Digilent Pmod i STM32 (7)

W siódmej części cyklu poświęconego modułom peryferyjnym Pmod firmy Digilent, zostaną omówione kolejne trzy moduły. Będą to: PmodBT2 – umożliwiający komunikację Bluetooth w standardzie 2.1, 2.0, 1.2 lub 1.0, PmodTC1 – zawierający termoparę do pomiaru temperatury i PmodCLP z wyświetlaczem znakowym.

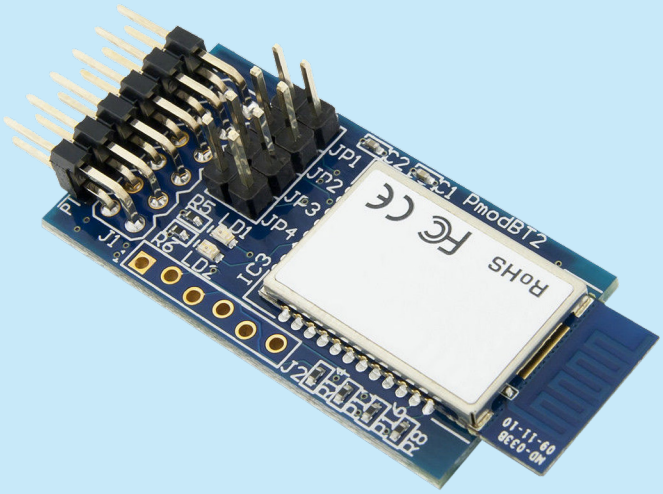
Przykłady dla wymienionych modułów zostały przygotowane dla środowiska Atollic TrueSTUDIO i zestawu uruchomieniowego KAmeLeon (*www.kameleonboard.org*) z wykorzystaniem biblioteki STM32Cube_FW_L4.

PmodBT2

Moduł PmodBT2 (**fotografia 1**) z układem RN42 dostępnym w ofercie firmy Microchip, pozwala na komunikację za pośrednictwem interfejsu bezprzewodowego Bluetooth. Układ wspiera wersje 2.1, 2.0, 1.2 i 1.0 standardu i jest urządzeniem klasy drugiej o mocy wyjściowej 2,5 mW zapewniającym zasięg komunikacji do 10 m. Układ RN42 ma wbudowany stos Bluetooth implementujący profile zależnie od modelu urządzenia. Znajdujący się w module PmodBT2, układ RN42-I/RM implementuje profil SPP (*Serial Port Profile*) emulujący port szeregowy podczas połączenia dwóch urządzeń.

Komunikacja z modułem odbywa się za pośrednictwem interfejsu UART z dodatkowymi sygnałami RTS i CTS odpowiedzialnymi za kontrolę przepływu danych. Za pośrednictwem tego interfejsu można przesyłać dane, które bezpośrednio trafiają na łącze radiowe i do urządzenia połączonego z układem. Moduł jest domyślnie skonfigurowany do połączenia zgodnie z następującą konfiguracją:

- 115200 baud,
- 8 bitów danych,
- brak parzystości,
- 1 bit stopu,
- sprzętowa kontrola przepływu (RTS/CTS).



Fotografia 1. Moduł PmodBT2

Moduł może także zostać wprowadzony w tryb komend, w którym można odczytać oraz zmienić jego konfigurację. Aby wejść do tego trybu należy wysłać sekwencję znaków: „$$$”, na którą układ RN42 powinien odpowiedzieć przez wysłanie ”CMD\r\n”. Każda komenda składa się z jednego lub dwóch znaków i opcjonalnych argumentów oddzielonych przecinkiem. Na końcu komendy należy wysłać znak ‘\r’, po którym można spodziewać się odpowiedzi od modułu. W przykładzie została użyta tylko jedna z komend, służąca do odczytu wersji firmware’u: „V\r”, na którą moduł odpowiada, wysyłając odpowiedni napis z wersją oprogramowania: „Ver 6.15 04/26/2013\n\r(c) Roving Networks\r\n”. Wszystkie wspierane komendy można znaleźć w dokumentacji: *http://bit.ly/2LtppPY*. Aby zakończyć tryb wprowadzania komend należy wysłać „---\r” (trzy minusy i powrót karetki), na co układ powinien odpowiedzieć przesyłając „END\n\r” i przechodząc z powrotem do trybu transmisji danych.

