Digilent Pmod i STM32 (3)

Biblioteki do obsługi modułów peryferyjnych

Kolejny artykuł z cyklu dotyczącego modułów peryferyjnych Pmod jest poświęcony modułom PmodACL z 3-osiowym akcelerometrem oraz PmodMAXSONAR z ultradźwiękowym czujnikiem odległości. Podobnie jak w poprzednik częściach, wszystkie opisywane przykłady zostały napisane w oparciu na bibliotece STM32Cube_FW_L4 i mogą być uruchomione w środowisku Atollic TrueSTUDIO na zestawie deweloperskim KAmeLeon (*www.kameleonboard.org*).

PmodACL

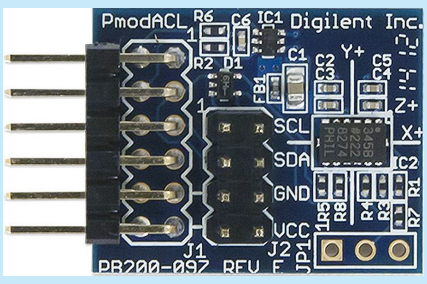
Moduł PmodACL (**fotografia 1**) zawiera 3-osiowy akcelerometr Analog Devices ADXL345 umożliwiający pomiar przyspieszenia w jednym z konfigurowanych zakresów: ±2g, ±4g, ±8g, ±16g. Rozdzielczość pomiaru może być ustawiona na 10 bitów niezależnie od zakresu, lub w trybie dynamicznym, który automatycznie ustawia rozdzielczość na 4 mg/LSB dla każdego zakresu pomiarowego. Akcelerometr ADXL345 posiada także możliwość wykrywania aktywności, pojedynczego, lub podwójnego stuknięcia i swobodnego spadania. Wszystkie z powyższych detekcji są konfigurowane w rejestrach akcelerometru przechowujących informacje o progach przyspieszenia poszczególnych aktywności i czasach ich trwania. Dodatkowo akcelerometr posiada wewnętrzną kolejkę FIFO przechowującą wyniki pomiarów i umożliwiającą zmniejszenie obciążenia mikrokontrolera.

Układ ADXL345 ma możliwość informowania mikrokontrolera o wykrytych zdarzeniach za pomocą dwóch linii przerwań: INT1 i INT2. Dla obu linii dostępne są następujące źródła przerwań:

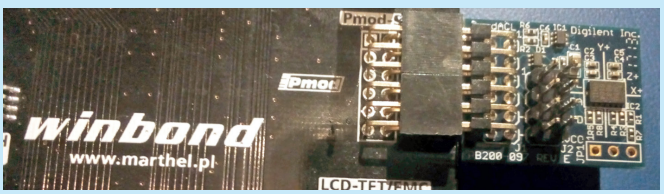
- DATA_READY – dostępne są nowe dane do odczytu.
- SINGLE_TAP – wykryto pojedyncze stuknięcie.
- DOUBLE_TAP – wykryto podwójne stuknięcie.
- Activity – wykryto stan aktywności.
- Inactivity – wykryto stan braku aktywności.
- FREE_FALL – wykryto swobodny spadek.
- Watermark – kolejka FIFO została zapełniona do ustawionego poziomu.
- Overrun – dostępne dane zostały nadpisane lub kolejka została przepełniona.

Każde z nich można niezależnie włączyć i przyporządkować do wybranej linii przerwań.

Dostęp do 8-bitowych rejestrów konfiguracyjnych i danych pomiarowych jest możliwy za pośrednictwem interfejsów SPI i I²C. Interfejs I²C jest aktywny, gdy pin SS (chip select) jest ustawiony, natomiast po wyzerowaniu pinu SS jest możliwa komunikacja za pośrednictwem SPI. Moduł PmodACL został wyposażony w złącza dla obu interfejsów – J2 odpowiada 8-pinowemu interfejsowi I²C Pmod, natomiast J1 jest 12-pinowym interfejsem SPI typu 2A z dodatkową linią przerwania w miejsce sygnału RESET. W opisywanym przykładzie komunikacja z PmodACL odbywa się za pośrednictwem interfejsu SPI, ponieważ złącze J1 jest kompatybilne ze złączem Pmod-SPI na płytce KAmeleon. Opis połączeń uwzględniający wykorzystane piny mikrokontrolera STM32L496ZG znajduje się w **tabeli 1**, natomiast na **fotografii 2** przedstawiono podłączony moduł.



