSTR &= ~RCC_AHB1RSTR_GPIOIRST;
    RCC->AHB1ENR |= RCC_AHB1ENR_GPIOIEN; //Povoleni hodin pro port I
    GPIOI->MODER |= GPIO_MODER_MODER1_0; //Nastaveni pinu 1 na portu I
    GPIOI->ODR |= GPIO_ODR_ODR_1;   //Zapis do vystupniho registru portu I PIN 1

    uint32_t i=0;
    while(1)
    {
        for(uint32_t i=0; i< 0xFFFF; i++)
        {;}
        GPIOI->ODR ^= GPIO_ODR_ODR_1;
    }
}
```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, always read as 0.

Bits 15:0 **ODRy[15:0]:** Port output data (y= 0 .. 15)
These bits can be read and written by software and can be accessed in Word mode only.

Note: For atomic bit set/reset, the ODR bits can be individually set and cleared by writing to the GPIOx_BSRR register (x = A .. G).

SPL

stále oblíbené

už nepodporované

```
void delay(unsigned int nCount);
GPIO_InitTypeDef GPIO_InitStruct;

int main (void)
{
    // Enable clock for GPIOA
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

    // Configure PA0 as push-pull output
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_2MHz;
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP;
    GPIO_Init(GPIOA, &GPIO_InitStruct);

    while (1)
    {
        /* Toggle LED on PA0 */
        // Reset bit will turn on LED (because the logic is interved)
        GPIO_ResetBits(GPIOA, GPIO_Pin_0);
        delay(1000);
        // Set bit will turn off LED (because the logic is interved)
        GPIO_SetBits(GPIOA, GPIO_Pin_0);
        delay(1000);
    }
}

// Delay function
void delay(unsigned int nCount)
{
    unsigned int i, j;

    for (i = 0; i < nCount; i++)
        for (j = 0; j < 0x2AFF; j++);
}
```

http://www.handsonembedded.com/stm32f103-spl-tutorial-2/

HAL (Hardware Abstraction Layer)

Základní inicializace

Nastavení periférií

->STM32CubeMX

```
void SystemClock_Config(void)
{
  RCC_ClkInitTypeDef clkinitstruct = {0};
  RCC_OscInitTypeDef oscinitstruct = {0};

  oscinitstruct.OscillatorType  = RCC_OSCILLATORTYPE_HSI;
  oscinitstruct.HSEState        = RCC_HSE_OFF;
  oscinitstruct.LSEState        = RCC_LSE_OFF;
  oscinitstruct.HSIState        = RCC_HSI_ON;
  oscinitstruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  oscinitstruct.HSEPredivValue    = RCC_HSE_PREDIV_DIV1;
  oscinitstruct.PLL.PLLState    = RCC_PLL_ON;
  oscinitstruct.PLL.PLLSource   = RCC_PLLSOURCE_HSI_DIV2;
  oscinitstruct.PLL.PLLMUL      = RCC_PLL_MUL16;
  if (HAL_RCC_OscConfig(&oscinitstruct)!= HAL_OK)
  {
    /* Initialization Error */
    while(1);
  }

  /* Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2
     clocks dividers */
  clkinitstruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
  clkinitstruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  clkinitstruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  clkinitstruct.APB2CLKDivider = RCC_HCLK_DIV1;
  clkinitstruct.APB1CLKDivider = RCC_HCLK_DIV2;
  if (HAL_RCC_ClockConfig(&clkinitstruct, FLASH_LATENCY_2)!= HAL_OK)
  {
    /* Initialization Error */
    while(1);
  }
}
```

HAL (Hardware Abstraction Layer)

```c
void SystemClock_Config(void)
{
  RCC_ClkInitTypeDef clkinitstruct = {0};
  RCC_OscInitTypeDef oscinitstruct = {0};

  oscinitstruct.OscillatorType  = RCC_OSCILLATOR
  oscinitstruct.HSEState        = RCC_HSE_OFF;
  oscinitstruct.LSEState        = RCC_LSE_OFF;
  oscinitstruct.HSIState        = RCC_HSI_ON;
  oscinitstruct.HSICalibrationValue = RCC_HSICAL
  oscinitstruct.HSEPredivValue    = RCC_HSE_PRED
  oscinitstruct.PLL.PLLState    = RCC_PLL_ON;
  oscinitstruct.PLL.PLLSource   = RCC_PLLSOURCE_
  oscinitstruct.PLL.PLLMUL      = RCC_PLL_MUL16;
  if (HAL_RCC_OscConfig(&oscinitstruct)!= HAL_OK
  {
    /* Initialization Error */
    while(1);
  }

