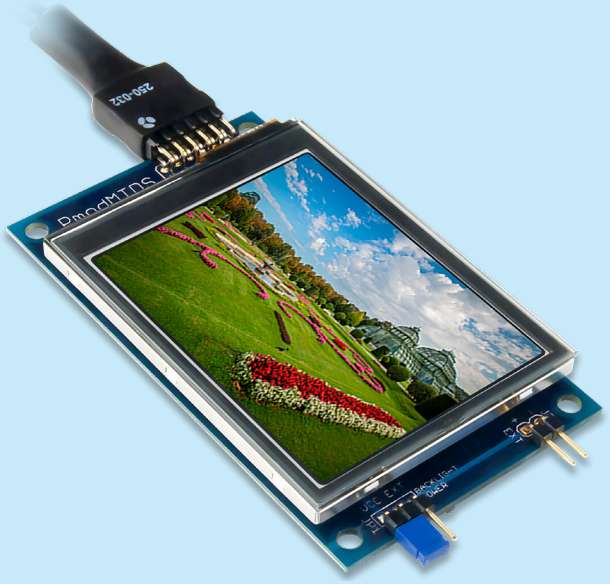
Digilent Pmod 9i STM32 (8)

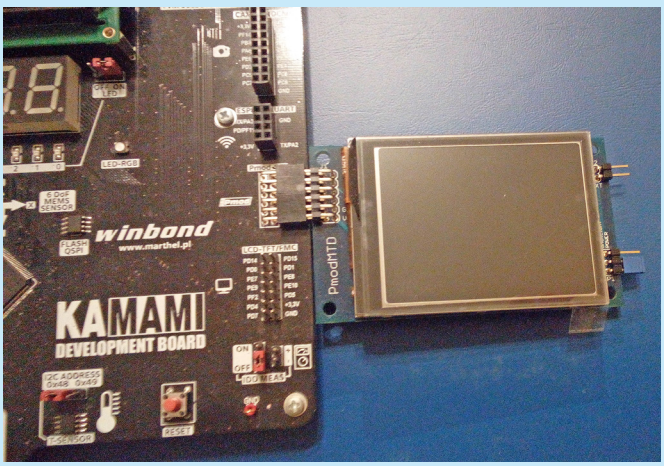
W ósmej części cyklu dotyczącego modułów Digilent Pmod zostaną przedstawione: PmodMTDS z wyświetlaczem graficznym o przekątnej 2,8" i rozdzielczości 320×240 pikseli, PmodRTCC zawierający układ zegara czasu rzeczywistego z kalendarzem i PmodCMPS2 z 3-osiowym czujnikiem pola magnetycznego. Wszystkie wymienione moduły mają biblioteki dostarczane przez producenta, dostosowane na potrzeby niniejszego artykułu do zestawu KAmeLeon. Przykłady zostały przygotowane dla środowiska Atollic TrueSTUDIO i zestawu uruchomieniowego KAmeLeon (*www.kameleonboard.org*) z wykorzystaniem biblioteki STM32Cube_FW_L4.



Fotografia 1. Wygląd modułu PmodMTDS

Moduł PmodMTDS ma kolorowy wyświetlacz graficzny o rozdzielczości 320×240 pikseli z panelem dotykowym mogącym obsłużyć do dwóch punktów dotyku. Dodatkowo moduł wyposażono w gniazdo microSD przeznaczone dla karty pamięci zawierającej grafiki do wykorzystania w aplikacji (**fotografia 1**). Do kontroli wyświetlacza służy mikrokontroler PIC32MZ, z którym można się komunikować za pośrednictwem interfejsu SPI.

Do obsługi kontrolera znajdującego się na płytce PmodMTDS służy dostarczona przez producenta biblioteka MTDS zaimplementowana w języku C++. Można ją pobrać ze strony: *http://bit.ly/2TIchZI*. Jest ona przeznaczona dla platformy Arduino, jednak na potrzeby niniejszego przykładu została zmodyfikowana, aby współpracowała z zestawem KAmeLeon.