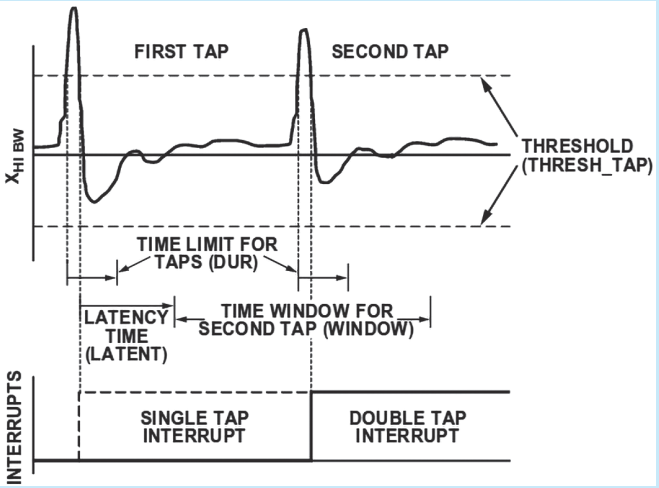
Fotografia 1. Moduł PmodACL



Fotografia 2. PmodACL podłączony do płytki KAmeLeon

Biblioteki opisane w artykule są dostępne bezpłatnie do pobrania na stronie *KAMAMI.pl*. W oknie wyszukiwarki trzeba wpisać nazwę modułu Pmod, na stronie wyrobu jest dostępny bezpośredni link do biblioteki.

Kod obsługujący moduł PmodACL znajduje się w pliku src/PmodACL.c. Zaimplementowano w nim funkcje do konfiguracji modułu



Rysunek 3. Algorytm detekcji podwójnego stuknięcia w akcelerometrze ADXL345 (źródło: dokumentacja ADXL345)

Tabela 1. Sygnały PmodACL oraz odpowiadające im piny mikrokontrolera; w tabeli pominięto sygnały (NC) i linie zasilania występujące na złączu Pmod

Sygnał	Numer pinu PmodACL (J1)	Pin STM32L496ZG (KAmeLeon Pmod-SPI)
~SS	1	PB0
MOSI	2	PA7
MISO	3	PE14
SCLK	4	PA1
INT2	7	PE12
INT1	8	PE13

(PmodACL_Config), odczytu danych z trzech osi (PmodACL_ReadXYZ) i odczytu stanu przerwań (PmodACL_ReadInterruptFlags). Funkcja konfiguracyjna została przedstawiona na **listngu 1**. Jest ona odpowiedzialna za konfigurację modułu SPI1 mikrokontrolera oraz wymaganych pinów. SPI konfigurowany jest w trybie dwukierunkowym (linie MISO i MOSI), 8 bitów danych, z programową kontrolą sygnału SS. Po inicjalizacji interfejsu SPI ustawiane są wartości rejestrów układu ADXL345. Przykładowa sekwencja konfiguracji włączająca detekcję podwójnego stuknięcia została przedstawiona w **tabeli 2**.

Wyjaśnienia może wymagać sposób detekcji konfigurowany za pomocą opisanych rejestrów. Akcelerometr wykrywa wszystkie zdarzenia o amplitudzie przyspieszenia przekraczającej wartość podaną w rejestrze THRESH_TAP i jednocześnie czasie nieprzekraczającym wartości w rejestrze DUR. Zdarzenia takie są klasyfikowane jako pojedyncze stuknięcia. Po pierwszym zdarzeniu akcelerometr odczekuje czas ustawiony w rejestrze LATENT i jeżeli wykryje kolejne zdarzenie w czasie ustawionym w rejestrze WINDOW, zgłasza przerwanie DOUBLE_TAP. Algorytm przedstawiono na **rysunku 3**.

