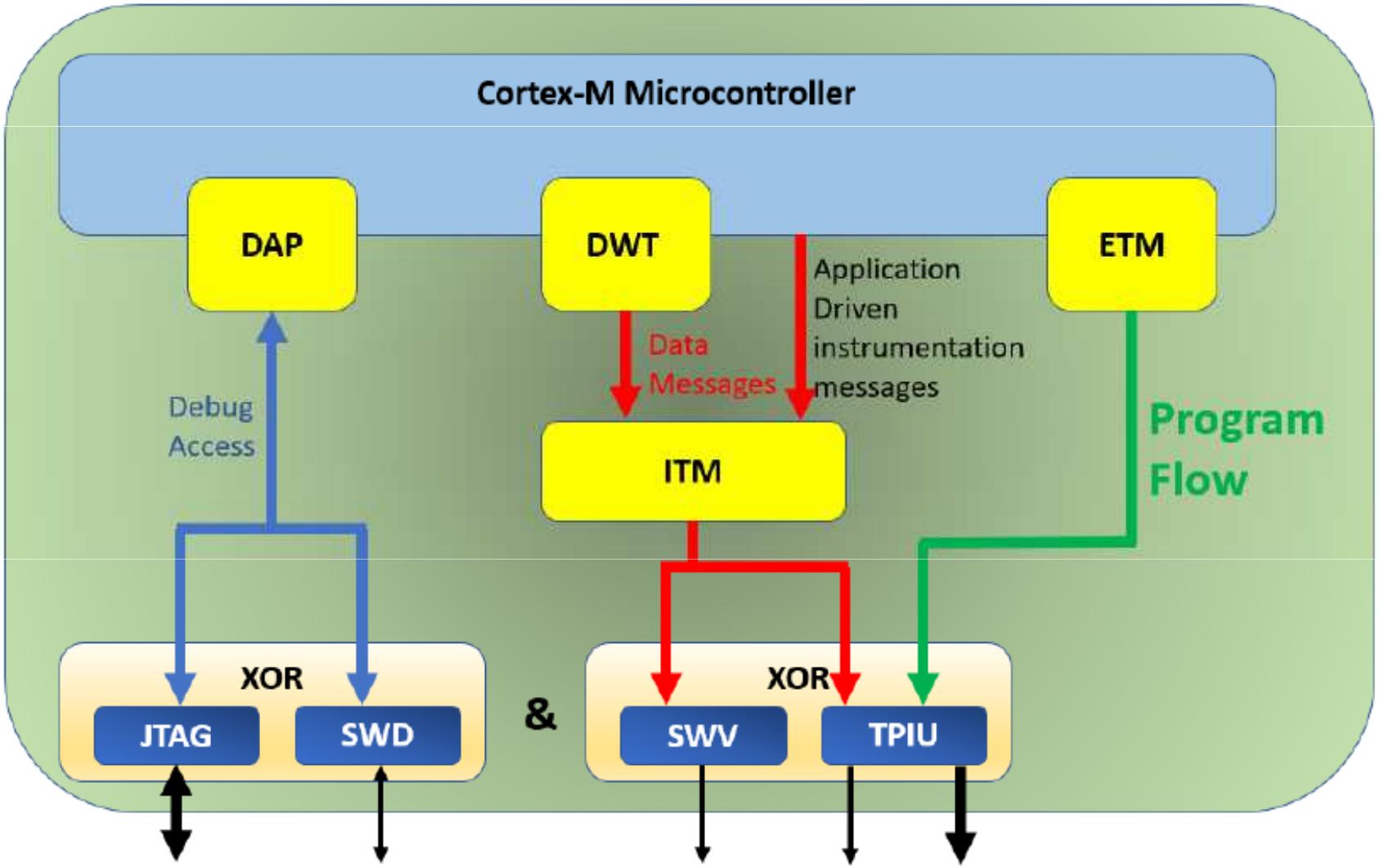
TRACE32 - VÝVOJOVÉ MOŽNOSTI CORTEX-M

Procesory Cortex-M jsou vybaveny trasovacím systémem, který umožňuje sledovat stav proměnných a událostí, ke kterým v běžícím programu došlo.



TRACE32 - VÝVOJOVÉ MOŽNOSTI CORTEX-M

- **Embedded Trace Macrocell (ETM) –** je nejvýkonnějším stupněm, který umožňuje přenést data z programu do jednotky **TPIU**. Po naformátování je umožněn jejich přenos mimo čip. Množství trasovacích dat je omezeno velikostí vyrovnávací paměti v externím komunikačním přípravku.
- **Instrumentation Trace Macrocell (ITM) -** Jednotka poskytuje tři hlavní funkce: trasování pomocí softwarových nástrojů; integrace trasovacích paketů z DWT do trasovacího streamu; generace časových značek pro trasovací pakety ITM a DWT.
- **Data Watchpoint and Trace (DWT) -** Tato volitelná součást poskytuje řadu funkcí podobných trasování: Přerušení trasování; Trasování dat; Spouštěč ETM; PC Sampler a Trigger. The výstup z DWT se přivádí do ITM pro formátování a uložení do trasovacího streamu.
- **Micro Trace Buffer (MTB) -** Umožňuje ukládat data z programu do interní SRAM. Data lze číst pomocí JTAG nebo sériového rozhraní SWD (Serial Wire Debug). Velikost SRAM a umístění vyrovnávací paměti jsou programově konfigurovatelné. Velikost vyrovnávací paměti SRAM omezuje množství trasovacích dat, které lze zachytit. Zápisy do SRAM mají přednost před systémovými zápisy do rozhraní AHB-Lite s jedním nebo více stavy čekání. To může ovlivnit dobu běhu programu ve sledované aplikaci.

- **Trace Port Interface Unit (TPIU)** – Jednotka zajišťující přenos dat ke konečnému cíli tj. mimo čip. TPIU může agregovat trasovací data z více zdrojů do jednoho streamu, což umožňuje sledování procesů ve vícejádrových návrzích. Jednotka podporuje dva režimy: Serial Wire Viewer **(SWV)** nebo Serial Wire Output **(SWO)** a paralelní trasování. SWV nebo SWO je jednobitový trasovací port určený pro trasování s nízkou rychlostí - obvykle z ITM / DWT. Je-li vybrána tato možnost TPIU „zahodí“ pakety ETM a nebude je přenášet. Při paralelním spojením s podporující jednotkou (ULINK*pro*) jsou všechny vybrané pakety ze všech zdrojů přeneseny. Paralelní port může mít šířku 1 až 4 bity a může být taktováno nezávisle na CPU.

