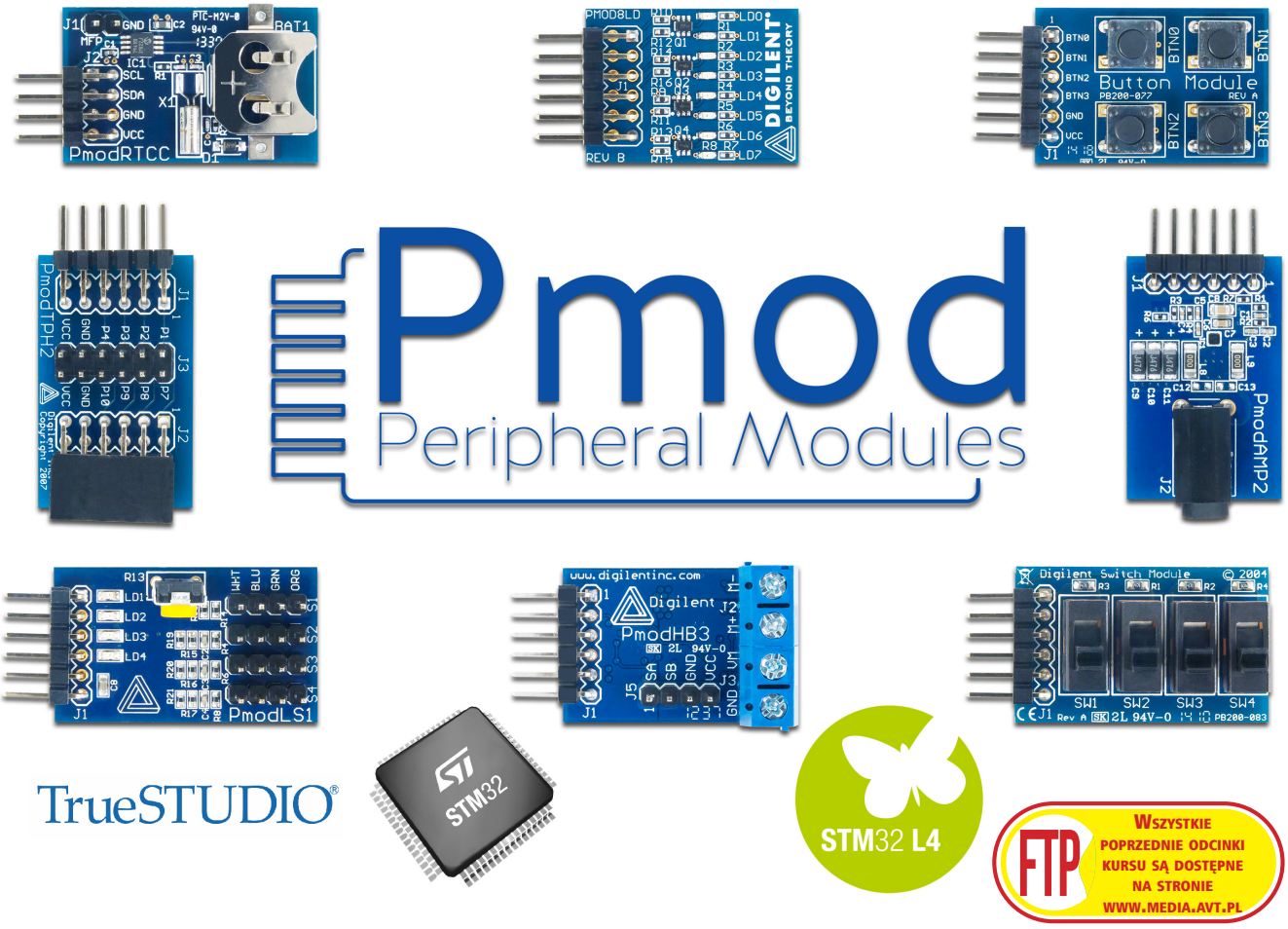


Digilent Pmod i STM32 (10)

W ostatnim odcinku cyklu opisującego wybrane moduły peryferyjne Pmod firmy Digilent zostaną omówione aplikacje modułów: PmodGYRO z 3-osiowym żyroskopem, PmodTMP3 z czujnikiem temperatury oraz PmodWi-Fi zawierający modem do komunikacji bezprzewodowej.

Przykłady przedstawione w niniejszym artykule zostały przygotowane dla środowiska Atollic TrueSTUDIO i zestawu uruchomieniowego KAmeLeon (*www.kameleonboard.org*) z wykorzystaniem biblioteki STM32Cube_FW_L4.