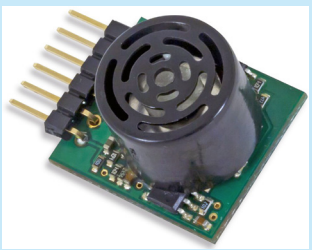
Na końcu funkcji PmodACL_Config jest konfigurowane przerwanie na pinie PE13, które odpowiada sygnałowi INT1. Domyślnie wszystkie przerwania akcelerometru są przypisane do tej linii (rejestr 0x25 – INT_MAP). W przykładzie linia przerwań INT2 nie jest używana.

Zgodnie z konwencją biblioteki STM32Cube, wywołanie funkcji bibliotecznej HAL_SPI_Init powoduje wywołanie funkcji HAL_SPI_MspInit, która została zaimplementowana w pliku src/PmodACL.c. Funkcja ta została przedstawiona na **listingu 2**. Jest ona odpowiedzialna za konfigurację pinów SS, SCLK, MISO oraz MOSI. Zgodnie z konfiguracją modułu SPI1, pin SS (PB0) jest sterowany programowo i zostaje pod kontrolą portu GPIO.

Podczas wysyłania adresu rejestru przez interfejs SPI, najstarsze dwa bity mają szczególne znaczenia. Bit 7. oznacza typ operacji – zapis (0), lub odczyt (1), natomiast bit 6. może być ustawiony do odczytu, lub zapisu wielu bajtów. Adres jest wówczas automatycznie inkrementowany, co umożliwia odczyt danych ze wszystkich osi podczas jednej transakcji SPI. Funkcja realizująca odczyt i konwersję danych jest przedstawiona na **listingu 3**. Przy ustawieniu rozdzielczości na 4 mg/LSB, odczyt każdej z osi należy pomnożyć przez 4, aby otrzymać wynik pomiaru w [mg]. Kolejka FIFO w przykładzie nie jest używana, dlatego zawsze odczytywane są wyniki z ostatniego pomiaru. Częstotliwość wykonywania pomiarów jest konfigurowana w rejestrze BW_RATE (0x2C).

Listing 1. Konfiguracja modułu PmodACL

```
void PmodACL_Config(void)
{
  // Enable clock for all required GPIO ports and SPI1.
  __HAL_RCC_SPI1_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOE_CLK_ENABLE();

  // Configure the SPI connected to the Pmod module.
  pmodAclSpi.Instance = SPI1;
  pmodAclSpi.Init.Mode = SPI_MODE_MASTER;
  pmodAclSpi.Init.Direction = SPI_DIRECTION_2LINES;
  pmodAclSpi.Init.DataSize = SPI_DATASIZE_8BIT;
  pmodAclSpi.Init.CLKPolarity = SPI_POLARITY_HIGH;
  pmodAclSpi.Init.CLKPhase = SPI_PHASE_2EDGE;
  pmodAclSpi.Init.NSS = SPI_NSS_SOFT;
  pmodAclSpi.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_128;
  pmodAclSpi.Init.FirstBit = SPI_FIRSTBIT_MSB;
  pmodAclSpi.Init.TIMode = SPI_TIMODE_DISABLE;
  pmodAclSpi.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  pmodAclSpi.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;

  HAL_SPI_Init(&pmodAclSpi);

  // Initialization of PmodACL.
  writeRegister(0x1d, 0x3C); // Set the tap threshold (THRESH_TAP register) to 60 * 62.5mg.
  writeRegister(0x21, 0x0A); // Set the tap duration (DUR register) to 10 * 625us.
  writeRegister(0x22, 0x10); // Set the latent (Latent register) to 16 * 1.25ms.
  writeRegister(0x23, 0x80); // Set the window duration (Window register) to 128 * 1.25m.
  writeRegister(0x2a, 0x07); // Enable all axes for tap detection (TAP_AXES register).
  writeRegister(0x2d, 0x08); // Enable measurement in POWER_CTL register.

  // Clear all flags before enabling the interrupt.
  PmodACL_ReadInterruptFlags();

  writeRegister(0x2e, 0x20); // Enable double tab interrupt in INT_ENABLE register.
  writeRegister(0x31, 0x28); // Set the interrupts to be active in low state and enable
                             // full resolution in DATA_FORMAT register.

  // Configure the INT1 (PE13) interrupt line with the lowest priority.
  GPIO_InitTypeDef GPIO_InitStruct;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Pin = GPIO_PIN_13;
  HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0x0F, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
}
```



