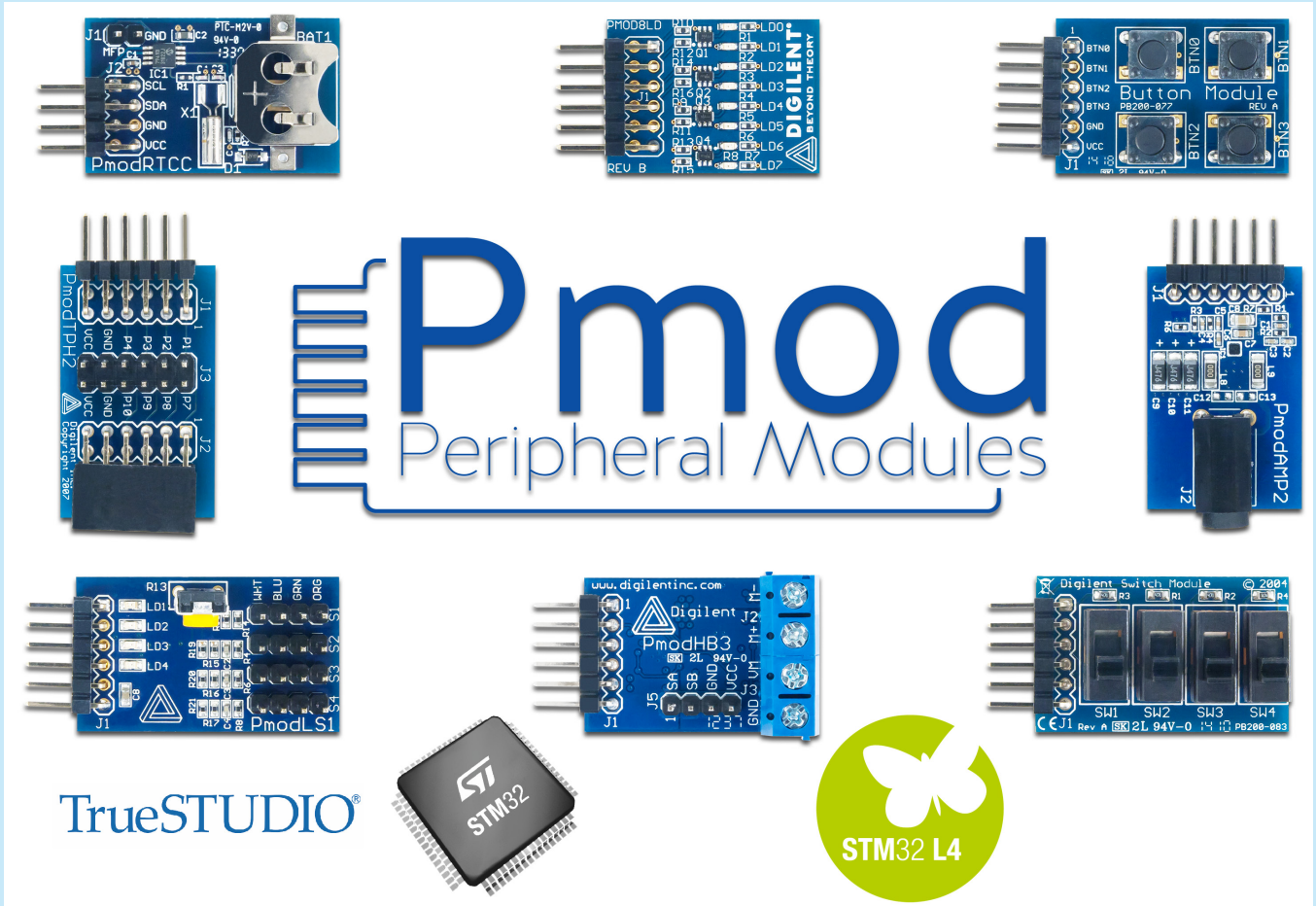


Digilent Pmod i STM32 (6)

Szósty odcinek cyklu poświęconego modułom Digilent Pmod obejmuje trzy kolejne układy: akcelerometr PmodACL2, potencjometr cyfrowy PmodDPOT i podwójny wyświetlacz 7-segmentowy PmodSSD. Przykłady dla wymienionych modułów zostały przygotowane dla środowiska Atollic TrueSTUDIO i zestawu uruchomieniowego KAmeLeon (*www.kameleonboard.org*) z wykorzystaniem biblioteki STM32Cube_FW_L4.