  Cortex-M0, M1 - Bez možnosti trasování.
  Cortex-M0+ - disponuje MTB
  Cortex-M23 - disponuje DWT, případně ETM or MTB.
  Cortex-M33 - disponuje ITM, DWT, ETM, MTB.
  Cortex-M3 - disponuje ITM, DWT, ETM.
  Cortex-M4 - disponuje ITM, DWT, ETM.
  Cortex-M7 - disponuje ITM, DWT, ETM.

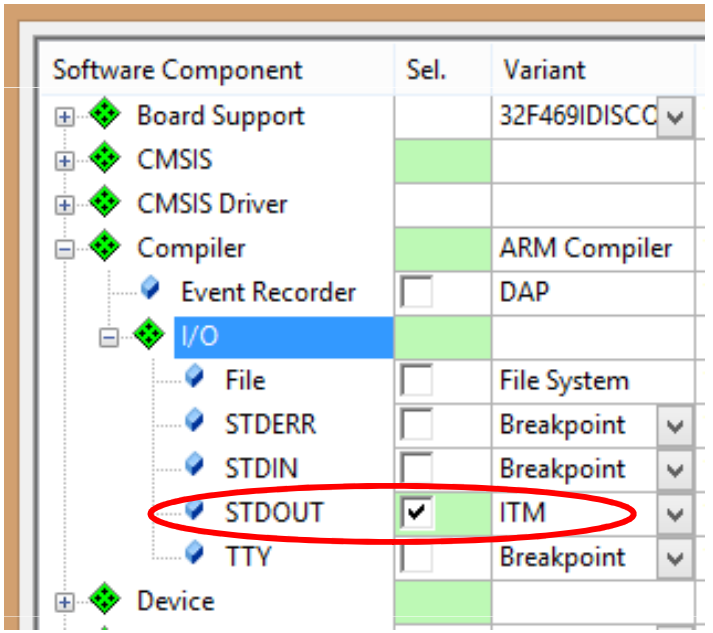
ITM - INSTRUMENTATION TRACE MACROCELL

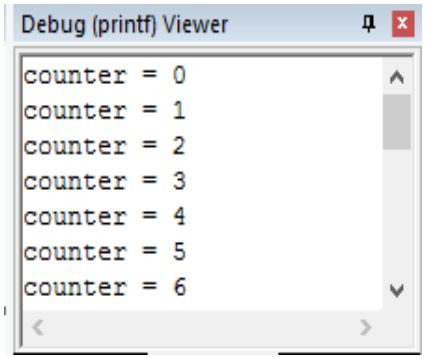
PORT ITM 0 – je k dispozici pro funkci ***printf***, která vyžaduje minimální uživatelský kód. Po zápisu do portu ITM nejsou potřeba žádné strojové cykly CPU, aby se data dostala z procesoru do zobrazení v okně **Debug *printf* Viewer** prostředí µVision. Tato data je možné odeslat z ITM do souboru.

NASTAVENÍ:

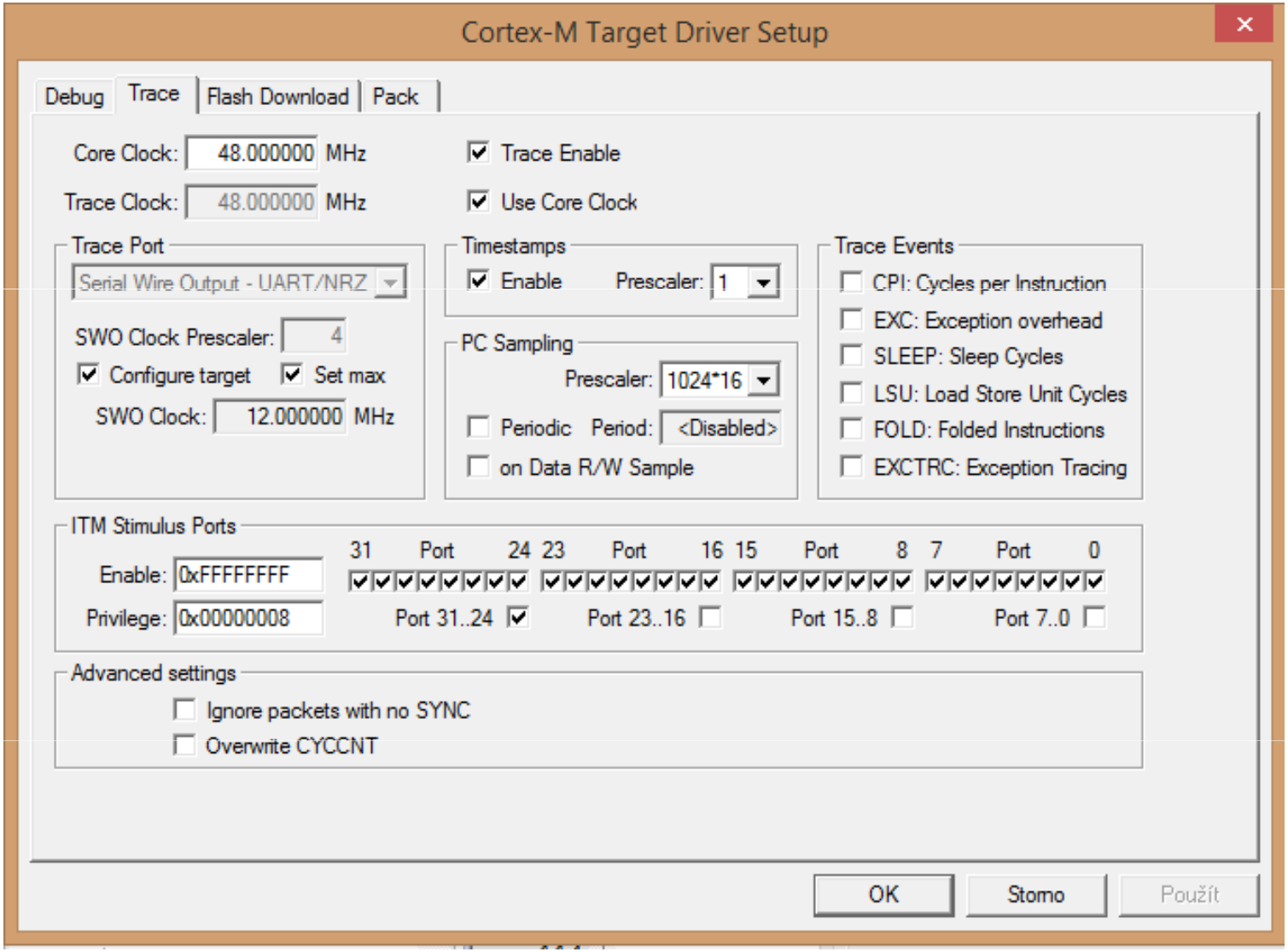
1. Otevřeme okno Manage Run-Time Environment (MRTE). 
2. Rozklikneme položku Compiler a potom I/O
3. Vybereme STDOUT a ITM. Tím je do projektu přidán soubor retarget_io.c.
4. Zkontrolujeme, zda jsou všechny bloky zelené.
5. Otevřeme okno *Option for Target – Debug – Settings – Trace*
6. Nastavíme správný hodinový kmitočet procesoru, Trace Enable a ITM Stimulus Ports podle následujícího obrázku (stačí jen ITM0).
7. Přidáme do našeho programu například

```
printf("counter = %d\n", counter);
```