Fotografia 4. Wygląd modułu PmodMAXSONAR

Tabela 3. Sygnały PmodMAXSONAR oraz odpowiadające im piny złącza ESP8266/UART i piny mikrokontrolera

Sygnał	Numer pinu PmodHYGRO	Numer pinu KAmeLeon ESP8266/UART	Pin mikrokontrolera
AN	1	-	-
RX	2	1 (TX/PA2)	PA2
TX	3	8 (RX/PA3)	PA3
PW	4	6 (PD/PF11)	PF11
GND	5	7 (GND)	-
VCC	6	2 (+3,3 V)	-

Tabela 2. Konfiguracja rejestrów ADXL345 do detekcji podwójnego stuknięcia

Rejestr	Wartość	Opis
0x1D (THRESH_TAP)	0x3C	Próg przyspieszenia dla detekcji stuknięcia (62,5 mg/LSB)
0x21 (DUR)	0x0A	Maksymalna czas trwania stuknięcia (625 µs/LSB)
0x22 (Latent)	0x10	Opóźnienie przed detekcją drugiego stuknięcia (1,25 ms/LSB)
0x23 (Window)	0x80	Czas oczekiwania na drugie stuknięcie (1,25 ms/LSB)
0x2A (TAP_AXES)	0x07	Włączenie osi do detekcji; wszystkie trzy są włączone
0x2D (POWER_CTL)	0x08	Włączenie pomiarów z wyłączonym trybem SLEEP
0x30 (INT_SOURCE)	odczyt	Odczyt i wyczyszczenie aktywnych flag przerwań; funkcja PmodACL_ReadInterruptFlags
0x2E (INT_ENABLE)	0x20	Włączenie przerwania podwójnego stuknięcia (DOUBLE_TAP)
0x31 (DATA_FORMAT)	0x28	Przerwanie aktywne w stanie niskim; automatyczna zmiana rozdzielczości (4 mg/LSB); wyrównanie danych do prawej; zakres pomiaru ±2 g

Poprzednie części kursu i dodatkowe materiały dostępne są na stronie www.media.avt.pl

W przykładzie ma on wartość domyślną 0x0A, co oznacza 100 Hz. Za realizację odczytu i zapisu przez SPI są odpowiedzialne dwie funkcje pomocnicze: readRegister i writeRegister pokazane na **listingu 4**.

Główna pętla programu w pliku src/main.c wykonuje odczyt danych co jedną sekundę i wysyła wartości przyspieszenia ze wszystkich trzech osi na port szeregowy LPUART1. Jego konfiguracja znajduje się w pliku src/serial.c. W pliku main.c znajduje się także obsługa przerwania konfigurowanego przez funkcję PmodACL_Config. W funkcji obsługi przerwania, przedstawionej na **listingu 5**, odczytywany jest rejestr akcelerometru przechowujący stan przerwań. Mimo, że w przykładzie używane jest tylko jedno źródło przerwania, to odczyt jest konieczny, ze względu na czyszczenie ich stanu w akcelerometrze.

Wystąpienie przerwania sygnalizowane jest przez mrugnięcie diody LED0 na płytce KAmeLeon. Użyta w przerwaniu funkcja HAL_Delay wprowadza opóźnienie w obsłudze przerwania, co nie jest dobrą praktyką i zostało zaimplementowane tylko ze względu na uproszczenie kodu. Aby opóźnienie mogło zadziałać, priorytet przerwania SysTick musi być wyższy od przerwania, w obsłudze którego jest ono użyte. Priorytet ten można zmienić w pliku stm32l4xx_hal_conf.h (TICK_INT_PRIORITY). Funkcje pomocnicze do obsługi diody zostały zaimplementowane w pliku src/led.c

PmodMAXSONAR

PmodMAXSONAR (**fotografia 4**) jest ultradźwiękowym czujnikiem odległości z modułem MB1010 LV-MaxSonar-EZ1. Urządzenie jest w stanie wykryć odległość od obiektów w zasięgu od 6 do 255 cali, a pomiary są wykonywane z rozdzielczością jednego cala. Wyniki pomiarów są udostępniane za pośrednictwem interfejsu UART, analogowo i w postaci sygnału PWM o zmiennym wypełnieniu: stan wysoki trwa 1 ms, a stan niski 147 µs na każdy cal wykrytej odległości od przeszkody. W omawianym przykładzie dane są odczytywane przez UART, a stan pinu PWM jest dodatkowo sygnalizowany przez diodę na płytce KAmeLeon.