Oprócz sygnałów odpowiedzialnych za komunikację za pośrednictwem interfejsu UART, PmodBT2 udostępnia także kilka linii GPIO do kontrolowania oraz monitorowania pracy układu RN42. Sygnały te zostały przedstawione w **tabeli 1**.

Sygnały podłączone do złączy JP1, JP2, JP3 i JP4 można ustawić w stan wysoki za pomocą zworek – domyślnie są one wszystkie w stanie niskim. PmodBT2 ma także złącze J2, na którym znajdują się sygnały interfejsu SPI, przez który można aktualizować oprogramowanie układu RN42.

Moduł PmodBT2 ma złącze Pmod dla interfejsu UART, typu 4A. W miejsce sygnału INT został podłączony sygnał STATUS (PIO2), informujący o stanie połączenia. W przykładzie, moduł został podłączony do złącza oznaczonego jako ARDUINO CONNECTOR na płytce KAmeLeon według **tabeli 2**.

Kod do obsługi opisywanego modułu został umieszczony w plikach: źródłowym src/PmodBT2.c oraz nagłówkowym inc/

Tabela 1. Sygnały PIO dostępne w module PmodBT2

Sygnał	Złącze	Opis
~RST	J1(8)	Reset, aktywny w stanie niskim
PIO2 (STATUS)	J1(7)	Stan połączenia, stan wysoki oznacza aktywne połączenie
PIO3	JP2	Automatyczne wykrywanie i parowanie; w stanie wysokim oznacza automatyczne poszukiwanie drugiego urządzenia o klasie 0x55AA (CoD, Class of Device) i zapisanie jego adresu. W stanie niskim moduł oczekuje na wykrycie przez inne urządzenie.
PIO4	JP1	Przywrócenie ustawień fabrycznych – pin powinien mieć stan wysoki przy włączeniu zasilania i dwukrotnie zmienić stan z sekundową przerwą.
PIO6	JP3	Automatyczne łączenie; w stanie wysokim moduł łączy się automatycznie z wcześniej znalezionym i sparowanym urządzeniem.
PIO7	JP4	Prędkość transmisji interfejsu UART, w stanie wysokim 9600, w stanie niskim według konfiguracji (domyślnie 115200).

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

Listing 1. Konfiguracja GPIO i UART dla modułu PmodBT2

```
void PmodBT2_Config(void)
{
    __HAL_RCC_GPIOB_CLK_ENABLE();
    GPIO_InitTypeDef GPIO_InitStruct;
    GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING_FALLING;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Pin = GPIO_PIN_12;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
    HAL_NVIC_SetPriority(EXTI15_10_IRQn, 2, 0);
    HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
    GPIO_InitStruct.Mode  = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull  = GPIO_PULLUP;
    GPIO_InitStruct.Pin = GPIO_PIN_15;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_RESET);
    HAL_Delay(500);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_SET);
    HAL_Delay(500);
    pmodBt2Uart.Instance = USART3;
    pmodBt2Uart.Init.BaudRate = 115200;
    pmodBt2Uart.Init.WordLength = UART_WORDLENGTH_8B;
    pmodBt2Uart.Init.StopBits = UART_STOPBITS_1;
    pmodBt2Uart.Init.Parity = UART_PARITY_NONE;
    pmodBt2Uart.Init.Mode = UART_MODE_TX_RX;
    pmodBt2Uart.Init.HwFlowCtl = UART_HWCONTROL_RTS_CTS;
    pmodBt2Uart.Init.OverSampling = UART_OVERSAMPLING_16;
    pmodBt2Uart.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
    HAL_UART_Init(&pmodBt2Uart);
}
```

Tabela 2. Sposób podłączenia modułu PmodBT2 do płytki KAmeLeon

Sygnał	Numer pinu PmodBT2	Numer pinu KAmeLeon ARDUINO CONNECTOR	Pin mikrokontrolera
RTS	1	D9	PB13
RX	2	D13	PB10
TX	3	D7	PB11
CTS	4	D12	PB14
GND	5	GND	–
VCC	6	+3,3	–
STATUS	7	D10	PB12
~RST	8	D11	PB15
NC	9	–	–
NC	10	–	–
GND	11	–	–
VCC	12	–	–

PmodBT2.h. Funkcja PmodBT2_Config konfiguruje piny dla sygnałów ~RST i STATUS. Pin PB12 jest ustawiony jako źródło przerwania na obu zboczach, tak aby można było zaobserwować zmiany stanu połączenia. Następnie układ RN42 jest resetowany pinem PB15 i konfigurowany jest interfejs UART3 zgodnie z domyślną konfiguracją modułu. Procedura konfiguracji została przedstawiona na **listingu 1**.

