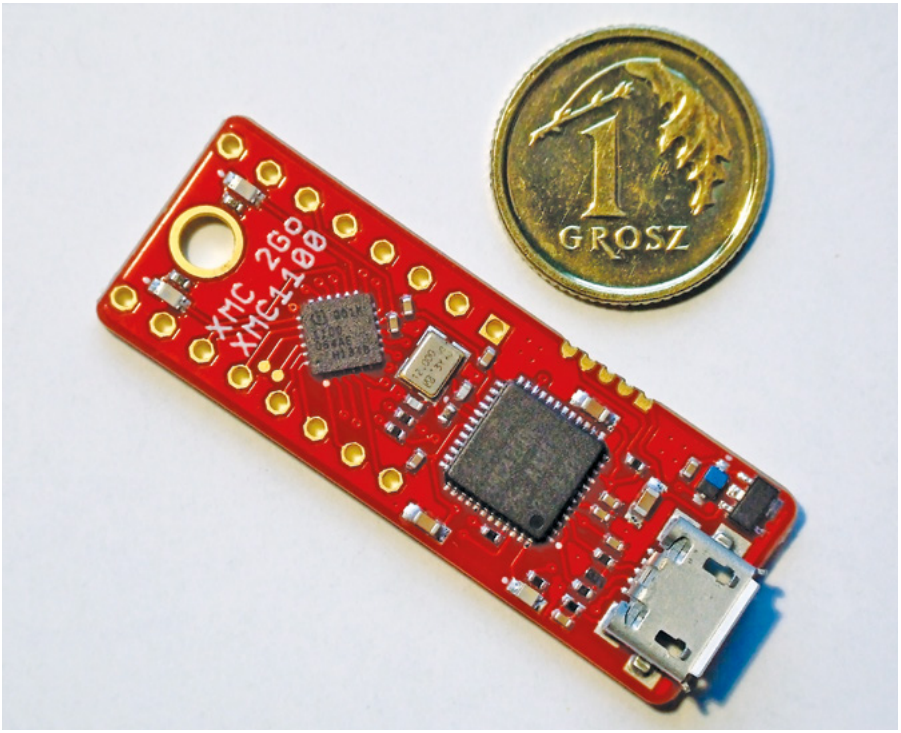
# Zestaw demonstracyjny XMC 2Go jako analizator stanów logicznych

*Zestawy demonstracyjne mikrokontrolerów po spełnieniu swej pierwotnej funkcji, jaką jest zapoznanie użytkownika z nowym układem, kończą zazwyczaj swój żywot na półce pokryte kurzem. Można zaryzykować stwierdzenie, ze taki los spotyka dany zestaw tym szybciej, im skromniej jest on wyposażony. Tymczasem nawet minimalistyczny zestaw ewaluacyjny mikrokontrolera może zostać przekształcony w narzędzie przydatne w pracowni elektronika i być dalej użyteczny. W artykule zaprezentowano sposób wykorzystania jednego z tego typu zestawów – modułu XMC 2Go w roli 8-wejściowego analizatora stanów logicznych.*



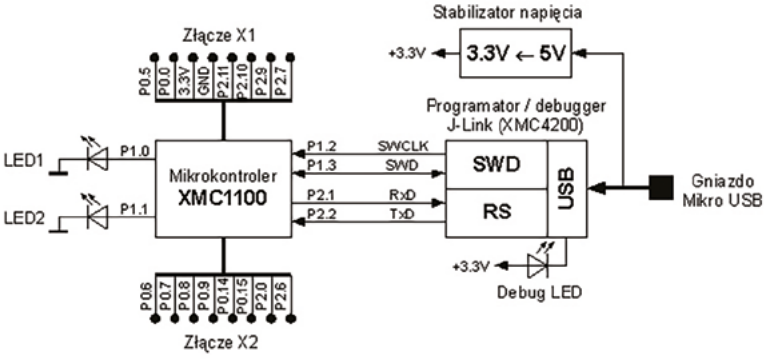

**Fotografia 1. Zestaw demonstracyjny XMC 2Go**

Pokazany na **fotografii 1** moduł XMC 2Go został opracowany przez firmę Infineon jako zestaw demonstracyjny mikrokontrolera typu XMC1100 oraz środowiska programistycznego o nazwie DAVE, przeznaczonego dla o mikrokontrolerów tej firmy. W momencie pojawienia się moduł był reklamowany jako prawdopodobnie najmniejszy na świecie zestaw ewaluacyjny mikrokontrolera. Rzeczywiście, jest mniejszy niż typowy pendrive, ponieważ moduł mierzy 38,5 mm długości i 14 mm szerokości.

Miniaturowe rozmiary zestawu XMC 2Go idą w parze z jego wyposażeniem, które jest bardziej niż skromne. Poza mikrokontrolerem XMC1100, zestaw jest wyposażony tylko w dwie diody LED, które mogą być sterowane przez program użytkownika. Dodatkowo, dla użytkownika jest dostępnych 14 linii I/O mikrokontrolera, które są wyprowadzone na pola lutownicze rozmieszczone wzdłuż krawędzi płytki drukowanej. Do programowania mikrokontrolera XMC1100 służy znajdujący się na tej samej płytce drukowanej programator/debugger zgodny z J-Link. Pełni on również rolę konwertera UART/USB dla układu transmisji szeregowej mikrokontrolera XMC1100. Schemat blokowy modułu XMC 2Go przedstawia **rysunek 2**.

Zastosowany w zestawie XMC 2Go układ XMC1100-Q24F0064 jest przedstawicielem najprostszej rodziny mikrokontrolerów z rdzeniem Cortex-M w ofercie firmy Infineon. Jest on wyposażony w rdzeń Cortex-M0, 8 kB pamięci ROM, 64 kB nieulotnej pamięci Flash i 16 kB pamięci SRAM. Maksymalna częstotliwość taktowania rdzenia to 32 MHz. Układy peryferyjne, w które jest wyposażony mikrokontroler XMC1100 to:

- trzy równoległe porty GPIO: P0, P1 i P2,
- 4-kanałowy układ typu Capture-Compare CCU4,
- okienkowy watchdog timer WDT,




**Rysunek 2. Schemat blokowy zestawu demonstracyjnego XMC 2Go**

- zegar czasu rzeczywistego RTC,
- 2-kanałowy uniwersalny interfejs szeregowy USIC0,
- układ zarządzania zdarzeniami i przerwaniami ERU0,
- 6-kanałowy, 12-bitowy przetwornik analogowo-cyfrowy VADC,
- generator pseudolosowy PRNG,
- czujnik temperatury TSE.

