PRIMENA MIKROKONTROLERA-MS1PMK

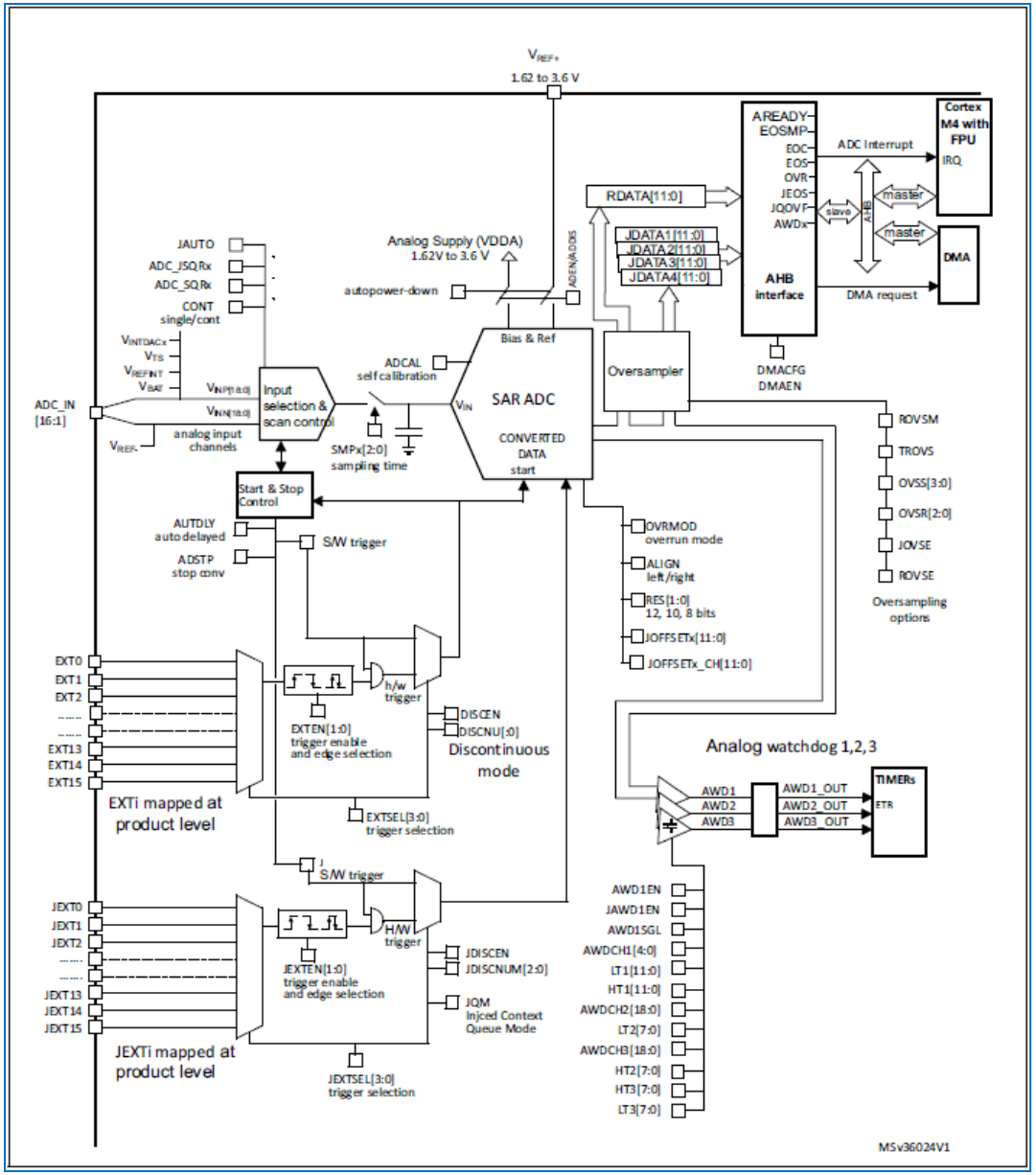
8. deo

2017

Nenad Jovičić

STM32L476 A/D konvertor

- **Osnovne karakteristike**
  - Trostruki 12-bitni A/D konvertor sa sukcesivnim aproksimacijama
  - AD konvertori mogu da rade sinhronizovano i ispreplitano
  - Svaki AD konvertor ima do 19 analognih ulaza
  - Merni opseg 0-Vdd
  - Učesnoatnost uzorkovanja 5 Msample/s
  - Veza sa integrisanim analognim temperaturnim senzorom.



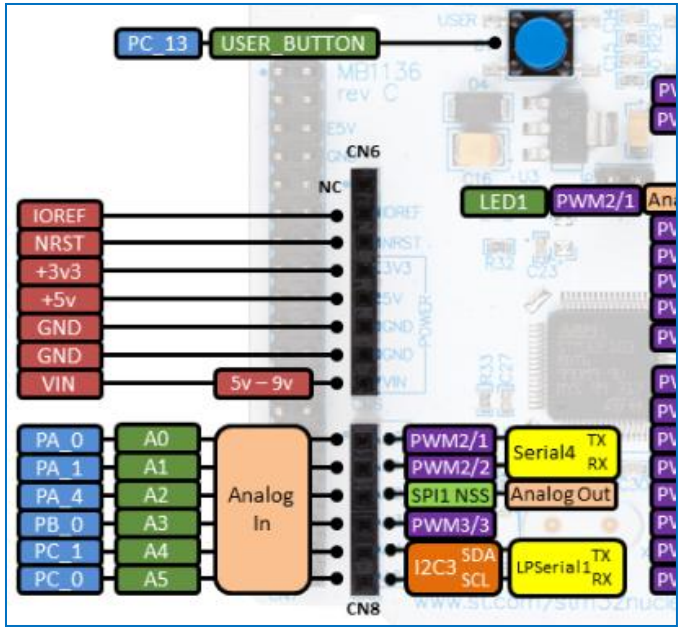
Modovi konverzije

- Prema načinu startovanja konverzije modovi su:
  - Jednostruki mod (single conversion)
    - Posle svake konverzije ADC se zaustavlja uz generisanje dozvoljenih prekida ili DMA zahteva.
  - Kontinulani mod (continuous conversion)
    - Nakon konverzije generišu se prekidi ili DMA zahtevi ali se automatski startuje nova konverzija.
  - Diskontinualni mod (discontinuous conversion)
    - Može se zadati konačan broj konverzija (n<=8) nakon čijeg izvršavanja se ADC zaustavlja.
- Prema broju kanala koji se konvertuju modovi su:
  - Single channel – konvertuje se samo jedan kanal i to u single ili kontinualnom modu
  - Scan mode – konvertuje se sekvenca kanala, i to i jednostrukom modu (single conversion), kontinualnom modu ili diskontinulanom modu. U ovom modu moguće je definisati do 16 proizvoljnih rageluarnih kanala ili do 4 injektovana kanala.

MBED Analogni ulazi

http://mbed.org/handbook/AnalogIn

```
//Turn on a LED when AnalogIn goes
    above 0.3
#include "mbed.h"
AnalogIn ain(A0);
DigitalOut led(LED1);
int main() {
        while (1){
                if(ain > 0.3) {
                        led = 1;
                }
                else {
                        led = 0;
                }
        }
}
```



AnalogIn Class Reference

```
#include <AnalogIn.h>
```

Public Member Functions

AnalogIn (PinName pin)
Create an **AnalogIn**, connected to the specified pin.

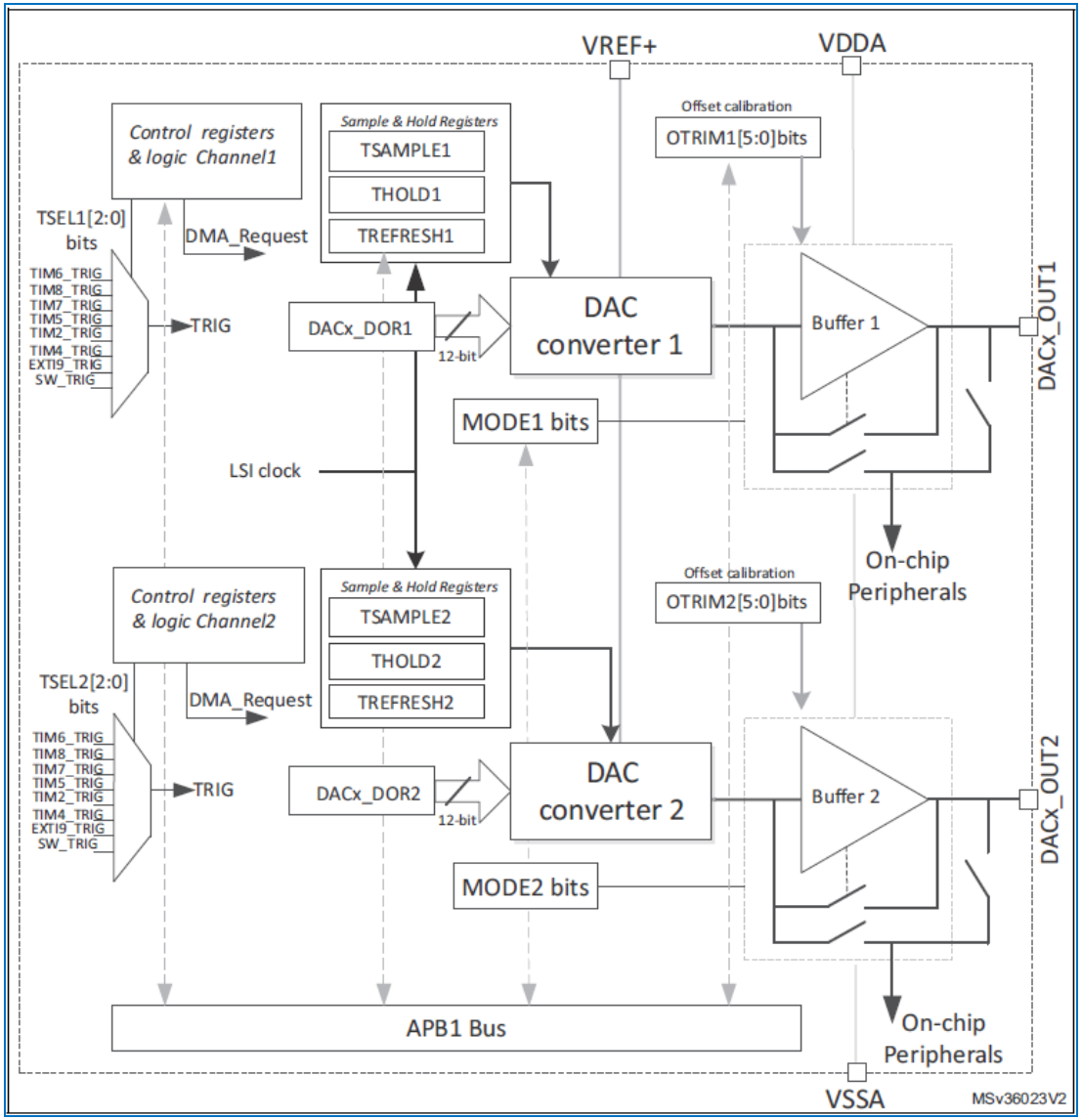
float read ()
Read the input voltage, represented as a float in the range [0.0, 1.0].