ITM - INSTRUMENTATION TRACE MACROCELL



8. Přeložíme projekt, otevřeme okno **Debug *printf* Viewer** (*View – Serial Window - Debug printf Viewer*) a spustíme program.

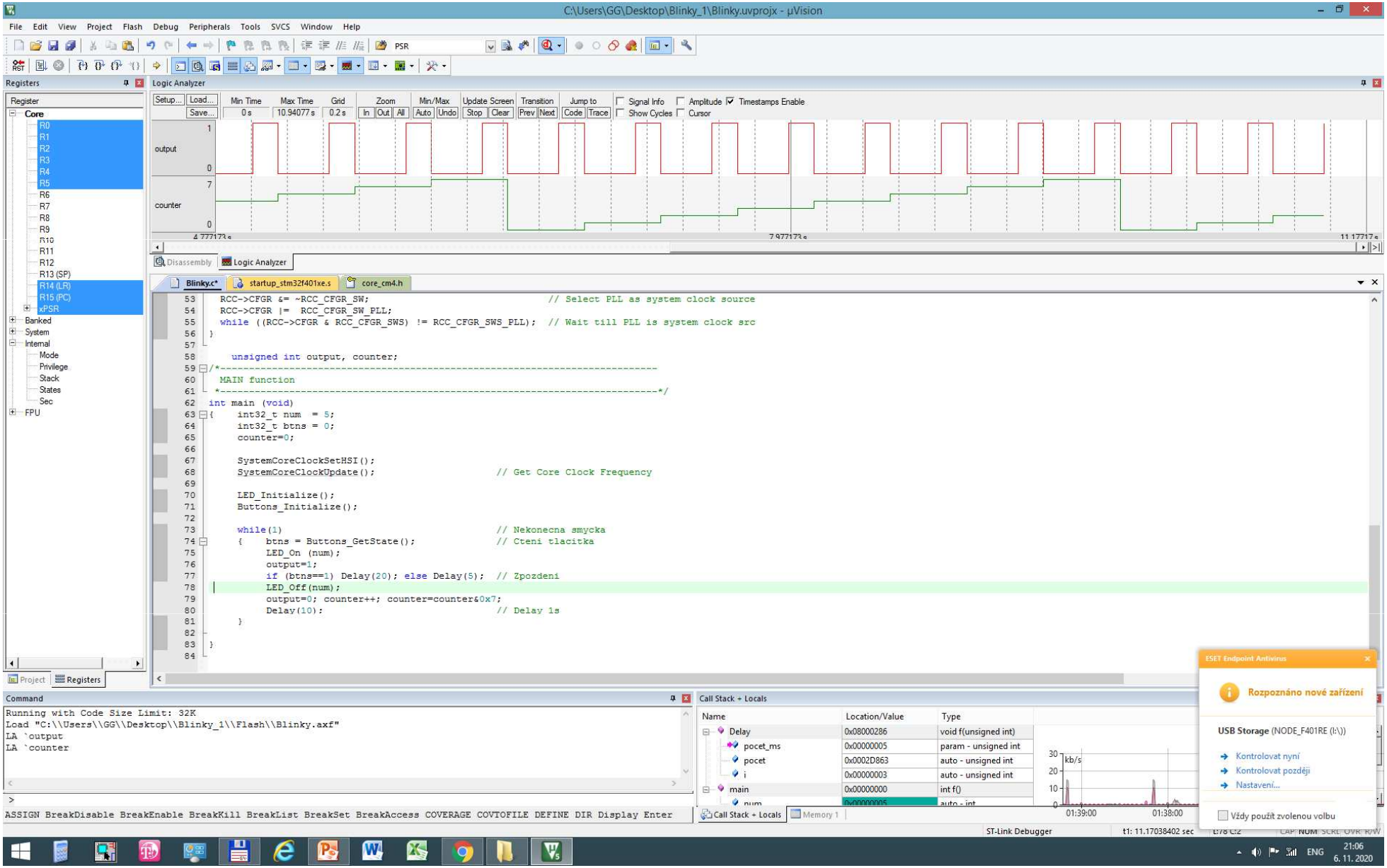
GRAFICKÉ ZOBRAZENÍ PROMĚNNÝCH V LOGICKÉM ANALYZÁTORU

Grafické zobrazení globálních proměnných nepřidává do programu žádný kód ani nezpomaluje prováděný program. Logický analyzátor používá Serial Wire Viewer a proto musí být okno Trace nakonfigurováno podle předcházejícího obrázku. V analyzátoru můžeme zobrazit až 4 proměnné. Tyto proměnné musí být globální, statické nebo raw adresy, například * ((unsigned*) 0x20000000).

NASTAVENÍ:

1. V okně *Debug* – *Settings* vybereme položku SW, která je nezbytná pro režim *SWV*.
2. V okně *Trace* nastavíme Trace Enable, nezvolíme Periodic a EXCTRC.
3. Spustíme testovaný program a otevřeme si okno *Logic Analyzer ( )*
4. Zapíšeme zobrazovanou proměnnou, určíme její typ, zobrazovaný rozsah a případné maskování bitů.