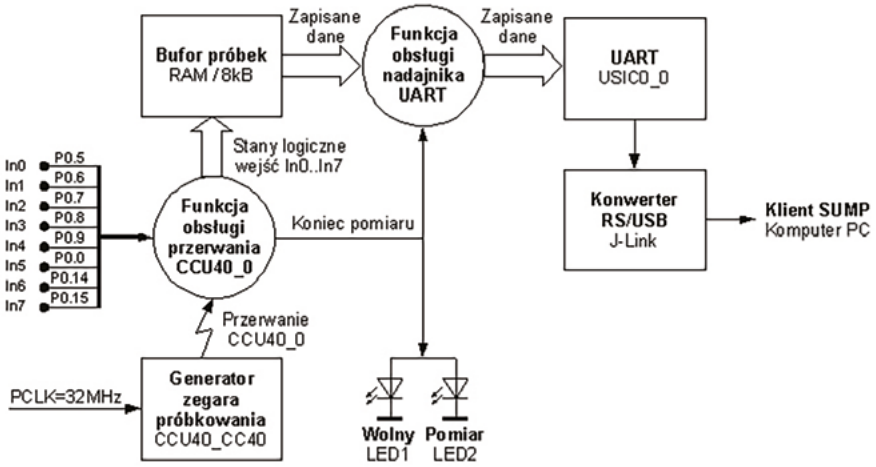
## Budowa analizatora

Typowy analizator stanów logicznych składa się z dwóch bloków: modułu próbkującego i zapamiętującego poziomy logiczne sygnałów doprowadzonych do wejść pomiarowych oraz modułu wyświetlającego zarejestrowane przebiegi. Pierwszy moduł zazwyczaj jest wykonany w postaci układu elektronicznego, natomiast prezentację zarejestrowanych przebiegów zwykle realizuje program uruchomiony na komputerze PC. Program ten stanowi też interfejs użytkownika umożliwiający obsługę przyrządu.

Opisywany w artykule analizator stanów logicznych ma identyczną budowę. Rolę modułu próbkującego i gromadzącego w swej pamięci stany logiczne mierzonych sygnałów pełni w nim zestaw ewaluacyjny XMC 2Go, natomiast prezentowanie wyników i programowanie parametrów pracy odbywa się za pomocą komputera PC.

Zrealizowany na bazie modułu XMC 2Go układ analizatora stanów logicznych ma następujące parametry:

- Liczba linii wejściowych: 8.
- Zakres napięcia wejściowego: 0…3,3 V.
- Częstotliwość próbkowania wejść (wybierana za pomocą menu): 10 Hz, 20 Hz, 50 Hz, 100 Hz, 200 Hz, 500 Hz, 1 kHz, 2 kHz, 5 kHz, 10 kHz, 20 kHz, 50 kHz, 100 kHz, 200 kHz i 500 kHz.




**Rysunek 3. Schemat działania analizatora stanów logicznych na bazie modułu XMC 2Go**

- Zegar próbkowania wejść: wewnętrzny.
- Wielkość bufora próbek (wybierana): 64 B, 128 B, 256 B, 512 B, 1024 B, 2048 B, 4096 B, 8192 B.
- Wyzwalanie pomiaru dowolną kombinacją poziomów logicznych wejść.
- Obserwacja sygnału przed momentem wyzwolenia w zakresie od 0 do 100% rozmiaru bufora próbek.
- Komunikacja z komputerem PC przez łącze USB (wirtualny port szeregowy 115200 b/s, 8N1).

Mikrokontroler XMC1100 nie należy do najszybszych obecnie dostępnych na rynku, wyposażonych w rdzeń Cortex-M. Nie on jest też wyposażony w żadne sprzętowe obwody wspomagające przesyłanie danych wewnątrz układu, np. kontroler DMA. W związku z tym, wszystkie funkcje analizatora związane z próbkowaniem stanów wejść oraz ich zapisem w pamięci muszą być realizowane przez mikrokontroler w sposób programowy. Schemat funkcjonalny układu zbudowanego według tej koncepcji przedstawia **rysunek 3**.

Mierzone sygnały są doprowadzone do wejść In0…In7 analizatora, których rolę pełni 8 linii portu P0. Są one skonfigurowane jako wejścia cyfrowe z aktywnym układem polaryzacji każdego wejścia do masy, dzięki czemu nieużywane w danym pomiarze linie portu P0 mają zawsze ustalony potencjał (poziom niski). Pomiar wejść jest aktywowany na polecenie z komputera PC. W momencie rozpoczęciu pomiaru startuje generator zegara próbkowania, który zaczyna generować przerwania z częstotliwością równą wybranej częstotliwości pomiaru wejść In0…In7. W roli generatora pracuje kanał CC40 układu Capture-Compare CCU4. Kanał ten jest skonfigurowany do pracy jako licznik zliczający w górę, który automatycznie restartuje się po osiągnięciu zaprogramowanej wartości. Licznik ten jest taktowany zegarem PCLK

**Listing 1. Funkcja konfiguracji układu CCU4_0**

```
void SamplingTimer_Init(void)
{
// Configuration of CCU40 system clock (clear CCU40 clock gating)
  SCU_GENERAL->PASSWD = 0x000000C0UL;  // disable bit protection
  SCU_CLOCK->CGATCLR0 |= SCU_CLOCK_CGATCLR0_CCU40_Msk; // clear gating of CCU40 clock
  SCU_GENERAL->PASSWD = 0x000000C3UL;  // enable bit protection
// Prescaler enable
  CCU40->GIDLC |= CCU4_GIDLC_SPRB_Msk;
// Configuration of global control register
  CCU40->GCTRL &= ~(CCU4_GCTRL_PCIS_Msk);  // prescaler input clock is module clock
// Configuration of CC40 timer slice
  CCU40_CC40->TC = 0x00000000;  // disabled multi channel mode,floating
                                // prescaler, dither and single shot mode;
                                // compare mode with shadow transfer on clr
                                // timer counting mode: edge aligned

// Configuration of timer prescaler (for test purposes, prescaler set prior to inputs capture)
//CCU40_CC40->PSC = 0x08;     // normal mode, prescaler value 256
// Configuration of timer period (for test purposes, period set prior to inputs capture)
//CCU40_CC40->TIMER = 0x0000; // timer clear
//CCU40_CC40->PRS = 32000-1;  // test period 100ms
//CCU40->GCSS |= CCU4_GCSS_S0SE_Msk;  // load shadow registers

// Interrupt configuration
  CCU40_CC40->SRS &= ~(CCU4_CC4_SRS_POSR_Msk);  // interrupt line: CC40SR0
  CCU40_CC40->INTE |= CCU4_CC4_INTE_PME_Msk;  // interrupt source: period match while counting up event
  NVIC_SetPriority(CCU40_0_IRQn, 0x01);  // NVIC: high interrupt priority
  NVIC_EnableIRQ(CCU40_0_IRQn);  // NVIC: interrupt enable

// Enable of CC40 timer slice
  CCU40->GIDLC |= CCU4_GIDLC_CS0I_Msk;
// Start CC40 timer (for test purposes, timer enabled prior to inputs capture)
//CCU40_CC40->TCSET = CCU4_CC4_TCSET_TRBS_Msk;
}
```