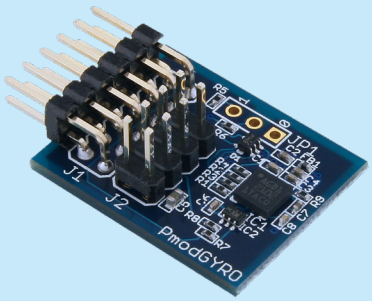
PmodGYRO

Jako pierwszy przedstawiony zostanie PmodGYRO (**fotografia 1**), czyli moduł z układem L3G4200D firmy STMicroelectronics. Jest to 3-osiowy żyroskop umożliwiający pomiar prędkości kątowej w jednym z wybranych zakresów:

- 250 dps (8,75 mdps/LSB),
- 500 dps (17,50 mdps/LSB),
- 2000 dps (70,00 mdps/LSB).

Dane pomiarowe mogą być odczytywane bezpośrednio z rejestrów układu lub zapisywane w wewnętrznej kolejce FIFO. Dodatkowo L3G4200D umożliwia włączenie filtrów górno- i dolnoprzepustowego do toru przetwarzania danych.

Układ L3G4200D umożliwia konfigurację źródeł przerwań przyporządkowanych do dwóch dostępnych linii: INT1 i INT2. Pierwsza z nich obsługuje przerwania od przekroczenia progów – niskiego, lub wysokiego, dla każdej z trzech osi. Dodatkowo można zdefiniować czas przekroczenia progu, po którym ma zostać zgłoszone przerwanie. Przerwanie od przekroczenia progu przestaje być aktywne natychmiast po powrocie mierzonej wielkości do dozwolonego zakresu lub po odczekaniu zdefiniowanego czasu bez ponownego przekroczenia progu. Sposób działania może być wybrany za pomocą bitu WAIT w rejestrze INT1_DURATION (0x38). Opisany mechanizm przerwań został przedstawiony na **rysunkach 2** i **3**.



Fotografia 1. Wygląd modułu PmodGYRO

Druga linia przerwań – INT2, może być użyta do sygnalizacji gotowości danych do odczytu lub poziomu zajętości kolejki FIFO. Dostępne źródła przerwań to:

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

- dane gotowe do odczytu,
- osiągnięta zdefiniowana liczba elementów w kolejce,
- przepełnienie kolejki,
- kolejka pusta.

Układ L3G4200D udostępnia także 8-bitowy czujnik temperatury o rozdzielczości 1°C/LSB i zakresie zgodnym z zakresem pracy całego układu: od –40°C do 85°C.

Do komunikacji z układem L3G4200D mogą być użyte interfejsy SPI i I²C. Są one dostępne na złączach J1 (złącze Pmod SPI typu 2A z sygnałami INT1 i INT2) i J2 (złącze Pmod I²C). W przykładzie wykorzystany został interfejs SPI ze względu na możliwość podłączenia złącza J1 bezpośrednio do złącza Pmod-SPI zestawu KAmeLeon, tak jak na **fotografii 4**. Piny mikrokontrolera i odpowiadające im sygnały modułu PmodGYRO zostały przedstawione w **tabeli 1**.

Obsługa modułu PmodGYRO znajduje się w plikach PmodGYRO.c i PmodGYRO.h. Za konfigurację modułu jest odpowiedzialna funkcja PmodGYRO_Config na **listingu 1**. Konfiguruje ona interfejs SPI1 w trybie 3 (CPOL = 1, CPHA = 1) z programową kontrolą pinu CS, a następnie ustawia wybrane rejestry układu L3G4200D:

- CTRL_REG1 (0x20): włączony pomiar na osi Z, częstotliwość pomiaru (ODR) 100 Hz, pasmo 12,5 Hz,
- INT1_THS_ZH (0x36), INT1_THS_ZL (0x37): próg generowania przerwania 35,84 dps (4096 * 8,75 mdps),
- INT1_CFG (0x30): włączenie przerwania od górnego progu na osi Z,
- CTRL_REG3 (0x22): włączenie przerwania na linii INT1.

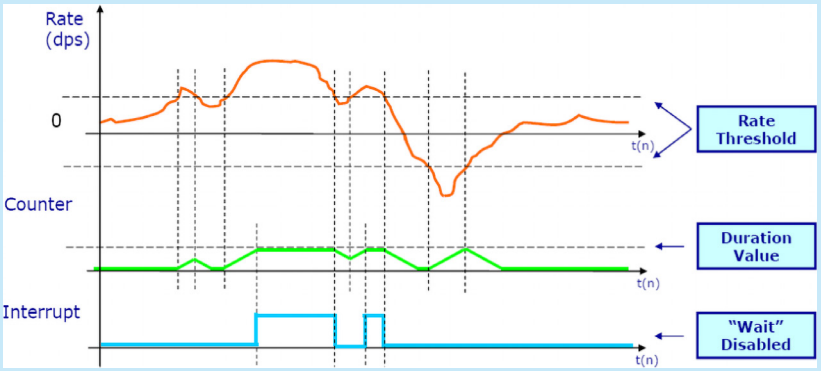
Na koniec konfigurowany jest pin PE12 w trybie przerwania aktywnego na zboczu narastającym.