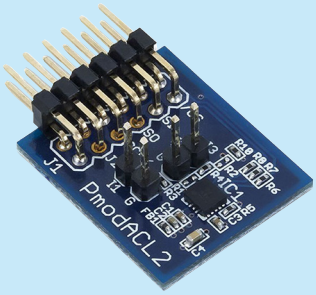
PmodACL2

Moduł PmodACL2 (**fotografia 1**) zawiera 3-osiowy akcelerometr MEMS – ADXL362, od firmy Analog Devices. Akcelerometr zapewnia 12-bitową rozdzielczość pomiaru oraz trzy możliwe do ustawienia zakresy: ±2 g, ±4 g, ±8 g, charakteryzujące się odpowiednio czułością: 1 mg/LSB, 2 mg/LSB, 4 mg/LSB. Układ ma również konfigurowalną częstotliwość odczytu danych, która może przyjąć jedną z wartości: 12,5 Hz, 25 Hz, 50 Hz, 100 Hz, 200 Hz, 400 Hz. Akcelerometr charakteryzuje się bardzo małym poborem prądu o maksymalnej wartości dochodzącej do 5 µA przy napięciu zasilania wynoszącym 3,5 V i maksymalnej prędkości odczytu danych. Poza pomiarem przyspieszenia, układ ADXL362 umożliwia także pomiar temperatury z rozdzielczością 12 bitów i czułością 0,065°C/LSB. Układ ADXL362 ma też szereg funkcji do przetwarzania danych pomiarowych, takich jak: detekcja aktywności, detekcja swobodnego spadku, kolejka FIFO dla danych pomiarowych.

Na szczególną uwagę zasługuje możliwość wykorzystania kolejki FIFO do zapisu historii pomiarów prowadzących do wykrycia aktywności definiowanej konfigurowalnymi progami dla każdej z osi. Umożliwia to mikrokontrolerowi odczyt całego profilu przyspieszenia po otrzymaniu przerwania, którego źródłem jest zdarzenie wykryte przez układ ADXL362.

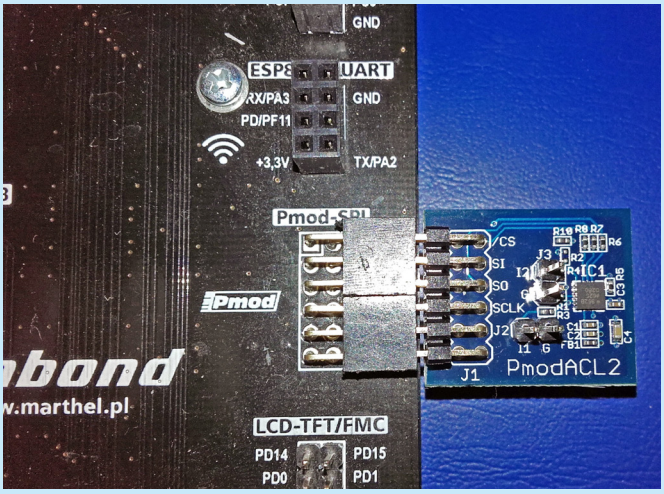
Akcelerometr udostępnia kilka źródeł przerwań, które mogą zostać przyporządkowane niezależnie do jednego z dwóch sygnałów INT1, lub INT2:

- AWAKE – wybudzenie przez wykrycie aktywności, lub uśpienie przez wykrycie braku aktywności,
- INACT – wykrycie braku aktywności, lub swobodnego spadku,
- ACT – wykrycie aktywności,
- FIFO_OVERRUN – przepełnienie kolejki FIFO,



Fotografia 1. Wygląd modułu PmodACL2

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



Fotografia 2. Moduł PmodACL2 podłączony do zestawu KAmeLeon

- FIFO_WATERMARK – zapełnienie kolejki do ustalonego poziomu,
- FIFO_READY – gotowość do odczytu co najmniej jednej wartości z kolejki FIFO,
- DATA_READY – gotowość do odbioru nowej zmierzonej wartości.

Zdarzenia dotyczące wykrywania aktywności lub jej braku wymagają zdefiniowania progu przyspieszenia oraz minimalnego czasu jego przekroczenia.

Z modułem PmodACL2 można komunikować się za pośrednictwem interfejsu SPI i trzech dostępnych komend:

- Write register (0x0A) – zapis rejestrów,
- Read register (0x0B) – odczyt rejestrów,
- Read FIFO (0x0D) – odczyt kolejki FIFO.