Moduł PmodMAXSONAR posiada 6-pinowy interfejs UART typu 4, jednak zamiast pinów kontroli przepływu CTS i RTS udostępnione są odpowiednio piny AN (odczyt analogowy) i PWM. W przykładzie moduł został podłączony do złącza ESP8266/UART na płytce KAmeleon. Złącze to udostępnia interfejs USART2 oraz dodatkowy pin PF11.

Lista pinów na złączu PmodMAXSONAR, wraz z odpowiadającymi im pinami złącza ESP8266/UART i pinami mikrokontrolera została przedstawiona w **tabeli 3**, a podłączony moduł można zobaczyć na **fotografii 5**.

Do komunikacji z modułem PmodMAXSONAR używany jest wyłącznie pin PA3, czyli RX po stronie mikrokontrolera. Pin PA2 powinien być na stałe w stanie wysokim. Jest to możliwe

Listing 2. Konfiguracja pinów używanych przez SPI

```
void HAL_SPI_MspInit(SPI_HandleTypeDef *hspi)
{
  // Initialize GPIO used by the SPI1 peripheral. The CS is controlled by the software.
  GPIO_InitTypeDef GPIO_InitStruct;

  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF5_SPI2;
  GPIO_InitStruct.Pin = GPIO_PIN_1 | GPIO_PIN_7;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  GPIO_InitStruct.Pin = GPIO_PIN_14;
  HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
}

void PmodACL_ReadXYZ(int16_t* x, int16_t* y, int16_t* z)
{
  uint8_t data[6];
  readRegister(0x32, data, 6);

  // The ADXL345 on PmodACL is configured for the highest resolution (4mg/LSB)
  // so the axes readings need to be multiplied by 4 to obtain the acceleration in [mg].
  *x = ((data[1] << 8) + data[0]) * 4;
  *y = ((data[3] << 8) + data[2]) * 4;
  *z = ((data[5] << 8) + data[4]) * 4;
}

uint8_t PmodACL_ReadInterruptFlags(void)
{
  uint8_t data;
  readRegister(0x30, &data, 1);
  return data;
}
```

Listing 3. Odczyt i konwersja danych z akcelerometru

```
static void readRegister(uint8_t address, uint8_t* data, uint8_t size)
{
  // Reading N bytes requires one more byte at the beginning for address transmission,
  // so the whole SPI transaction has to be N + 1 bytes long.
  uint8_t txbuf[MAX_SPI_RX_BUFER_LEN + 1] = {0x00};
  uint8_t rxbuf[MAX_SPI_RX_BUFER_LEN + 1] = {0x00};
  txbuf[0] = address | SPI_READ_FLAG | SPI_MB_FLAG;

  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  HAL_SPI_TransmitReceive(&pmodAclSpi, txbuf, rxbuf, size + 1, 100);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);

  for(int i=0; i<size; i++)
    data[i] = rxbuf[i+1];
}
```

Listing 4. Funkcje pomocnicze do zapisu i odczytu danych przez SPI

```
static void writeRegister(uint8_t address, uint8_t data)
{
  uint8_t txbuf[2] = {address | SPI_WRITE_FLAG | SPI_MB_FLAG, data};

  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  HAL_SPI_Transmit(&pmodAclSpi, txbuf, 2, 100);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
}

static void readRegister(uint8_t address, uint8_t* data, uint8_t size)
{
  // Reading N bytes requires one more byte at the beginning for address transmission,
  // so the whole SPI transaction has to be N + 1 bytes long.
  uint8_t txbuf[MAX_SPI_RX_BUFER_LEN + 1] = {0x00};
  uint8_t rxbuf[MAX_SPI_RX_BUFER_LEN + 1] = {0x00};
  txbuf[0] = address | SPI_READ_FLAG | SPI_MB_FLAG;

  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  HAL_SPI_TransmitReceive(&pmodAclSpi, txbuf, rxbuf, size + 1, 100);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);

  for(int i=0; i<size; i++)
    data[i] = rxbuf[i+1];
}
```