unsigned short read_u16 ()
Read the input voltage, represented as an unsigned short in the range [0x0, 0xFFFF].

operator float ()
An operator shorthand for read()

STM32L476 D/A konvertor

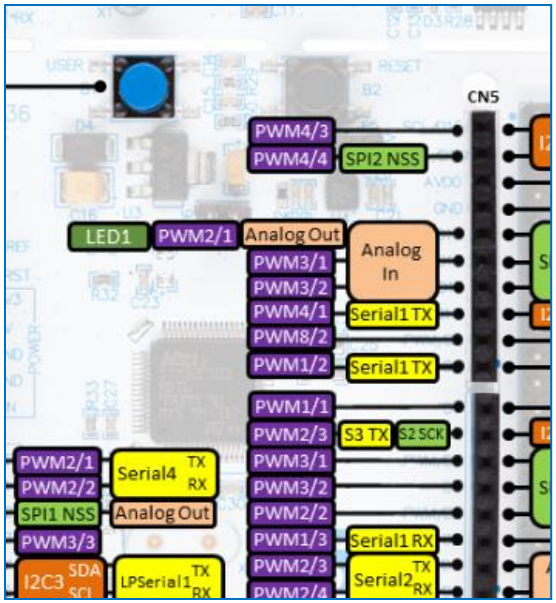
- Dvostruki 12-bitni D/A kovertor sa otpornim razdelinicima
- Eksterni trigeri
- Vreme smirivanja manje od 3us
- Podrška za DMA prenos
- Generator šuma
- Generator trougaonog talasnog oblika



MBED Analogni izlazi

http://mbed.org/handbook/AnalogIOut

```
#include "mbed.h"
AnalogOut signal(pA_5);
int main() {
    while(1) {
        for(float i=0.0; i<1.0; i+=0.1) {
            signal = i;
            wait(0.0001);
        }
    }
}
```



Public Member Functions

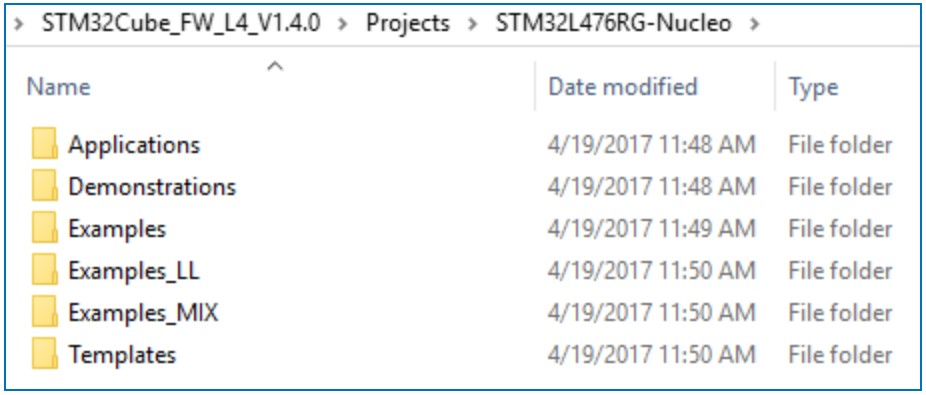
	AnalogOut (PinName pin) Create an **AnalogOut** connected to the specified pin.
void	write (float value) Set the output voltage, specified as a percentage (float)
void	write_u16 (unsigned short value) Set the output voltage, represented as an unsigned short in the range [0x0, 0xFFFF].
float	read () Return the current output voltage setting, measured as a percentage (float)
AnalogOut &	operator= (float percent) An operator shorthand for write() operator float () An operator shorthand for read()

Zadatak 1 – ADC i DAC u MBED okruženju

- Unakrsno povezati AD i DA konvertore dva MBED-a.
- Na jednom MBED-u se izvršava program koji vrši DA konverziju podatka serijski poslatog sa računara iz hiper terminala.
- Na drugom MBED-u se izvršava program koji radi AD konverziju podatka i serijski ga šalje na računar u hiper terminal.

STM CUBE

- Pored standardnih HAL drajvera postoji podrška i za implementaciju sa nižim nivoom abstrakcije, preko takozvanih Low Level drajvera.



LL drajveri

- LL drajveri su namenjeni ekspertima, i postoje za komponente kod kojih optimizacija ima smisla.
- Nema ih za komponente koje zahtevaju kompleksne softevrske drajvere (USB).
- LL drajveri uglavnom ne zahtevaju dodatne strukture podataka i implementiraju se preko inline funkcija.
- Direktno pristupaju hardverskim registrima i obezbeđuju atomične operacije.
- Iz tog razloga često ne postoji drajverski C fajl već samo H fajl u kome su inline funkcije.

Istovremeno korišćenje HAL and LL drajvera

Mixed use of Low Layer APIs and HAL drivers

In this case the Low Layer APIs are used in conjunction with the HAL drivers to achieve direct and register level based operations.

Mixed use is allowed, however some consideration should be taken into account:

- It is recommended to avoid using simultaneously the HAL APIs and the combination of Low Layer APIs for a given peripheral instance. If this is the case, one or more private fields in the HAL PPP handle structure should be updated accordingly.
- For operations and processes that do not alter the handle fields including the initialization structure, the HAL driver APIs and the Low Layer services can be used together for the same peripheral instance.
- The Low Layer drivers can be used without any restriction with all the HAL drivers that are not based on handle objects (RCC, common HAL, flash and GPIO).

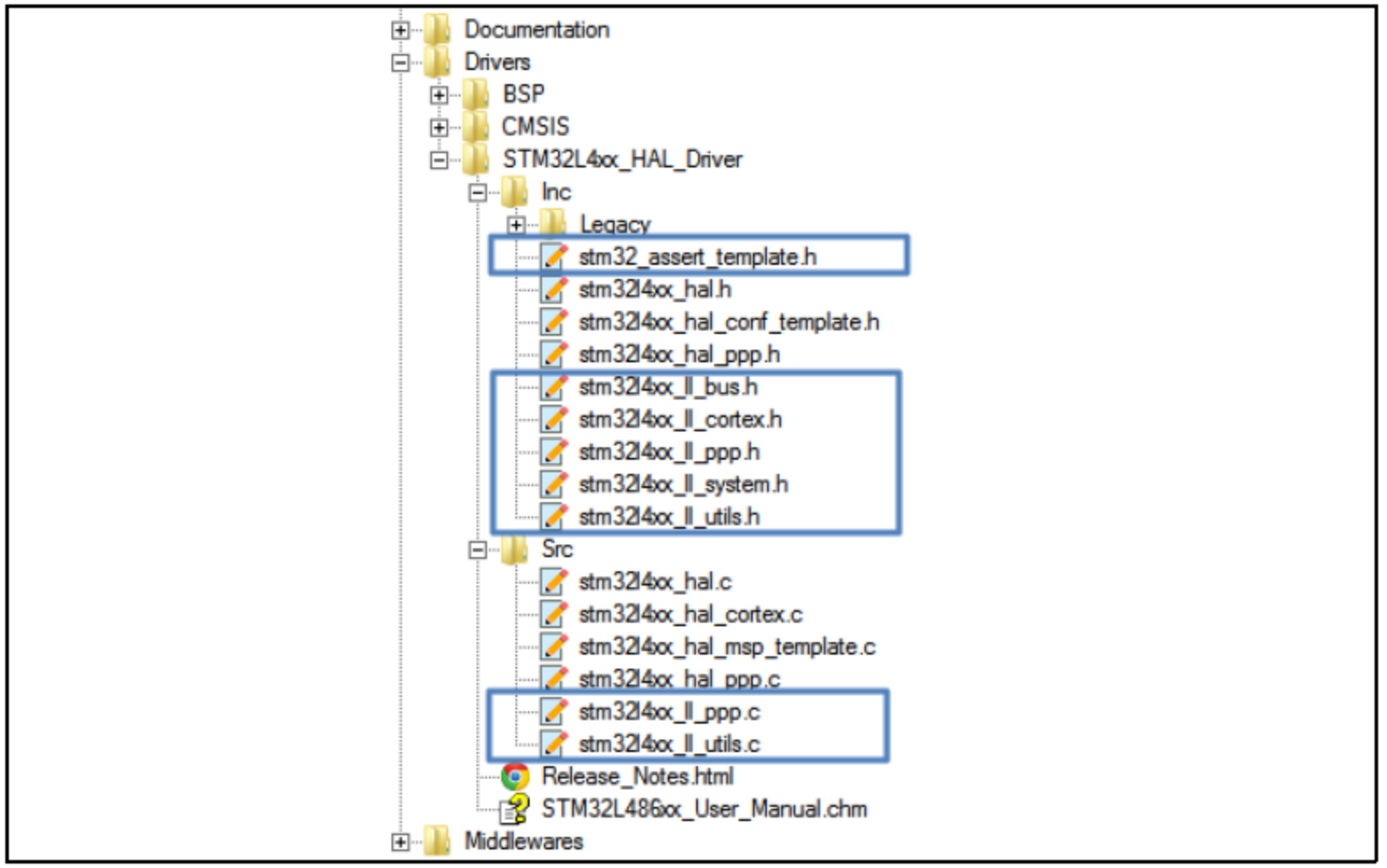
Several examples showing how to use HAL and LL in the same application are provided within stm32l4 firmware package (refer to Examples_MIX projects).