Komendom do obsługi rejestrów musi zawsze towarzyszyć 8-bitowy adres rejestru, a następnie co najmniej jeden bajt danych. W przypadku wielu bajtów danych, adres rejestru jest automatycznie inkrementowany wewnątrz układu ADXL362. Komenda odczytu FIFO nie wymaga dodatkowych parametrów – po jej wysłaniu układ automatycznie odpowiada danymi znajdującymi się w kolejce.

Moduł PmodACL2 posiada złącze SPI typu 2A, z dwoma sygnałami przerwań. Złącze to może być podłączone do gniazda Pmod-SPI zestawu KAmeLeon, tak jak zostało to przedstawione na **fotografii 2**. Piny mikrokontrolera podłączone do odpowiednich sygnałów modułu zostały przedstawione w **tabeli 1**.

Obsługa modułu PmodACL2 w prezentowanym przykładzie znajduje się w plikach inc/PmodACL2.h i src/PmodACL2.c i składa się z czterech funkcji:

- PmodACL2_Config – konfiguracja peryferiów mikrokontrolera i rejestrów akcelerometru,
- PmodACL2_ReadFifo – odczyt danych z kolejki FIFO,

Tabela 1. Sygnały PmodACL2 oraz odpowiadające im piny mikrokontrolera; w tabeli pominięto sygnały niepołączone (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

- PmodACL2_ReadStatus – odczyt aktualnego stanu źródeł przerwań,
- PmodACL2_ConvertFifoEntry – konwersja wartości odczytanej z kolejki na liczbę całkowitą ze znakiem.

Pierwsza z funkcji została przedstawiona na **listingu 1**. W pierwszej kolejności konfigurowany jest interfejs SPI w trybie 0 (CPOL=0, CPHA=0) i z programową kontrolą sygnału CS. Długość danych jest ustawiona na 8-bitów. W pliku PmodACL2.c została także zdefiniowana funkcja HAL_SPI_MspInit, wywoływana wewnątrz funkcji bibliotecznej HAL_SPI_Init i odpowiedzialna za konfigurację GPIO dla interfejsu SPI (MISO, MOSI, SCLK, CS) i włączenie odpowiednich sygnałów zegarowych.

Kolejnym zadaniem funkcji PmodACL2_Config jest konfiguracja rejestrów akcelerometru ADXL362. W pierwszej kolejności wykonywany jest reset układu w wyniku zapisu wartości 0x52 do rejestru SOFT_RESET, po którym należy odczekać co najmniej 0,5 ms. Reset ma na celu przywrócenie wszystkich rejestrów do ich domyślnych wartości. Są to m. in. zakres i częstotliwość odczytu danych

Listing 1. Konfiguracja modułu PmodACL2

```
void PmodACL2_Config(void)
{
  // Configure the SPI connected to the Pmod module.
  pmodAcl2Spi.Instance = SPI1;
  pmodAcl2Spi.Init.Mode = SPI_MODE_MASTER;
  pmodAcl2Spi.Init.Direction = SPI_DIRECTION_2LINES;
  pmodAcl2Spi.Init.DataSize = SPI_DATASIZE_8BIT;
  pmodAcl2Spi.Init.CLKPolarity = SPI_POLARITY_LOW;
  pmodAcl2Spi.Init.CLKPhase = SPI_PHASE_1EDGE;
  pmodAcl2Spi.Init.NSS = SPI_NSS_SOFT;
  pmodAcl2Spi.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_128;
  pmodAcl2Spi.Init.FirstBit = SPI_FIRSTBIT_MSB;
  pmodAcl2Spi.Init.TIMode = SPI_TIMODE_DISABLE;
  pmodAcl2Spi.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  pmodAcl2Spi.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;

  HAL_SPI_Init(&pmodAcl2Spi);

  writeRegister(0x1F, 0x52); // Reset the ADXL362 (SOFT_RESET register)
  HAL_Delay(1); // Wait to make sure that the ADXL362 is ready after reset.

  uint16_t activityValueThreshold = 400;    // 400mg / Sensitivity (1mg/LSB) = 400
  uint16_t activityTimeThreshold = 5;       // 50ms * ODR (100Hz) = 5
  uint16_t inactivityValueThreshold = 100;  // 100mg / Sensitivity (1mg/LSB) = 100;
  uint16_t inactivityTimeThreshold = 50;    // 500ms * ODR (100Hz) = 50
  // Initialization of the ADXL362 registers.
  writeRegister(0x20, activityValueThreshold & 0xFF);           // Set the THRESH_ACT_L register
  writeRegister(0x21, (activityValueThreshold >> 8) & 0xFF);    // Set the THRESH_ACT_H register
  writeRegister(0x22, activityTimeThreshold);                   // Set the TIME_ACT register
  writeRegister(0x23, inactivityValueThreshold & 0xFF);         // Set the THRESH_INACT_L register
  writeRegister(0x24, (inactivityValueThreshold >> 8) & 0xFF);  // Set the THRESH_INACT_H register
  writeRegister(0x25, activityTimeThreshold & 0xFF);            // Set the TIME_INACT_L register
  writeRegister(0x26, (inactivityTimeThreshold >> 8) & 0xFF);   // Set the TIME_INACT_H register
  writeRegister(0x27, 0x3F); // Enable loop mode and activity/inactivity detection
                             // in referenced mode (ACT_INACT_CTL register)
  writeRegister(0x28, 0x02); // Enable the fifo in stream mode (FIFO_CONTROL register)
  writeRegister(0x2A, 0x90); // Set the INT1 active in low state, enable activity interrupt
                             // (INTMAP1 register)

  writeRegister(0x2D, 0x02); // Enable measurement mode (POWER_CTL 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);
}
```