Fotografia 2. Moduł PmodMTDS podłączony do płytki KAmeLeon

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

Sygnał	Numer pinu PmodMTDS (J1)	Pin STM32L496ZG (KAmeLeon Pmod-SPI)
CS	1	PB0
MOSI	2	PA7
MISO	3	PE14
SCLK	4	PA1
RESET	8	PE13

Biblioteka składa się z dwóch warstw: mtds oraz MyDisp. Pierwsza z nich zawiera interfejs umożliwiający komunikację z kontrolerem wyświetlacza i udostępniający podstawowe operacje graficzne. Moduł MyDisp stanowi warstwę nadrzędną, ułatwiającą korzystanie z wyświetlacza oraz rozszerzającą funkcjonalność warstwy niższej. W przykładzie pokazano, w jaki sposób można korzystać z obu warstw w celu zbudowania prostego interfejsu graficznego. Ze względu na wielkość interfejsów obu warstw zostaną opisane tylko te metody, które wykorzystano w przykładowej aplikacji. Umożliwiają one wyświetlanie tekstu na wyświetlaczu oraz tworzenie przycisków obsługiwanych za pomocą panelu dotykowego.

Moduł PmodMTDS ma 12-pinowe złącze typu 2A z interfejsem SPI oraz sygnałem RESET. Może być ono dołączone do gniazda Pmod-SPI zestawu KameLeon, tak jak na **fotografii 2**. Piny mikrokontrolera i odpowiadające im sygnały modułu PmodMTDS zostały przedstawione w **tabeli 1**.

Tabela 2. Użyte w przykładzie funkcje warstwy MyDisp

Funkcja	Opis
Begin	Inicjalizacja biblioteki i wyświetlacza.
clearDisplay	Ustawienie wszystkich pikseli na wybrany kolor.
createButton	Utworzenie przycisku.
enableButton	Włączenie obsługi przycisku przez panel dotykowy.
drawButton	Rysowanie przycisku na wyświetlaczu.
setForeground	Ustawienie koloru pierwszego planu dla rysowanych obiektów.
setBackground	Ustawienie koloru tła dla rysowanych obiektów.
setPen	Ustawienie typu pisaka.
setTransparency	Włączenie lub wyłączenie przezroczystości.
drawText	Rysowanie tekstu na wyświetlaczu.
checkTouch	Aktualizacja stanu panelu dotykowego.
isTouched	Sprawdzenie stanu wciśnięcia wybranego przycisku.

Konfiguracja i obsługa interfejsu SPI dla modułu jest zaimplementowana w pliku MtdsHal.cpp w warstwie mtds. Znajdujące się tam funkcje zostały zmodyfikowane tak, aby moduł PmodMTDS mógł działać razem z zestawem KAmeLeon.

Ich fragmenty znajdują się na **listingach 1** i **2**. Pierwsza z nich – MtdsHalInitSpi, jest odpowiedzialna za inicjalizację interfejsu SPI1 w trybie 0 (CPOL = 0, CPHA = 0) z programową kontrolą linii CS, natomiast druga – MtdsHalPutSpiByte, realizuje transmisję i odczyt bajtu przez SPI.

Oprócz obsługi SPI, w pliku MtdsHal.cpp zostały zmodyfikowane także funkcje odpowiedzialne za zarządzanie czasem:

- **MtdsHalTmsElapsed** – zwracająca liczbę milisekund od startu programu,
- **MtdsHalDelayMs** i **MtdsHalDelayUs** – wprowadzające zadane opóźnienie.

Funkcje zostały zaimplementowane przy użyciu wywołań z biblioteki STM32Cube: HAL_GetTick i HAL_Delay. Do obsługi modułu zmieniona została także funkcja MtdsHalResetDisplay, ustawiająca niski stan na wyprowadzeniu RESET przez 1 ms w celu wyzerowania układu.

Obsługa wyświetlacza w prezentowanym przykładzie odbywa się przy użyciu obu warstw dostarczonej biblioteki. Warstwa MyDisp zawiera wysokopoziomowe funkcje do inicjalizacji wyświetlacza oraz rysowania podstawowych komponentów interfejsu, takich jak przyciski, tekst i proste kształty geometryczne. Umożliwia ona też korzystanie z panelu dotykowego. Lista funkcji warstwy MyDisp użytych w przykładzie znajduje się w **tabeli 2**.