Listing 2. Konfiguracja pinów dla interfejsu USART3

```
void PmodBT2_HAL_UART_MspInit(UART_HandleTypeDef *huart)
{
    __HAL_RCC_USART3_CLK_ENABLE();
    __HAL_RCC_GPIOB_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_USART3;
    GPIO_InitStruct.Pin = GPIO_PIN_10 | GPIO_PIN_11 | GPIO_PIN_13 | GPIO_PIN_14;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
    HAL_NVIC_SetPriority(USART3_IRQn, 1, 0);
    HAL_NVIC_EnableIRQ(USART3_IRQn);
}
```

Listing 3. Wybór konfiguracji jednego z dwóch interfejsów

```
void HAL_UART_MspInit(UART_HandleTypeDef *huart)
{
    if(huart->Instance == USART3) PmodBT2_HAL_UART_MspInit(huart);
    else if(huart->Instance == LPUART1) Serial_HAL_UART_MspInit(huart);
}
```

Listing 4. Odczyt wersji oprogramowania układu RN42 w trybie komend

```
bool PmodBT2_ReadVersion(char* data, int* len)
{
    char cmdResponse[5];
    HAL_UART_Transmit(&pmodBt2Uart, (uint8_t*)"$$$", 3, 100);
    HAL_UART_Receive(&pmodBt2Uart, (uint8_t*)cmdResponse, 5, 100);
    if(strncmp(cmdResponse, „CMD\r\n", 5) != 0) return false;
    HAL_UART_Transmit(&pmodBt2Uart, (uint8_t*)"V\r", 2, 100);
    HAL_UART_Receive(&pmodBt2Uart, (uint8_t*)data, 100, 100);
    HAL_UART_Transmit(&pmodBt2Uart, (uint8_t*)"---\r", 4, 100);
    HAL_UART_Receive(&pmodBt2Uart, (uint8_t*)cmdResponse, 5, 100);
    if(strncmp(cmdResponse, „END\r\n", 5) != 0) return false;
    *len = 0;
    while(data[(*len)++] != '\n');
    return true;
}
```