1. When the HAL Init/DeInit APIs are not used and are replaced by the Low Layer macros, the InitMsp() functions are not called and the MSP initialization should be done in the user application.
2. When process APIs are not used and the corresponding function is performed through the Low Layer APIs, the callbacks are not called and post processing or error management should be done by the user application.
3. When the LL APIs is used for process operations, the IRQ handler HAL APIs cannot be called and the IRQ should be implemented by the user application. Each LL driver implements the macros needed to read and clear the associated interrupt flags.

LL drajverski fajlovi

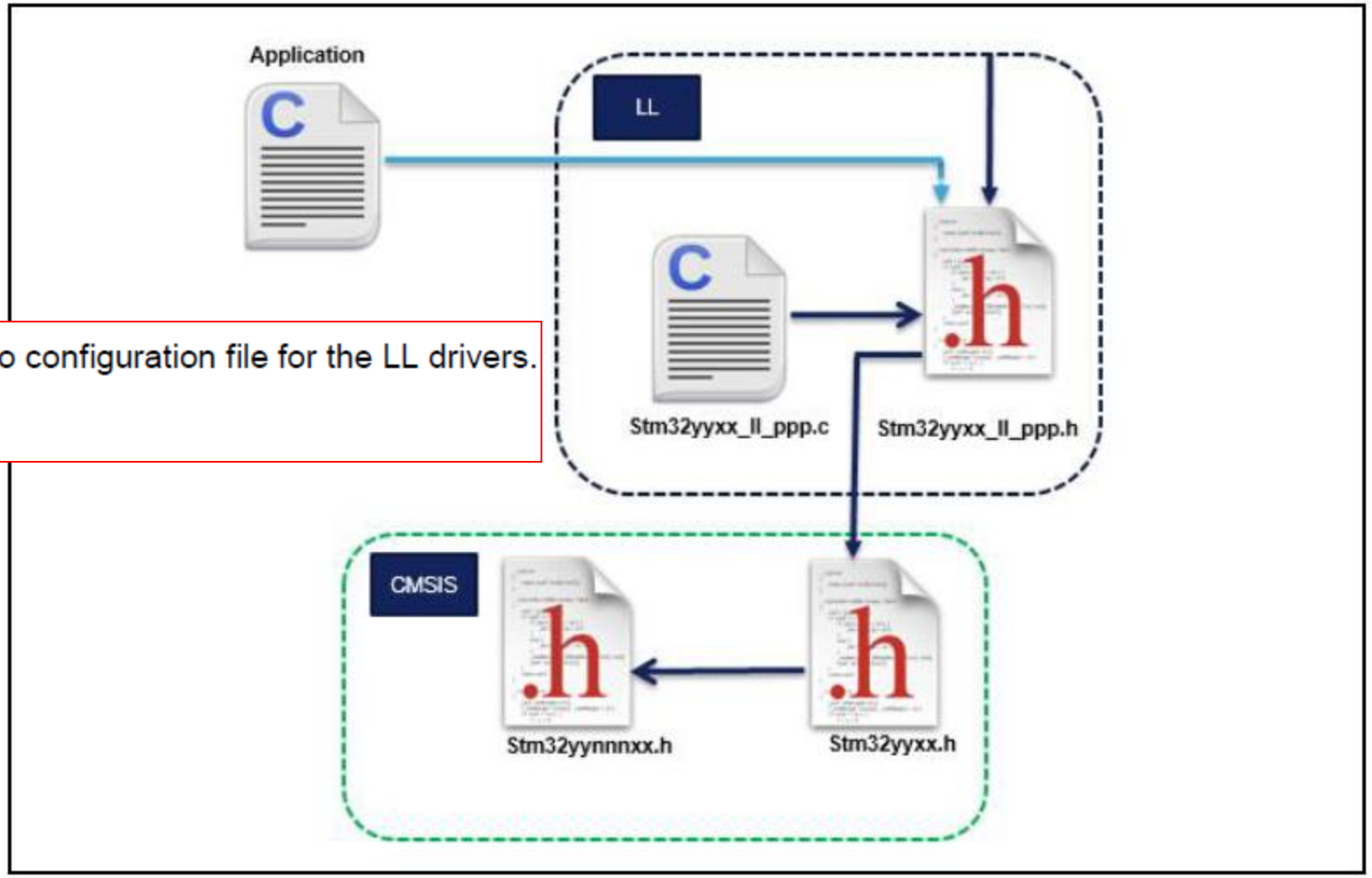
File	Description
stm32l4xx_ll_bus.h	This is the h-source file for core bus control and peripheral clock activation and deactivation *Example: LL_AHB2_GRP1_EnableClock*
stm32l4xx_ll_ppp.h/.c	stm32l4xx_ll_ppp.c provides peripheral initialization functions such as LL_PPP_Init(), LL_PPP_StructInit(), LL_PPP_DeInit(). All the other APIs are definined within stm32l4xx_ll_ppp.h file. The Low Layer PPP driver is a standalone module. To use it, the application must include it in the xx_ll_ppp.h file.
stm32l4xx_ll_cortex.h	Cortex-M related register operation APIs including the Systick, Low power (LL_SYSTICK_xxxxx, LL_LPM_xxxxx "Low Power Mode" ...)
stm32l4xx_ll_utils.h/.c	This file covers the generic APIs: • Read of device unique ID and electronic signature • Timebase and delay management • System clock configuration.
*stm32l4*xx_ll_system.h	System related operations (LL_SYSCFG_xxx, LL_DBGMCU_xxx, LL_FLASH_xxx and LL_VREFBUF_xxx)
stm32_assert_template.h	Template file allowing to define the assert_param macro, that is used when run-time checking is enabled. This file is required only when the LL drivers are used in standalone mode (without calling the HAL APIs). It should be copied to the application folder and renamed to stm32_assert.h.

LL drajverski fajlovi

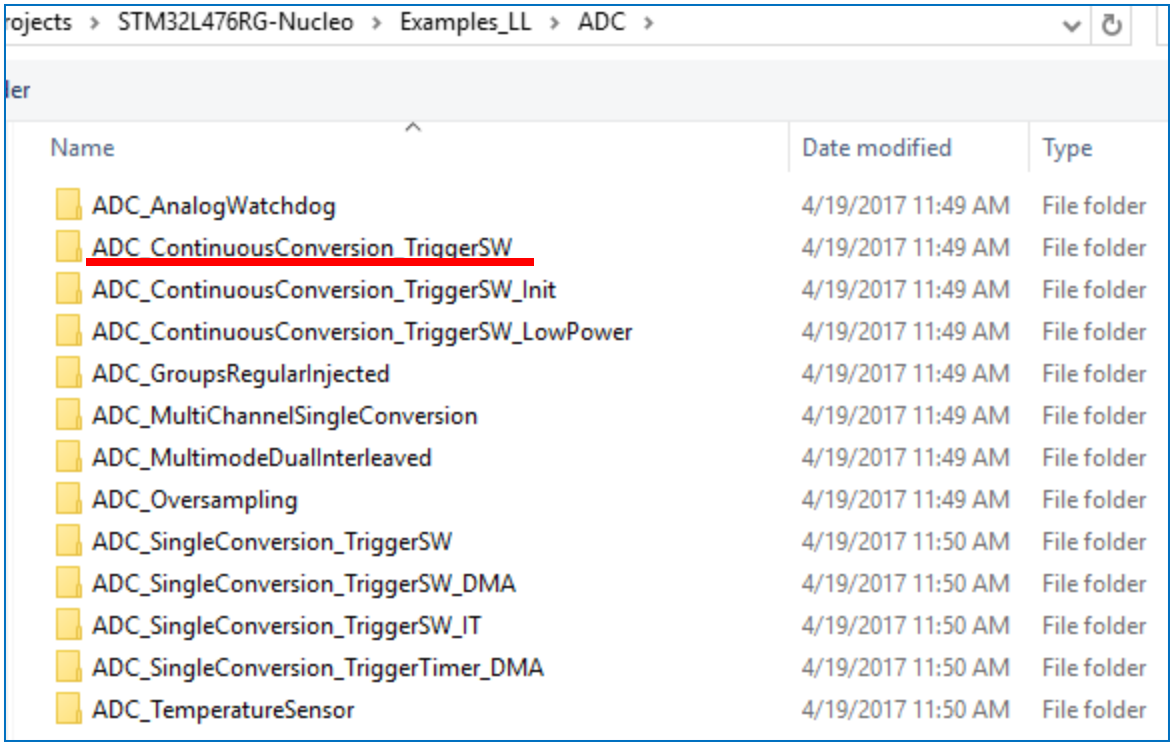


LL drajverski fajlovi

There is no configuration file for the LL drivers.