Druga z warstw – mtds, również może być użyta do rysowania kształtów i tekstu, jednak w przykładzie została wykorzystana jej możliwość generowania bitmap dla przycisków warstwy MyDisp. Tworzone bitmapy mogą być używane wielokrotnie, dzięki czemu można na przykład utworzyć różne motywy graficzne dla przycisków.

Lista użytych funkcji warstwy mtds została przedstawiona w **tabeli 3**.

Przykładowy interfejs jest generowany w funkcji main, która została przedstawiona na **listingach 3** i **4**. Pierwszy z nich zawiera generowanie bitmapy dla przycisku. Znajdują się na nim dwie zmienne – HDS – reprezentująca kontekst graficzny oraz HBMP będąca uchwytem do tworzonej bitmapy. Sama bitmapa jest tworzona przez odpowiednie ustawienie powierzchni rysowania (SetDrawingSurface) i nanoszenie na nią kształtów i tekstu.

W pozostałej części funkcji main (listing 4) jest inicjalizowany moduł PmodMTDS oraz tworzone są tekst i przycisk, który dodatkowo wykorzystuje przygotowaną bitmapę. W pętli głównej programu cyklicznie aktualizowany jest stan panelu dotykowego. Jeżeli wykryte zostało wciśnięcie przycisku, to dodatkowo zwiększany jest licznik

Tabela 3. Użyte w przykładzie funkcje warstwy mtds

Funkcja	Opis
GetDs	Pobranie aktualnego kontekstu graficznego
CreateBitmap	Utworzenie nowej bitmapy o zadanym rozmiarze i formacie kolorów.
SetDrawingSurface	Ustawienie powierzchni rysowania, np. na bitmapę.
SetBgColor	Ustawienie koloru tła.
SetFgColor	Ustawienie koloru pierwszego planu.
Rectangle	Rysowanie prostokąta.
SetPen	Ustawienie typu pisaka.
SetFont	Ustawienie czcionki.
TextOut	Rysowanie tekstu.
ReleaseDs	Zwolnienie kontekstu graficznego.

Listing 1. Inicjalizacja SPI dla modułu PmodMTDS

```
void MtdsHalInitSpi(uint32_t pspiInit, uint32_t frq)
{
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();
    __HAL_RCC_GPIOE_CLK_ENABLE();
    __HAL_RCC_SPI1_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_AF5_SPI1;
    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_PULLUP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Pin = GPIO_PIN_0;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
    pmodMtdsSpi.Instance = SPI1;
    pmodMtdsSpi.Init.Mode = SPI_MODE_MASTER;
    pmodMtdsSpi.Init.Direction = SPI_DIRECTION_2LINES;
    pmodMtdsSpi.Init.DataSize = SPI_DATASIZE_8BIT;
    pmodMtdsSpi.Init.CLKPolarity = SPI_POLARITY_LOW;
    pmodMtdsSpi.Init.CLKPhase = SPI_PHASE_1EDGE;
    pmodMtdsSpi.Init.NSS = SPI_NSS_SOFT;
    pmodMtdsSpi.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_32;
    pmodMtdsSpi.Init.FirstBit = SPI_FIRSTBIT_MSB;
    pmodMtdsSpi.Init.TIMode = SPI_TIMODE_DISABLE;
    pmodMtdsSpi.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
    HAL_SPI_Init(&pmodMtdsSpi);
}
```