Listing 5. Funkcja obsługi przerwań generowanych przez PmodACL

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  // Read the flags to clear the interrupt state register.
  PmodACL_ReadInterruptFlags();

  // Blink LED to indicate that the interrupt was detected.
  // When using HAL_Delay within the interrupt handler, the SysTick interrupt priority
  // has to be higher than the priority of the currently processed interrupt.
  // Otherwise the delay counter inside the HAL library will not increment.
  // The SysTick interrupt priority is defined inside the stm32l4xx_hal_conf.h.
  // The GPIO priority is set inside the PmodACL_Config() function.
  Led_TurnOn(LED0);
  HAL_Delay(100);
  Led_TurnOff(LED0);
}
```

dlatego, ponieważ moduł PmodMAXSONAR nie wymaga żadnej konfiguracji. Po włączeniu zasilania wykonuje kalibrację i rozpoczyna od razu wysyłać dane pomiarowe przez UART oraz piny AN i PWM. Dane są wysyłane raz na 49 ms. Konfiguracja modułu w prezentowanym przykładzie znajduje się w funkcji PmodMAXSONAR_Config, znajdującej się w pliku src/PmodMAXSONAR.c i przedstawionej na **listingu 6**.

Listing 6. Konfiguracja interfejsu UART i linii przerwań do komunikacji z modułem PmodMAXSONAR

```
void PmodMAXSONAR_Config(void)
{
  __HAL_RCC_GPIOF_CLK_ENABLE();

  // Configure the interrupts on PF11 (PWM) pin. The interrupts are active on both edges
  // to show the PWM signal duty cycle.
  GPIO_InitTypeDef GPIO_InitStruct;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING_FALLING;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Pin = GPIO_PIN_11;
  HAL_GPIO_Init(GPIOF, &GPIO_InitStruct);

  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 2, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

  // The UART interface is used only for data reception. The TX pin should be kept high.
  pmodMaxsonarUart.Instance = USART2;
  pmodMaxsonarUart.Init.BaudRate = 9600;
  pmodMaxsonarUart.Init.WordLength = UART_WORDLENGTH_8B;
  pmodMaxsonarUart.Init.StopBits = UART_STOPBITS_1;
  pmodMaxsonarUart.Init.Parity = UART_PARITY_NONE;
  pmodMaxsonarUart.Init.Mode = UART_MODE_RX;
  pmodMaxsonarUart.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  pmodMaxsonarUart.Init.OverSampling = UART_OVERSAMPLING_16;
  pmodMaxsonarUart.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;

  HAL_UART_Init(&pmodMaxsonarUart);
}
```

W pierwszym kroku konfigurowane jest przerwanie na pinie PF11, które będzie informowało o zmianie stanu sygnału PWM. Przerwanie to jest aktywne na obu zboczach sygnału wejściowego. Interfejs UART musi być skonfigurowany zgodnie z wymaganiami modułu PmodMAXSONAR, czyli 8 bitów danych, 1 bit stopu, brak parzystości i prędkość transmisji 9600 b/s. Podobnie, jak w przypadku innych interfejsów, wywołanie HAL_UART_Init powoduje wywołanie funkcji konfiguracyjnej HAL_UART_MspInit, w której można dokonać dalszej konfiguracji zegara i GPIO.

Listing 7. Konfiguracja interfejsów UART występujących w przykładzie

```
void HAL_UART_MspInit(UART_HandleTypeDef *huart)
{
  if(huart->Instance == USART2)
    PmodMAXSONAR_HAL_UART_MspInit(huart);
  else if(huart->Instance == LPUART1)
    Serial_HAL_UART_MspInit(huart);
}
```

Z uwagi na to, że w przykładzie są wykorzystywane dwa interfejsy szeregowe, wspomniana funkcja jest zaimplementowana w pliku src/main.c (**listing 7**). Sprawdza ona, który interfejs ma być konfigurowany i wywołuje odpowiednio funkcję PmodMAXSONAR_HAL_UART_MspInit lub Serial_HAL_UART_MspInit. Pierwsza z nich, pokazana na **listingu 8**, ustawia pin PA2 w stanie wysokim, natomiast pin PA3 jako funkcję alternatywną dla interfejsu UART2. Na koniec włączane jest przerwanie od UART-a, które jest potrzebne do odbioru danych pomiarowych.

Listing 8. Konfiguracja pinów do komunikacji z modułem PmodMAXSONAR