Piny dla interfejsu SPI są konfigurowane w funkcji HAL_SPI_MspInit, wywoływanej przez biblioteczną funkcję HAL_SPI_Init. Sygnały MISO, MOSI i SCLK są ustawiane w funkcji alternatywnej dla SPI1, natomiast CS jest konfigurowany jako wyjście GPIO.

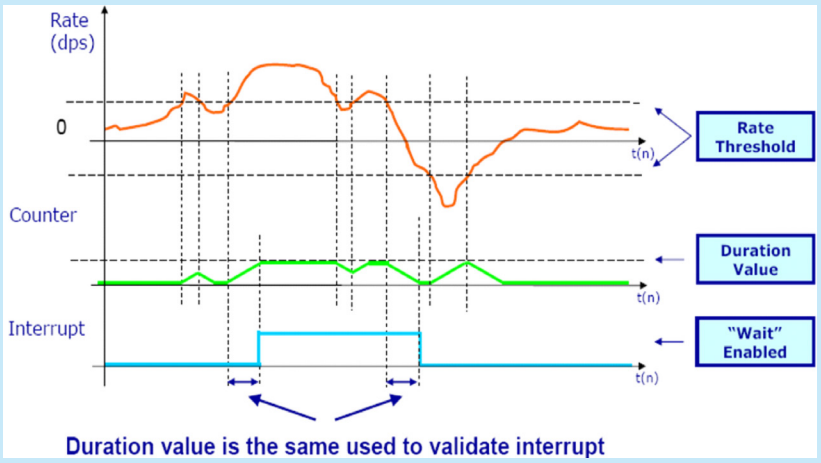
Po wystąpieniu przerwania na linii INT1 konieczne jest odczytanie rejestru INT1_SRC (0x31), w którym znajduje się informacja o zdarzeniach będących źródłem przerwania. Odczytanie tego rejestru powoduje wyczyszczenie bitu IA oraz ustawienie stanu niskiego na linii INT1. Odczyt rejestru jest realizowany przez funkcję PmodGYRO_ReadInterruptFlags.

Do odczytu i zapisu rejestrów układu L3G4200D zostały zaimplementowane funkcje pomocnicze readRegister i writeRegister, przedstawione na **listingu 2**. Na początku każdej transakcji wysyłany jest bajt zawierający adres rejestru oraz dwa bity sterujące: RW definiujący, czy jest to odczyt (1), czy zapis (0) oraz MS włączający inkrementację adresu (1) w przypadku transmisji wielu bajtów. W przykładzie wszystkie transakcje zawierają dwa bajty: adresu oraz danych.

W przykładowej aplikacji nie został zaimplementowany odczyt danych, ponieważ reaguje ona wyłącznie na skonfigurowane przerwania od przekroczenia zadanego progu. Funkcja obsługująca przerwanie – HAL_GPIO_EXTI_Callback, odczytuje rejestr INT1_SRC za pośrednictwem opisanej funkcji PmodGYRO_ReadInterruptFlags a następnie zapala diodę LED0, podłączoną do pinu PC6, na 100 ms. Z uwagi na fakt, że opóźnienie jest realizowane przez funkcję HAL_Delay, przerwanie od GPIO musi mieć niższy priorytet niż przerwanie od licznika SysTick. Pierwsze z nich jest konfigurowane w funkcji PmodGYRO_Config (priorytet 0x0F), natomiast priorytet drugiego z przerwań jest ustawiany w pliku stm32l4xx_hal_conf.h jako TICK_INT_PRIORITY (0x0A).