Listing 2. Transmisja i odczyt bajtu za pośrednictwem interfejsu SPI

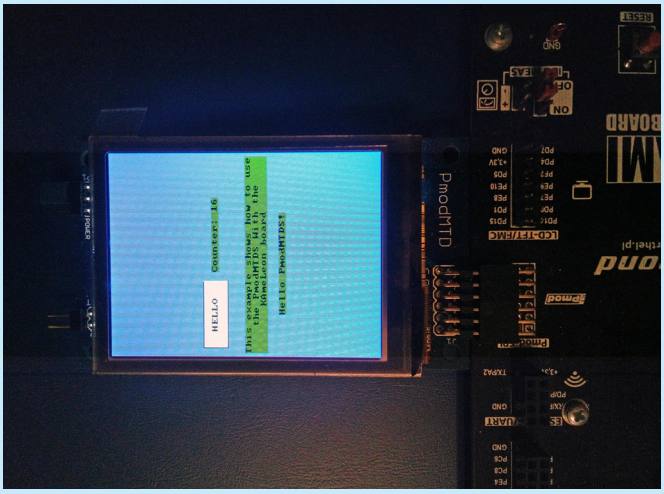
```
uint8_t MtdsHalPutSpiByte(uint8_t bSnd)
{
    uint8_t bRcv;
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_RESET);
    HAL_SPI_TransmitReceive(&pmodMtdsSpi, &bSnd, &bRcv, 1, 100);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
    return bRcv;
}
```

Listing 3. Generowanie bitmapy za pomocą funkcji warstwy mtds

```
HDS hds;
HBMP hbmp;
hds  = mtds.GetDs();
hbmp = mtds.CreateBitmap(80, 30, 16);
mtds.SetDrawingSurface(hds, hbmp);
mtds.SetBgColor(hds, clrWhite);
mtds.SetFgColor(hds, clrBlack);
mtds.Rectangle(hds, 0, 0, 100, 50);
mtds.SetPen(hds, penSolid);
mtds.SetFont(hds, hfntConsole);
mtds.TextOut(hds, 20, 10, 5, „HELLO”);
mtds.ReleaseDs(hds);
```

Listing 4. Tworzenie tekstu i przycisku oraz obsługa panelu dotykowego w funkcji main

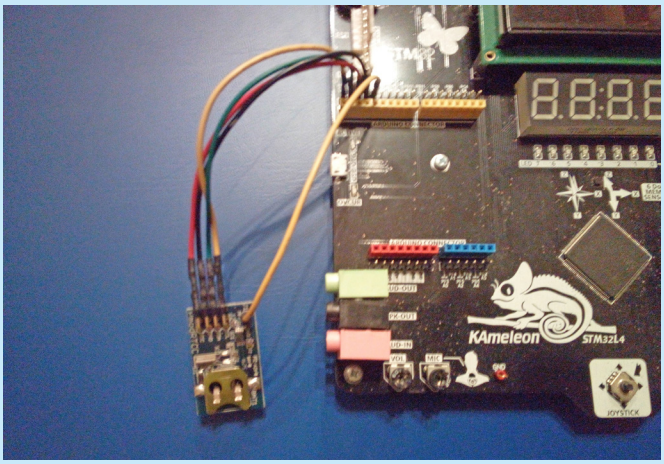
```
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    mydisp.begin();
    mydisp.clearDisplay(clrMedBlueGray);
//...
//Tu tworzenie bitmapy z Listingu 3.
//...
    mydisp.createButton(0, hbmp, hbmp, 10, 110);
    mydisp.enableButton(0, true);
    mydisp.drawButton(0, BUTTON_UP);
    mydisp.setForeground(clrBlack);
    mydisp.setBackground(clrDkGreen);
    mydisp.setPen(penSolid);
    mydisp.setTransparency(false);
    mydisp.drawText((char*) „Hello PmodMTDS!”, 50, 200);
    mydisp.drawText((char*) „This example shows how to use”, 4,
160);
    mydisp.drawText((char*) „    the PmodMTDS With the    „, 4,
169);
    mydisp.drawText((char*) „       KAmeLeon board        „, 4,
178);
    char textBuffer[32];
    int touchCounter = 0;
    while (1)
    {
        mydisp.checkTouch();
        if (mydisp.isTouched(0))
        {
            sprintf(textBuffer, „Counter: %d”, touchCounter++);
            mydisp.drawText(textBuffer, 100, 120);
        }
    }
}
```



Fotografia 3. Efekt działania przykładowej aplikacji dla modułu PmodMTDS

Tabela 4. Sposób podłączenia modułu PmodRTCC do złącza ARDUINO CONNECTOR zestawu KAmeLeon