**Listing 2. Funkcja obsługi przerwania CCU4_0**

```
void CCU40_0_IRQHandler(void) __attribute__((section(".ISRAMCode")));

void CCU40_0_IRQHandler(void)
{
  uint32_t in_sample;  // input sample
  // Sample inputs
  in_sample = PORT0->IN;  // read input port P0
  in_sample ^= (in_sample >> 13);  // b2b1b0 = /P0.15/P0.14/P0.0
  in_sample ^= (in_sample << 10);  // b12b11b10 = P0.15 P0.14 P0.0
  in_sample >>= 5;  // b7..b0 = P0.15P0.14P0.0P0.9P0.8P0.7P0.6P0.5

  // Store inputs sample in buffer
  sample_buf.data[sample_buf.wr_idx] = in_sample;   // store input sample in buffer
  sample_buf.wr_idx ++;  // index the next free position in buffer
  sample_buf.wr_idx &= (SAMPLE_BUF_SIZE - 1);  // if buf. overflow, start from the beginning

  // Service trigger
  if (sample_buf.trig_mask != 0){  // if sampling not triggered
    // Check trigger condition
    if ((in_sample & sample_buf.trig_mask) == sample_buf.trig_value){
      sample_buf.trig_mask = 0;  // trigger released
    }
  }
  else {  // if sampling triggered
    sample_buf.trig_cntr --;  // count samples after trigger
    if (sample_buf.trig_cntr == 0){  // if specified number of samples has been reached
      USIC0_CH0->RBCTR &= ~USIC_CH_RBCTR_SRBIEN_Msk;// disable UART RxBuf irq
      CCU40_CC40->TCCLR = CCU4_CC4_TCCLR_TRBC_Msk;  // stop sampling timer
      NVIC_ClearPendingIRQ(CCU40_0_IRQn);  // clear pending timer irq
      LED2_OFF;  // switch off LED2
      SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;  // start SysTick Timer
      sample_tx.state = ON;  // activate samples transfer
    }
  }
}
```

o częstotliwości 32 MHz podzielonym w wewnętrznym preskalerze kanału CC40. Okres zliczania licznika jest więc równy:

*T = 1/f_PCLK * CC40_PSC * (CC40_PRS + 1)*

gdzie :

f_PCLK – częstotliwość zegara układu CCU4 (f_PCLK = 32 MHz).

CC40_PSC – stopień podziału wewnętrznego prescalera kanału CC40.

CC40_PRS – górny próg zliczania licznika w kanale CC40.

Stopień podziału prescalera *CC40_PSC* oraz okres zliczania *CC40_PRS* są każdorazowo programowane, aby uzyskać wybraną częstotliwość pomiaru wejść In0…In7. Osiągnięcie górnej wartości przez licznik skutkuje wygenerowaniem przez kanał CC40 układu CCU4 przerwania. Ponieważ w mikrokontrolerze XMC1100 źródła przerwań z układów peryferyjnych w ramach danego peryferium nie są na stałe przypisane do określonego wektora przerwań tylko są programowane, w prezentowanym układzie do sygnalizacji przeładowania licznika użyto pierwszego wolnego przerwania układu CCU4, tj. wektora CCU40_0. Szczegóły konfiguracji układu CCU4 oraz jego przerwań przedstawia **listing 1**.

Właściwy pomiar sygnałów doprowadzonych do wejść analizatora jest realizowany przez funkcję obsługi przerwania CCU4_0 (**listing 2**). Odczytuje ona stany logiczne wejść In0…In7, kompletuje pojedynczy bajt i zapisuje w buforze próbek, który jest zorganizowany jako bufor kołowy o pojemności 8 kB. W trakcie pomiaru zawsze wykorzystywana jest cała pojemność tego bufora, niezależnie od ustawionego danym pomiarze rozmiaru N bufora próbek, co upraszcza kod funkcji i przyśpiesza jej wykonywanie. Dopiero przy transmisji zgromadzonych danych do komputera PC z bufora jest pobierane i wysyłane tylko N ostatnich próbek. Po zapisie danych do bufora kontrolowany jest warunek wyzwolenia pomiaru oraz zliczana jest liczba próbek stanów logicznych wejść In0…In7 zgromadzonych od momentu wyzwolenia pomiaru. Po osiągnięciu zadanej liczby próbek N, ich dalsza akwizycja jest zatrzymywana i następuje wywołanie funkcji transmisji danych do komputera PC.

Szybkość obsługi przerwań z kanału CC40 układu CCU4 przez mikrokontroler decyduje o maksymalnej częstotliwości, z jaką analizator jest w stanie próbkować poziomy na swoich wejściach. Przy częstotliwości próbkowania analizatora równej 500 kHz wywołanie funkcji obsługi przerwania oraz jej wykonanie musi trwać nie dłużej niż 64 takty zegara MCLK (przy częstotliwości MCLK równej 32 MHz). Jest to niewiele, jeżeli uwzględnimy, że w układzie XMC1100 samo opóźnienie pomiędzy momentem zarejestrowania przez układ zgłoszenia przerwania a pobraniem pierwszej instrukcji funkcji obsługi tego przerwania jest równe 21 taktom zegara MCLK, więc na wykonanie kodu funkcji obsługi przerwania pozostają 43 takty zegara MCLK. W tej liczbie zawierają się również opóźnienia związane z pobieraniem kodu programu z pamięci Flash. Co prawda, karta katalogowa mikrokontrolera XMC1100 nie podaje dokładnych wartości tych opóźnień, ale na podstawie informacji dostępnych na forum poświęconym mikrokontrolerom XMC wiadomo, że opóźnienie związane z odczytem pamięci Flash w układzie XMC1100 nie jest deterministyczne, ponieważ zależy ono od rodzaju dostępu do pamięci Flash, temperatury, a nawet egzemplarza układu. Przy sygnale zegarowym MCLK większym lub równym od 16 MHz należy liczyć się z tym, że odczyt pamięci Flash będzie obarczony od 1 do 3 cyklami oczekiwania [2.]. Aby uniknąć powyższych opóźnień, funkcja obsługi przerwania CCU40_0 jest wykonywana z pamięci SRAM, która jest wystarczająco szybka i nie wprowadza żadnych cykli oczekiwania. W tym celu funkcja ta została umieszczona w wydzielonej sekcji programu o nazwie *ISRAMCode*. W kompilatorze gcc operacja taka jest wymuszana przez dodanie przy deklaracji funkcji polecenia *__attribute__((section(„.ISRAMCode")))* (**listing 2**). Odpowiednio zmodyfikowany skrypt linkera *XMC_2Go.ld* zapewnia, że sekcja *ISRAMCode* jest linkowana do pamięci SRAM. Automatyczne kopiowanie zawartości tej sekcji z pamięci Flash do pamięci SRAM wykonuje z kolei odpowiednio zmodyfikowany kod startowy zawarty w pliku *startup_XMC1100.S*.