  /* Select PLL as system clock source and confi
     clocks dividers */
  clkinitstruct.ClockType = (RCC_CLOCKTYPE_SYSCL
  clkinitstruct.SYSCLKSource = RCC_SYSCLKSOURCE_
  clkinitstruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  clkinitstruct.APB2CLKDivider = RCC_HCLK_DIV1;
  clkinitstruct.APB1CLKDivider = RCC_HCLK_DIV2;
  if (HAL_RCC_ClockConfig(&clkinitstruct, FLASH_
  {
    /* Initialization Error */
    while(1);
  }
}
```

```c
int main(void)
{

  HAL_Init();

  /* Configure the system clock to 64 MHz */
  SystemClock_Config();

  /* -1- Enable GPIO Clock (to be able to program the configuration registers) */
  LED2_GPIO_CLK_ENABLE();

  /* -2- Configure IO in output push-pull mode to drive external LEDs */
  GPIO_InitStruct.Mode  = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull  = GPIO_PULLUP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;

  GPIO_InitStruct.Pin = LED2_PIN;
  HAL_GPIO_Init(LED2_GPIO_PORT, &GPIO_InitStruct);

  /* -3- Toggle IO in an infinite loop */
  while (1)
  {
    HAL_GPIO_TogglePin(LED2_GPIO_PORT, LED2_PIN);
    /* Insert delay 100 ms */
    HAL_Delay(100);
  }
}
```

LL API (Low-Layer API)

Základní inicializace

Nastavení periférií

->STM32CubeMX

```
void SystemClock_Config(void)
{
  /* Set FLASH latency */
  LL_FLASH_SetLatency(LL_FLASH_LATENCY_2);

  /* Enable HSE oscillator */
  LL_RCC_HSE_EnableBypass();
  LL_RCC_HSE_Enable();
  while(LL_RCC_HSE_IsReady() != 1)
  {
  };

  /* Main PLL configuration and activation */
  LL_RCC_PLL_ConfigDomain_SYS(LL_RCC_PLLSOURCE_HSE_DIV_1, LL_RCC_PLL_MUL_9);

  LL_RCC_PLL_Enable();
  while(LL_RCC_PLL_IsReady() != 1)
  {
  };

  /* Sysclk activation on the main PLL */
  LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
  LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_PLL);
  while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_PLL)
  {
  };

  /* Set APB1 & APB2 prescaler*/
  LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_2);
  LL_RCC_SetAPB2Prescaler(LL_RCC_APB2_DIV_1);

  /* Set systick to 1ms in using frequency set to 72MHz */
  LL_Init1msTick(72000000);

  /* Update CMSIS variable (which can be updated also through SystemCoreClockUpdate function) */
  LL_SetSystemCoreClock(72000000);
}
```

LL API (Low-Layer API)

```c
void SystemClock_Config(void)
{
  /* Set FLASH latency */
  LL_FLASH_SetLatency(LL_FLASH_LATENCY_2);

  /* Enable HSE oscillator */
  LL_RCC_HSE_EnableBypass();
  LL_RCC_HSE_Enable();
  while(LL_RCC_HSE_IsReady() != 1)
  {
  };

  /* Main PLL configuration and activation */
  LL_RCC_PLL_ConfigDomain_SYS(LL_RCC_PLLSOURCE_HSE_DIV_1, LL_RCC_P

  LL_RCC_PLL_Enable();
  while(LL_RCC_PLL_IsReady() != 1)
  {
  };

  /* Sysclk activation on the main PLL */
  LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
  LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_PLL);
  while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_PL
  {
  };