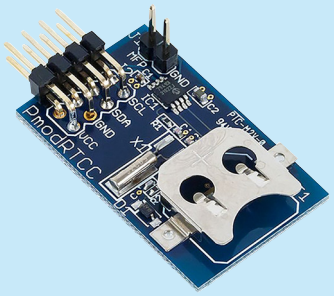
Sygnał	Numer pinu PmodRTCC	Numer pinu KAmeLeon ARDUINO CONNECTOR	Pin mikrokontrolera
SCL	1 (J2)	D15	PF14
SDA	2 (J2)	D14	PF15
MFP	1 (J1)	D13	PB10



Fotografia 5. Moduł PmodRTCC podłączony do płytki KAmeLeon

Listing 5. Konfiguracja interfejsu I²C do komunikacji z modułem PmodRTCC

```
void RTCCI2C::begin()
{
    __HAL_RCC_I2C4_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();
    __HAL_RCC_GPIOF_CLK_ENABLE();
    GPIO_InitTypeDef GPIO_InitStruct;
    GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Pin = GPIO_PIN_10;
    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_AF_OD;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF4_I2C4;
    GPIO_InitStruct.Pin = GPIO_PIN_14 | GPIO_PIN_15;
    HAL_GPIO_Init(GPIOF, &GPIO_InitStruct);
    this->i2c.Instance = I2C4;
    this->i2c.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
    this->i2c.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
    this->i2c.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
    this->i2c.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
    this->i2c.Init.OwnAddress1 = 0x01;
    this->i2c.Init.Timing = 0x10563046;
    HAL_I2C_Init(&this->i2c);
}
```



Fotografia 4. Wygląd modułu PmodRTCC

wyświetlany na ekranie. Efekt działania aplikacji został przedstawiony na **fotografii 3**.

PmodRTCC jest drugim z prezentowanych w tej części modułów (**fotografia 4**). Jest to zegar czasu rzeczywistego z kalendarzem oparty na układzie Microchip MCP79410. Układ ten umożliwia dodatkowo ustawienie dwóch alarmów, generowanie fali prostokątnej oraz korzystanie z pamięci EEPROM (128B) i SRAM (64B). Do obsługi modułu PmodRTCC została udostępniona biblioteka RTCCI2C dostępna na stronie producenta: ***http://bit.ly/2Uj9SJn***. Jest ona przeznaczona dla środowiska MPIDE, dlatego na potrzeby opisywanego przykładu musiała ona zostać modyfikowana tak, aby mogła być użyta w środowisku Atollic z mikrokontrolerem STM32L496ZGT6.

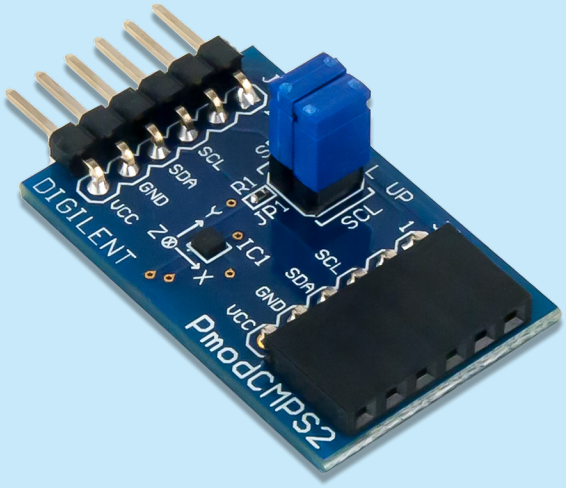
Moduł PmodRTCC ma 8-pinowe złącze dla interfejsu I²C (J2) i 2-pinowe złącze J1 zawierające piny MFP (Multi-Function Pin) oraz GND. Pin MFP może pełnić różne funkcje, zależnie od konfiguracji układu – w prezentowanym przykładzie będzie on źródłem przerwań wywołanych wystąpieniem alarmu. Sygnały modułu PmodRTCC zostały podłączone do złącza oznaczonego jako ARDUINO CONNECTOR na płytce KAmeLeon zgodnie z **tabelą 4** i **fotografią 5**.