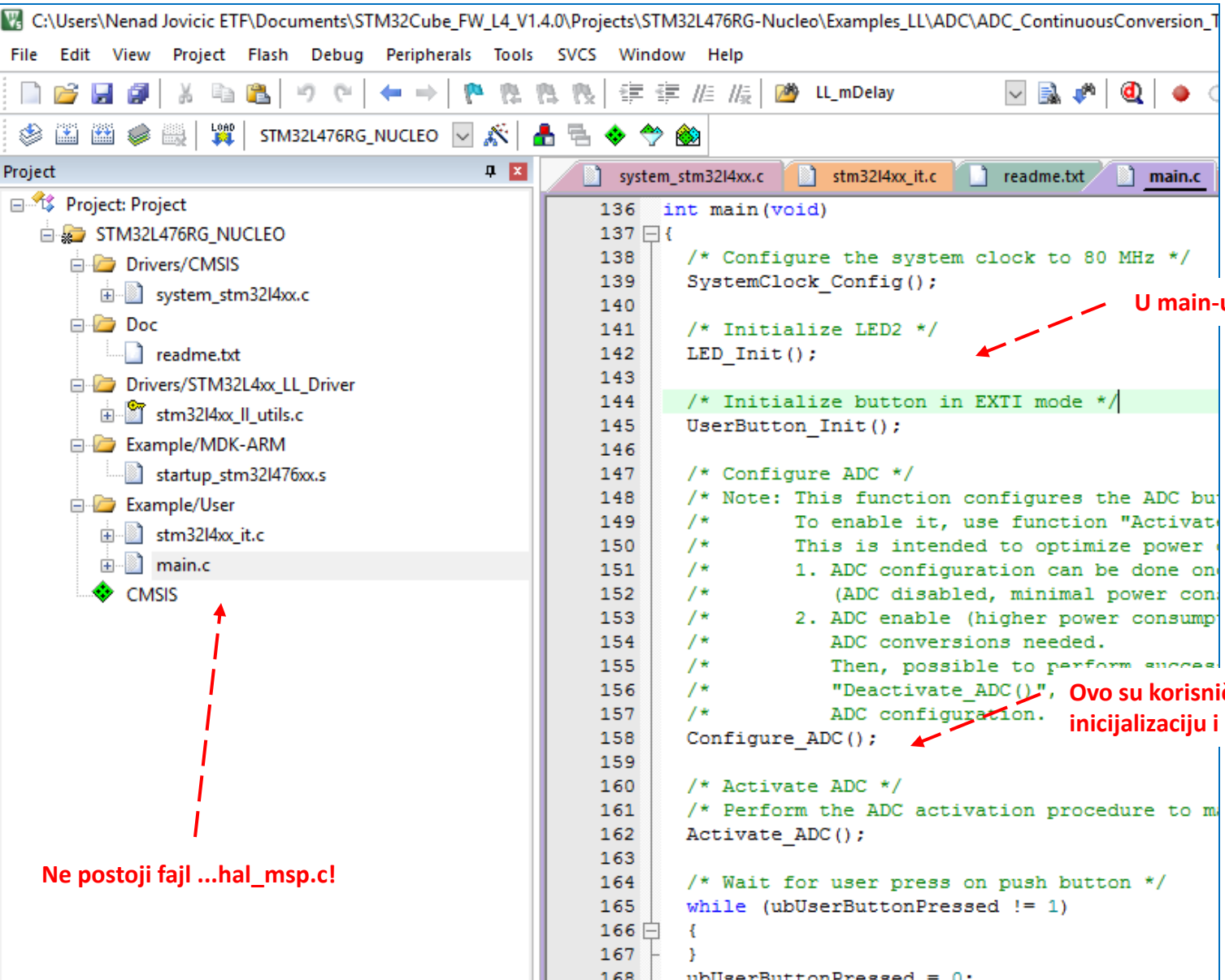


Projekti za ADC



- Testiramo projekat ADC_ContinuousConversion_TriggerSW
- Akvizija se vrši kontinulano sa kanala PA4, tj. Arduino ulaza A2.

Projekat ADC_ContinuousConversion_TriggerSW



system_stm32l4xx.c | stm32l4xx_it.c | readme.txt | main.c | startup_stm32l476xx.s | stm32l4xx_ll_ad

```
172
173   /* Reset status variable of ADC unitary conversion before performing      */
174   /* a new ADC conversion start.                                            */
175   /* Note: Optionally, for this example purpose, check ADC unitary          */
176   /*       conversion status before starting another ADC conversion.        */
177
178   if (ubAdcGrpRegularUnitaryConvStatus != 0)
179   {
180     ubAdcGrpRegularUnitaryConvStatus = 0;
181   }
182
183   /* Init variable containing ADC conversion data */
184   uhADCxConvertedData = VAR_CONVERTED_DATA_INIT_VALUE;
185
186   /* Perform ADC group regular conversion start, poll for conversion        */
187   /* completion.                                                            */
188   ConversionStartPoll_ADC_GrpRegular();
189
190   /* Retrieve ADC conversion data */
191   /* (data scale corresponds to ADC resolution: 12 bits) */
192   uhADCxConvertedData = LL_ADC_REG_ReadConversionData12(ADC1);
193
194   /* Computation of ADC conversions raw data to physical values             */
195   /* using LL ADC driver helper macro.                                      */
196   uhADCxConvertedData_Voltage_mVolt = __LL_ADC_CALC_DATA_TO_VOLTAGE(VDDA_APPLI,
197
198   /* Update status variable of ADC unitary conversion */
199   ubAdcGrpRegularUnitaryConvStatus = 1;
200
201   /* Set LED depending on ADC unitary conversion status */
202   /* - Turn-on if ADC unitary conversion is completed */
203   /* - Turn-off if ADC unitary conversion is not completed */
204   LED_On();
```

Prva konverzija i njen rezutat.

system_stm32l4xx.c | stm32l4xx_it.c | readme.txt | main.c | startup_stm32l476xx.s | stm32l4xx_ll_adc.

```
645   */
646  void ConversionStartPoll_ADC_GrpRegular(void)
647  {
648    #if (USE_TIMEOUT == 1)
649    uint32_t Timeout = 0; /* Variable used for timeout management */
650    #endif /* USE_TIMEOUT */
651
652    /* Start ADC group regular conversion */
653    /* Note: Hardware constraint (refer to description of the function          */
654    /*       below):                                                            */
655    /*       On this STM32 family, setting of this feature is conditioned to    */
656    /*       ADC state:                                                         */
657    /*       ADC must be enabled without conversion on going on group regular,  */
658    /*       without conversion stop command on going on group regular.         */
659    /* Note: In this example, all these checks are not necessary but are        */
660    /*       implemented anyway to show the best practice u
661    /*       corresponding to reference manual procedure.
662    /*       Software can be optimized by removing some of these checks, if     */
663    /*       they are not relevant considering previous settings and actions    */
664    /*       in user application.                                               */
665    if ((LL_ADC_IsEnabled(ADC1) == 1)                 &&
666        (LL_ADC_IsDisableOngoing(ADC1) == 0)          &&
667        (LL_ADC_REG_IsConversionOngoing(ADC1) == 0)    )
668    {
669      LL_ADC_REG_StartConversion(ADC1);
670    }
671    else
672    {
673      /* Error: ADC conversion
674      LED_Blinking(LED_BLINK_ERROR);
675    }
676
677    #if (USE_TIMEOUT == 1)
678    Timeout = ADC_UNITARY_CONVERSION_TIMEOUT_MS;
679    #endif /* USE_TIMEOUT */
680
681    while (LL_ADC_IsActiveFlag_EOC(ADC1) == 0)
682    {
683    #if (USE_TIMEOUT == 1)
684      /* Check Systick counter flag to decrement the time-out value */
685      if (LL_SYSTICK_IsActiveCounterFlag())
686      {
687        if(Timeout-- == 0)
688        {
689        /* Time-out occurred. Set LED to blinking mode */
```

Korisnička funkcija

Funkcije LL drajvera uvek počinju sa LL.

system_stm32l4x

```
172
173    /* Re
174    /* a
175    /* Nc
176    /*
177
178    if (u
179    {
180      ubA
181    }
182
183    /* In
184    uhADC
185
186    /* Pe
187    /* cc
188    Conve
189
190    /* Re
191    /* (d
192    uhADC
193
194    /* Cc
195    /* us
196    uhADC
197
198    /* Up
199    ubAdc
200
201    /* Se
202    /* -
203    /* -
204    LED_C
```