  /* Set APB1 & APB2 prescaler*/
  LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_2);
  LL_RCC_SetAPB2Prescaler(LL_RCC_APB2_DIV_1);

  /* Set systick to 1ms in using frequency set to 72MHz */
  LL_Init1msTick(72000000);

  /* Update CMSIS variable (which can be updated also through Syst
  LL_SetSystemCoreClock(72000000);
}
```

```c
int main(void)
{
  /* Configure the system clock to 72 MHz */
  SystemClock_Config();

  /* -2- Configure IO in output push-pull mode to drive external LED */
  Configure_GPIO();

  /* Toggle IO in an infinite loop */
  while (1)
  {
    LL_GPIO_TogglePin(LED2_GPIO_PORT, LED2_PIN);

    /* Insert delay 250 ms */
    LL_mDelay(250);
  }
}


void Configure_GPIO(void)
{
  /* Enable the LED2 Clock */
  LED2_GPIO_CLK_ENABLE();

  /* Configure IO in output push-pull mode to drive external LED2 */
  LL_GPIO_SetPinMode(LED2_GPIO_PORT, LED2_PIN, LL_GPIO_MODE_OUTPUT);

}
```

Porovnání HAL a LL API

```
//Inicialization of I2C transfer - writing
HAL_I2C_Master_Transmit(&hi2c1, u8_slaveAddress, u8_transData, 1, 100);
HAL_Delay(100);

HAL_I2C_Master_Receive(&hi2c1, u8_slaveAddress, u8_recData, 2, 100);
HAL_Delay(100);
```

```
//Inicializace zapisu a startovaci podminky
LL_I2C_HandleTransfer(I2C1, u8_slaveAddress, LL_I2C_ADDRSLAVE_7BIT, 1,
      LL_I2C_MODE_AUTOEND,LL_I2C_GENERATE_START_WRITE);

while(!LL_I2C_IsActiveFlag_STOP(I2C1))//cekani na priznak STOP bitu
{
   if(LL_I2C_IsActiveFlag_TXE(I2C1))//prazdny TX
   {
      LL_I2C_TransmitData8(I2C1, 0xE3);
      //LL_I2C_GenerateStopCondition(I2C1); //pokud pouziju LL_I2C_MODE_I2C
   }

}
LL_I2C_ClearFlag_STOP(I2C1);  //smaz priznak STOP bitu


//Inicializace cteni
LL_I2C_HandleTransfer(I2C1, u8_slaveAddress, LL_I2C_ADDRSLAVE_7BIT, 2,
        LL_I2C_MODE_AUTOEND,LL_I2C_GENERATE_START_READ);

while(!LL_I2C_IsActiveFlag_STOP(I2C1))
{
   if(LL_I2C_IsActiveFlag_RXNE(I2C1)) //prazdny RX
   {
      u8_recData[u8_rxIndex++] = LL_I2C_ReceiveData8(I2C1);
   }
}
u8_rxIndex = 0;  //ZAKOMENTOVAT, POKUD CHCETE VYZKOUSET HardFault Handler
LL_I2C_ClearFlag_STOP(I2C1);  //smaz priznak STOP bitu
```

Porovnání HAL a LL API

```
//Inicialization of I2C transfer - writing
HAL_I2C_Master_Transmit(&hi2c1, u8_slaveAddress, u8_transData, 1, 100);
```

```
HAL_StatusTypeDef HAL_I2C_Master_Transmit(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t Time
{
 uint32_t tickstart = 0U;

 if(hi2c->State == HAL_I2C_STATE_READY)
 {
  /* Process Locked */
  __HAL_LOCK(hi2c);

  /* Init tickstart for timeout management*/
  tickstart = HAL_GetTick();

  if(I2C_WaitOnFlagUntilTimeout(hi2c, I2C_FLAG_BUSY, SET, I2C_TIMEOUT_BUSY, tickstart) != HAL_OK)
  {
   return HAL_TIMEOUT;
  }

  hi2c->State     = HAL_I2C_STATE_BUSY_TX;
  hi2c->Mode      = HAL_I2C_MODE_MASTER;
  hi2c->ErrorCode = HAL_I2C_ERROR_NONE;