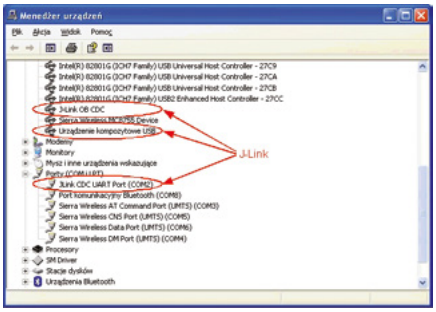
Zagadnienia modyfikacji kodu startowego i skryptu linkera są dość złożone i wykraczają poza zakres tego artykułu. Osoby zainteresowane tym problemem powinny zapoznać się z dokumentem firmy Infineon pt. *„XMC1000 Microcontroller Series for Industrial Applications. C - Start and Device Initialization. Device Guide"* [3.]. Pomocna może też być analiza dołączonych do artykułu kodów źródłowych. W każdym bądź razie z punktu widzenia programisty piszącego kod funkcji obsługi przerwania CCU40_0, poza wymaganą deklaracją *__attribute__(...)*, nie jest konieczne wgłębianie się w zawiłości umieszczania kodu funkcji w pamięci SRAM. Cały ten proces jest dla niego niewidoczny. Programista musi jednak zadbać, aby kod funkcji obsługi przerwania wykonywał się w ciągu maksymalnie wspomnianych wcześniej 43 taktów zegara MCLK. W tym celu kod funkcji *CCU40_0_IRQHandler* został

napisany pod kątem maksymalnej szybkości wykonania. Z tego właśnie powodu kompletacja stanów logicznych poszczególnych wejść jest realizowana przy pomocy operacji XOR, zamiast typowo spotykanej kombinacji funkcji AND i OR. Z tego samego powodu poszczególne zmienne używane przez funkcję *CCU40_0_IRQHandler* zostały zdefiniowane jako pola jednej globalnej struktury *sample_buf*, a za znacznik wyzwolenia pomiaru służy nie dedykowana zmienna, lecz wyzerowana maska *sample_buf.trig_mask* sekwencji wyzwalającej pomiar. Fakty te należy mieć na uwadze przy ewentualnym dokonywaniu modyfikacji kodu, ponieważ nawet prosta zamiana kolejności dwóch kolejnych wierszy kodu funkcji *CCU40_0_IRQHandler* może spowodować, że wzrośnie całkowity czas jej wykonania i nie uda się osiągnąć zakładanej maksymalnej częstotliwości pracy układu analizatora.

Jak wspomniano, wyświetlanie zarejestrowanych poziomów logicznych wejść In0…In7 oraz sterowanie analizatorem są realizowane z poziomu programu uruchomionego na komputerze PC. Komunikację pomiędzy modułem XMG 2Go a komputerem PC obsługuje układ UART. W jego roli wykorzystano kanał 0 uniwersalnego interfejsu szeregowego USIC0, który został skonfigurowany do pracy w trybie UART o prędkości transmisji 115200 bit/s i formacie znaku 8N1. Wbudowany w mikrokontroler XMC1100 interfejs USIC0 jest bardzo elastyczny i może on pracować w trybach UART, SPI, I²C, I²S lub LIN, dlatego jego konfigurowanie nie ogranicza się tylko do zaprogramowania generatora prędkości transmisji i wyboru formatu znaku (jak ma to zazwyczaj miejsce przy programowaniu typowego UARTa), lecz obejmuje także takie parametry jak: protokół transmisji, długość i liczba

**Listing 3. Funkcja konfiguracji układu USIC0 jako UART**

```
void UART_Init(void)
{
  // Configuration of USIC0 system clock (clear USIC0 clock gating)
  SCU_GENERAL->PASSWD = 0x000000C0UL;  // disable bit protection
  SCU_CLOCK->CGATCLR0 |= SCU_CLOCK_CGATCLR0_USIC0_Msk;  // clear gating of USIC0 clock
  SCU_GENERAL->PASSWD = 0x000000C3UL;  // enable bit protection
  // - enable USIC0 kernel clock and USIC0 functionality
  USIC0_CH0->KSCFG |= USIC_CH_KSCFG_MODEN_Msk | USIC_CH_KSCFG_BPMODEN_Msk;

  // Configuration of the baud rate generator
  // UART transmission clock frequency equals to:
  //  fASC = (fPB * STEP/1024)/((PPPEN+1)*(CTQSEL)) * 1/(PDIV+1) * 1/(DCTQ+1) * 1/(PCTQ+1)
  // - fractional divider configuration
  USIC0_CH0->FDR &= ~(USIC_CH_FDR_DM_Msk | USIC_CH_FDR_STEP_Msk);
  USIC0_CH0->FDR |= (0x02UL << USIC_CH_FDR_DM_Pos) |        // fractional divider mode
                    (UART_FDR_STEP << USIC_CH_FDR_STEP_Pos); // set division step
  // - baud rate generator configuration
  USIC0_CH0->BRG &= ~(USIC_CH_BRG_CLKSEL_Msk |  // set fractional divider as clock
                      USIC_CH_BRG_PPPEN_Msk |  // 1:2 fPPP divider disabled
                      USIC_CH_BRG_PDIV_Msk | USIC_CH_BRG_DCTQ_Msk | USIC_CH_BRG_PCTQ_Msk);
  USIC0_CH0->BRG |= (UART_BRG_PDIV << USIC_CH_BRG_PDIV_Pos)| // set divider to generate fPDIV
                    (UART_BRG_DCTQ << USIC_CH_BRG_DCTQ_Pos)| // set denominator for quanta cntr.
                    (UART_BRG_PCTQ << USIC_CH_BRG_PCTQ_Pos); // set  pre-divider for quanta cntr.

  // Configuration of USIC shift control
  USIC0_CH0->SCTR &= ~(USIC_CH_SCTR_WLE_Msk | USIC_CH_SCTR_FLE_Msk | USIC_CH_SCTR_TRM_Msk);
  USIC0_CH0->SCTR |= (0x07UL << USIC_CH_SCTR_WLE_Pos) |  // set word length to 8 bits
                     (0x07UL << USIC_CH_SCTR_FLE_Pos) |  // set frame length to 8 bits
                     (0x01UL << USIC_CH_SCTR_TRM_Pos) |      // control signal active when high
                     USIC_CH_SCTR_PDL_Msk;  // passive data level high

  // Configuration of USIC Transmit Control/Status Register
  USIC0_CH0->TCSR &= ~(USIC_CH_TCSR_TDEN_Msk);
  USIC0_CH0->TCSR |= (0x01UL << USIC_CH_TCSR_TDEN_Pos) |  // transmission starts at TDV=1
                     USIC_CH_TCSR_TDSSM_Msk;  // word-by-word data transmission

  // Configuration of protocol
  USIC0_CH0->PCR &= ~(USIC_CH_PCR_ASCMode_PL_Msk |  // pulse length equal to bit length
                      USIC_CH_PCR_ASCMode_SP_Msk |
                      USIC_CH_PCR_ASCMode_STPB_Msk);  // stop bit = 1
  USIC0_CH0->PCR |= (8UL << USIC_CH_PCR_ASCMode_SP_Pos) |  // sample point = 8
                    USIC_CH_PCR_ASCMode_SMD_Msk;  // three samples per bit

  // Configuration of transmit buffer
  USIC0_CH0->TBCTR &= ~(USIC_CH_TBCTR_SIZE_Msk |
                        USIC_CH_TBCTR_DPTR_Msk);
  USIC0_CH0->TBCTR |= ((0x05UL << USIC_CH_TBCTR_SIZE_Pos)|  // buffer size = 32 entries
                       (0x00 << USIC_CH_TBCTR_DPTR_Pos));  // data pointer = 0

  // Configuration of receive buffer
  USIC0_CH0->RBCTR &= ~(USIC_CH_RBCTR_SIZE_Msk | USIC_CH_RBCTR_RCIM_Msk |
                        USIC_CH_RBCTR_SRBINP_Msk |  // standard node pointer = out.SR0
                        USIC_CH_RBCTR_DPTR_Msk);
  USIC0_CH0->RBCTR |= (USIC_CH_RBCTR_RNM_Msk |  // RCI mode of receiver notification
                       (0x05UL << USIC_CH_RBCTR_SIZE_Pos) |  // buffer size = 32 entries
                       (0x02UL << USIC_CH_RBCTR_RCIM_Pos) |  // 0&WLEN receiver control mode
                       (32UL << USIC_CH_RBCTR_DPTR_Pos));  // data pointer = 32

  // Configuration of channel control register
  USIC0_CH0->CCR &= ~(USIC_CH_CCR_PM_Msk |  // parity generation disabled
                      USIC_CH_CCR_MODE_Msk);
  USIC0_CH0->CCR |= 0x02UL << USIC_CH_CCR_MODE_Pos;  // ASC(SCI,UART) operating mode

  // Configuration of TxD and RxD pins (P2.1 as TxD, P2.2 as RxD)
  P2_1_set_mode(OUTPUT_PP_AF6);  // set P2.1 as alt.function output
  P2_1_enable_digital();  // switch analog pin to digital

  P2_2_set_mode(INPUT);  // set P2.2 as input
  USIC0_CH0->DX3CR &= ~(USIC_CH_DX3CR_DSEL_Msk);  // select P2.2 as input for DX3->DX0
  USIC0_CH0->DX0CR &= ~(USIC_CH_DX0CR_DSEL_Msk);  // \ route USIC DX3 input to
  USIC0_CH0->DX0CR |= 6UL << USIC_CH_DX0CR_DSEL_Pos;  // /   USIC DX0 (DSEL=DX0G)
  P2_2_enable_digital();  // switch analog pin to digital

  // Interrupt configuration (int.line: SR0, int.source: receive buffer event on OUTR updating;
  // int. enabled when in need)
  NVIC_SetPriority(USIC0_0_IRQn, 0x00);  // NVIC: the highest int. priority
  NVIC_EnableIRQ(USIC0_0_IRQn);  // NVIC: interrupt enabled
}
```