LL drajverske funkcije

system_stm32l4xx.c | stm32l4xx_it.c | readme.txt | main.c | startup_stm32l476xx.s | stm32l4xx_ll_adc.h

```
5804   * @brief  Start ADC group regular conversion.
5805   * @note   On this STM32 family, this function is relevant for both
5806   *         internal trigger (SW start) and external trigger:
5807   *         - If ADC trigger has been set to software start, ADC conversion
5808   *           starts immediately.
5809   *         - If ADC trigger has been set to external trigger, ADC conversion
5810   *           will start at next trigger event (on the selected trigger edge)
5811   *           following the ADC start conversion command.
5812   * @note   On this STM32 family, setting of this feature is conditioned to
5813   *         ADC state:
5814   *         ADC must be enabled without convers
5815   *         without conversion stop command on
5816   * @rmtoll CR       ADSTART        LL_ADC_REG_StartConversion
5817   * @param  ADCx ADC instance
5818   * @retval None
5819   */
5820  __STATIC_INLINE void LL_ADC_REG_StartConversion(ADC_TypeDef *ADCx)
5821  {
5822    /* Note: Write register with some additional bits forced to state reset     */
5823    /*       instead of modifying only the selected bit for this function,     */
5824    /*       to not interfere with bits with HW property "rs".                 */
5825    MODIFY_REG(ADCx->CR,
5826               ADC_CR_BITS_PROPERTY_RS,
5827               ADC_CR_ADSTART);
5828  }
```

Ovo je primer INLINE funkcije i stoga se njena definicija nalazi u .h fajlu

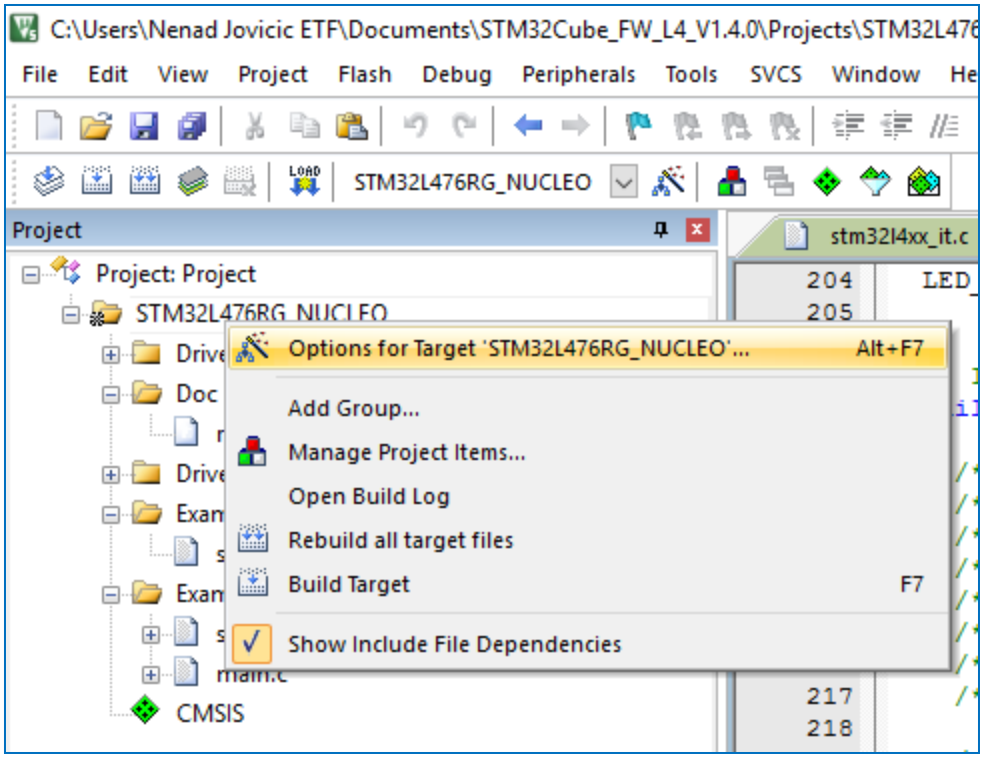
Glavni program

stm32l4xx_it.c | readme.txt | main.c* | startup_stm32l476xx.s | stm32l4xx_ll_adc.h | stm32l4xx_ll_utils.c

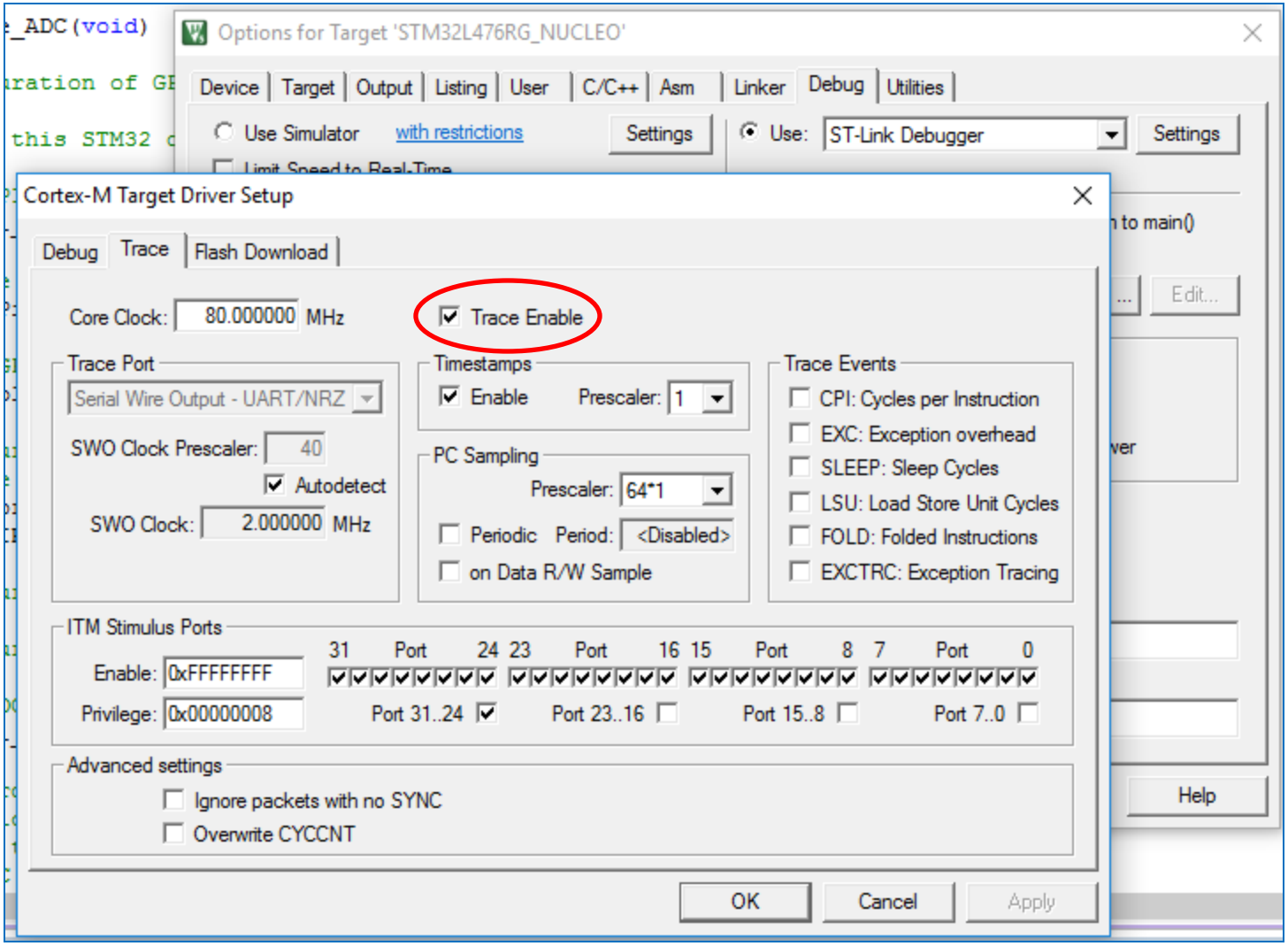
```
06
07    /* Infinite loop */
08    while (1)
09    {
10      /* Note: At this step, ADC is performing ADC conversions continuously,    */
11      /*       indefinitely (ADC continuous mode enabled in this example).      */
12      /*       Main program reads frequently ADC conversion data                */
13      /*       (without waiting for end of each conversion: soft
14      /*       when main program execution pointer is available
15      /*       some ADC conversions data unread and overwritten
16      /*       overrun IT is disabled and overrun flag is ignore
17      /*       and stores it into the same variable.
18
19      /* Retrieve ADC conversion data */
20      uhADCxConvertedData = LL_ADC_REG_ReadConversionData12(ADC1);
21
22      /* Computation of ADC conversions raw data to physical values             */
23      /* using LL ADC driver helper macro.                                      */
24      uhADCxConvertedData_Voltage_mVolt = __LL_ADC_CALC_DATA_TO_VOLTAGE(VDDA_APPLI, uhADCxConvertedData, LL_ADC_R
25
26
27    }
```