Za obsługę interfejsu I²C odpowiedzialne są trzy zmodyfikowane metody klasy RTCCI2C: begin, readValue i writeValue, znajdujące się w bibliotece RTCCI2C. Pierwsza z nich, przedstawiona na **listingu 5**, jest odpowiedzialna za konfigurację interfejsu I2C4 oraz pinu PB10 do obsługi przerwania. Przerwanie jest wykrywane na zboczu opadającym sygnału MFP w momencie wystąpienia alarmu. Dla komunikacji I²C konfigurowane są dwa piny – PF14 i PF15 oraz interfejs I2C4. Dla tego ostatniego konieczne jest podanie wartości rejestru TIMINGR, przedstawionego na **rysunku 6**. Jest on odpowiedzialny za generowanie odpowiednich przebiegów czasowych na liniach SDA i SCL: SCLL i SCLH definiują długość stanu niskiego i wysokiego

Tabela 5. Metody klasy RTCCI2C użyte w przykładzie

Metoda klasy RTCCI2C	Opis
startClock/stopClock	Uruchomienie/zatrzymanie zegara przez włączenie/wyłączenie oscylatora.
setSec/getSec	Zapis/odczyt sekund zegara lub alarmu.
setMin/getMin	Zapis/odczyt minut zegara lub alarmu.
setHour/getHour	Zapis/odczyt godziny zegara lub alarmu.
setDay/getDay	Zapis/odczyt dnia tygodnia (0x01 – 0x07) zegara lub alarmu.
setDate/getDate	Zapis/odczyt dnia miesiąca (0x01 – 0x32) zegara lub alarmu.
setMonth/getMonth	Zapis/odczyt miesiąca (0x01 – 0x12) zegara lub alarmu.
setYear/getYear	Zapis/odczyt roku (0x00 – 0x99) zegara lub alarmu.
enableAlarm	Włączenie alarmu
getAmPm	Czas dla zegara 12-godzinnego (AM/PM).

Rysunek 6. Definicja rejestru I2C_TIMINGR (źródło: dokumentacja mikrokontrolera STM32L496ZG)



Fotografia 7. Wygląd modułu PmodCMPS2

sygnału SCL, SCLDEL oznacza opóźnienie pomiędzy ustawieniem wartości na linii danych a zboczem narastającym sygnału zegarowego, natomiast SDADEL opóźnienie pomiędzy zboczem opadającym zegara a zmianą stanu na linii danych. Pole PRESC wyznacza dzielnik sygnału zegarowego taktującego układ I²C mikrokontrolera.

Funkcje odpowiedzialne za komunikację z modułem PmodRTCC zostały przedstawione na **listingu 6**. Wykorzystują one blokujące wywołania HAL_I2C_Master_Transmit i HAL_I2C_Master_Receive biblioteki STM32Cube.

Oprócz wymienionych funkcji klasa RTCCI2C zawiera także metody konfigurujące poszczególne rejestry układu MCP79410. W **tabeli 5** wymienione zostały te użyte w kodzie przykładu. Wszystkie wartości są zapisywane i odczytywane w kodzie BCD.

Funkcja main przykładowej aplikacji została przedstawiona na **listingu 7**. Konfiguruje ona moduł PmodRTCC i ustawia datę: niedziela, 4 marca 2018 23:15:30. Następnie ustawiany jest alarm na 10 sekund od zdefiniowanej daty. W pętli głównej co sekundę odczytywana jest data i godzina za pomocą funkcji pomocniczej printTime. Odczytane wartości są od razu wysyłane na port szeregowy LPUART1. Alarm jest sygnalizowany przez zapalenie diody LED1 podłączonej do pinu PC7. Port szeregowy LPUART1 jest konfigurowany i obsługiwany za pomocą kodu znajdującego się w pliku serial.c, natomiast sterownik diody został umieszczony w pliku led.c.