Rysunek 2. Działanie przerwania INT1 z bitem WAIT ustawionym na 0



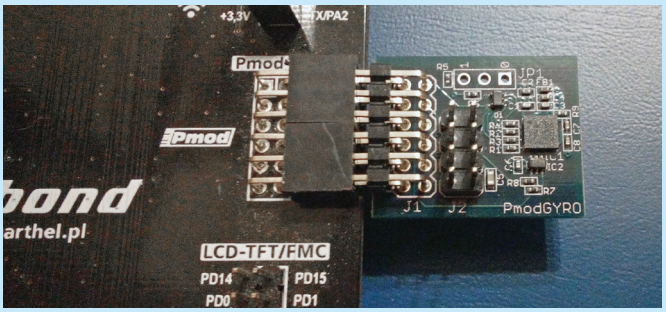
Rysunek 3. Działanie przerwania INT1 z bitem WAIT ustawionym na 1

PmodTMP3

Drugim z modułów jest PmodTMP3 (**fotografia 5**) – czujnik temperatury z układem Microchip TCN75A. Zapewnia on pomiary z dokładnością ±2°C (typowo ±1°C) w zakresie od –40°C do +125°C

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

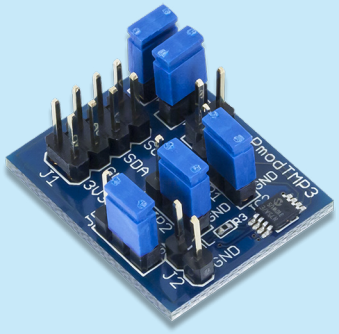
Sygnał	Numer pinu PmodGYRO (J1)	Pin STM32L496ZG (Kameleon Pmod-SPI)
~CS	1	PB0
MOSI	2	PA7
MISO	3	PE14
SCLK	4	PA1
INT1	7	PE12
INT2	8	PE13



Fotografia 4. Moduł PmodGYRO podłączony do zestawu KAmeLeon

i regulowaną rozdzielczością 0,5°C (9 bitów), 0,25°C (10 bitów), 0,125°C (11 bitów) lub 0,0625°C (12 bitów).

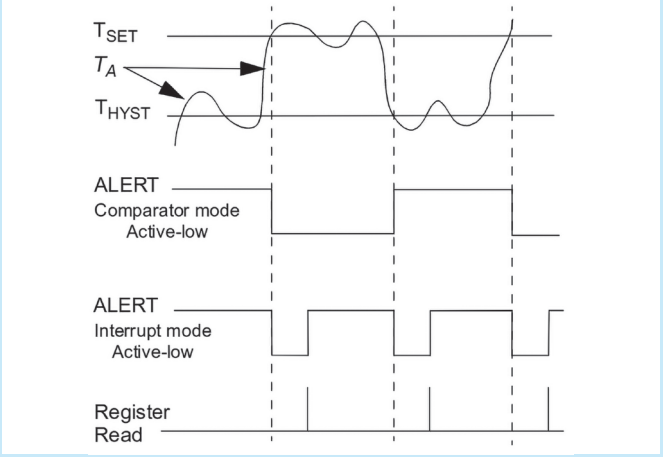
Układ TCN75A umożliwia konfigurację progu temperatury, którego przekroczenie wyzwala przerwanie na pinie AL. Przerwanie to jest aktywne do momentu odczytu dowolnego rejestru (w trybie przerwań) lub do momentu, kiedy temperatura spadnie poniżej progu z uwzględnieniem histerezy (w trybie komparatora). Opisana zasada działania została przedstawiona na **rysunku 6**. Wartości progu temperatury i histerezy są konfigurowane za pośrednictwem rejestrów.



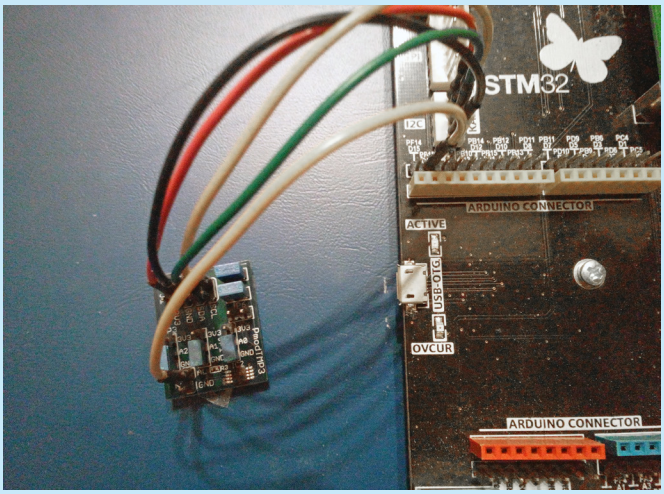
Fotografia 5. Wygląd modułu PomodTEM3

Moduł PmodTMP3 zawiera 8-pinowe złącze Pmod I²C. W przykładzie zostało ono podłączone do złącza Kamod I²C Expander zgodnie z **tabelą 2** i **fotografią 7**. Pin AL znajdujący się na złączu J2 został podłączony do pinu D14 (PF15) na złączu ARDUINO CONNECTOR zestawu KAmeLeon. Adres I²C modułu w przykładzie został ustawiony na 0x48, dlatego wszystkie zworki odpowiedzialne za jego konfigurację (JP1, JP2, JP3) powinny zostać zwarte do masy.