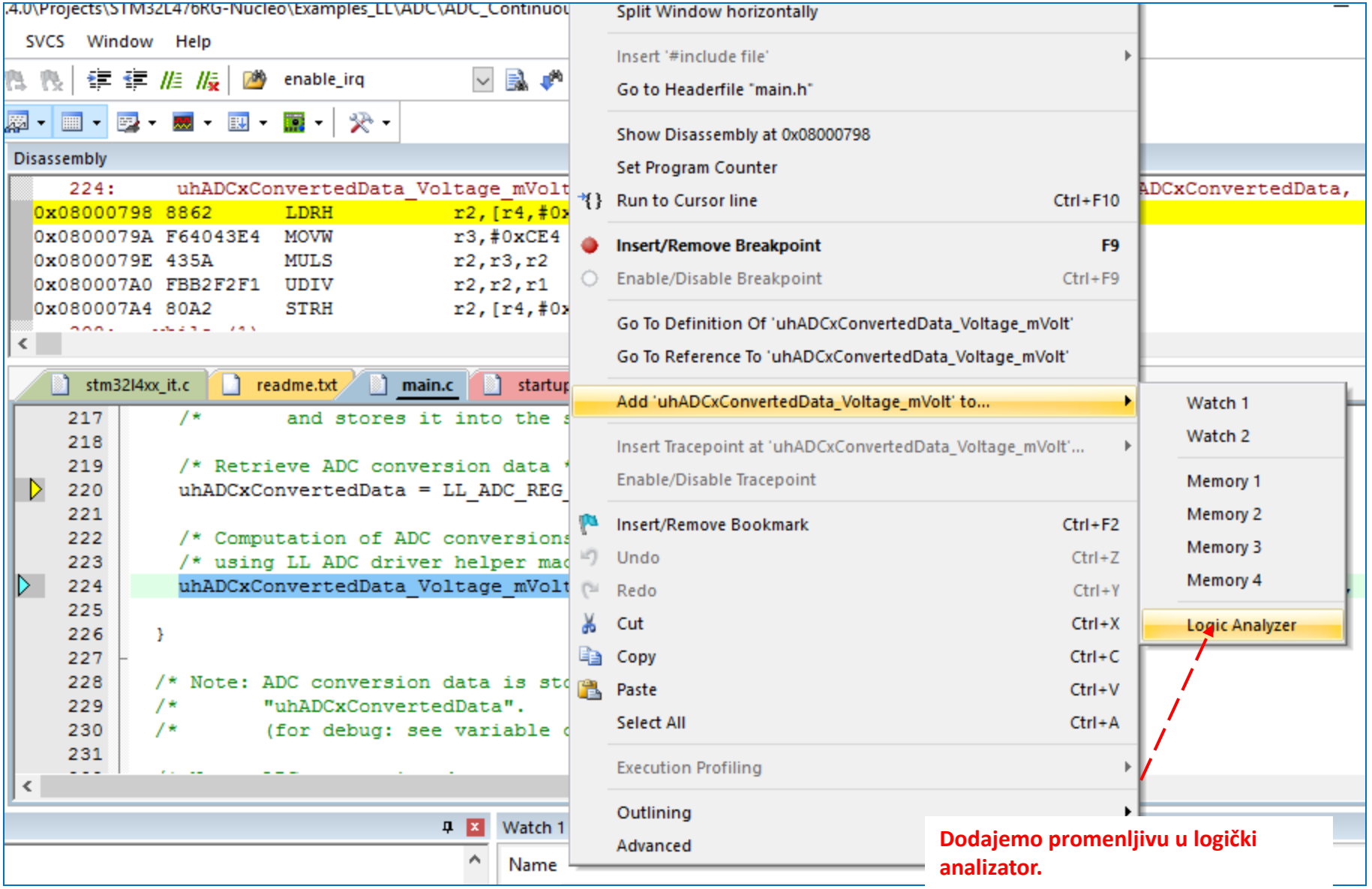
Konverzija se vrši kontinulano maksimalnom brzinom određenom sample/hold vremenom i taktom ADC-a, a rezultat se očitava po potrebi.