**Rysunek 4. Identyfikacja modułu XMC 2Go w systemie Windows XP**

taktów zegara przypadających na jeden bit transmitowanych danych, czy też moment próbkowania sygnału RxD. Szczegóły konfiguracji układu USIC0 przedstawia **listing 3**. Ze względu na szczupłość dostępnego miejsca, komentarze w ramce zostały skrócone w stosunku do kodu źródłowego. Pełną wersję opisu kodu konfiguracji układu USIC0 zawierają pliki dołączone do artykułu.

W zestawie XMC 2Go kanał 0 układu USIC0 jest podłączony do programatora / debuggera J-Link, który realizuje funkcję konwertera interfejsu RS na USB. Dzięki temu analizator stanów logicznych komunikuje się z komputerem PC przez łącze USB. Po stronie komputera PC odwrotną konwersję zapewniają sterowniki programatora / debuggera J-Link. Podłączony do portu USB moduł XMC 2Go jest widoczny w systemie Windows jako urządzenie kompozytowe USB składające się z dwóch urządzeń: *J-Link OB CDC* oraz *Jlink CDC UART Port* (**rysunek 4**). Analizator stanów logicznych XMC 2Go LA wykorzystuje drugie z nich, tj. *Jlink CDC UART Port*.

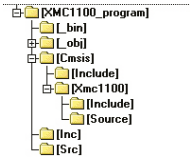
Pomimo przedsięwzięcia przedstawionych przy opisie funkcji *CCU40_0_IRQHandler* środków zaradczych, przy częstotliwości próbkowania analizatora równej 500kHz mikrokontroler XMC1100 praktycznie cały swój czas spędza w funkcji obsługi przerwania *CCU40_0*. Mogłoby to powodować niedogodności z obsługą analizatora z poziomu programu sterującego w przypadku, gdy ustawiony warunek wyzwolenia pomiaru nigdy nie występuje. W takiej sytuacji analizator bez końca gromadziłby dane oczekując na wyzwolenie i nie reagował na komendę przerywania pomiaru wysyłaną przez użytkownika z poziomu komputera PC, ponieważ analiza komend przesyłanych łączem szeregowym jest realizowana w analizatorze w pętli głównej programu. W celi zapobieżenia tej sytuacji, w programie na czas próbkowania stanów wejść In0…In7 aktywowane jest pomocnicze przerwanie *USIC0_0* z interfejsu szeregowego USIC0 (**listing 3**). Przerwanie to ma zaprogramowany wyższy priorytet niż przerwanie z układu CCU4 i jego jedynym zadaniem jest, po odebraniu znaku przez układ UART, wyłączenie próbkowania wejść In0…In7 i przełączenie

**Listing 4. Funkcja obsługi przerwania USIC0_0**

```
void USIC0_0_IRQHandler(void)
{
  USIC0_CH0->RBCTR &= ~USIC_CH_RBCTR_SRBIEN_Msk;  // disable UART RxBuf irq
  CCU40_CC40->TCCLR = CCU4_CC4_TCCLR_TRBC_Msk;  // stop sampling timer
  LED2_OFF;  // switch off LED2
  SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;  // start SysTick Timer
}
```