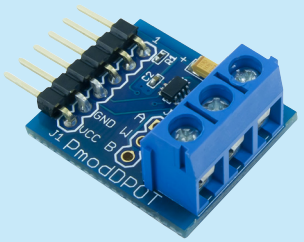
(rejestr 0x2C FILTER_CTL) wynoszące odpowiednio ±2 g i 100 Hz. Następnie ustawiane są progi detekcji aktywności. Ustawiane wartości odnoszą się do aktualnie ustawionej rozdzielczości – dla wartości 400 i rozdzielczości 1 mg/LSB, aktywność będzie rozpoznawana po przekroczeniu progu 400 mg. Oprócz wartości progu, wpisywany jest także minimalny czas jego przekroczenia. Czas ten podawany jest w próbkach i jest ściśle związany z ustawioną częstotliwością odczytu danych (ODR). Warto zwrócić uwagę, że wszystkie wartości, poza minimalnym czasem przekroczenia progu aktywności, czyli progi aktywności, braku aktywności i czas przekroczenia progu braku aktywności, są podzielone na dwa 8-bitowe rejestry (MSB i LSB), które należy ustawić niezależnie. Następnym rejestrem jest 0x27 ACT_INACT_CTL, który zawiera dalszą konfigurację zdarzeń związanych z aktywnością. W przykładzie, układ jest

B15	B14	B13	B12	B11	B10	B9	B8
Data Type: 00: X-Axis 01: Y-Axis 10: Z-Axis 11: Temp		Sign Extension		MSB	Data		

B7	B6	B5	B4	B3	B2	B1	B0
Data							LSB

Rysunek 3. Struktura elementów kolejki FIFO

konfigurowany w trybie *loop* (wykrywanie zdarzeń aktywności i jej braku jest włączane naprzemiennie, a przerwania są potwierdzanie automatycznie bez ingerencji mikrokontrolera) z pomiarem referencyjnym (względem przyspieszenia referencyjnego obliczanego przez układ) uwzględniającym położenie początkowe. Rejestr 0x28 FIFO_CONTROL to konfiguracja kolejki FIFO – włączenie w trybie *stream*, w którym kolejka zawiera zawsze najnowsze dane, a najstarsze są usuwane. Bit 2. ustawiony na 0 oznacza, że kolejka nie przechowuje informacji o temperaturze. Zapis do kolejnego rejestru – 0x2A INTMAP1, ustawia przerwania na linii INT1 aktywne w stanie niskim i włącza przerwanie od wykrycia aktywności. Ostatnia operacja zapisu – do rejestru 0x2D POWER_CTL, uruchamia pomiary.



Fotografia 4. Wygląd modułu PmodDPOT

Do realizacji zapisu, wykorzystywana jest funkcja pomocnicza - writeRegister, która kontroluje sygnał CS, wysyła komendę zapisu 0x0A, adres rejestru i wartość. Do obsługi interfejsu SPI używana jest biblioteczna funkcja HAL_SPI_Transmit.

Na koniec, funkcja PmodACL2_Config konfiguruje pin PE13, podłączony do linii INT1 modułu PmodACL2, jako wejście przerwań wyzwalanych zboczem opadającym.