Listing 5. Obsługa przerwań interfejsu UART

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if(textBuffer[bufferIndex] == '\n') PmodBT2_Write(echoText, echoTextLen);
    else
    {
        bufferIndex++;
        PmodBT2_Read(&textBuffer[bufferIndex], 1);
    }
}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
    if(bufferIndex == 0)
    {
        PmodBT2_Read(textBuffer, 1);
        return;
    }
    PmodBT2_Write(textBuffer, bufferIndex + 1);
    bufferIndex = 0;
}
```

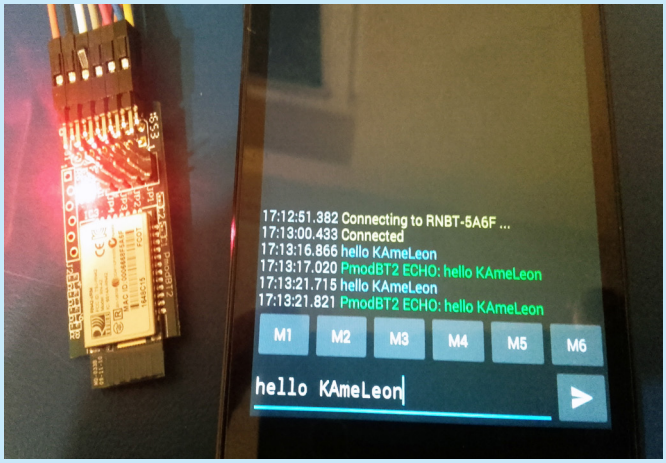
Wszystkie cztery piny podłączone do interfejsu USART3 są konfigurowane w funkcji PmodBT2_HAL_UART_MspInit, która podobnie jak w przypadku innych modułów wykorzystujących ten interfejs, jest wywoływana w funkcji HAL_UART_MspInit w pliku main.c. Decyduje ona o tym, który interfejs powinien zostać skonfigurowany i wywołuje konfigurację USART3 dla PmodBT2, lub LPUART1 dla modułu Serial. Wspomniane funkcje znajdują się na **listingach 2** i **3**.

Kolejna z funkcji znajdujących się w pliku PmodBT2.c – PmodBT2_ReadVersion, stanowi przykład użycia trybu komend układu RN42. Funkcja przełącza tryb, odczytuje wersję oprogramowania i przywraca tryb danych. Po każdym kroku sprawdzana jest poprawność otrzymanej odpowiedzi. Dane o wersji wpisywane są do przekazanego w argumencie bufora, a długość pierwszej linii odpowiedzi układu wpisywana jest do odpowiedniego wskaźnika. Cała funkcja znajduje się na **listingu 4**.

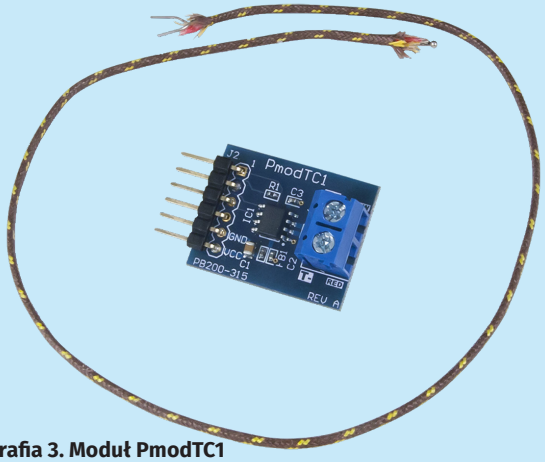
Ostatnie dwie funkcje: PmodBT2_Write i PmodBT2_Read są jedynie wrapperami na funkcje biblioteczne inicjalizujące odpowiednio zapis i odczyt z wykorzystaniem przerwań.

W przykładzie zaimplementowane zostały dwie funkcje obsługujące przerwania od interfejsu UART, wykonywane po zakończeniu odbioru znaku i wysłaniu bufora. Pierwsza z nich zapisuje dane do bufora i jeżeli odebrany znak jest nową linią ('\n'), to inicjalizuje transmisję odebranych danych poprzedzonych napisem „PmodBT2 ECHO: ". Kod obsługujący koniec transmisji sprawdza, czy w buforze znajdują się dane do wysłania. Jeżeli tak, to rozpoczynany jest kolejny zapis, w przeciwnym razie następnie odczyt kolejnej porcji danych. Opisane funkcje obsługi przerwań przedstawione zostały na **listingu 5**.

Główna funkcja programu – main, zajmuje się konfiguracją systemu oraz modułu PmodBT2. Następnie odczytywana jest wersja oprogramowania,



Fotografia 2. Efekt działania programu



Fotografia 3. Moduł PmodTC1