**Tabela 1. Komendy SUMP obsługiwane przez analizator stanów logicznych XMC 2Go LA v1.0**

| Lp | Komenda | Kod | Dane | Odpowiedź analizatora |
|---|---|---|---|---|
| 1 | Reset ustawień analizatora | 0x00 | Brak | Brak |
| 2 | Start pomiaru | 0x01 | Brak | Brak |
| 3 | Identyfikacja analizatora | 0x02 | Brak | 1ALS |
| 4 | Odczyt metadanych analizatora | 0x04 | Brak | 0x01 – identyfikator pola nazwy urządzenia<br>XMC2GO LA v1.0 – nazwa<br>0x00 – znacznik końca nazwy<br>0x02 – identyfikator pola wersji<br>1.0 – numer wersji<br>0x00 – znacznik końca wersji<br>0x21 – identyfikator pola rozmiaru dostępnej pamięci próbek<br>0x00, 0x00, 0x20, 0x00 – rozmiar pamięci w bajtach (tj. 8192B)<br>0x23 – identyfikator pola maksymalnej częstotliwości próbkowania<br>0x00, 0x07, 0xA1, 0x20 – maksymalna częstotliwość próbkowania w Hz (tj. 500000Hz)<br>0x40 – identyfikator pola liczby wejść analizatora<br>0x08 – liczba wejść<br>0x41 – identyfikator pola skróconego numeru wersji<br>0x02 – skrócony numer wersji<br>0x00 – znacznik końca metadanych |
| 5 | Maska wyzwalania pomiaru (stan 0) | 0xC0 | Bajt 1 – maski bitowe dla wejść In8..In1<br>Bajt 2 – ignorowany (oryginalnie maski bitowe dla wejść In16..In9)<br>Bajt 3 – ignorowany (oryginalnie maski bitowe dla wejść In24..In17 )<br>Bajt 4 – ignorowany (oryginalnie maski bitowe dla wejść In32..In25) | Brak |
| 6 | Wartość wyzwalająca pomiar (stan 0) | 0xC1 | Bajt 1 – wartości bitowe dla wejść In8..In1<br>Bajt 2 – ignorowany (oryginalnie wartości bitowe dla wejść In16..In9)<br>Bajt 3 – ignorowany (oryginalnie wartości bitowe dla wejść In24..In17 )<br>Bajt 4 – ignorowany (oryginalnie wartości bitowe dla wejść In32..In25) | Brak |
| 7 | Wartość dzielnika częstotliwości | 0x80 | Bajt 1 – LSB stopnia podziału dzielnika częstotliwości<br>Bajt 2 – drugi bajt stopnia podziału dzielnika częstotliwości<br>Bajt 3 – MSB stopnia podziału dzielnika częstotliwości<br>Bajt 4 – ??? – ignorowany | Brak |
| 8 | Liczba próbek i opóźnienie wyzwolenia pomiaru | 0x81 | Bajt 1 – LSB liczby próbek N<br>Bajt 2 – MSB liczby próbek N<br>Bajt 3 – LSB opóźnienia wyzwalania<br>Bajt 4 – MSB opóźnienia wyzwalania | Brak |
| 9 | Zarejestrowane stany logiczne wejść In0..In7 | Brak | Brak | Blok N bajtów zawierających kolejne próbki stanów logicznych wejść In0..In7 |




**Rysunek 5. Struktura projektu programu analizatora stanów logicznych XMC 2Go LA**

analizatora w stan oczekiwania na nowe komendy. Kod funkcji obsługi tego przerwania przedstawia **listing 4**.

Stan pracy analizatora jest obrazowany przez dwie diody LED, w które jest wyposażony moduł XMC 2Go. Mruganie diody LED1 z częstotliwością 0,5 Hz sygnalizuje, że układ jest wolny i oczekuje na skonfigurowanie oraz komendę rozpoczęcia pomiaru, natomiast sam pomiar jest sygnalizowany przez ciągłe świecenie diody LED2.

Oprogramowanie analizatora stanów logicznych zostało stworzone wyłącznie przy użyciu kompilatora GCC, bez użycia wspomnianego wcześniej środowiska programistycznego DAVE. Strukturę projektu programu przedstawia **rysunek 5**. Foldery *_bin* i *_obj* zawierają pliki wynikowe oraz obiektowe projektu. Z kolei folder *Cmsis* zawiera standardowe pliki nagłówkowe rdzeni Cortex-M (katalog *Cmsis*/*Include*) i plik nagłówkowy z definicjami rejestrów układów peryferyjnych mikrokontrolera XMC1100 (katalog *Cmsis*/*Xmc1100*/*Include*). W folderze tym znajduje się również plik źródłowy realizujący inicjalizację systemu (*Cmsis*/*Xmc1100*/*Source*/*system_XMC1100.c*) oraz plik z kodem startowym (*Cmsis*/*Xmc1100*/*Source*/*startup_XMC1100.S*). Z kolei foldery *Inc* oraz *Src* zawierają odpowiednio pliki nagłówkowe i źródłowe programu głównego analizatora stanów logicznych.

## Protokół komunikacyjny

Jako protokół komunikacyjny analizatora został zastosowany protokół SUMP [4., 5.]. Pierwotnie był on używany przez analizator stanów logicznych o nazwie *Sump Logic Analyzer*, lecz później protokół ten był implementowany w wielu projektach analizatorów stanów logicznych tworzonych na licencji Open Source. Najbardziej znanym z nich jest chyba *Openbench Logic Sniffer* (w skrócie OLS). Główną zaletą implementacji protokołu SUMP we własnym projekcie jest możliwość wykorzystania w charakterze programu sterującego analizatorem oprogramowania stworzonego dla analizatora OLS. Ponieważ prezentowany układ analizatora na bazie zestawu XMC 2Go ma znacznie mniejsze możliwości niż analizator OLS, nie było potrzeby implementacji pełnego protokołu SUMP. **Tabela 1** zawiera listę komend protokołu SUMP zaimplementowanych w oprogramowaniu analizatora XMC 2Go LA.

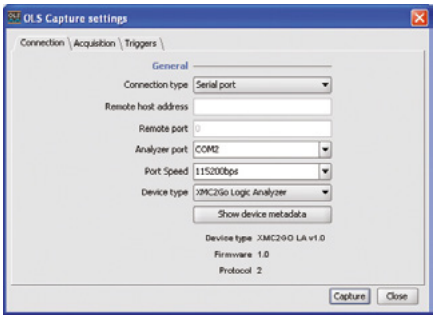
## Program dla komputera PC

Jak już wspomniano, do ustawiania parametrów pracy analizatora XMC 2Go LA, oraz do obrazowania zarejestrowanych poziomów logicznych sygnałów doprowadzonych do jego wejść, może zostać użyty dowolny program klienta obsługujący protokół SUMP. W prezentowanym projekcie zastosowano do tego celu program o nazwie *OpenBench LogicSniffer* (w skrócie OLS), którego autorem jest Jan Willem Janssen [6.]. Jest to chyba najlepszy program klienta SUMP dostępny na licencji Open Source. W chwili pisania artykułu ostatnia opublikowana wersja programu OLS nosiła numer 0.9.7.1, jednak jak wykazały próby, nie zawsze pracuje ona stabilnie. Zaobserwowano, że występują w niej problemy z automatyczną identyfikacją dołączonego modułu analizatora. W związku z tym do pracy z analizatorem XMC 2Go LA polecana jest poprzednia wersja klienta OLS o numerze 0.9.7.