GRAFICKÉ ZOBRAZENÍ PROMĚNNÝCH V LOGICKÉM ANALYZÁTORU

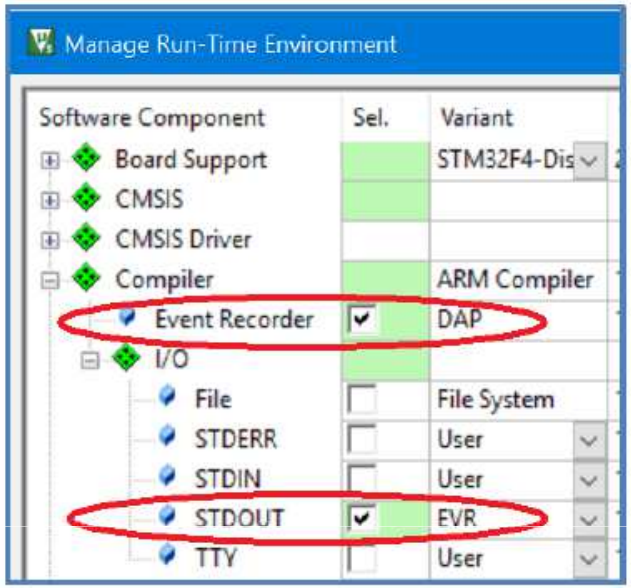



```
53    RCC->CFGR &= ~RCC_CFGR_SW;                          // Select PLL as system clock source
54    RCC->CFGR |=  RCC_CFGR_SW_PLL;
55    while ((RCC->CFGR & RCC_CFGR_SWS) != RCC_CFGR_SWS_PLL);  // Wait till PLL is system clock src
56  }
57
58      unsigned int output, counter;
59  /*----------------------------------------------------------------------------
60    MAIN function
61   *----------------------------------------------------------------------------*/
62  int main (void)
63  {    int32_t num  = 5;
64       int32_t btns = 0;
65       counter=0;
66
67       SystemCoreClockSetHSI();
68       SystemCoreClockUpdate();                    // Get Core Clock Frequency
69
70       LED_Initialize();
71       Buttons_Initialize();
72
73       while(1)                                    // Nekonecna smycka
74       {    btns = Buttons_GetState();             // Cteni tlacitka
75            LED_On (num);
76            output=1;
77            if (btns==1) Delay(20); else Delay(5);  // Zpozdeni
78            LED_Off(num);
79            output=0; counter++; counter=counter&0x7;
80            Delay(10);                             // Delay 1s
81       }
82
83  }
84
```

NAHRÁVÁNÍ UDÁLOSTÍ - EVENT RECORDER

Event Recorder je další funkcí, která může být přidána do zdrojového kódu, pro sledování událostí v běžícím programu. Keil RTX5 a Keil Middleware jsou již vybaveny záznamníkem událostí. Jedná se o stejnou technologii DAP, která se používá v oknech Watch, Memory a Peripheral. Můžete doplňovat Vaše zdroje o zobrazení událostí. Může realizovat funkci printf bez použití SWV.

NASTAVENÍ:



1. Otevřeme okno Manage Run-Time Environment (MRTE). 
2. Rozklikneme položku *Compiler* a potom *I/O* a zaškrtneme *EventRecorder* a v *STDOUT* a EVR. Tím budou do projektu do skupiny Compiler přidány soubory retarget_io.c a EventRecorder.c
3. Do hlavního programu přidáme řádky #include "stdio.h" a #include "EventRecorder.h".
4. Na začátku programu main() vložíme řádek EventRecorderInitialize (EventRecordAll, 1);
5. Proměnnou, kterou chceme sledovat, vyjádříme pomocí funkce printf jako v popisu ITM.
6. Soubor uložíme a přeložíme Rebuild the source files. 

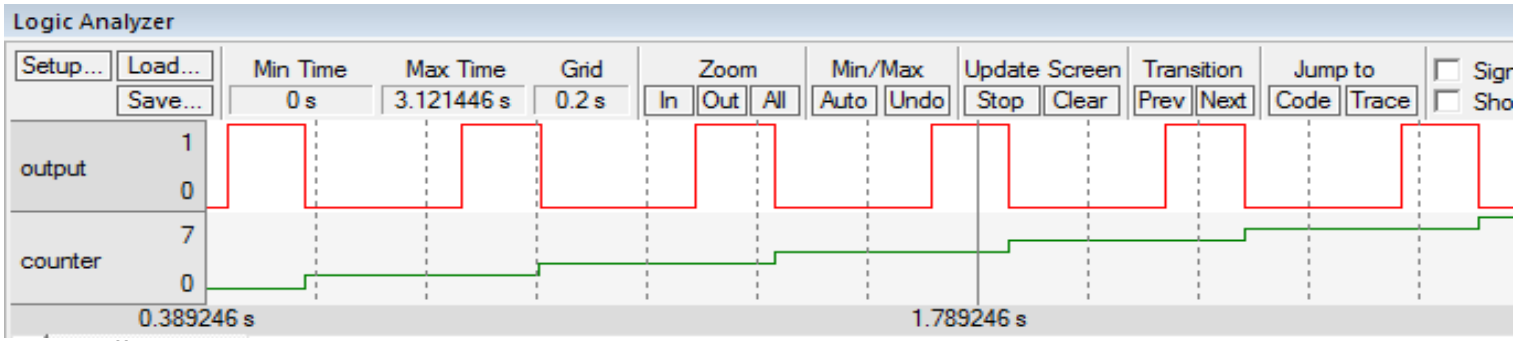
NAHRÁVÁNÍ UDÁLOSTÍ - EVENT RECORDER

7. Přejdeme do *Debug* módu, vybereme okno *View / Serial Windows* a aktivujeme *Debug (printf) Viewer*.
8. Otevřeme okno *View – Analysis windows* - Event Recorder viz.obrázek.
9. Zápis realizovaný funkcí ***printf("cout=%d\n", counter);*** je zobrazen vpravo na spodním obrázku.

Event Recorder

Enable Recorder: Mark: All Operations Stopped

Event	Time (sec)	Component	Event Property	Value
0	0.00064035	EvCtrl	EventRecorderInitialize	Restart Count = 1
1	0.00064552	EvCtrl	EventRecorderStart	
2	0.14261004	STDIO	stdout	0x63,0x6F,0x75,0x74,0x3D,0x31,0x0A,0x00
3	0.56824856	STDIO	stdout	0x63,0x6F,0x75,0x74,0x3D,0x32,0x0A,0x00
4	0.99388692	STDIO	stdout	0x63,0x6F,0x75,0x74,0x3D,0x33,0x0A,0x00
5	1.41952527	STDIO	stdout	0x63,0x6F,0x75,0x74,0x3D,0x34,0x0A,0x00
6	1.84516363	STDIO	stdout	0x63,0x6F,0x75,0x74,0x3D,0x35,0x0A,0x00
7	2.27080198	STDIO	stdout	0x63,0x6F,0x75,0x74,0x3D,0x36,0x0A,0x00
8	2.69644033	STDIO	stdout	0x63,0x6F,0x75,0x74,0x3D,0x37,0x0A,0x00
9	3.12207869	STDIO	stdout	0x63,0x6F,0x75,0x74,0x3D,0x38,0x0A,0x00