  /* Prepare transfer parameters */
  hi2c->pBuffPtr  = pData;
  hi2c->XferCount = Size;
  hi2c->XferISR   = NULL;

  /* Send Slave Address */
  /* Set NBYTES to write and reload if hi2c->XferCount > MAX_NBYTE_SIZE and generate RESTART */
  if(hi2c->XferCount > MAX_NBYTE_SIZE)
  {
   hi2c->XferSize = MAX_NBYTE_SIZE;
   I2C_TransferConfig(hi2c, DevAddress, hi2c->XferSize, I2C_RELOAD_MODE, I2C_GENERATE_START_WRITE);
  }
  else
```

```
//Inicializace zapisu a startovaci podminky
LL_I2C_HandleTransfer(I2C1, u8_slaveAddress, LL_I2C_ADDRSLAVE_7BIT, 1,
    LL_I2C_MODE_AUTOEND,LL_I2C_GENERATE_START_WRITE);

while(!LL_I2C_IsActiveFlag_STOP(I2C1))//cekani na priznak STOP
{
  if(LL_I2C_IsActiveFlag_TXE(I2C1))//prazdny TX
  {
    LL_I2C_TransmitData8(I2C1, 0xE3);
    //LL_I2C_GenerateStopCondition(I2C1); //pokud pouziju LL_I2C_MODE_I2C
  }

}
LL_I2C_ClearFlag_STOP(I2C1);  //smaz priznak STOP bitu


//Inicializace cteni
LL_I2C_HandleTransfer(I2C1, u8_slaveAddress, LL_I2C_ADDRSLAVE_7BIT, 2,
      LL_I2C_MODE_AUTOEND,LL_I2C_GENERATE_START_READ);

while(!LL_I2C_IsActiveFlag_STOP(I2C1))
{
  if(LL_I2C_IsActiveFlag_RXNE(I2C1)) //prazdny RX
  {
    u8_recData[u8_rxIndex++] = LL_I2C_ReceiveData8(I2C1);
  }
}
u8_rxIndex = 0;  //ZAKOMENTOVAT, POKUD CHCETE VYZKOUSET HardFault Handler
LL_I2C_ClearFlag_STOP(I2C1);  //smaz priznak STOP bitu
```

```
__STATIC_INLINE void LL_I2C_HandleTransfer(I2C_TypeDef *I2Cx, uint32_t SlaveAdd
                                           uint32_t TransferSize, uint32_t EndMode, uint32_t Request)
{
  MODIFY_REG(I2Cx->CR2, I2C_CR2_SADD | I2C_CR2_ADD10 | I2C_CR2_RD_WRN | I2C
             I2C_CR2_NBYTES | I2C_CR2_AUTOEND | I2C_CR2_HEAD10R,
             SlaveAddr | SlaveAddrSize | TransferSize << I2C_CR2_NBYTES_Pos | EndMode
}
```

Porovnání HAL a LL API

```
//Inicialization of I2C transfer - writing
HAL_I2C_Master_Transmit(&hi2c1, u8_slaveAddress, u8_transData, 1, 100);
```

```
HAL_StatusTypeDef HAL_I2C_Master_Transmit(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t Time
{
  else
  {
    hi2c->XferSize = hi2c->XferCount;
    I2C_TransferConfig(hi2c, DevAddress, hi2c->XferSize, I2C_AUTOEND_MODE, I2C_GENERATE_START_WRITE);
  }

  while(hi2c->XferCount > 0U)
  {
    /* Wait until TXIS flag is set */
    if(I2C_WaitOnTXISFlagUntilTimeout(hi2c, Timeout, tickstart) != HAL_OK)
    {
      if(hi2c->ErrorCode == HAL_I2C_ERROR_AF)
      {
        return HAL_ERROR;
      }
      else
      {
        return HAL_TIMEOUT;
      }
    }
    /* Write data to TXDR */
    hi2c->Instance->TXDR = (*hi2c->pBuffPtr++);
    hi2c->XferCount--;
    hi2c->XferSize--;

    if((hi2c->XferSize == 0U) && (hi2c->XferCount!=0U))
    {
      /* Wait until TCR flag is set */
      if(I2C_WaitOnFlagUntilTimeout(hi2c, I2C_FLAG_TCR, RESET, Timeout, tickstart) != HAL_OK)
      {
        return HAL_TIMEOUT;
      }
```