Druga z funkcji obsługujących moduł PmodACL2 – PmodACL2_ReadFifo, przedstawiona na **listingu 2**, jest odpowiedzialna za odczyt danych znajdujących się w kolejce FIFO akcelerometru. W pierwszej kolejności funkcja odczytuje 10-bitową liczbę elementów kolejki z dwóch rejestrów: 0x0C FIFO_ENTRIES_L i 0x0D FIFO_ENTRIES_H. Dane z kolejki są zapisywane do wewnętrznego bufora o długości MAX_FIFO_READ_LEN, a następnie kopiowane pod adres podany w argumencie, wewnątrz pomocniczej funkcji readFifo. Funkcja ta steruje linią CS, wysyła komendę odczytu kolejki 0x0D, a następnie odczytuje dane z akcelerometru. Wszystko odbywa się w ramach jednej transakcji SPI, co zostało przedstawione na **listingu 3**.

Listing 2. Odczyt liczby elementów z kolejki FIFO
```
void PmodACL2_ReadFifo(uint8_t* data, uint32_t* len)
{
  // Read the number of entries currently stored in fifo.
  uint8_t entriesLsb = readRegister(0x0C); // FIFO_ENTRIES_L register.
  uint8_t entriesMsb = readRegister(0x0D); // FIFO_ENTRIES_H register.
  // Each fifo entry is 2 bytes long.
  *len = (entriesLsb | (entriesMsb << 8)) * 2;
  // Make sure that the number of bytes does not exceed the fifo limit.
  if(*len > MAX_FIFO_READ_LEN)
    *len = MAX_FIFO_READ_LEN;
  // Do not read from empty fifo.
  if(*len == 0)
    return;
  readFifo(data, *len);
}
```

Listing 3. Odczyt kolejki FIFO
```
static void readFifo(uint8_t* data, uint32_t len)
{
  // The local buffers used for SPI transaction. The additional byte is required
  // for transmitting the command.
  uint8_t txbuf[MAX_FIFO_READ_LEN + 1] = {0x00};
  uint8_t rxbuf[MAX_FIFO_READ_LEN + 1] = {0x00};
  txbuf[0] = SPI_READ_FIFO;

  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  HAL_SPI_TransmitReceive(&pmodAcl2Spi, txbuf, rxbuf, len + 1, 100);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
  if(data == NULL) return;
  // Copy the data to given buffer without the command byte.
  for(uint32_t i = 0; i < len; i++)
    data[i] = rxbuf[i + 1];
}
```

Następna z funkcji do obsługi akcelerometru realizuje odczyt rejestru stanu 0x0B STATUS. Wykorzystuje ona funkcję pomocniczą readRegister, która wysyła komendę odczytu rejestru – 0x0B, oraz adres, a następnie odczytuje wartość znajdującą się w rejestrze. Tak jak w przypadku innych funkcji pomocniczych, tutaj także wywoływane są funkcje z biblioteki STM32Cube: HAL_GPIO_WritePin do kontroli sygnału CS i HAL_SPI_TransmitReceive do transmisji danych po SPI. Obie funkcje – PmodACL2_ReadStatus i readRegister zostały przedstawione na **listingach 4** i **5**.

Listing 4. Odczyt rejestru stanu akcelerometru ADXL362
```
uint8_t PmodACL2_ReadStatus(void)
{
  return readRegister(0x0B);
}
```

Listing 5. Funkcja pomocnicza do odczytu rejestru
```
static uint8_t readRegister(uint8_t address)
{
  // Reading register requires sending read command and the address
  // before obtaining the data byte.
  uint8_t txbuf[3] = {SPI_READ_REG, address, 0x00};
  uint8_t rxbuf[3] = {0x00};

  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  HAL_SPI_TransmitReceive(&pmodAcl2Spi, txbuf, rxbuf, 3, 100);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
  return rxbuf[2];
}
```

Ostatnia z opisywanych funkcji – PmodACL2_ConvertFifoEntry, pełni pomocniczą rolę przy przetwarzaniu danych odczytanych z kolejki FIFO. Jej zadaniem jest konwersja pojedynczej wartości znajdującej się w dwóch bajtach odczytanych z kolejki na liczbę 16-bitową ze znakiem. Elementy FIFO zakodowane są zgodnie ze schematem przedstawionym na rysunku 3: dwa najstarsze bity (14-15) zawierają informację o przyporządkowaniu wartości do osi, lub o pomiarze temperatury, dwa kolejne bity (12-13) przechowują znak, a pozostałe – wartość zmierzoną. Z tego powodu, oprócz łączenia dwóch bajtów w jedną wartość 16-bitową, konieczne jest także rozszerzenie bitów znaku na bity 14. i 15. Operacja ta jest przedstawiona na **listingu 6**.