a odpowiedź układu wysyłana jest na interfejs LPUART1 dostępny jako wirtualny port szeregowy po podłączeniu KAmeLeona do komputera. Na koniec program przechodzi do nasłuchu nadchodzących danych, które automatycznie odsyłane są do nadawcy. Dodatkowo stan połączenia sygnalizowany jest za pomocą diody LED1 podłączonej do pinu PC7. Efekt działania programu można zaobserwować na **fotografii 2**. Do celów demonstracji została użyta aplikacja Serial Bluetooth Terminal dla systemu Android.

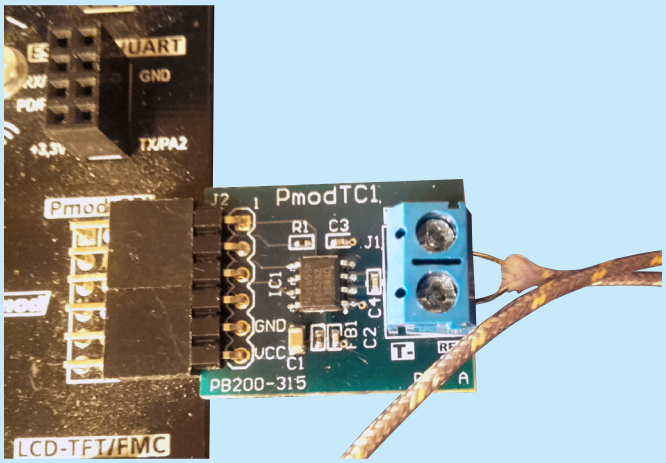
PmodTC1

PmodTC1 (**fotografia 3**) jest modułem do pomiaru temperatury w zakresie od –73°C do 482°C i z dokładnością ±2°C za pomocą termopary. Składa się on z dwóch elementów – termopary typu K oraz układu MAX31855, będącego przetwornikiem z kompensacją temperatury zimnego złącza. Jest to konieczne ze względu na konieczność utrzymywania tzw. zimnych końców termopary w ściśle określonej temperaturze, najczęściej 0°C. Zastosowanie kompensacji usuwa to ograniczenie przez pomiar temperatury otoczenia, która jest jednocześnie temperaturą zimnego złącza.

Moduł PmodTC1 ma dwa złącza – J1 do podłączenia termopary oraz J2 z interfejsem SPI. Przy podłączaniu termopary należy zwrócić uwagę na oznaczenia poszczególnych przewodów – kolor oplotu każdego z nich (żółty i czerwony) odpowiadają oznaczeniom obu terminali złącza (T+ YLW, T– RED). Drugie ze złączy odpowiada standardowemu złączu Pmod SPI typu 2, bez sygnału MOSI, ponieważ komunikacja odbywa się tylko w jednym kierunku. Moduł może być podłączony do złącza Pmod-SPI zestawu KAmeLeon, w którym schemat połączenia z pinami mikrokontrolera wygląda tak, jak zostało to przedstawione w **tabeli 3**. Podłączony moduł jest widoczny na **fotografii 4**.

Tabela 3. Podłączenie modułu PmodTC1 do złącza Pmod-SPI zestawu KAmeLeon; w tabeli pominięto sygnały niepołączone (NC) oraz linie zasilania

Sygnał	Numer pinu złącza Pmod	Pin mikrokontrolera
CS	1	PB0
MISO	3	PE14
SCK	4	PA1



Fotografia 4. Moduł PmodTC1 podłączony do zestawu KAmeLeon

	14-BIT THERMOCOUPLE TEMPERATURE DATA				RES	FAULT BIT	12-BIT INTERNAL TEMPERATURE DATA				RES	SCV BIT	SCG BIT	OC BIT
BIT	D31	D30	...	D18	D17	D16	D15	D14	...	D4	D3	D2	D1	D0
VALUE	Sign	MSB 2^{10} (1024°C)	...	LSB 2^{-2} (0.25°C)	Reserved	1 = Fault	Sign	MSB 2^{6} (64°C)	...	LSB 2^{-4} (0.0625°C)	Reserved	1 = Short to V_{CC}	1 = Short to GND	1 = Open Circuit

Rysunek 5. Kodowanie danych wysyłanych przez moduł PmodTC1

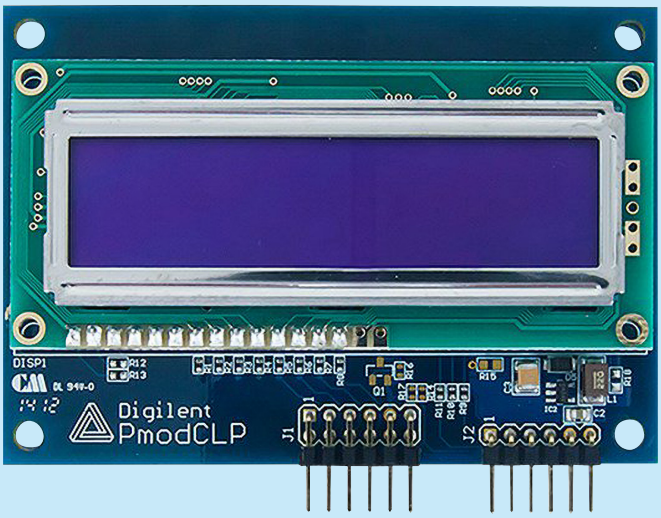
Tabela 4. Znaczenie danych przesyłanych przez moduł PmodTC1