```
//Inicializace zapisu a startovaci podminky
LL_I2C_HandleTransfer(I2C1, u8_slaveAddress, LL_I2C_ADDRSLAVE_7BIT, 1,
    LL_I2C_MODE_AUTOEND,LL_I2C_GENERATE_START_WRITE);

while(!LL_I2C_IsActiveFlag_STOP(I2C1))//cekani na priznak STOP
{
  if(LL_I2C_IsActiveFlag_TXE(I2C1))//prazdny TX
  {
    LL_I2C_TransmitData8(I2C1, 0xE3);
    //LL_I2C_GenerateStopCondition(I2C1); //pokud pouziju LL_I2C_MODE_I2C
  }

}
LL_I2C_ClearFlag_STOP(I2C1);  //smaz priz

//Inicializace cteni
LL_I2C_HandleTransfer(I2C1, u8_slaveAddress, LL_I2C_ADDRSLAVE_7BIT, 2,
      LL_I2C_MODE_AUTOEND,LL_I2C_GENERATE_START_READ);

while(!LL_I2C_IsActiveFlag_STOP(I2C1))
{
  if(LL_I2C_IsActiveFlag_RXNE(I2C1)) //prazdny RX
  {
    u8_recData[u8_rxIndex++] = LL_I2C_ReceiveData8(I2C1);
  }
}
u8_rxIndex = 0;  //ZAKOMENTOVAT, POKUD CHCETE VYZKOUSET HardFault Handler
LL_I2C_ClearFlag_STOP(I2C1);  //smaz priznak STOP bitu
```

```
__STATIC_INLINE void LL_I2C_HandleTransfer(I2C_TypeDef *I2Cx, uint32_t SlaveAdd
                                           uint32_t TransferSize, uint32_t EndMode, uint32_t Request)
{
  MODIFY_REG(I2Cx->CR2, I2C_CR2_SADD | I2C_CR2_ADD10 | I2C_CR2_RD_WRN | I2C
             I2C_CR2_NBYTES | I2C_CR2_AUTOEND | I2C_CR2_HEAD10R,
             SlaveAddr | SlaveAddrSize | TransferSize << I2C_CR2_NBYTES_Pos | EndMode
}
```

```
__STATIC_INLINE void LL_I2C_TransmitData8(I2C_TypeDef *I2Cx, uint8_t Data)
{
  WRITE_REG(I2Cx->TXDR, Data);
}
```

Porovnání HAL a LL API

```
//Inicialization of I2C transfer - writing
HAL_I2C_Master_Transmit(&hi2c1, u8_slaveAddress, u8_transData, 1, 100);
```

```
HAL_StatusTypeDef HAL_I2C_Master_Transmit(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t Time
{
  else
  {
    if(hi2c->XferCount > MAX_NBYTE_SIZE)
    {
      hi2c->XferSize = MAX_NBYTE_SIZE;
      I2C_TransferConfig(hi2c, DevAddress, hi2c->XferSize, I2C_RELOAD_MODE, I2C_NO_STARTSTOP);
    }
    else
    {
      hi2c->XferSize = hi2c->XferCount;
      I2C_TransferConfig(hi2c, DevAddress, hi2c->XferSize, I2C_AUTOEND_MODE, I2C_NO_STARTSTOP);
    }
  }
}

/* No need to Check TC flag, with AUTOEND mode the stop is automatically generated */
/* Wait until STOPF flag is set */
if(I2C_WaitOnSTOPFlagUntilTimeout(hi2c, Timeout, tickstart) != HAL_OK)
{
  if(hi2c->ErrorCode == HAL_I2C_ERROR_AF)
  {
    return HAL_ERROR;
  }
  else
  {
    return HAL_TIMEOUT;
  }
}

/* Clear STOP Flag */
__HAL_I2C_CLEAR_FLAG(hi2c, I2C_FLAG_STOPF);