Pozostałe operacje związane z konwersją danych odczytanych z kolejki FIFO zostały zaimplementowane

w funkcji main i przedstawione na **listingu 7**. Po otrzymaniu przerwania sygnalizowanego flagą activityFlag, następuje odczytanie danych z kolejki FIFO za pomocą opisywanej wcześniej funkcji PmodACL2_ReadFifo. Następnie w pętli for, każde dwa bajty analizowane są pod kątem przynależności do jednej z osi (pomiary temperatury w przedstawionej konfiguracji nie są zapisywane do kolejki) i konwertowane za pomocą funkcji PmodACL2_ConvertFifoEntry.

Po zebraniu danych ze wszystkich trzech osi, co jest sygnalizowane za pomocą flag readX, readY, readZ, są one wysyłane za pośrednictwem portu szeregowego LPUART1, którego obsługa znajduje się w pliku src/serial.c.

Listing 6. Konwersja 2-bajtowego elemntu kolejki FIFO na wartość 16-bitową ze znakiem

```
int16_t PmodACL2_ConvertFifoEntry(uint8_t* data)
{
  int16_t convertedValue = data[0] | (data[1] << 8);

  if(convertedValue & 0x3000)
    return convertedValue | 0xC000;
  else
    return convertedValue & 0x3FFF;
}
```

Listing 7. Odczyt i konwersja danych z kolejki FIFO

```
while(activityFlag == 0);
PmodACL2_ReadFifo(fifoData, &fifoLen);
int sampleIndex = 1;
for(int i = 0; i < fifoLen; i += 2) {
  switch(fifoData[i+1] & 0xC0) {
    case 0x00:
      x = PmodACL2_ConvertFifoEntry(&fifoData[i]);
      readX = 1;
      break;
    case 0x40:
      y = PmodACL2_ConvertFifoEntry(&fifoData[i]);
      readY = 1;
      break;
    case 0x80:
      z = PmodACL2_ConvertFifoEntry(&fifoData[i]);
      readZ = 1;
      break;
  }
}
```

PmodDPOT

Drugim z opisywanych modułów jest PmodDPOT (**fotografia 4**) z potencjometrem cyfrowym AD5160 od Analog Devices. Potencjometr posiada 256 pozycji, umożliwiających podział całkowitej rezystancji widzianej pomiędzy portami A i B (R_{AB}) i wynoszącej 10 kΩ. Zależność pomiędzy ustawieniem ślizgacza, a rezystancją pomiędzy portami W i B (R WB) jest opisana wzorem:

$$R_{WB}(D) = \frac{D}{256} \cdot R_{AB} + R_W ,$$

w którym D jest konfigurowaną pozycją ślizgacza w zakresie od 0 do 255, natomiast R_W jest jego rezystancją wynoszącą 60 Ω. Analogicznie, przedstawia się zależność dla portów W i A:

$$R_{WA}(D) = \frac{256 - D}{256} \cdot R_{AB} + R_W .$$

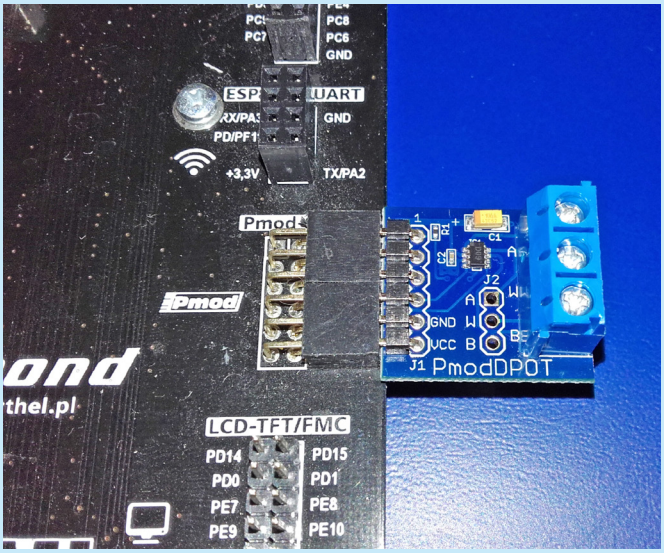
Do komunikacji z układem służy interfejs SPI, za pomocą którego można konfigurować pozycję ślizgacza. Odbywa się to przez wysłanie pojedynczego bajtu, zawierającego żądaną pozycję, która jest jednocześnie wartością D w przedstawionych wcześniej zależnościach.