PmodCMPS2 jest ostatnim z prezentowanych w tej części cyklu modułów. PmodCMPS2 (**fotografia 7**) wyposażono w układ Memsic MMC34160PJ. Jest to 3-osiowy czujnik pola magnetycznego o zakresie ±16G i konfigurowanej rozdzielczości 12, 14 lub 16 bitów. Do obsługi modułu została udostępniona biblioteka przygotowana z myślą o platformie Arduino. Kod biblioteki może być pobrany bezpośrednio ze strony firmy Digilent ***http://bit.ly/2CI8jL4***. W jej skład wchodzi pojedyncza klasa – CMPS2 służąca do konfiguracji i komunikacji z modułem PmodCMPS2. Oprócz podstawowych funkcji do odczytu pomiarów zawiera ona także metodę readHeading, umożliwiającą odczyt kierunku kompasowego, jednak

Listing 6. Funkcje realizujące odczyt i zapis rejestru

```
uint8_t RTCCI2C::readValue(uint8_t address)
{
    uint8_t value = 0;
    HAL_I2C_Master_Transmit(&this->i2c, RTCC_I2C_ADDR, &address, 1, 100);
    HAL_I2C_Master_Receive(&this->i2c, RTCC_I2C_ADDR, &value, 1, 100);
    return value;
}

void RTCCI2C::writeValue(uint8_t address, uint8_t value)
{
    uint8_t data[2] = {address, value};
    HAL_I2C_Master_Transmit(&this->i2c, RTCC_I2C_ADDR, data, 2, 100);
}
```

Listing 7. Kod główny przykładowej aplikacji

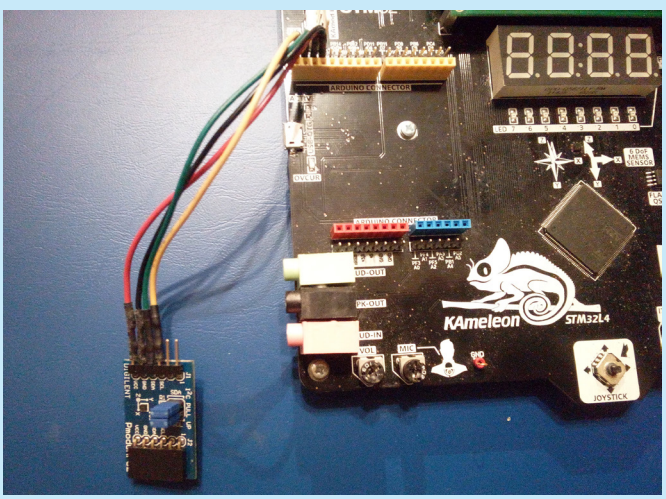
```
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    Serial_Config();
    Led_Config();
    RTCCI2C myRTCC;
    myRTCC.begin();
    //set the real time clock
    myRTCC.stopClock();
    myRTCC.setSec(RTCC_RTCC, 0x30);
    myRTCC.setMin(RTCC_RTCC, 0x15);
    myRTCC.setHour(RTCC_RTCC, 0x11, RTCC_PM);
    myRTCC.setDay(RTCC_RTCC, 0x07);
    myRTCC.setDate(RTCC_RTCC, 0x04);
    myRTCC.setMonth(RTCC_RTCC, 0x03);
    myRTCC.setYear(0x18);
    // Set the alarm for 10 seconds after written time.
    myRTCC.setSec(RTCC_ALM0, 0x40);
    myRTCC.setMin(RTCC_ALM0, 0x15);
    myRTCC.setHour(RTCC_ALM0, 0x11, RTCC_PM);
    myRTCC.setDay(RTCC_ALM0, 0x07);
    myRTCC.setDate(RTCC_ALM0, 0x04);
    myRTCC.setMonth(RTCC_ALM0, 0x03);
    myRTCC.enableAlarm(RTCC_ALM0, RTCC_ALMC2 | RTCC_ALMC1 | RTCC_ALMC0);
    myRTCC.startClock();
    while(1)
    {
        HAL_Delay(1000);
        printTime(&myRTCC, RTCC_RTCC);
    }
}
```