Debug (printf) Viewer

```
cout=1
cout=2
cout=3
cout=4
cout=5
cout=6
cout=7
cout=8
```

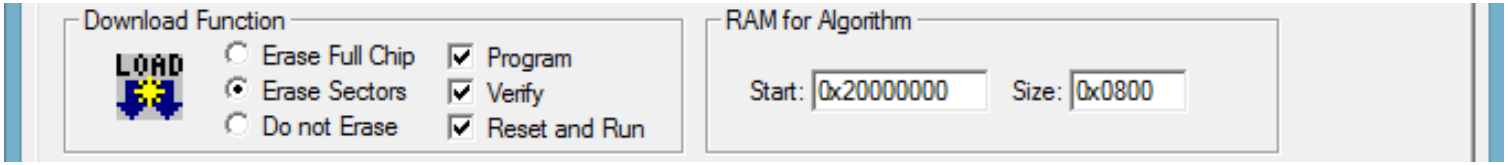
NAHRÁVÁNÍ UDÁLOSTÍ - EVENT RECORDER

Event Recorder nám poskytuje časovou informaci od DWT, kdy k dané události dochází. Protože uvedený záznam není zrovna pohodlný, ukážeme další možnosti mimo funkci *printf*. Pro Event Recorder využívající kanál DWT je potřeba rezervovat určitou oblast SRAM, která není inicializovaná a nevyužívá ji program i programování Flash. Pro nepřerušované ukládání by měla být SRAM pro Event Recorder umístěna v oblasti paměti, která není vymazána (inicializována) resetem systému a která se liší od oblasti používané pro programování Flash. Velikost paměti se vypočítá podle vzorce:

164 + 16 x Number_of_Records (defined by \c EVENT_RECORD_COUNT in EventRecorderConf.h)

V příkladu volíme 0x800 (2kB), které bohatě stačí pro více jak 64 záznamů.

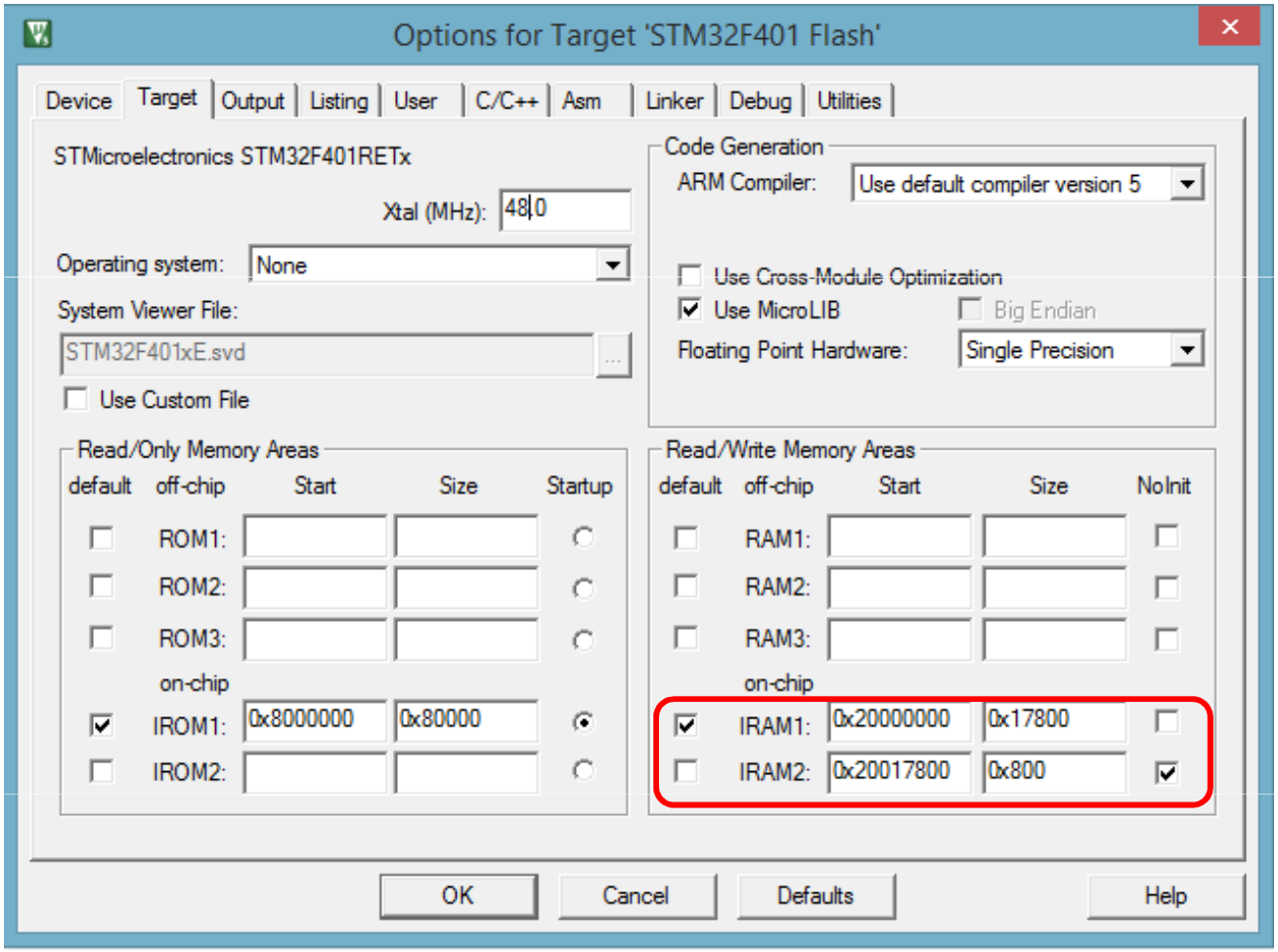
1. Nejprve zvolíme velikost paměti pro programování. Otevřeme okno *Options for Target – Debug – Settings – Flash Download* a v položce *RAM for Algorithm* nastavíme velikost RAM.



2. Následně rozdělíme paměť SRAM na dvě části IRAM1 a IRAM2. Volíme okno *Options for Target – Target,* v kterém vytvoříme oblast IRAM2 a příslušně zmenšíme oblast IRAM1.