```
void PmodMAXSONAR_HAL_UART_MspInit(UART_HandleTypeDef *huart)
{
  // Enable the clocks for GPIO pins used by the USART2 port (PA2, PA3).
  __HAL_RCC_USART2_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();

  GPIO_InitTypeDef GPIO_InitStruct;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF7_USART2;
  GPIO_InitStruct.Pin = GPIO_PIN_3;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  // The TX pin (PA2)  RX of the PmodMAXSONAR should be left in the high state
  // and not used for communication.
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Pin = GPIO_PIN_2;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_2, GPIO_PIN_SET);

  // Enable the interrupts generated by USART2 port.
  HAL_NVIC_SetPriority(USART2_IRQn, 1, 0);
  HAL_NVIC_EnableIRQ(USART2_IRQn);
}
```

PmodMAXSONAR wysyła dane w określonym formacie: wielka litera 'R', zmierzona trzycyfrowa odległość wyrażona w calach (od 006 do 255) i znak powrotu karetki '\r' (0x0D). Funkcja obsługi przerwania UART-a, realizująca odbiór danych jest przedstawiona na **listingu 9**. Sprawdza ona czy odebranym znakiem jest '\r'. Jeżeli tak, to ustawia globalną flagę dataReady odczytywaną w głównej pętli programu, informując tym samym o gotowych danych. W przeciwnym razie znak dodawany jest do bufora i rozpoczyna się oczekiwanie na kolejny.

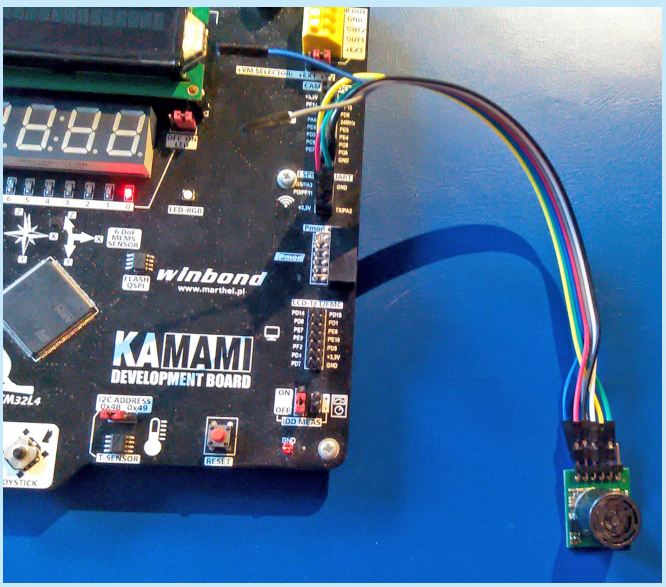
Oczekiwanie na znak inicjalizuje funkcja PmodMAXSONAR_Read, będąca jedynie wrapperem na funkcję biblioteczną HAL_UART_Receive_IT. Główna pętla programu, znajdująca się w funkcji main, oczekuje na ustawienie flagi dataReady, po czym wysyła otrzymane dane na port szeregowy LPUART1 i inicjalizuje oczekiwanie na kolejne znaki.

Sygnał PWM modułu PmodMAXSONAR, zgodnie z konfiguracją, wywołuje przerwania na pinie PF11 mikrokontrolera. Funkcja obsługująca te przerwania znajduje się również w pliku src/main.c i zmienia stan diody LED0 płytki KAmeLeon, dzięki czemu można zaobserwować zmieniające się wypełnienie sygnału PWM wraz ze zmianą odległości czujnika od przeszkody.

Krzysztof Chojnowski

Listing 9. Funkcja obsługi przerwania UART

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  // Look for the end of the line and notify the main loop.
  // Otherwise receive next character.
  if(dataBuffer.buffer[dataBuffer.index] == '\r') {
    dataBuffer.index++;
    dataReady = 1;
} else {
  dataBuffer.index++;
    PmodMAXSONAR_Read(&dataBuffer.buffer[dataBuffer.index], 1);
  }
}
```



Fotografia 5. Moduł PmodMAXSONAR przyłączony do płytki KAmeLeon