Listing 8. Inicjalizacja interfejsu I2C dla modułu PmodCMPS2

```
void CMPS2::init()
{
    __HAL_RCC_I2C4_CLK_ENABLE();
    __HAL_RCC_GPIOF_CLK_ENABLE();
    GPIO_InitTypeDef GPIO_InitStruct;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF4_I2C4;
    GPIO_InitStruct.Pin = GPIO_PIN_14 | GPIO_PIN_15;
    HAL_GPIO_Init(GPIOF, &GPIO_InitStruct);
    this->i2cHandle.Instance = I2C4;
    this->i2cHandle.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
    this->i2cHandle.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
    this->i2cHandle.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
    this->i2cHandle.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
    this->i2cHandle.Init.OwnAddress1 = 0x01;
    this->i2cHandle.Init.Timing = 0x10563046;
    HAL_I2C_Init(&this->i2cHandle);
    this->writeReg(intConReg0, 0x20);
    HAL_Delay(10);
    this->writeReg(intConReg1, 0x00);
}
```

Tabela 6. Sposób podłączenia modułu PmodCMPS2 do złącza ARDUINO CONNECTOR zestawu KAmeLeon

Sygnał	Numer pinu PmodCMPS2	Numer pinu KAmeLeon ARDUINO CONNECTOR	Pin mikrokontrolera
SCL	3 (J1)	D15	PF14
SDA	4 (J1)	D14	PF15



Fotografia 8. Moduł PmodCMPS2 podłączony do płytki KAmeLeon

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

ze względu na brak kompensacji pochylenia układu daje ona poprawne wyniki wyłącznie podczas pracy na płaszczyźnie XY.

Moduł PmodCMPS2 został wyposażony w 6-pinowe złącze interfejsu I²C z dwoma nieużywanymi pinami. W przykładzie zostało ono przyłączone do magistrali I²C wyprowadzonej na złącze ARDUINO CONNECTOR na płytce KAmeLeon, tak jak na **fotografii 8**. Opis połączeń znajduje się w **tabeli 6**. Za konfigurację interfejsu I²C jest odpowiedzialna metoda init klasy CMPS2, przedstawiona na **listingu 8**. Została ona zmodyfikowana tak, aby mogła być użyta z mikrokontrolerem STM32L496ZGT6.

Konfiguracja I2C4 jest analogiczna do tej z poprzedniego przykładu, jednak tym razem konfigurowane są także rejestry Internal Control 0 (0x07) i Internal Control 1 (0x08) – pojedynczy pomiar z rozdzielczością 16 bitów i czasem trwania 7,92 ms.

Do komunikacji z układem MMC34160PJ wykorzystywane są funkcje readReg i writeReg przedstawione na **listingu 9**. Tym razem do ich realizacji użyte zostały biblioteczne funkcje HAL_I2C_Mem_Read i HAL_I2C_Mem_Write umożliwiające dostęp do określonego adresu w ramach jednego urządzenia.

Kod głównej funkcji programu widoczny jest na **listingu 10**. Znajduje się w nim inicjalizacja modułu PmodCMPS2 (myCMPS.init) oraz cykliczny odczyt kierunku do 10 ms (myCMPS.readHeading).

Listing 9. Funkcje do zapisu i odczytu rejestrów układu MMC34160PJ
```
uint8_t CMPS2::readReg(uint8_t addr)
{
    uint8_t data;
    HAL_I2C_Mem_Read(&this->i2cHandle, this->deviceID, addr, 1, &data, 1, 100);
    return data;
}

void CMPS2::writeReg(uint8_t addr, uint8_t data)
{
  HAL_I2C_Mem_Write(&this->i2cHandle, this->deviceID, addr, 1, &data, 1, 100);
}
```

Listing 10. Funkcja main aplikacji przykładowej dla PmodCMPS2
```
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    Led_Config();
    CMPS2 myCMPS;
    myCMPS.init();
    while(1)
    {
        HAL_Delay(10);
        int degree = myCMPS.readHeading( );
        if(degree < 10 || degree > 350) Led_TurnOn(LED1);
        else Led_TurnOff(LED1);
    }
}
```

Dodatkowo, jeżeli kierunek kompasowy znajduje się przedziale od 350 do 10 stopni, to zapalana jest dioda LED1 podłączona do linii PC7 i sygnalizująca wykrycie północy.

Krzysztof Chojnowski