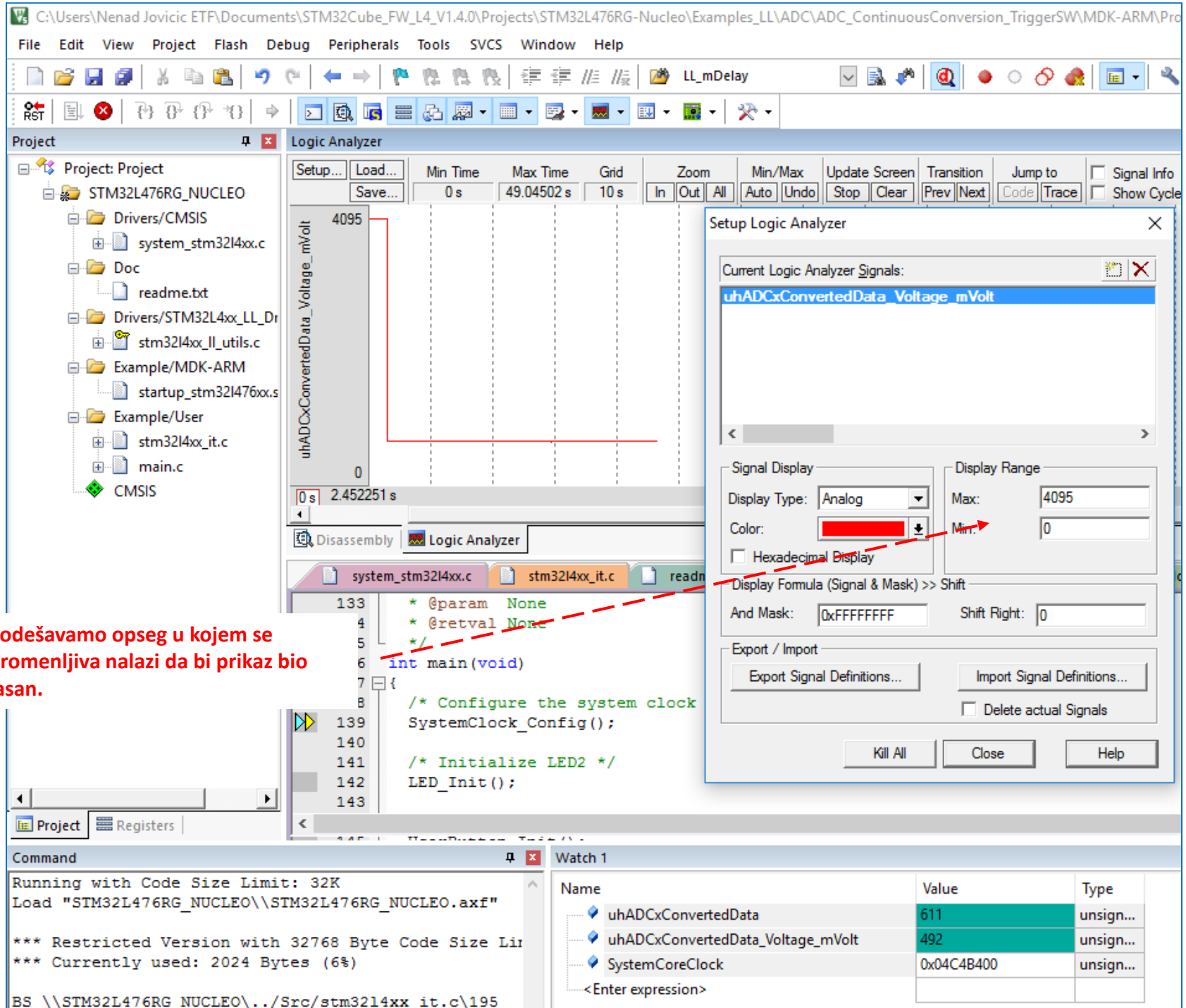
Cortex-M Trace



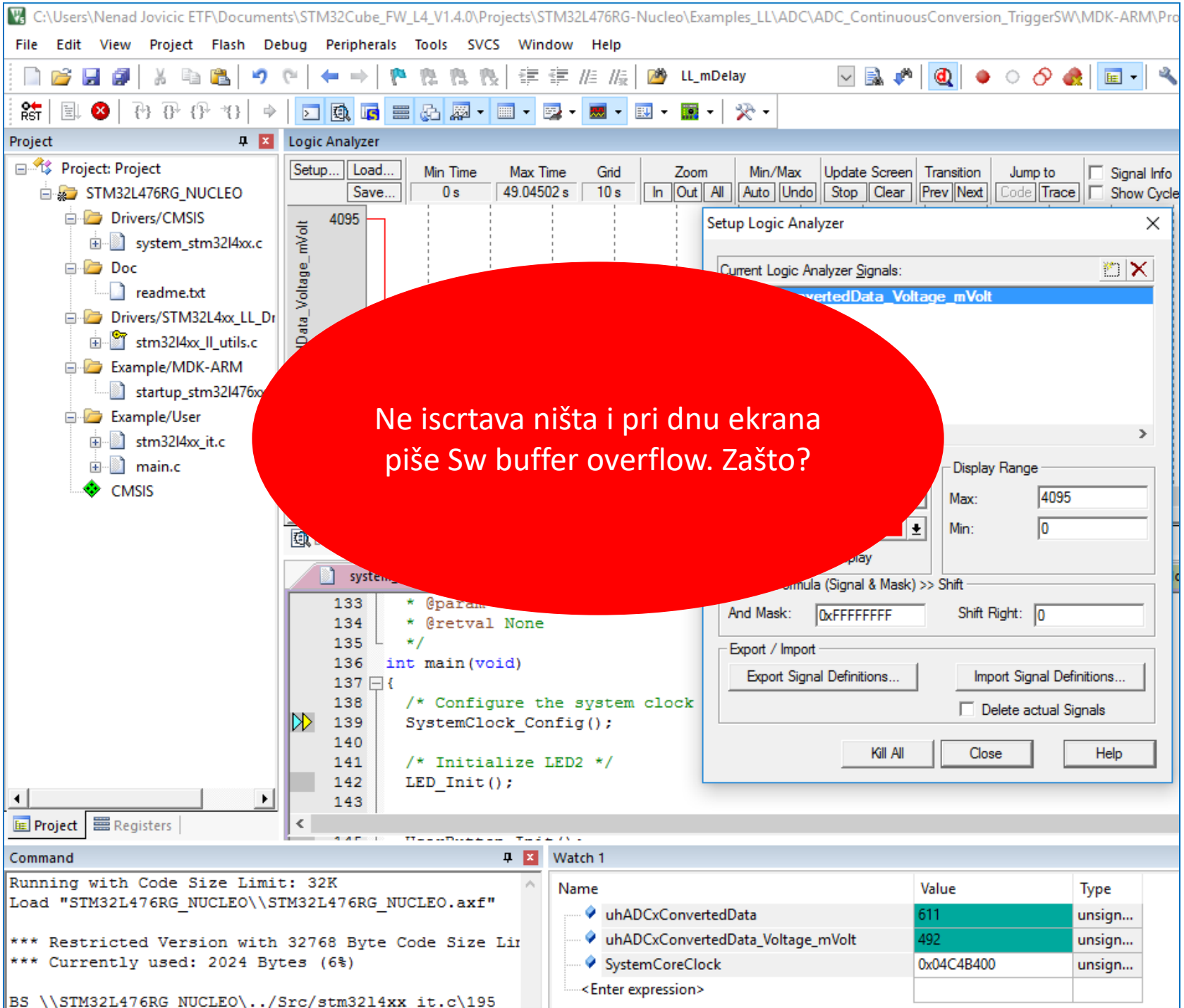
Cortex-M Trace



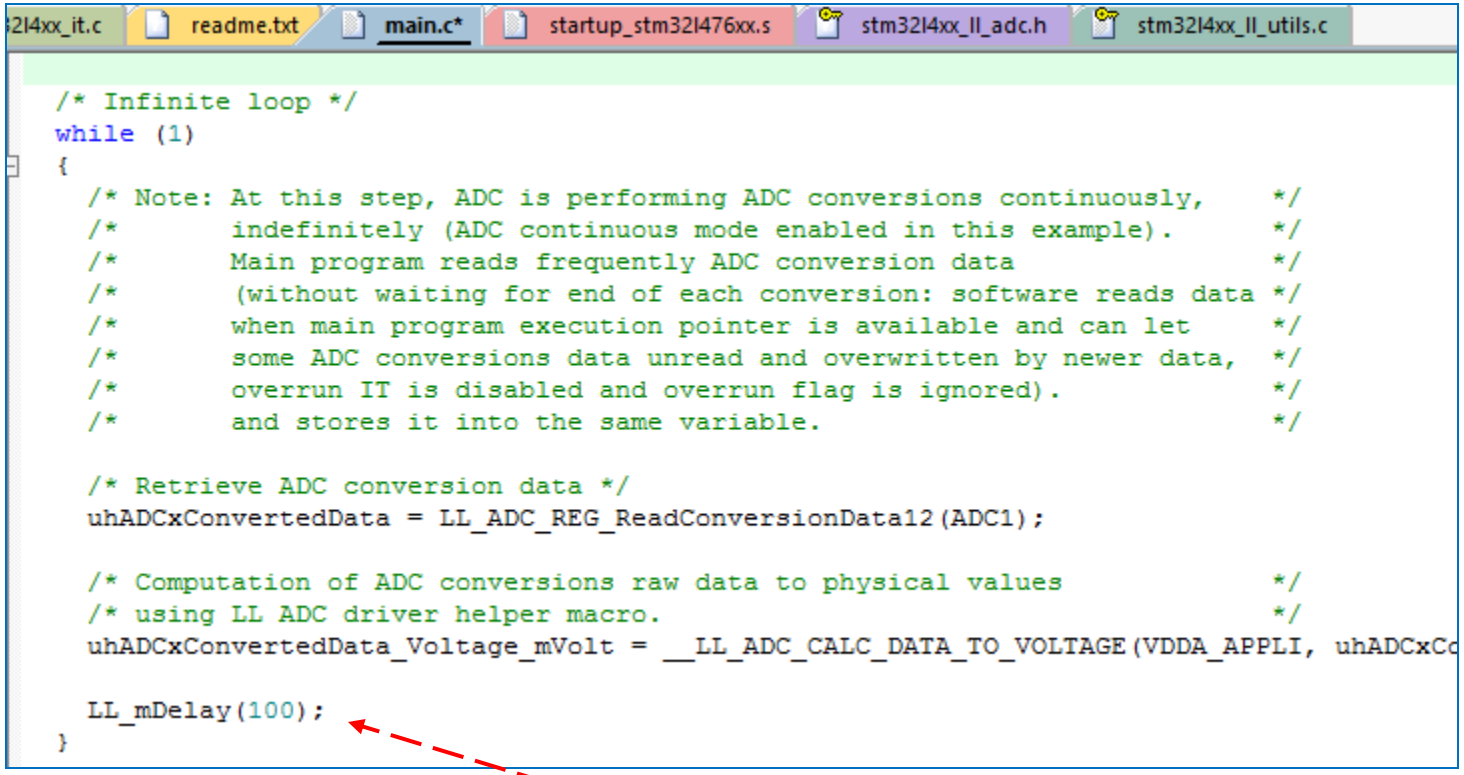

SVCS Window Help
enable_irq
Disassembly
224: uhADCxConvertedData_Voltage_mVolt
0x08000798 8862 LDRH r2,[r4,#0x
0x0800079A F64043E4 MOVW r3,#0xCE4
0x0800079E 435A MULS r2,r3,r2
0x080007A0 FBB2F2F1 UDIV r2,r2,r1
0x080007A4 80A2 STRH r2,[r4,#0x
stm32l4xx_it.c
readme.txt
main.c
startup
217 /* and stores it into the
218
219 /* Retrieve ADC conversion data
220 uhADCxConvertedData = LL_ADC_REG_
221
222 /* Computation of ADC conversions
223 /* using LL ADC driver helper mac
224 uhADCxConvertedData_Voltage_mVolt
225
226 }
227
228 /* Note: ADC conversion data is sto
229 /* "uhADCxConvertedData".
230 /* (for debug: see variable
231
Split Window horizontally
Insert '#include file'
Go to Headerfile "main.h"
Show Disassembly at 0x08000798
Set Program Counter
Run to Cursor line Ctrl+F10
Insert/Remove Breakpoint F9
Enable/Disable Breakpoint Ctrl+F9
Go To Definition Of 'uhADCxConvertedData_Voltage_mVolt'
Go To Reference To 'uhADCxConvertedData_Voltage_mVolt'
Add 'uhADCxConvertedData_Voltage_mVolt' to...
Insert Tracepoint at 'uhADCxConvertedData_Voltage_mVolt'...
Enable/Disable Tracepoint
Insert/Remove Bookmark Ctrl+F2
Undo Ctrl+Z
Redo Ctrl+Y
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Select All Ctrl+A
Execution Profiling
Outlining
Advanced
Watch 1
Memory 1
Memory 2
Memory 3
Memory 4
Logic Analyzer
Watch 1
Name
ADCxConvertedData,
Dodajemo promenljivu u logički analizator.



Podešavamo opseg u kojem se promenljiva nalazi da bi prikaz bio jasan.


C:\Users\Nenad Jovicic ETF\Documents\STM32Cube_FW_L4_V1.4.0\Projects\STM32L476RG-Nucleo\Examples_LL\ADC\ADC_ContinuousConversion_TriggerSW\MDK-ARM\Pro
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
LL_mDelay
Project
Project: Project
STM32L476RG_NUCLEO
Drivers/CMSIS
system_stm32l4xx.c
Doc
readme.txt
Drivers/STM32L4xx_LL_Dr
stm32l4xx_ll_utils.c
Example/MDK-ARM
startup_stm32l476xx
Example/User
stm32l4xx_it.c
main.c
CMSIS
Project
Registers
Logic Analyzer
Setup... Load... Save...
Min Time 0 s
Max Time 49.04502 s
Grid 10 s
Zoom In Out All
Min/Max Auto Undo
Update Screen Stop Clear
Transition Prev Next
Jump to Code Trace
Signal Info
Show Cycle
4095
Data_Voltage_mVolt
Setup Logic Analyzer
Current Logic Analyzer Signals:
vertedData_Voltage_mVolt
Display Range
Max: 4095
Min: 0
mula (Signal & Mask) >> Shift
And Mask: 0xFFFFFFFF
Shift Right: 0
Export / Import
Export Signal Definitions...
Import Signal Definitions...
Delete actual Signals
Kill All
Close
Help
Ne iscrtava ništa i pri dnu ekrana
piše Sw buffer overflow. Zašto?
133 * @param
134 * @retval None
135 */
136 int main(void)
137 {
138 /* Configure the system clock
139 SystemClock_Config();
140
141 /* Initialize LED2 */
142 LED_Init();
143
Command
Running with Code Size Limit: 32K
Load "STM32L476RG_NUCLEO\\STM32L476RG_NUCLEO.axf"
*** Restricted Version with 32768 Byte Code Size Lir
*** Currently used: 2024 Bytes (6%)
BS \\STM32L476RG_NUCLEO\../Src/stm32l4xx_it.c\195
Watch 1
Name Value Type
uhADCxConvertedData 611 unsign...
uhADCxConvertedData_Voltage_mVolt 492 unsign...
SystemCoreClock 0x04C4B400 unsign...
<Enter expression>

Delay....



Bez ovog delay-a pri svakom novom ažuriranju vrednosti sa ADC-a generiše se prenos što ST-link V2 lite ne može dovoljno brzo da obradi.