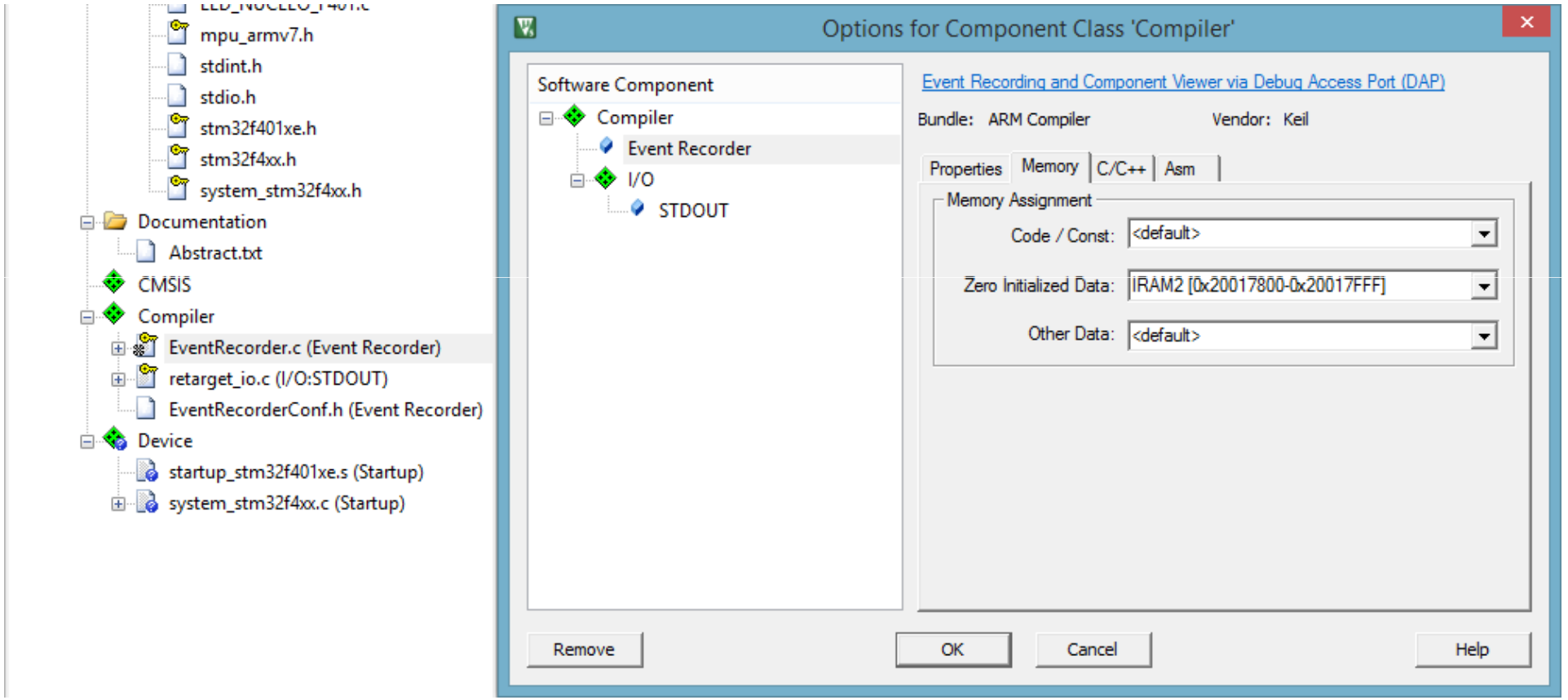
NAHRÁVÁNÍ UDÁLOSTÍ - EVENT RECORDER

Například tak, jak je ukázáno na obrázku.



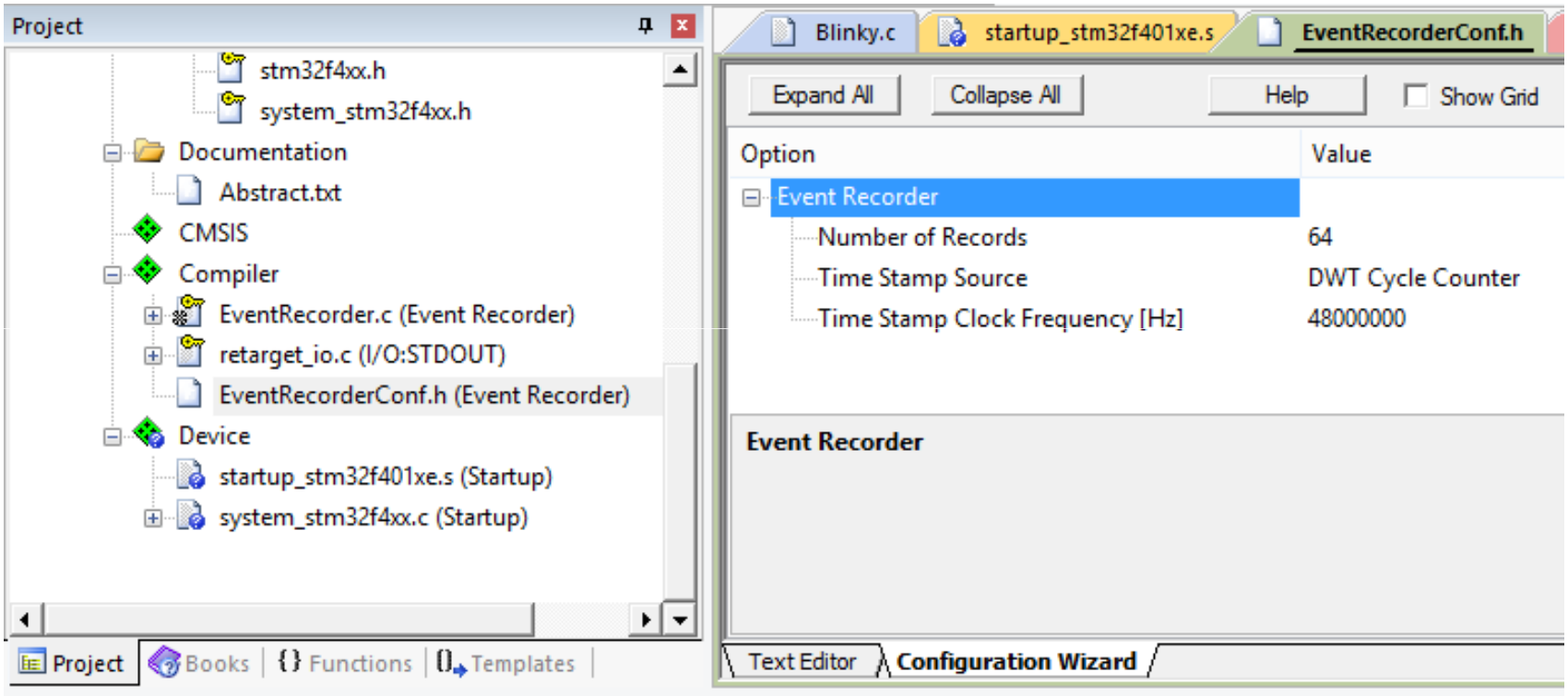
3. V okně *Options for Component Class 'Compiler'*, které otevřeme pravým tlačítkem na souboru EventRecorder.c v okně projektu, v položce *Memory* vybereme z nabídky velikost oblasti IRAM2 viz. obrázek.