/* Clear Configuration Register 2 */
I2C_RESET_CR2(hi2c);
```

```
//Inicializace zapisu a startovaci podminky
LL_I2C_HandleTransfer(I2C1, u8_slaveAddress, LL_I2C_ADDRSLAVE_7BIT, 1,
    LL_I2C_MODE_AUTOEND,LL_I2C_GENERATE_START_WRITE);

while(!LL_I2C_IsActiveFlag_STOP(I2C1))//cekani na priznak STOP
{
  if(LL_I2C_IsActiveFlag_TXE(I2C1))//prazdny TX
  {
    LL_I2C_TransmitData8(I2C1, 0xE3);
    //LL_I2C_GenerateStopCondition(I2C1); //pokud pouziju LL_I2C_MODE_I2C
  }

}
LL_I2C_ClearFlag_STOP(I2C1);  //smaz priz

//Inicializa
LL_I2C_Ha
    LL_I
while(!LL_
{
  if(LL_I2C_IsActiveFlag_RXNE(I2C1)) //prazdny RX
  {
    u8_recData[u8_rxIndex++] = LL_I2C_ReceiveData8(I2C1);
  }
}
u8_rxIndex = 0;  //ZAKOMENTOVAT, POKUD CHCETE VYZKOUSET HardFault Handler
LL_I2C_ClearFlag_STOP(I2C1);  //smaz priznak STOP bitu
```

```
__STATIC_INLINE void LL_I2C_HandleTransfer(I2C_TypeDef *I2Cx, uint32_t SlaveAdd
                                           uint32_t TransferSize, uint32_t EndMode, uint32_t Request)
{
  MODIFY_REG(I2Cx->CR2, I2C_CR2_SADD | I2C_CR2_ADD10 | I2C_CR2_RD_WRN | I2C
             I2C_CR2_NBYTES | I2C_CR2_AUTOEND | I2C_CR2_HEAD10R,
             SlaveAddr | SlaveAddrSize | TransferSize << I2C_CR2_NBYTES_Pos | EndMode
}
```

```
__STATIC_INLINE void LL_I2C_TransmitData8(I2C_TypeDef *I2Cx, uint8_t Data)
{
  WRITE_REG(I2Cx->TXDR, Data);
}
```