Obsługa modułu PmodTMP3, zaimplementowana w plikach PmodTMP3.c i PmodTMP3.h, sprowadza się do konfiguracji układu oraz odczytu zmierzonych wartości temperatury. Funkcja konfiguracyjna – PmodTMP3_Config, przedstawiona na **listingu 3**, konfiguruje interfejs I2C2 oraz pin PF15 w trybie przerwania wyzwalanego



Rysunek 6. Generowanie przerwania na pinie AL



Fotografia 7. Moduł PmodTMP3 podłączony do zestawu KAmeLeon

Listing 1. Konfiguracja interfejsu SPI i przerwania dla modułu PmodGYRO

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

  HAL_SPI_Init(&pmodGyroSpi);

  writeRegister(0x20, 0x0C);
  writeRegister(0x36, 0x10);
  writeRegister(0x37, 0x00);
  writeRegister(0x30, 0x20);
  writeRegister(0x22, 0x80);

  __HAL_RCC_GPIOE_CLK_ENABLE();

  GPIO_InitTypeDef GPIO_InitStruct;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Pin = GPIO_PIN_12;
  HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

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

Listing 2. Odczyt i zapis rejestrów układu L3G4200D za pośrednictwem interfejsu SPI

```
uint8_t readRegister(uint8_t address)
{
  uint8_t txbuf[2] = {address | 0x80, 0x00};
  uint8_t rxbuf[2] = {0x00};

  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  HAL_SPI_TransmitReceive(&pmodGyroSpi, txbuf, rxbuf, 2, 100);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);

  return rxbuf[1];
}

void writeRegister(uint8_t address, uint8_t data)
{
  uint8_t txbuf[2] = {address, data};
  uint8_t rxbuf[2] = {0x00};

  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
  HAL_SPI_TransmitReceive(&pmodGyroSpi, txbuf, rxbuf, 2, 100);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
}
```

dowolnym zboczem. Tak jak w poprzednich przykładach, tutaj także piny należące do magistrali I²C są konfigurowane wewnątrz funkcji HAL_I2C_MspInit wywoływanej przez bibliotekę STM32Cube.

Na zakończenie konfiguracji zapisywane są dwa rejestry układu TCN75A: TEMPERATURE LIMIT-SET REGISTER (0x03) oraz TEMPERATURE HYSTERESIS REGISTER (0x02). Ustawiają one odpowiednio wartości progu temperatury oraz histerezy zgodnie z argumentami przekazanymi do funkcji. Obsługiwane są wyłącznie wartości całkowite, dlatego drugi z zapisywanych do rejestrów bajtów, zawierający część ułamkową, jest wyzerowany. Rejestr konfiguracyjny SENSOR CONFIGURATION REGISTER (0x01) ma domyślną wartość 0x00, która ustawia rozdzielczość pomiaru na 0,5°C, a przerwanie w trybie komparatora.

Odczyt temperatury został zaimplementowany w funkcji PmodTMP3_GetTemperature. Odczytuje ona dwa bajty z rejestru AMBIENT TEMPERATURE REGISTER (0x00), a następnie generuje zmienną typu int16_t przechowującą wartość temperatury. Jest ona dodatkowo mnożona przez 10, aby zachować ustawioną rozdzielczość. Dzielenie przez 256 wynika z przesunięcia części całkowitej w rejestrach układu o 8 bitów. Funkcja PmodTMP3_GetTemperature została przedstawiona na **listingu 4**.

Główna funkcja programu, znajdująca się w pliku main.c, rozpoczyna się konfiguracją peryferiów mikrokontrolera oraz modułu PmodTMP3. Opisana wcześniej funkcja konfiguracyjna ustawia próg alarmowy na 28°C, natomiast histereza przyjmuje wartość

26°C. Następnie odczytywana jest temperatura i programowo sprawdzane jest przekroczenie progu, aby odpowiednio ustawić stan początkowy diody LED0 (PC6) informującej o alarmie. W pętli głównej, co 1000 ms, odczytywana jest temperatura, a jej wartość wysyłana jest za pośrednictwem portu szeregowego LPUART1. Opisana procedura została przedstawiona na **listingu 5**.

W funkcji obsługi przerwania na linii AL zmieniany jest stan diody LED0 po każdorazowej sygnalizacji przekroczenia progu lub powrotu do dopuszczalnej wartości.