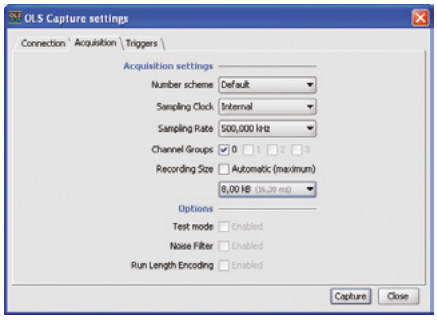
Program OLS został napisany w języku Java. Może więc być uruchamiany w wielu

**Listing 5. Plik konfiguracyjny analizatora XMC 2Go LA dla programu OLS**

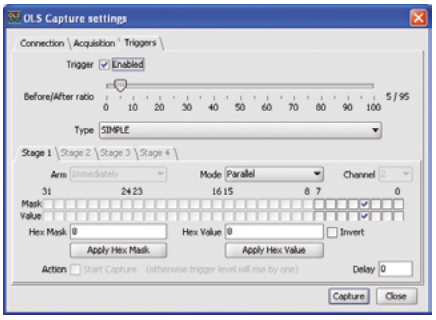
```
# Configuration for XMC 2Go Logic Analyzer profile
# The short (single word) type of the device described in this profile
device.type = XMC2GO
# A longer description of the device
device.description = XMC2Go Logic Analyzer
# The device interface, SERIAL only
device.interface = SERIAL
# The device's native clockspeed, in Hertz.
device.clockspeed = 32000000
# The clockspeed used in the divider calculation, in Hertz. Defaults to 100MHz as most devices appear to use this.
device.dividerClockspeed = 32000000
# Whether or not double-data-rate is supported by the device (also known as the "demux"-mode).
device.supports_ddr = false
# Supported sample rates in Hertz, separated by comma's
device.samplerates = 10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000, 20000, 50000, 100000, 200000, 500000
# What capture clocks are supported
device.captureclock = INTERNAL
# The supported capture sizes, in bytes
device.capturesizes = 64, 128, 256, 512, 1024, 2048, 4096, 8192
# Whether or not the noise filter is supported
device.feature.noisefilter = false
# Whether or not Run-Length encoding is supported
device.feature.rle = false
# Whether or not a testing mode is supported
device.feature.testmode = false
# Whether or not triggers are supported
device.feature.triggers = true
# The number of trigger stages
device.trigger.stages = 1
# Whether or not "complex" triggers are supported
device.trigger.complex = false

# The total number of channels usable for capturing
device.channel.count = 8
# The number of channels groups, together with the channel count determines the channels per group
device.channel.groups = 1
# Whether the capture size is limited by the enabled channel groups
device.capturesize.bound = false
# Which numbering does the device support
device.channel.numberingschemes = DEFAULT

# Is a delay after opening the port and device detection needed? (0 = no delay, >0 = delay in milliseconds)
device.open.portdelay = 0
# The receive timeout for the device (in milliseconds, 100 = default, <=0 = no timeout)
device.receive.timeout = 100
# Does the device need a high or low DTR-line to operate correctly? (high = true, low = false)
device.open.portdtr = true
# Which metadata keys correspond to this device profile? Value is a comma-separated list of (double quoted) names...
device.metadata.keys = "XMC2GO LA v1.0"

# In which order are samples sent back from the device? false = last sample first, true = first sample first
device.samples.reverseOrder = true
```




**Rysunek 6. Konfiguracja połączenia klienta OLS z modułem analizatora**




**Rysunek 7. Wybór parametrów akwizycji sygnału**




**Rysunek 8. Konfiguracja warunków wyzwolenia pomiaru**

systematach operacyjnych, w tym w systemie MS Windows oraz Linux. Sam program OLS nie wymaga instalacji. Po pobraniu go ze strony autora i zapisaniu na dysku w wybranym katalogu należy jedynie w folderze *\plugins* dodatkowo umieścić plik konfiguracyjny o nazwie *ols.profile-xmc2go.cfg* (**listing 5**). Plik ten jest dołączony do kodu źródłowego opisywanego analizatora i zawiera definicje obsługiwanych przez analizator XMC 2Go LA trybów pracy, dostępnych częstotliwości próbkowania wejść, rozmiarów buforów próbek, kolejności przesyłanych próbek sygnałów, itp.

Najważniejszym parametrem konfiguracyjnym analizatora zdefiniowanym w pliku *ols.profile-xmc2go.cfg* jest natywna częstotliwość sygnału zegarowego analizatora *device.dividerClockspeed*. Na jej podstawie program OLS wylicza, w zależności od wybranej częstotliwości próbkowania stanów wejść, wartość podziału dzielnika częstotliwości, która jest programowana w urządzeniu. **W przypadku analizatora XMC 2Go wartość parametru *device.dividerClockspeed* musi być równa 32000000 (tj. 32 MHz). Inaczej rzeczywista częstotliwość próbkowania stanów wejść analizatora nie będzie odpowiadała wartości wybranej w programie OLS.** Należy to mieć na względzie przy samodzielnej modyfikacji pliku konfiguracyjnego, lub też używaniu konfiguracji innego urządzenia, ponieważ większość plików konfiguracyjnych dla popularnych analizatorów stanów logicznych dostarczana razem w programem OLS zakłada, że natywna częstotliwość sygnału zegarowego analizatora jest równa 100 MHz.

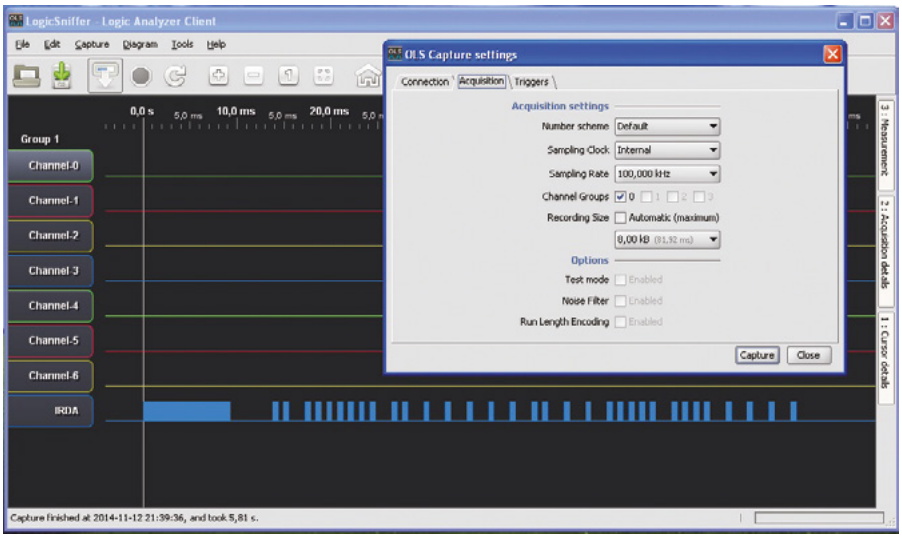
Drugim ważnym parametrem konfiguracyjnym zapisanym w pliku *ols.profile-xmc2go.cfg* jest definicja klucza tekstowego *device.metadata.keys*. Program klienta OLS wykorzystuje ją do automatycznego rozpoznawania typu dołączonego analizatora stanów logicznych. W przypadku modułu XMC 2Go LA wartość tego klucza powinna być równa *„XMC2GO LA v1.0”*.