```
__STATIC_INLINE void LL_I2C_ClearFlag_STOP(I2C_TypeDef *I2Cx)
{
  SET_BIT(I2Cx->ICR, I2C_ICR_STOPCF);
}
```

Porovnání HAL a LL API

Examples from Cube Package on STM32L0						
	GPIO toggle			ADC conversions + DMA		
	HAL	LL	Diff	HAL	LL	Diff
CODE + RO DATA	3398	948	72%	7 292	1 996	73%
SRAM	1052	1028	2%	1 268	1 048	17%

Porovnání HAL a LL API

		Přenositelnost	Optimalizace kódu	Jednoduchost	Porozumnění kódu	Podpora HW
STM32 Cube	HAL API	+++	+	++	++	+++
	LL API	+	+++	+	+++	++

STM32CubeMX

Grafický nástroj pro inicializaci mikrokontroléru

- nastavení hodinových zdrojů a hodinových sběrnic
- nastavení periférií
- nastavení přerušení
- generování kódu pro 3 IDE (IAR, Keil, SW4STM32 (AC6 a Atollic TrueStudio))
- orientační výpočet spotřeby na základě vložených údajů
- a spousta dalšího

STM32 a Arduino IDE

Oficiální podpora několika vývojových kitů a mikrokontrolérů pro Arduino IDE (Wiring)

Vložení jednoho *.json souboru

Podpora GPIO, I2C, SPI, DAC, ADC, Čítače/časovače, USART, PWM, USB HID, ...

www.stm32duino.eu

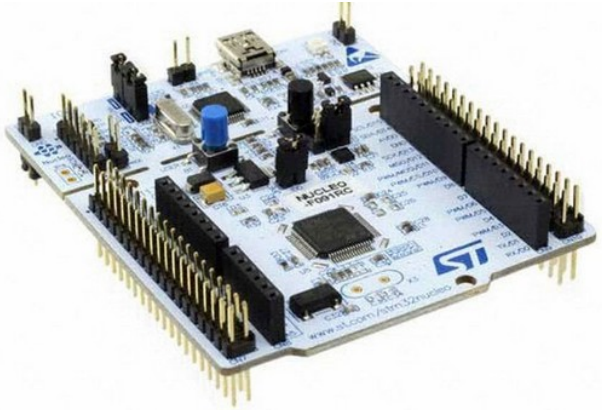
Jak začít, co nastavit, jaký vývojový kit vybrat, vzorové kódy ...

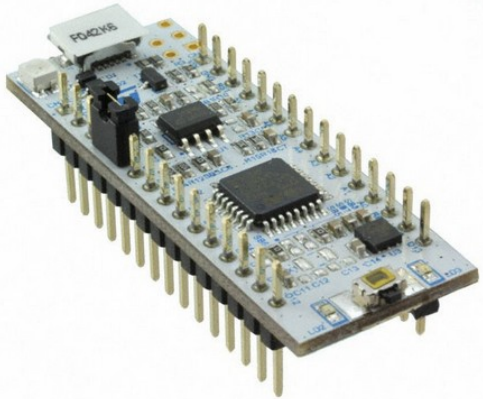
STM32 a Arduino

- rozmanitost podporovaných mikrokontrolérů (velikost FLASH, RAM, počet GPIO, různé periférie, vysoký výkon, nízká spotřeba, ...)
- v případě použití originálních STM32 vývojových kitů nahrání programu přes vestavěný programátor/debugger
- Podpora některých čínských vývojových kitů

Vývojové desky

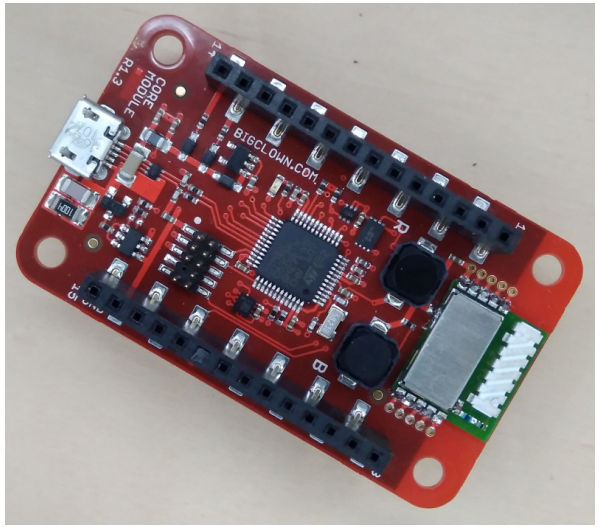
Oficiální (Nucleo32, Nucleo64, Nucleo144, Discovery)

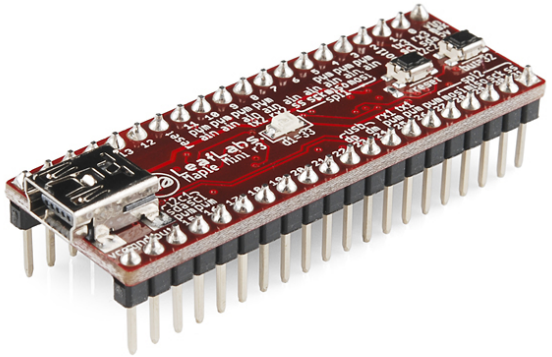