ULINK*pro*: Debug, Serial Wire and Streaming Trace

JTAG
SWD
SWO 100 Mpbs
ETM Streaming

- Flash Programming + Run-Control
- Memory + Breakpoint (access while running)
- Serial Wire Trace capturing up to **100 Mbit/sec** (Manchester mode)
- 50 MHz JTAG clock speed
- ETM Trace Capturing up to **800 Mbit/sec**
- **Streaming Trace**: Instruction Trace, Code Coverage, Performance Analysis

ULINK*pro* D: Debug and Fast Serial Wire Trace

JTAG
SWD
SWO 100 Mpbs

- Flash Programming + Run-Control
- Memory + Breakpoint (access while running)
- Serial Wire Trace capturing up to **100 Mbit/sec** (Manchester mode)
- 50 MHz JTAG clock speed

Performanse TRACE-a zavise od debugger-a ali i do softvera i drajvera na PC-ju...

ULINK*plus*: Debug, Serial Wire Trace, Test I/O, and Power Measurement

JTAG
SWD
SWO 50 Mpbs
Power & GPIO

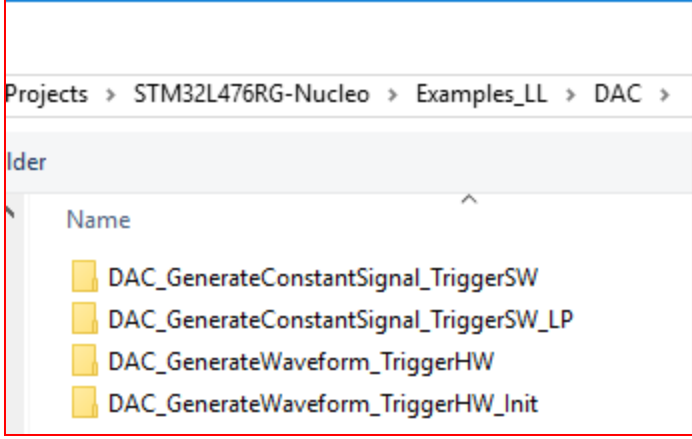
- Flash Programming + Run-Control
- Memory + Breakpoint (access while running)
- Serial Wire Trace capturing up to 50 Mbit/sec (UART mode)
- 20 MHz JTAG clock speed
- Power measurement for efficient source code
- I/Os for test automation and continuous integration

ULINK2: Debug and Serial Wire Trace

JTAG
SWD
SWO 1 Mpbs

- Flash Programming + Run-Control
- Memory + Breakpoint (access while running)
- Serial Wire Trace capturing up to 1 Mbit/sec (UART mode)
- 10 MHz JTAG clock speed

DAC Cube



- Testiramo projekat
- DAC_GenerateWaveform_TriggerHW
- Korišćenjem tajmerskog prekida i DMA prenosa generiše se periodičan promenljiv signal na DA konvertoru, izlazu PA4, tj. arduino kanalu A2.

Projekat
DAC_GenerateWaveform_TriggerHW

em_stm32l4xx.c | main.h | readme.txt | startup_stm32l476xx.s | main.c

```
/** @addtogroup DAC_GenerateWaveform_TriggerHW
  * @{
  */

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/

/* Definitions of environment analog values */
  /* Value of analog reference voltage (Vref+), connected to analog voltage   */
  /* supply Vdda (unit: mV).                                                   */
  #define VDDA_APPLI                       ((uint32_t)3300)

/* Definitions of data related to this example */
  /* Full-scale digital value with a resolution of 12 bits (voltage range     */
  /* determined by analog voltage references Vref+ and Vref-,                  */
  /* refer to reference manual).                                               */
  #define DIGITAL_SCALE_12BITS             ((uint32_t)0x0FFF)

/* Definitions of waveform generation values */
  /* Waveform generation: parameters of waveform */
  /* Waveform amplitude (unit: mV) */
  #define WAVEFORM_AMPLITUDE          (VDDA_APPLI)
  /* Waveform amplitude (unit: Hz) */
  #define WAVEFORM_FREQUENCY          ((uint32_t)1)
  /* Size of array containing DAC waveform samples */
  #define WAVEFORM_SAMPLES_SIZE       (sizeof (WaveformSine_12bits_32samples) / sizeof (uint16_t))

  /* Waveform generation: parameters of timer (used as DAC trigger) */
  /* Timer frequency (unit: Hz). With a timer 16 bits and time base           */
  /* freq min 1Hz, range is min=1Hz, max=32kHz.                                */
  #define WAVEFORM_TIMER_FREQUENCY                (WAVEFORM_FREQUENCY * WAVEFORM_SAMPLES_SIZE)
  /* Timer minimum frequency (unit: Hz), used to calculate frequency range.   */
  /* With a timer 16 bits, maximum frequency will be 32000 times this value.  */
  #define WAVEFORM_TIMER_FREQUENCY_RANGE_MIN      ((uint32_t)     1)
  /* Timer prescaler maximum value (0xFFFF for a timer 16 bits) */
  #define WAVEFORM_TIMER_PRESCALER_MAX_VALUE      ((uint32_t)0xFFFF-1)
```

Učestanost sinusoide na izlazu.
Smanjiti na 1Hz da bi bila “vidljiva”

Projekat DAC_GenerateWaveform_TriggerHW

tem_stm32l4xx.c | main.h | readme.txt | startup_stm32l476xx.s | main.c

```
  * @retval None
  */
int main(void)
{
  /* Configure the system clock to 80 MHz */
  SystemClock_Config();

  /* Initialize LED2 */
  LED_Init();

  /* Initialize button in EXTI mode */
  UserButton_Init();

  /* Wait for User push-button press */
  WaitForUserButtonPress();

  /* Turn-off LED2 */
  LED_Off();

  /* Configure DMA for data transfer from DAC */
  Configure_DMA();

  /* Configure timer as a time base used to trig DAC conversio
  Configure_TIM_TimeBase_DAC_trigger();

  /* Configure DAC channel */
  Configure_DAC();

  /* Activate DAC channel*/
  /* Enable DAC channel */
  LL_DAC_Enable(DAC1, LL_DAC_CHANNEL_1);

  /* Note: In this example, DAC channel trigger is set just af
  /*       DAC channel enable.
  /*       However, in user application, depending on accuracy
  /*       a delay may be required between DAC channel enable
  /*       DAC channel trigger.
  /*       Refer to description of function @ref LL_DAC_Enable

  /* Enable DAC channel trigger */
  LL_DAC_EnableTrigger(DAC1, LL_DAC_CHANNEL_1);

  /* Turn-on LED2 */
  LED_On();

  /* Infinite loop */
  while (1)
  {
  }
}
```

Tajmer trigeruje DA konverziju i DAM prenos

system_stm32l4xx.c | main.h | readme.txt | startup_stm32l476xx.s | main.c

```
102
103  const uint16_t WaveformSine_12bits_32samples[] =
104  {
105  __WAVEFORM_AMPLITUDE_SCALING(2048),
106  __WAVEFORM_AMPLITUDE_SCALING(2447),
107  __WAVEFORM_AMPLITUDE_SCALING(2831),
108  __WAVEFORM_AMPLITUDE_SCALING(3185),
109  __WAVEFORM_AMPLITUDE_SCALING(3495),
110  __WAVEFORM_AMPLITUDE_SCALING(3750),
111  __WAVEFORM_AMPLITUDE_SCALING(3939),
112  __WAVEFORM_AMPLITUDE_SCALING(4056),
113  __WAVEFORM_AMPLITUDE_SCALING(4095),
114  __WAVEFORM_AMPLITUDE_SCALING(4056),
115  __WAVEFORM_AMPLITUDE_SCALING(3939),
116  __WAVEFORM_AMPLITUDE_SCALING(3750),
117  __WAVEFORM_AMPLITUDE_SCALING(3495),
118  __WAVEFORM_AMPLITUDE_SCALING(3185),
119  __WAVEFORM_AMPLITUDE_SCALING(2831),
120  __WAVEFORM_AMPLITUDE_SCALING(2447),
121  __WAVEFORM_AMPLITUDE_SCALING(2048),
122  __WAVEFORM_AMPLITUDE_SCALING(1649),
123  __WAVEFORM_AMPLITUDE_SCALING(1265),
124  __WAVEFORM_AMPLITUDE_SCALING(911),
125  __WAVEFORM_AMPLITUDE_SCALING(601),
126  __WAVEFORM_AMPLITUDE_SCALING(346),
127  __WAVEFORM_AMPLITUDE_SCALING(157),
128  __WAVEFORM_AMPLITUDE_SCALING(40),
129  __WAVEFORM_AMPLITUDE_SCALING(0),
130  __WAVEFORM_AMPLITUDE_SCALING(40),
131  __WAVEFORM_AMPLITUDE_SCALING(157),
132  __WAVEFORM_AMPLITUDE_SCALING(346),
133  __WAVEFORM_AMPLITUDE_SCALING(601),
134  __WAVEFORM_AMPLITUDE_SCALING(911),
135  __WAVEFORM_AMPLITUDE_SCALING(1265),
136  __WAVEFORM_AMPLITUDE_SCALING(1649)
137  };
```

Tabela iz koje DMA periodično uzima odbirke i prebacuje u DAC.

Zadatak 2

- Povezati dve Nucleo pločice, tako da su im povezane mase i analogni arduino ulaz/izlaz A2.
- Na jednoj pločici pokrenuti projekat DAC_GenerateWaveform_TriggerHW a na drugoj projekat ADC_ContinuousConversion_TriggerSW
- Trace bi trebalo da daje talasni oblik u logic analyzer prozoru.