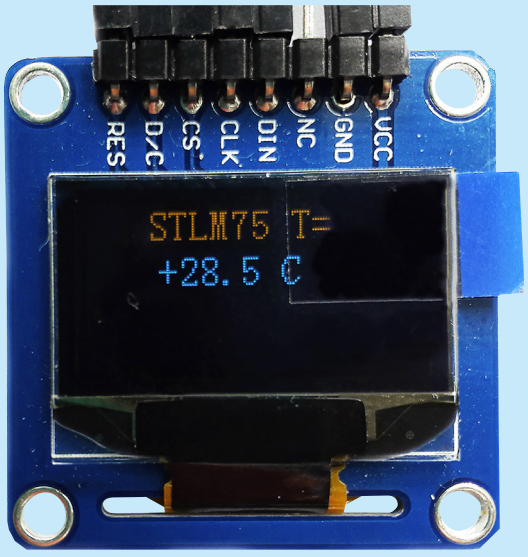
Renesas Synergy – interfejsy szeregowe (3)

Interfejsy szeregowe są bardzo ważnym elementem budowania systemów mikroprocesorowych. Wiele elementów zewnętrznych takich jak: wyświetlacze, czujniki, moduły komunikacyjne i inne mają wbudowane szeregowe interfejsy komunikacyjne z jednej strony, a mikrokontrolery układy peryferyjne obsługujące transmisję szeregową z drugiej strony. Moduły komunikacyjne mikrokontrolerów są często bardzo rozbudowane. Przychodzące i wysyłane dane mogą być przesyłane kanałami DMA lub są buforowane w FIFO. Bardziej rozbudowane interfejsy na przykład I2C mogą pracować jako master w magistrali z wieloma masterami. Konfigurowanie tego typu peryferii jest prawdziwą udręką programistów. Konieczność zapisania wielu rejestrów konfiguracyjnych i wzajemne czasami skomplikowane zależności pomiędzy bitami konfiguracyjnymi powodują, że łatwo się pomylić i bardzo trudno znaleźć przyczynę pomyłki. Żeby ułatwić i przyspieszyć konfigurację stosuje się dwa wzajemnie się uzupełniające elementy. Pierwszy z nich to najczęściej graficzny konfigurator, a drugi to gotowe biblioteki warstwy HAL.

Programową obsługę termometru rozpoczniemy od procedury otwarcia i procedury zamknięcia kanału I²C drivera STM75 (**listing 13**). Funkcja Open został już opisana wyżej. Warto testować w programie wywołującym tę funkcję, czy został zwrócony kod błędu SSP_SUCCES. Jeżeli nie, to kanał nie zostanie otwarty lub zamknięty i użytkownik musi na to zareagować.

Po zainicjowaniu modułu IIC trzeba zainicjować termometr STLM75 przez zapisanie rejestru konfiguracji. Całą procedurę konfiguracji termometru wraz otwarciem kanału I²C pokazano na **listingu 14**. Zapisanie rejestru konfiguracyjnego polega na wysłaniu dwóch bajtów: adresu rejestru (0x01) i bajtu konfiguracji. Wspominałem już, że do testowania zakończenia transmisji można użyć mechanizmu potwierdzenia callback, jak to zrobiono w wypadku interfejsu SPI. Z opisu funkcji drivera IIC wynika, że callback nie jest niezbędny i jeżeli nie zostanie zdefiniowany (nazwa NULL), to funkcje transferu będą działać jako blokujące. Oznacza to, że funkcja będzie czekać na zakończenie transmisji. W swoim programie postanowiłem użyć callback i testować zakończenie transmisji. Jak w wypadku SPI, moje procedury po wykryciu błędu lub braku potwierdzenia zakończenia transmisji wchodzą w pętle nieskończone. Użytkownik powinien zadbać o obsługę zdarzeń tego typu, aby program nie był trwale blokowany.

Na **listingu 15** zamieszczono procedurę callback testującą zdarzenia zakończenia wysyłania danych I2C_EVENT_TX_COMPLETE, zakończenia odbierania danych I2C_EVENT_RX_COMPLETE i niepowodzenia (przerwania) transmisji I2C_EVENT_ABORDED. Do testowania



Rysunek 28. Pomiar temperatury

Listing 13. Funkcje otwarcia i zamknięcia kanału I²C drivera STM75 IIC

```
//otwarcie kanału I2C drivera stm75 IIC
ssp_err_t OpenI2CSTLM75(void)
{
 ssp_err_t   err;
    err=R_RIIC_MasterOpen(STM75.p_ctrl, STM75.p_cfg);
    return(err);
}

//zamknięcie kanali I2C drivera STM75 IIC
ssp_err_t CloseI2CSTLM75(void)
{
 ssp_err_t   err;
    err=R_RIIC_MasterClose(STM75.p_ctrl, STM75.p_cfg);
    return(err);
}
```