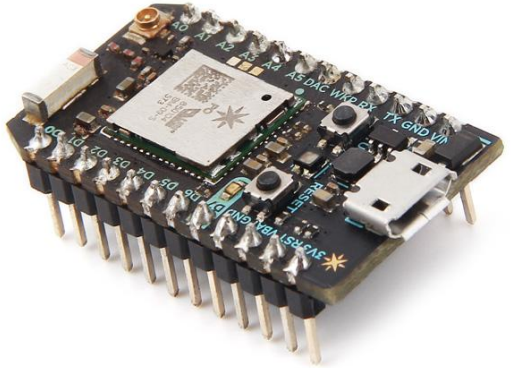


Vývojové desky

Třetích stran (BigClown, LeafLabs - Maple Mini, Particle Photon, RedBear Duo)







Vývojové desky

Desky z Číny: BluePill, BlackPill (Minimum development board for STM32)

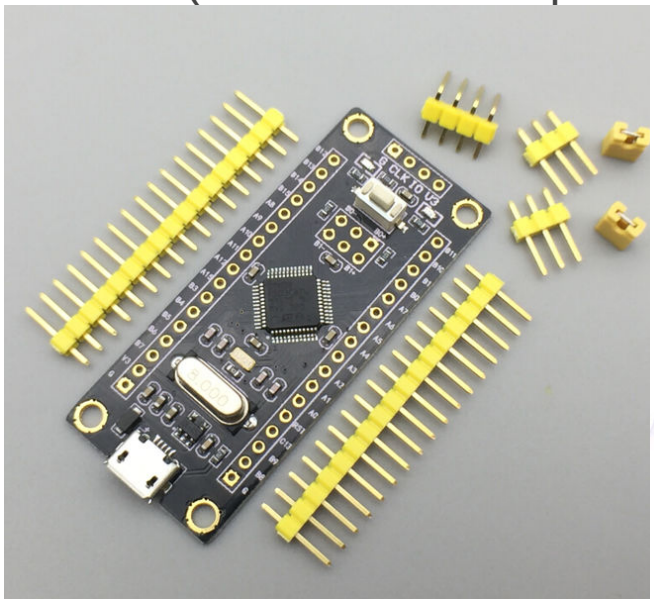
STM32F103

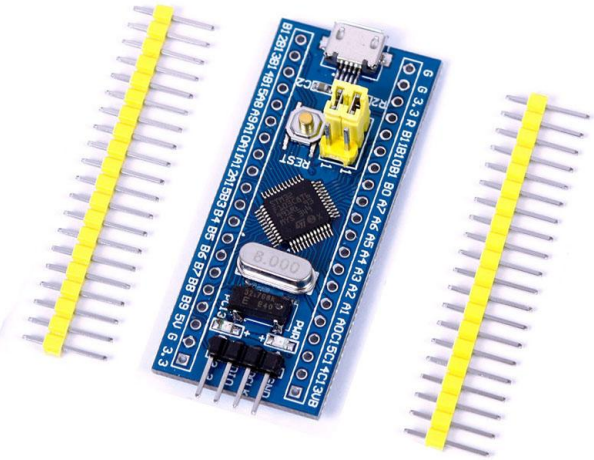
bez programátoru

nahrání FW:

USB, UART bootloader

externí programátor





aliexpress.com

Vývojová prostředí

IAR - placený

Keil - zdarma do 32 kB, zdarma pro řady L0 a F0

AC6 (System Workbench for STM32) - zdarma, bez limitu kódu

Atollic TRUEStudio - zdarma, bez omezení kódu - koupeno STMicroelectronics

Arduino IDE - zdarma

Sloeber - zdarma, alternativa k Arduino IDE

Embed.com - zdarma, programování a nahrávání kódu přes webový prohlížeč

Vybíráme vhodný STM32 mikrokontrolér

- v čem to chci programovat
  - Assembler, HAL -> všechny uC podporovány
  - LL API -> velká část uC podporována
  - Arduino -> 24 mikrokontrolérů na 26 vývojových deskách
- kolik chci investovat do vývojového kitu
  - originální kity od 250 Kč
  - čínské vývojové desky od 30 Kč
  - vývojové kity třetích stran Particle Photon od 450 Kč, BigClown od 490 Kč
- jaké mám požadavky na velikost FLASH, RAM, typy periférií (ADC, DAC, CAN, USB, Ethernet, ...), spotřebu uC, ...

Odkazy k připomenutí

STM32 a Arduino IDE (a vzorové kódy) -> stm32duino.eu

STM32 a Embed.com -> chiptron.cz (sérii článků napsal Lukáš Beran)

BigClown -> chiptron.cz (seznámení a několik vzorových projektů napsal já)

Máte dotaz?

Napište mi na facebook, twitter nebo google plus -> @chiptronCZ

www.chiptron.cz

Soutěž Cena bastlířů: hlasujte a vyhrajte na www.cenabastliru.cz