Tabela 2. Sygnały PmodTMP3 oraz odpowiadające im piny złączy ARDUINO, I²C Expander oraz wyprowadzenia mikrokontrolera

Sygnał	Numer pinu PmodTMP3	Pin ARDUINO CONNECTOR	Pin Kamod I²C Expander	Pin mikrokontrolera
SCL	1 (J1)	-	2	PF1
SDA	2 (J1)	-	3	PF0
GND	3 (J1)	-	4	-
3V3	4 (J1)	-	1	-
AL	1 (J2)	D14	-	PF15

PmodWi-Fi

Ostatnim z przedstawianych modułów jest PmodWi-Fi (**fotografia 8**), mający modem radiowy Wi-Fi MRF24WG0MA firmy Microchip. Modem jest zgodny ze standardami IEEE 802.11b/g/n i umożliwia transmisję danych na odległość do 400 m, przy prędkości transmisji danych wynoszącej 1 lub 2 Mbps. Moduł PmodWi-Fi zawiera antenę PCB, dzięki czemu do jego użycia nie są potrzebne żadne dodatkowe komponenty. Do komunikacji z modemem służy interfejs SPI oraz generowane przez niego przerwania.

PmodWi-Fi ma 12-pinowe złącze SPI z dodatkowymi sygnałami WP (Write-protected) i HIBERATE. Sygnały te są podłączone przez rezystory do masy, dzięki czemu nie muszą być obsługiwane przez mikrokontroler. W przykładzie nie są one używane ze względu na brak podłączonych do nich pinów mikrokontrolera na złączu Pmod-SPI. Podłączenie modułu PmodWi-Fi do mikrokontrolera za pośrednictwem złącza Pmod-SPI, użytego w przykładzie, zostało przedstawione w **tabeli 3** i na **fotografii 9**.

Do obsługi układu MRF24WG0MA firma Microchip udostępniła programowy stos TCP/IP przeznaczony do mikrokontrolerów z rodziny PIC i dostępny na stronie ***www.microchip.com/wireless***. Jednak z uwagi na to, że przykład został przygotowany dla mikrokontrolera z rodziny STM32, zastosowany został stos CycloneTCP przygotowany przez firmę ORYX embedded i dostępny do pobrania ze strony ***www.oryx-embedded.com***. Stos ten jest dostępny za darmo na licencji GPLv2 lub na jednej z licencji komercyjnych, a jego dużą zaletą jest duża liczba wspieranych mikrokontrolerów oraz modemów. W przykładzie pokazano skanowanie widocznych sieci Wi-Fi, dlatego w katalogu Drivers znajduje się tylko sterownik modułu MRF24WG0MA (CycloneTCP_SSL_Crypto_Open_1_8_0/third_party/microchip/devices/mrf24wg/). Pozostałe źródła wchodzące w skład stosu są nieużywane, dlatego zostały usunięte z plików projektu.

Tabela 3. Sygnały PmodWi-Fi oraz odpowiadające im piny mikrokontrolera; w tabeli pominięto sygnały niepodłączone do mikrokontrolera (WP i HYBERNATE) oraz linie zasilania występujące na złączu Pmod

Sygnał	Numer pinu PmodWi-Fi (J1)	Pin STM32L496ZG (KAmeLeon Pmod-SPI)
CS	1	PB0
SDI	2	PA7
SDO	3	PE14
SCK	4	PA1
INT	7	PE12
RESET	8	PE13

Listing 3. Konfiguracja interfejsu SPI i przerwania dla modułu Pmod

```
void PmodTMP3_Config(int8_t alarm, int8_t hyst)
{
  __HAL_RCC_GPIOF_CLK_ENABLE();
  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_15;
  HAL_GPIO_Init(GPIOF, &GPIO_InitStruct);

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

  pmodTmp3I2c.Instance = I2C2;
  pmodTmp3I2c.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  pmodTmp3I2c.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  pmodTmp3I2c.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  pmodTmp3I2c.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  pmodTmp3I2c.Init.OwnAddress1 = 0x01;
  pmodTmp3I2c.Init.Timing = 0x10563046;

  HAL_I2C_Init(&pmodTmp3I2c);

  uint8_t tAlarm[2] = {alarm, 0};
  uint8_t tHyst[2] = {hyst, 0};

  HAL_I2C_Mem_Write(&pmodTmp3I2c, PMODTMP3_ADDRESS, 0x03, 1, tAlarm, 2, 100);
  HAL_I2C_Mem_Write(&pmodTmp3I2c, PMODTMP3_ADDRESS, 0x02, 1, tHyst, 2, 100);
}
```