Pozostałe parametry konfiguracyjne zdefiniowane w pliku *ols.profile-xmc2go.cfg* nie mają już tak newralgicznego znaczenia dla działania analizatora. Informują one program OLS o poszczególnych trybach pracy, które są obsługiwane przez urządzenie. Dzięki temu w programie OLS aktywne są tylko

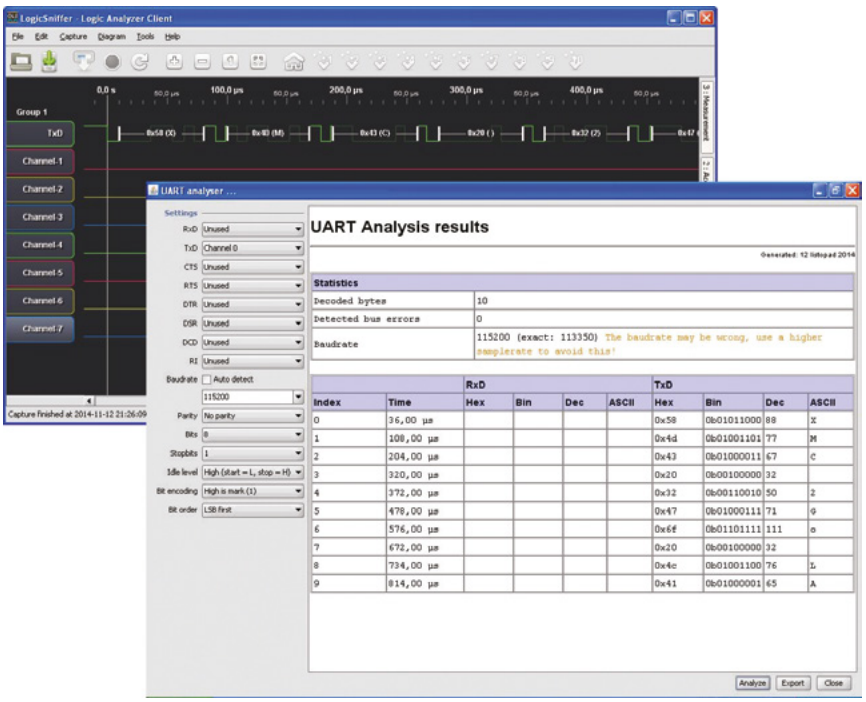
te funkcje analizatora, które są obsługiwane przez podłączony moduł.

Sposób obsługi programu OLS nie różni się znacząco od obsługi innych tego typu aplikacji. Po uruchomieniu programu należy wybrać numer portu szeregowego COM, do którego podłączony jest analizator i przyciskiem *Show device metadata* dokonać identyfikacji modułu analizatora. Jest to jednocześnie test poprawności komunikacji programu klienta OLS z analizatorem. Jeżeli operacja przebiegnie pomyślnie, program OLS powinien wyświetlić typ urządzenia i jego podstawowe dane oraz uaktywnić odpowiadającą mu konfigurację pracy (**rysunek 6**). Rodzaj dołączonego modułu analizatora może też zostać wybrany ręcznie z rozwijanej listy. W kolejnym kroku należy określić częstotliwość, z jaką analizator będzie próbkował stany swoich wejść oraz rozmiar bufora próbek (**rysunek 7**). Całkowity czas rejestrowania sygnału będzie w takiej sytuacji oczywiście równy iloczynowi wielkości bufora próbek i okresu próbkowania. W razie potrzeby możliwe jest też określenie momentu wyzwolenia pomiaru poziomów logicznych wejść analizatora oraz sekwencji wyzwalającej pomiar (**rysunek 8**). W obecnej wersji oprogramowania analizator XMC 2Go LA obsługuje tylko tryb wyzwalania równoległego pojedynczą kombinacją poziomów logicznych wejść In0…In7. Nie jest natomiast obsługiwane ani wyzwalanie sekwencją kolejnych stanów logicznych wejść In0…In7, ani też sekwencją szeregową stanów logicznych wybranego wejścia. Po zakończeniu pracy analizatora, program OLS powinien wyświetlić zarejestrowane przez moduł analizatora stany logiczne wejść In0…In7. Przykładowy, uzyskany w trakcie testów układu analizatora sygnał przedstawia **rysunek 9**.

Możliwości programu OLS nie ograniczają się tylko do prezentacji w postaci graficznej zarejestrowanych przez analizator stanów logicznych wejść. Sygnały te mogą zostać poddane w programie OLS dalszej analizie. Poza standardowymi pomiarami parametrów sygnału za pomocą kursorów, program OLS oferuje też automatyczną analizę magistral szeregowych. Na liście obsługiwanych magistral znajdują się między innymi takie interfejsy jak: 1-wire, DMX512, i2c, SPI, JTAG oraz UART. Przykład automatycznej analizy przez program OLS sygnału portu szeregowego przedstawia **rysunek 10**.




**Rysunek 9. Zarejestrowany przez analizator XMC 2Go LA sygnał z pilota AverMedia sterującego kartą przetwarzania sygnału video**




**Rysunek 10. Automatyczna analiza zarejestrowanego sygnału nadajnika UART w programie OLS**

## Podsumowanie

Przedstawiony analizator stanów logicznych nie bije ani rekordów prędkości rejestracji sygnałów, ani też rekordów w liczbie obsługiwanych wejść. Pod tym względem nie może się on równać z profesjonalnymi przyrządami pomiarowymi. Tym niemniej, jak pokazują przykłady z rys. 9 i 10, do obserwacji i analizy sygnałów o średniej szybkości zmian jest on zupełnie wystarczający. W tym zakresie analizator XMC 2Go LA może realnie wspomóc konstruktora przy diagnozowaniu błędów w budowanym układzie czy też w pisanym programie. Dysponując zestawem ewaluacyjnym mikrokontrolera o większych możliwościach, w oparciu o powyższy projekt można zbudować analizator stanów logicznych działający szybciej, obsługujący większą liczbę wejść, czy też mający większy bufor danych.

**Aleksander Borysiuk**
**alex_priv@wp.pl**

**Bibliografia:**

1. Evaluation Board For XMC1100 Family. XMC 2Go Kit with XMC1100. Kit Version 1.0. Board User's Manual, Revision 1.0, Infineon Technologies AG, 2014
2. Infineon Forums. Microcontroller Forum. XMC Forum. XMC1100 NVM wait states documentation, *http://goo.gl/Hxxawq*
3. XMC1000 Microcontroller Series for Industrial Applications. C - Start and Device Initialization. Device Guide, V1.0, Infineon Technologies AG, 2013
4. SUMP Communications Protocol, *http://goo.gl/7IKs57*
5. The Logic Sniffer's extended SUMP protocol, *http://goo.gl/EIqRE8*
6. OpenBench LogicSniffer Client, *http://goo.gl/pXWD4E*