Numer bitu	Opis
31–18	Zmierzona wartość temperatury w postaci 14-bitowej liczby stałoprzecinkowej ze znakiem; rozdzielczość wynosi 0,25°C/LSB.
16	Wartość 1 oznacza, że wystąpił jeden z błędów zakodowanych na bitach 2–0.
15–4	Temperatura zimnego złącza, mierzona wewnątrz układu MAX31855; wartość zmierzona ma postać 12-bitowej liczby stałoprzecinkowej ze znakiem, o rozdzielczości 0,0625°C/LSB.
2	Wartość 1 oznacza błąd zwarcia termopary do napięcia zasilania.
1	Wartość 1 oznacza błąd zwarcia termopary do masy.
0	Wartość 1 oznacza błąd niepodłączenia termopary.

Układ MAX31855 nie wymaga konfiguracji, a odczyt danych odbywa się za pośrednictwem interfejsu SPI. Dane zwracane przez moduł są zakodowane na 32 bitach, tak ja na **rysunku 5**. Poszczególne pola bitowe zostały wyjaśnione w **tabeli 4**.

Konwersja temperatury nie wymaga wyzwalania, ponieważ jest wykonywana przez cały czas. Czas trwania pojedynczej konwersji to 100 ms. Wartości zmierzone przechowywane w pamięci, a także flagi błędów są aktualizowane tylko wtedy kiedy sygnał CS jest w stanie wysokim, czyli nie ma aktywnej transmisji danych. Należy także pamiętać, że układ jest gotowy do pracy dopiero po 200 ms od włączenia zasilania.

Kod do konfiguracji modułu PmodTC1 znajduje się w funkcji PmodTC1_Config w pliku src/PmodTC1.c. Funkcja ta konfiguruje



Fotografia 6. Wyświetlacz znakowy modułu PmodCLP

interfejs SPI1 w trybie 1 (CPOL = 0, CPHA = 1), z programową kontrolą sygnału CS i danymi o długości 16 bitów. Dodatkowo SPI jest konfigurowany w trybie tylko do odczytu ze względu na brakujący sygnał MOSI. Opisana konfiguracja została przedstawiona na **listingu 6**. Inicjalizacja GPIO znajduje się tak jak zwykle w funkcji HAL_SPI_MspInit, która jest wołana przez bibliotekę STM32Cube wewnątrz funkcji HAL_SPI_Init.

Za odczyt danych odpowiedzialna jest funkcja PmodTC1_GetValues, która odczytuje 32-bitowe dane w postaci dwóch słów 16-bitowych i konwertuje je na dwie wartości zmiennoprzecinkowe. W każdym z dwóch słów sprawdzany jest bit znaku znajdujący się na najstarszym bicie. Jego wartość jest zapamiętywana i uwzględniana po konwersji liczby stałoprzecinkowej na zmiennoprzecinkową. Należy pamiętać o tym, że każda z wartości jest przesunięta ze względu na bity przechowujące kody błędów i bity nieużywane. Procedura konwersji została pokazana na **listingu 7**.

W głównej pętli programu dane z modułu są odczytywane co 500 ms, a zmierzone wartości temperatury i kody błędów są wysyłane na port szeregowy LPUART1, za pośrednictwem funkcji Serial_Write.