Moduł ma 6-pinowe złącze SPI typu 2, które może zostać podłączone do gniazda Pmod-SPI zestawu KAmeLeon, jak pokazano na **fotografii 5**. Lista wyprowadzeń mikrokontrolera połączonych z pinami modułu PmodDPOT została przedstawiona w **tabeli 2**. Komunikacja odbywa się tylko w jedną stronę: od mikrokontrolera do modułu.

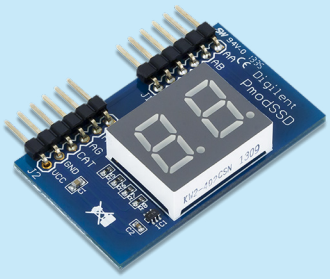
Kod do obsługi modułu PmodDPOT w prezentowanym przykładzie znajduje się w plikach src/PmodDPOT.c i inc/PmodDPOT.h. Konfiguracja interfejsu SPI, znajdująca się w funkcji PmodDPOT_Config, wygląda analogicznie jak dla opisywanego wcześniej modułu PmodACL2 – tryb 0 z programową kontrolą sygnału ~CS. Piny GPIO są jak zwykle konfigurowane przez funkcję HAL_SPI_MspInit wywoływaną wewnątrz funkcji bibliotecznej HAL_SPI_Init. Ostatnia z funkcji – PmodDPOT_SetValue, jest odpowiedzialna za zmianę ustawienia ślizgacza. Została ona przedstawiona na **listingu 8**.

Listing 8. Funkcja PmodDPOT_SetValue

```
void PmodDPOT_SetValue(uint8_t value)
{
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  HAL_SPI_Transmit(&pmodDpotSpi, (uint8_t*)&value, 1, 100);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
}
```



Fotografia 5. Moduł PmodDPOT podłączony do zestawu KAmeLeon



Fotografia 6. Wygląd modułu PmodSSD

Główna pętla programu znajdująca się w funkcji main, wykonuje co 100 ms zmianę ustawienia ślizgacza w zakresie od 0 do 255, co można obserwować przez pomiar rezystancji R_{WA}, lub R_{WB}.

PmodSSD

Moduł PmodSSD (**fotografia 6**) jest wyposażony w dwucyfrowy wyświetlacz siedmiosegmentowy. Obie cyfry wyświetlacza mają wspólną katodę, a wybór aktywnej odbywa się za pośrednictwem jednego z sygnałów. PmodSSD posiada dwa sześciopinowe złącza GPIO typu 1. Z tego względu w przykładzie moduł został podłączony do złącza Arduino zestawu KAmeLeon zgodnie z **tabelą 3** (**fotografia 7**).

Za obsługę wyświetlacza odpowiedzialny jest kod znajdujący się w plikach inc/PmodSSD.h i src/PmodSSD.c. Na **listingu 9** znajdują się definicje pinów i portów GPIO oraz konfiguracja segmentów

Tabela 2. Sygnały PmodDPOT oraz odpowiadające im piny mikrokontrolera; w tabeli pominięto sygnały niepołączone (NC) i linie zasilania występujące na złączu Pmod

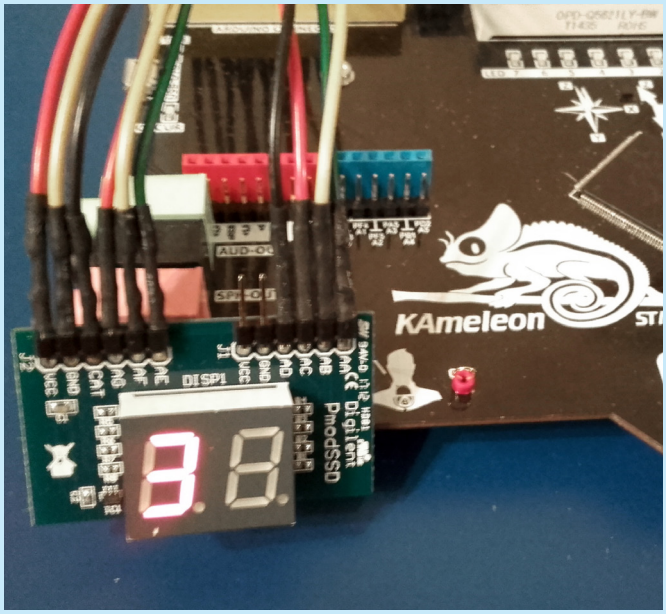
Sygnał	Numer pinu PmodACL (J1)	Pin STM32L496ZG (KAmeLeon Pmod-SPI)
~CS	1	PB0
MOSI	2	PA7
SCLK	4	PA1