NAHRÁVÁNÍ UDÁLOSTÍ - EVENT RECORDER



4. Nakonec nastavíme kmitočet hodinového signálu, abychom se vyhnuli problémům v okně System Analyzer (*View - Analysis Windows - System Analyzer)*. Okno, v kterém realizujeme nastavení, se otevře otevřením souboru *EventRecorderConf.h* a ve spodní liště klepneme na *Configuration Wizadr* viz. obrázek na další stránce. V okně můžeme vyjma hodinového kmitočtu nastavit počet záznamů a zdroj časových značek.

NAHRÁVÁNÍ UDÁLOSTÍ - EVENT RECORDER



Časovač EventRecorder je 32 bitový a proto u něj dochází v relativně krátké době k přetečení. Proto je vhodné v rozsahu 32 bitů realizovat značku, která usnadní identifikaci času i skutečnost, že aplikace stále běží.

NAHRÁVÁNÍ UDÁLOSTÍ - EVENT RECORDER

Chceme-li monitorovat dynamické chování vyvíjeného programu, vložíme do programového kódu jeden z následujících záznamů.

- uint32_t EventRecordData (uint32_t id, const void *data, uint32_t len);
- uint32_t EventRecord2 (uint32_t id, uint32_t val1, uint32_t val2);
- uint32_t EventRecord4 (uint32_t id, uint32_t val1, uint32_t val2, uint32_t val3, uint32_t val4);

Hodnota ***id*** nám identifikuje typ záznamu. V prvém případě můžeme do záznamu umístit textový řetězec např.

```
char string[10] = "MyTest";
 :
EventRecordData (3+EventLevelOp,string, sizeof(string));
```

Zbývající dva záznamy umožňují zobrazit dvě nebo čtyři 32 bitové hodnoty např.

```
EventRecord2(2+EventLevelOp,hodnota1,hodnota2);
EventRecord4(4+EventLevelOp,hod1,hod2,hod3,hod4);
```

Umístíme-li tyto záznamy do programu, můžeme získat následující záznam Event Recorderu, který je zobrazen na následující stránce.

NAHRÁVÁNÍ UDÁLOSTÍ - EVENT RECORDER

Blinky.c | startup_stm32f401xe.s | EventRecorderConf.h

```
80  {    char string[10] ="MyTest";   // some
81       int32_t num  = 5;
82       int32_t btns = 0;
83       EventRecorderInitialize (EventRecord
84       SystemCoreClockSetHSI();
85       SystemCoreClockUpdate();
86       counter=0;
87       LED_Initialize();
88       Buttons_Initialize();
89
90       while(1)
91       {    btns = Buttons_GetState();
92            LED_On (num);
93            output=1;
94            if (btns==1) Delay(20); else De
95
96           counter++;
97  //          printf("cout=%d\n", counter);
98            LED_Off(num);
99            EventRecord2(0x01, counter,3);
100           EventRecordData (3+EventLevelOp,string, sizeof(string));
101           output=0;
102           if (btns==1) Delay(20); else Delay(5);  // Zpozdeni
103      }
104
105 }
```