Tabela 5. Sygnały sterujące modułem PmodCLP

Sygnał	Opis
DB0-DB7	8-bitowa szyna danych i instrukcji
RS	Tryb instrukcji (stan niski) i danych (stan wysoki)
R/W	Tryb zapisu (stan niski) i odczytu (stan wysoki)
E	Wyzwalanie zapisu (na zboczu opadającym), lub tryb odczytu (w stanie wysokim)

PmodCLP

Ostatni z opisywanych w tej części cyklu – PmodCLP (**fotografia 6**), ma wyświetlacz znakowy LCD zawierający 2 linie po 16 znaków. W roli kontrolera wyświetlacza użyto układu KS0066 firmy Samsung obsługującego wyświetlacze jedno- i dwuliniowe. Komunikacja z nim odbywa się za pomocą interfejsu równoległego, przez który są wysyłane zarówno komendy jak i dane do wyświetlenia.

Układ KS0066 ma trzy typy wewnętrznej pamięci:

- CGROM – zawiera predefiniowany zestaw 192 znaków o rozmiarze 5×8 pikseli,
- CGRAM – umożliwiająca zdefiniowanie do 8 znaków użytkownika,
- DDRAM – przechowująca aktualnie wyświetlany zestaw znaków.

Wyświetlenie napisu jest możliwe przez zapis do pamięci DDRAM za pośrednictwem interfejsu równoległego, którego sygnały przedstawiono w **tabeli 5**. Za jego pomocą można także konfigurować wyświetlacz wysyłając do niego instrukcje. Część dostępnych instrukcji, użytych w przykładzie, razem z wybranymi parametrami została przedstawiona w **tabeli 6**. Szczegóły dotyczące wszystkich instrukcji oraz ich struktury znajdują się w dokumentacji modułu, na stronie: ***http://bit.ly/2SfhLLy***

Ze względu na dużą liczbę sygnałów, moduł PmodCLP został w przykładzie podłączony do złącza ARDUINO CONNECTOR według **tabeli 7**. W przykładzie używane są tylko komendy do zapisu danych i instrukcji, dlatego piny mikrokontrolera mogą być skonfigurowane wyłącznie jako wyjścia.

Obsługa wyświetlacza znajduje się w plikach inc/PmodCLP.h oraz src/PmodCLP.c. W pierwszym z nich znajdują się definicje użytych pinów, portów oraz funkcji włączających sygnały zegarowe. Służą one ułatwieniu konfiguracji ze względu na dużą liczbę sygnałów. Za samą konfigurację odpowiedzialna jest funkcja PmodCLP_Config. Włącza sygnały zegarowe dla kolejnych portów oraz konfiguruje piny jako wyjścia. Następnie przeprowadzana jest inicjalizacja wyświetlacza, którą przedstawiono na **listingu 8**. Jest ona zgodna z procedują inicjalizacji przedstawioną w dokumentacji modułu, natomiast użyte w niej komendy zostały wyjaśnione w tabeli 6. Zastosowane pomiędzy instrukcjami opóźnienia zostały wydłużone

Listing 6. Konfiguracja interfejsu SPI dla modułu PmodTC1