Listing 9. Definicje pinów i portów GPIO oraz masek bitowych dla cyfr 0 – 9

```
uint16_t segmentPins[SEGMENTS_COUNT] = {GPIO_PIN_10, GPIO_PIN_11, GPIO_PIN_11, GPIO_PIN_13, GPIO_PIN_12, GPIO_PIN_15, GPIO_PIN_14};

GPIO_TypeDef* segmentPorts[SEGMENTS_COUNT] = {GPIOD, GPIOB, GPIOD, GPIOB, GPIOB, GPIOB, GPIOB};

uint8_t segmentsConfig[10] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F};
```



Fotografia 7. Moduł PmodSSD podłączony do złącza Arduino zestawu KAmeleon

Tabela 3. Sygnały modułu PmodSSD podłączone do złącza ARDUINO CONNECTOR zestawu KAmeLeon

Sygnał	Numer pinu PmodSSD	Numer pinu KAmeLeon ARDUINO CONNECTOR	Pin mikrokontrolera
AA	1 (J1)	D6	PD10
AB	2 (J1)	D7	PB11
AC	3 (J1)	D8	PD11
AD	4 (J1)	D9	PB13
GND	5 (J1)	GND	-
VCC	6 (J1)	+3,3	-
AE	1 (J2)	D10	PB12
AF	2 (J2)	D11	PB15
AG	3 (J2)	D12	PB14
C	4 (J2)	D13	PB10
GND	5 (J2)	-	-
VCC	6 (J2)	-	-

dla cyfr od 0 do 9. Cyfry są zakodowane jako maski bitowe, w których jedynka oznacza, że segment jest zapalony. Segment A w każdej masce jest zakodowany na zerowym bicie, a segment G na szóstym.

Za konfigurację wyprowadzeń mikrokontrolera podłączonych do wyświetlacza odpowiedzialna jest funkcja PmodSSD_Config włączająca niezbędne sygnały zegarowe i ustawiająca piny jako wyjścia w stanie niskim. Oprócz pinów obecnych w powyższych definicjach, funkcja konfiguruje także pin PB10, który służy do wybierania aktywnej cyfry wyświetlacza. Funkcja PmodSSD_Config została przedstawiona na **listingu 10**.

Druga z funkcji znajdujących się w pliku src/PmodSSD.c – PmodSSD_DisplayNumber, służy do wyświetlania wybranej cyfry. Została ona przedstawiona na **listingu 11**. Funkcja ustawia stan linii C, wybierając w ten sposób jedną z cyfr, a następnie ustawia pozostałe linie GPIO zgodnie z wybraną maską bitową. Aktywna cyfra wybierana jest za pomocą argumentu typu DisplayId, zdefiniowanego w pliku inc/PmodSSD.h.

Pętla główna programu w funkcji main wyświetla naprzemiennie cyfry na wyświetlaczu w kolejności rosnącej, po prawej stronie i malejącej, po lewej stronie wyświetlacza.

Krzysztof Chojnowski

Listing 10. Konfiguracja wyprowadzeń podłączonych do modułu PmodSSD

```
void PmodSSD_Config(void)
{
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  GPIO_InitTypeDef GPIO_InitStruct;
  GPIO_InitStruct.Mode  = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull  = GPIO_PULLDOWN;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;

  for(int i=0; i<SEGMENTS_COUNT; i++) {
    GPIO_InitStruct.Pin = segmentPins[i];
    HAL_GPIO_Init(segmentPorts[i], &GPIO_InitStruct);
    HAL_GPIO_WritePin(segmentPorts[i], segmentPins[i], GPIO_PIN_RESET);
  }
  GPIO_InitStruct.Pin = GPIO_PIN_10;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET);
}
```

Listing 11. Wyświetlanie wybranej cyfry na wyświetlaczu

```
void PmodSSD_DisplayNumber(DisplayId id, uint8_t number)
{

  if (number > 9)
    return;
  if(id == DisplayId_0)
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_RESET);
  else
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_10, GPIO_PIN_SET);

  for(int i=0; i<SEGMENTS_COUNT; i++) {
    if((segmentsConfig[number] >> i) & 0x01)
      HAL_GPIO_WritePin(segmentPorts[i], segmentPins[i], GPIO_PIN_SET);
    else
      HAL_GPIO_WritePin(segmentPorts[i], segmentPins[i], GPIO_PIN_RESET);
  }
}
```


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