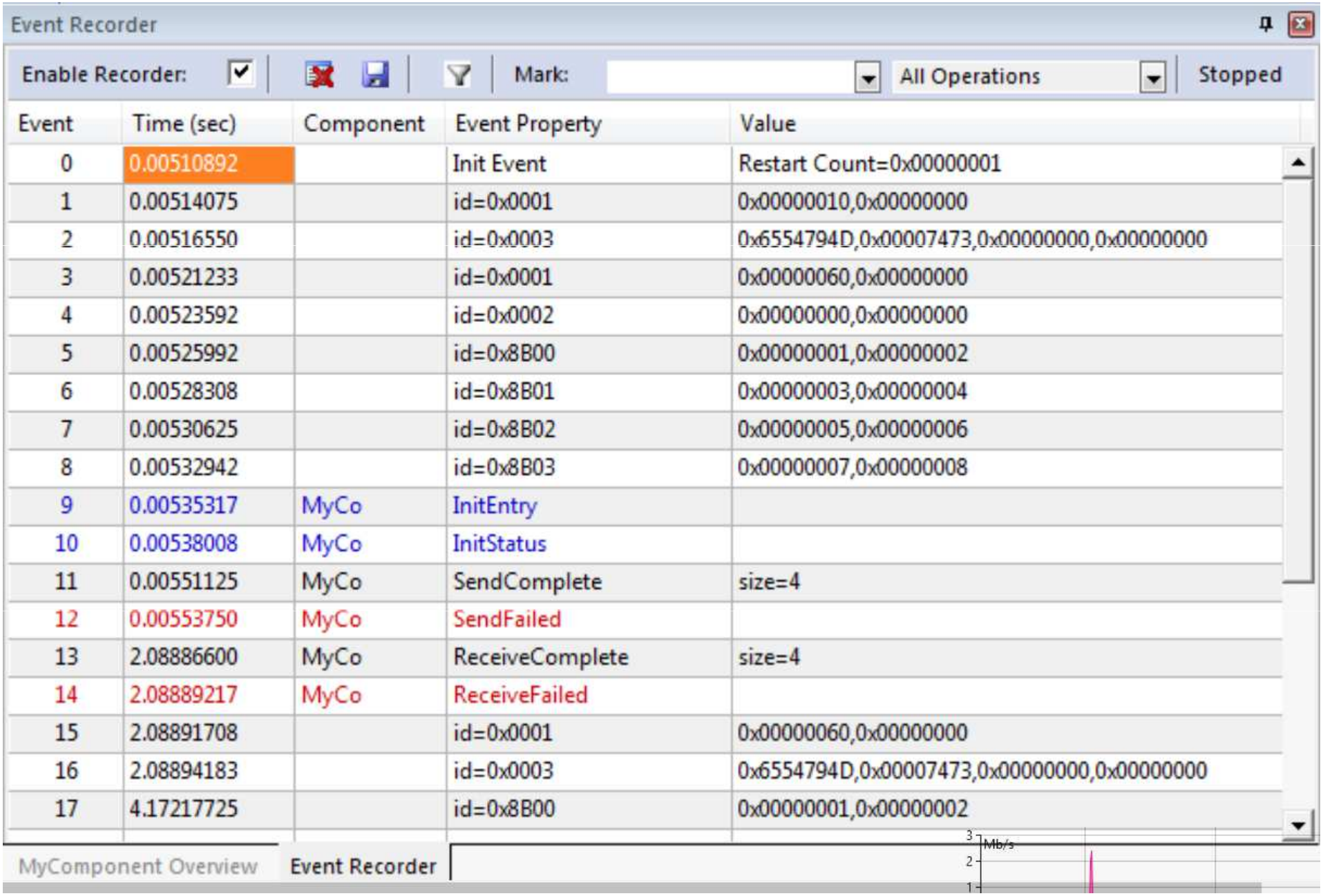
Event Recorder

Enable Recorder: ☑ | Mark: 0001 | Details

Event	Time (sec)	Component	Event Property	Value
0	0.00067154	EvCtrl	EventRecorderInitialize	Restart Count = 8
1	0.00067712	EvCtrl	EventRecorderStart	
2	0.14512185		id=0x0001	0x00000001,0x00000003
3	0.14512942		id=0x0003	0x6554794D,0x00007473,0x00000000,0x00000000
4	0.43388927		id=0x0001	0x00000002,0x00000003
5	0.43389662		id=0x0003	0x6554794D,0x00007473,0x00000000,0x00000000
6	0.72266325		id=0x0001	0x00000003,0x00000003
7	0.72267060		id=0x0003	0x6554794D,0x00007473,0x00000000,0x00000000

Ze získaného záznamu je zřejmé, že užitečnými funkcemi jsou Event-Record2 a EventRecord4, kterými můžeme sledovat proměnné v celém programu nebo jen v jeho určité části za použití funkcí *EventRecorderStart (void), EventRecorderStop (void), EventRecorderEnable (uint32_t recording, uint32_t comp_start, uint32_t comp_end)* a *EventRecorderDisable (uint32_t recording, uint32_t comp_start, uint32_t comp_end).* EventRecorderData je v řádcích označených id=0003, kde jsou jednotlivé ASCII kódy písmen řetězce v pořadí od LSB k MSB bytu a zbytek v druhém slově. Výstup nahrávaných událostí lze formátovat pomocí souboru *.SCVD, jak je ukázáno na obrázku.

FORMÁTOVÁNÍ NAHRÁVANÝCH UDÁLOSTÍ - EVENT RECORDER



Event	Time (sec)	Component	Event Property	Value
0	0.00510892		Init Event	Restart Count=0x00000001
1	0.00514075		id=0x0001	0x00000010,0x00000000
2	0.00516550		id=0x0003	0x6554794D,0x00007473,0x00000000,0x00000000
3	0.00521233		id=0x0001	0x00000060,0x00000000
4	0.00523592		id=0x0002	0x00000000,0x00000000
5	0.00525992		id=0x8B00	0x00000001,0x00000002
6	0.00528308		id=0x8B01	0x00000003,0x00000004
7	0.00530625		id=0x8B02	0x00000005,0x00000006
8	0.00532942		id=0x8B03	0x00000007,0x00000008
9	0.00535317	MyCo	InitEntry	
10	0.00538008	MyCo	InitStatus	
11	0.00551125	MyCo	SendComplete	size=4
12	0.00553750	MyCo	SendFailed	
13	2.08886600	MyCo	ReceiveComplete	size=4
14	2.08889217	MyCo	ReceiveFailed	
15	2.08891708		id=0x0001	0x00000060,0x00000000
16	2.08894183		id=0x0003	0x6554794D,0x00007473,0x00000000,0x00000000
17	4.17217725		id=0x8B00	0x00000001,0x00000002

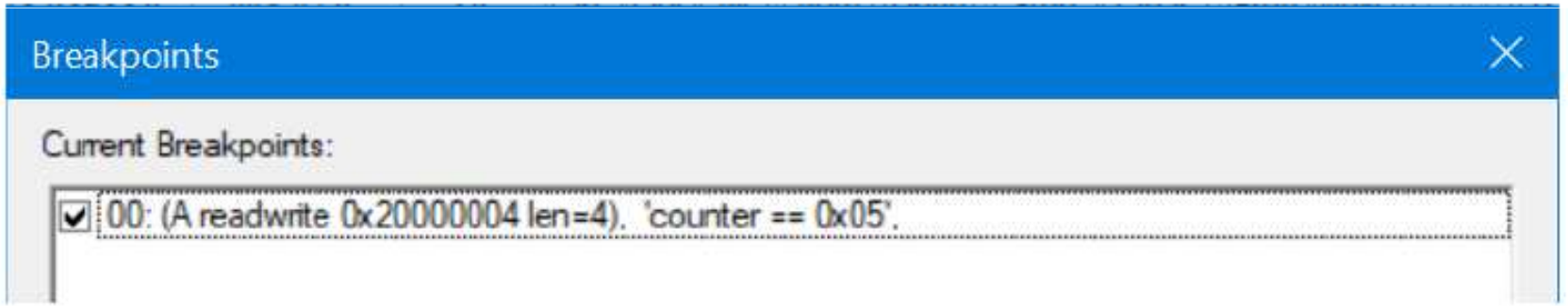
PROHLÍŽECÍ BODY – PODMÍNĚNÉ BREAKPOINTY

Procesory STM32 mají 6 hardwarových breakpointů, které lze nastavit za chodu bez zastavení CPU. STM32 má také dva sledovací body (Watchpoints), které lze považovat za podmíněné breakpointy. Logický analyzátor používá ve svých operacích stejné komparátory jako Watchpoints a musí být sdíleny. To znamená, že v µVision musíte mít v logickém analyzátoru volné dvě proměnné, abyste mohli používat Watchpoints. Watchpoints se také označují jako přístupové body (Access Breakpoints). Například použijeme globální proměnnou *counter* k ukázání funkce Watchpoint.

NASTAVENÍ:

1. Přejdeme po překladu do debugovacího režimu
2. V okně *Debug* vyberte breakpoints nebo stiskneme Ctrl-B.
3. V okně Breakpoints volíme Read a Write
4. V řádku Expression box zapíšeme např. **Counter==0x07.**
5. Zmáčkneme Define a Close.
6. Můžeme proměnnou counter sledovat v okně Watch 1.
7. Opustíme Debug a v prostředí otevřeme *Debug - Debug Settings* a volíme tabulku Trace. Označíme “on Data R/W sample”, EXTRC bude neoznačeno.
8. Přejdeme do *Debug* otevřeme *Trace Records window* a spustíme program.

PROHLÍŽECÍ BODY – PODMÍNĚNÉ BREAKPOINTY



Chování programu s podmíněným breakpointem je uložen v projektu Podmíněný Break.