```
void PmodTC1_Config(void)
{
    // Configure the SPI connected to the Pmod module. Only the MISO line is required.
    pmodTc1Spi.Instance = SPI1;
    pmodTc1Spi.Init.Mode = SPI_MODE_MASTER;
    pmodTc1Spi.Init.Direction = SPI_DIRECTION_2LINES_RXONLY;
    pmodTc1Spi.Init.DataSize = SPI_DATASIZE_16BIT;
    pmodTc1Spi.Init.CLKPolarity = SPI_POLARITY_LOW;
    pmodTc1Spi.Init.CLKPhase = SPI_PHASE_2EDGE;
    pmodTc1Spi.Init.NSS = SPI_NSS_SOFT;
    pmodTc1Spi.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_64;
    pmodTc1Spi.Init.FirstBit = SPI_FIRSTBIT_MSB;
    pmodTc1Spi.Init.TIMode = SPI_TIMODE_DISABLE;
    pmodTc1Spi.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
    pmodTc1Spi.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
    HAL_SPI_Init(&pmodTc1Spi);
}
```

Listing 7. Odczyt i konwersja danych z modułu PmodTC1

```
void PmodTC1_GetValues(float* temp, float* tempRef, uint8_t* status)
{
    uint16_t data[2] = {0};

    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
    HAL_SPI_Receive(&pmodTc1Spi, (uint8_t*)data, 2, 100);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
    uint16_t tempSign = data[0] & 0x8000;
    data[0] &= ~0x8000;
    *temp = (data[0] >> 2) / 4.0;
    if(tempSign != 0) *temp *= -1;
    uint16_t tempRefSign = data[1] & 0x8000;
    data[1] &= ~0x8000;
    *tempRef = (data[1] >> 4) / 16.0;
    if(tempRefSign != 0) *tempRef *= -1;
    *status = data[1] & 0x03;
}
```

Tabela 6. Opis komend użytych w przykładzie, wraz z ich parametrami

RS	R/W	DB7-DB0	Opis
0	0	0x38	8-bitowe dane (DB4 = 1), 2 linie (DB3 = 1), rozmiar znaku 5x8 (DB2 = 0)
0	0	0x0C	Włączenie wyświetlacza (DB2 = 1), niewidoczny kursor (DB1 = 0)
0	0	0x01	Wyczyszczenie wyświetlacza
0	0	0x06	Przesuwanie kursora w prawo (DB1 = 1), wyłączenie przesuwania napisu (DB0 = 0)
0	0	0x80	Ustawienie kursora na początku pierwszej linii
0	0	0xC0	Ustawienie kursora na początku drugiej linii

Tabela 7. Sposób podłączenia modułu PmodCLP do płytki KAmeLeon

Sygnał	Numer pinu PmodCLP	Numer pinu KAmeLeon ARDUINO CONNECTOR	Pin mikrokontrolera
DB0	1 (J1)	D0	PC5
DB1	2 (J1)	D1	PC4
DB2	3 (J1)	D2	PD6
DB3	4 (J1)	D3	PB6
GND	5 (J1)	GND	–
VCC	6 (J1)	+3,3	–
DB4	7 (J1)	D4	PB9
DB5	8 (J1)	D5	PD9
DB6	9 (J1)	D6	PD10
DB7	10 (J1)	D7	PB11
GND	11 (J1)	–	–
VCC	12 (J1)	–	–
RS	1 (J2)	D8	PD11
R/W	2 (J2)	D9	PB13
E	3 (J2)	D10	PB12
NC	4 (J2)	–	–
GND	5 (J2)	–	–
VCC	6 (J2)	–	–

ze względu na ograniczenie funkcji bibliotecznej HAL_Delay, która przyjmuje jedynie całkowitą liczbę milisekund.

Podczas konfiguracji wykorzystywane są następujące funkcje pomocnicze ułatwiające zarządzanie pinami i korzystanie z interfejsu równoległego:

- configGpio – konfiguruje podany pin GPIO jako wyjście,
- setGpio – ustawia podany pin w stan wysoki,
- resetGpio – ustawia podany pin w stan niski,
- writeGpio – ustawia podany pin w dowolny stan przekazany jako argument,
- setDataBits – wystawia podaną wartość 8-bitową na liniach DB7 – DB0,
- writeCommand – wysyła podaną instrukcję,
- writeData – wysyła podany bajt danych.

Do wyświetlania napisów została zaimplementowana funkcja PmodCLP_Write, przedstawiona na **listingu 9**. Przyjmuje ona numer linii oraz napis wraz z jego długością, a następnie wysyła go do wyświetlacza. Wybór linii (0, lub 1), odbywa się przez zapis instrukcji ustawiającej kursor na odpowiednim adresie pamięci DDRAM. Dane do wyświetlacza mogą być wysłane bezpośrednio w kodzie ASCII, ze względu na zastosowane mapowanie kodów znaków w pamięci CGROM.

Główna funkcja programu – main, wywołuje funkcję konfiguracji modułu PmodCLP oraz wyświetla teksty:

Hello PmodCLP!
KAmeleon Board

Efekt działania programu jest widoczny na **fotografii 7**.

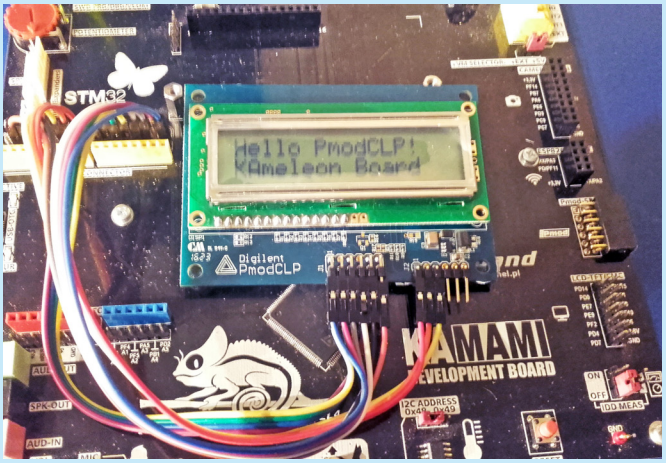
Krzysztof Chojnowski

Listing 8. Sekwencja inicjalizacji wyświetlacza

```
HAL_Delay(20);
writeCommand(0x38);
HAL_Delay(1);
writeCommand(0x0C);
HAL_Delay(1);
writeCommand(0x01);
HAL_Delay(2);
writeCommand(0x06);
```

Listing 9. Wysyłanie napisu na wyświetlacz

```
void PmodCLP_Write(uint8_t line, char* text, uint32_t len)
{
    if(line >= MAX_LINES || len > MAX_LINE_CHARACTERS) return;
    if(line == 0) writeCommand(0x80); else writeCommand(0xC0);
    for(uint32_t i = 0; i < len; i++) writeData((uint8_t)text[i]);
}
```



Fotografia 7. Wyświetlanie napisu na module PmodCLP

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