Listing 4. Odczyt wartości temperatury z modułu PmodTMP3

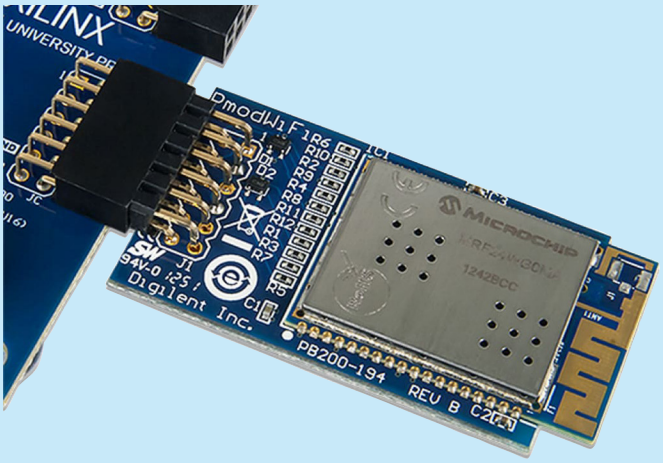
```
int16_t PmodTMP3_GetTemperature()
{
  uint8_t data[2] = {0};
  HAL_I2C_Mem_Read(&pmodTmp3I2c, PMODTMP3_ADDRESS, 0x00, 1, data, 2, 100);

  int16_t temperature = (data[0] << 8) | data[1];
  temperature = (temperature * 10) / 256;
  return temperature;
}
```

Listing 5. Konfiguracja modułu PmodTMP3 i odczyt temperatury

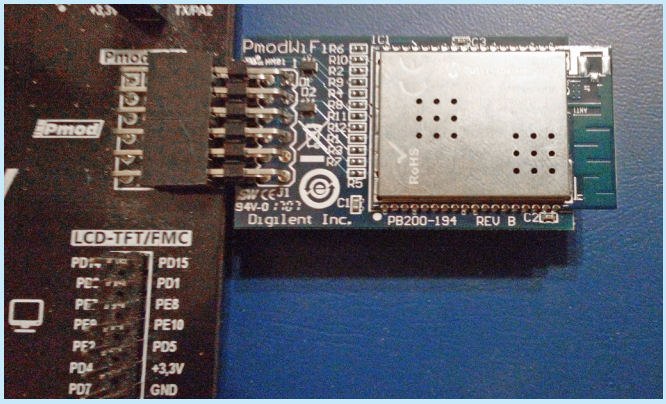
```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  Serial_Config();
  Led_Config();

  int16_t alarm = 28;
  int16_t hyst = 26;

  PmodTMP3_Config(alarm, hyst);

  int16_t temperature = PmodTMP3_GetTemperature();
  if (temperature > alarm * 10)
    Led_TurnOn(LED0);

  char textBuffer[128];

  while(1) {
    HAL_Delay(1000);

    temperature = PmodTMP3_GetTemperature();
    sprintf(textBuffer, „Temperature: %d\n”, temperature);
    Serial_Write(textBuffer, strlen(textBuffer));
  }
}
```

Listing 6. Funkcja main przykładowej aplikacji dla modułu PmodWi-Fi

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  Serial_Config();
  WF_Init();

  while(1)
  {
    WF_Task();
  }
}
```



Fotografia 8. Wygląd modułu PmodWi-Fi



Fotografia 9. Moduł PmodWi-Fi podłączony do zestawu KAmeLeon

Sterownik mrf24wg wymaga zdefiniowania kilku dodatkowych funkcji i definicji odpowiedzialnych za obsługę GPIO, SPI, przerwań oraz zliczania czasu. Zostały one umieszczone w plikach znajdujących się w katalogach inc i src w katalogu głównym projektu:

- wf_customize.h – konfiguracja sterownika, kolejności bajtów i komunikatów diagnostycznych,
- wf_eint_stub.c – konfiguracja i obsługa przerwań od pinu INT,
- wf_gpio_stub.c – konfiguracja i obsługa pinu RESET,
- wf_spi_stub.c – konfiguracja i obsługa interfejsu SPI,
- wf_timer_stub.c – informacja o czasie systemowym.

Opisy funkcji zaimplementowanych w powyższych plikach znajdują się w **tabeli 4**.

Mając zaimplementowane wszystkie funkcje wymagane przez sterownik układu MRF24WG0MA, można przejść do kodu głównego przykładowej aplikacji. Inicjalizacja wszystkich modułów znajduje się w funkcji *main*, przedstawionej na **listingu 6**. Za konfigurację sterownika odpowiada funkcja biblioteczna WF_Init, która z kolei wywołuje niezbędne funkcje opisane wyżej. Dodatkowo, w głównej pętli programu wywoływana jest funkcja WF_Task, napędzająca maszynę stanów sterownika odpowiedzialną za sterowanie modemem oraz przetwarzanie danych.

Druga z funkcji w pliku main.c – HAL_GPIO_EXTI_Callback wywołuje jedynie wspomnianą wcześniej funkcję ext1IrqHandler odpowiedzialną za obsługę przerwania od pinu INT.

Ostatnie dwie funkcje – WF_ProcessEvent oraz WF_RxPacketReady, są wymagane przez sterownik. Pierwsza z nich, przedstawiona na **listingu 7**, obsługuje zgłaszane przez niego zdarzenia. W przykładzie są to: WF_EVENT_INITIALIZATION wysyłane po zakończeniu inicjalizacji i WF_EVENT_SCAN_RESULTS_READY wysyłane, gdy zakończy się skanowanie widocznych sieci Wi-Fi. Obsługa wszystkich zdarzeń zależy od aplikacji – w opisywanym programie wyniki skanowania są jedynie wypisywane na port szeregowy. Na listingu znajduje się także wywołanie – DumpEventInfo, które wypisuje informacje diagnostyczne dotyczące otrzymanych zdarzeń. Druga z funkcji – WF_RxPacketReady nie jest używana w przykładzie, jednak jest wymagana przez sterownik, dlatego została pozostawiona pusta.

Do przykładu został dołączony plik tiny_printf.c zawierający proste implementacje funkcji printf, fprintf i sprintf. Są one używane przez stos CycloneTCP do wypisywania informacji diagnostycznych. Do poprawnego działania tych funkcji konieczne jest zaimplementowanie funkcji _write odpowiedzialnej za wypisywanie znaków na port szeregowy. W przykładzie użyto do tego celu funkcji Serial_Write, zaimplementowanej w pliku serial.c i obsługującej port LPUART1.

Krzysztof Chojnowski

Listing 7. Obsługa zdarzeń dostarczanych przez sterownik mrf24wg

```
void WF_ProcessEvent(uint8_t eventType, uint32_t eventData)
{
  DumpEventInfo(eventType, eventData);

  if(eventType == WF_EVENT_INITIALIZATION && eventData == WF_INIT_SUCCESSFUL) {
      WF_Scan();
  } else if (eventType == WF_EVENT_SCAN_RESULTS_READY) {
      for(int i=0; i<eventData; i++) {
        t_scanResult res;
        WF_ScanResultGet(i,  &res);
        res.ssid[res.ssidLen] = '\0';
        printf(„%d. [%s] RSSI:%d\r\n”,i , res.ssid, res.rssi);
      }
  }
}
```

Tabela 4. Funkcje wymagane przez sterownik modemu mrf24wg

Funkcja	Opis
WF_EintInit	Inicjalizacja przerwania od pinu INT aktywnego na zboczu opadającym.
WF_EintEnable	Włączenie przerwania od pinu INT.
WF_EintDisable	Wyłączenie przerwania od pinu INT.
WF_isEintDisabled	Sprawdzenie stanu przerwania od pinu INT (włączone lub nie).
ext1IrqHandler	Obsługa przerwania od pinu INT wyłączająca przerwania i wywołująca wewnętrzną funkcję sterownika odpowiedzialną za dalsze przetwarzanie przerwania.
WF_GpioInit	Inicjalizacja pinu RESET.
WF_GpioSetReset	Ustawienie żądanego stanu na pinie RESET.
WF_GpioSetHibernate	Ustawienie żądanego stanu na pinie HIBERNATE. Funkcja jest wymagana przez sterownik, jednak w przykładzie jest pusta ze względu na niepodłączony sygnał HIBERNATE.
WF_SpiInit	Inicjalizacja interfejsu SPI w trybie 0 (CPOL = 0, CPHA = 0) z programową obsługą pinu CS; inicjalizacja sygnałów SDI, SDO i SCK.
WF_SpiEnableChipSelect	Ustawienie pinu CS w stanie niskim (aktywnym).
WF_SpiDisableChipSelect	Ustawienie pinu CS w stanie wysokim (nieaktywnym).
WF_SpiTxRx	Wysyłanie i odbiór danych przez interfejs SPI.
WF_TimerInit	Inicjalizacja licznika podającego liczbę milisekund od uruchomienia systemu. Funkcja jest pusta, ponieważ w przykładzie używany jest licznik obsługiwany przez bibliotekę STM32Cube.
WF_TimerRead	Odczyt liczby milisekund od startu systemu. Funkcja wykorzystuje bibliotekę STM32Cube (HAL_GetTick).