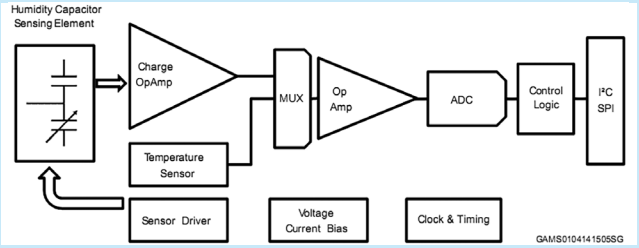
Listing 14. Konfigurowanie termometru STLM75

```
//************************************************
//inicjalizacja interfejsu I2C
//************************************************

ssp_err_t STM75Config (uint8_t config)
{
    ssp_err_t   err;
        uint8_t buf[2];
        buf[0]=0x01;//adres rejestru konfiguracji
        buf[1]=config;//bajt konfiguracyjny
        OpenI2CSTLM75();//otwarcie kanału I2C drivera STLM75
        i2c_data_tx=0;
        err=R_RIIC_MasterWrite(STM75.p_ctrl,buf,2,false);
        if(err!=SSP_SUCCESS)
          while(1);
        while(i2c_data_tx==0);
        CloseI2CSTLM75();//zamkniecie kanału I2C drivera STLM75
 }       return(err);
```

Listing 15. procedura callback dla drivera IIC termometru STM75

```
//*********************************************************
//prototyp funkcji calback dla kontroli transmisji I2C
//*********************************************************
void STM75_callback(i2c_callback_args_t * p_args)
{
    if(I2C_EVENT_TX_COMPLETE == p_args->event)
    {
        i2c_data_tx=1;
    }
    if(I2C_EVENT_RX_COMPLETE == p_args->event)
    {
       i2c_data_rx=1;
    }
    if(I2C_EVENT_ABORTED == p_args->event)
    {
        i2c_data_rx=10;i2c_data_tx=10;
    }
}
```



Rysunek 29. Schemat blokowy czujnika HTS221

Listing 16. Sekwencja odczytywania 2 rejestrów temperatury z czujnika STM75

```
ssp_err_t   err;
uint8_t data_write[6];
uint8_t data_read[6];

OpenI2CSTLM75();//otwarcie kanału I2C drivera STLM75
data_write[0] = 0;//rejestr temperatury
stlm_data_tx=0;
err=R_RIIC_MasterWrite(STM75.p_ctrl,data_write,1,true);//bez
bitu stopu
if(err!=SSP_SUCCESS) while(1);
while(stlm_data_tx==0); stlm_data_rx=0;
err=R_RIIC_MasterRead(STM75.p_ctrl,data_read,2,false);//z bitem
stopu
if(err!=SSP_SUCCESS) while(1);
while(stlm_data_rx==0);
```

zakończenia transmisji zdefiniowałem dwie zmienne globalne stlm_data_rx, oraz stlm_data_tx. Przed wywołaniem funkcji zapisu, lub odczytu danych przez driver IIC odpowiednia zmienna jest zerowana. Po zakończeniu transferu danych i wystawieniu sekwencji STOP funkcja callback wpisuje do tych zmiennych „1" i program „wie", że można transferować kolejne dane lub zamknąć kanał komunikacji.

Po prawidłowym otwarciu kanału i skonfigurowaniu termometru można wysyłać sekwencję odczytu rejestrów temperatury. Jak pokazano na **rysunku 27**, składa się ona z dwóch części. Pierwsza to wysłanie sekwencji START, adresu Slave z bitem R/W=0 oraz bajtu adresu rejestru 0x00 bez bitu STOP W drugiej części trzeba ponownie wysłać sekwencję START, adres slave z bitem R/W=1, oraz odczytać z magistrali 2 bajty rejestru temperatury. Całość kończy się sekwencją STOP. Na **listingu 16** pokazano fragment programu wykonujący te czynności.

Mamy już wszystkie elementy składowe do napisania funkcji, która:

- Otwiera kanał transmisyjny I²C.
- Konfiguruje termometr STLM75.
- Odczytuje rejestr temperatury.
- Wykonuje konwersję zgodnie z zasadą pokazaną na rys. 23.
- Wyświetla temperaturę na ekranie wyświetlacza OLED.
- Zamyka kanał I²C.

Listing 17 prezentuje kompletną procedurę odczytującą i wyświetlającą temperaturą otoczenia mierzoną przez STLM75. Każde wywołanie **STLMTempRead()** otwiera i zamyka kanał I²C drivera STM75. Używając tylko tego drivera można by było raz otworzyć kanał i go nie zamykać, jednak przy planowanej obsłudze innych układów dołączonych do magistrali niezamknięty kanał nie pozwoli na otworzenie kanału kolejnego drivera i zostanie przy próbie otwarcia zostanie zgłoszony kod błędu SSP_ERR_IN_USE.

W czasie testu procedura **STLMTempRead()** była wywoływana w niekończonej pętli. Dotkniecie czujnika palcem powodowało zwiększanie odczytywanej temperatury. Wynik działania programu z **listingu 18** pokazano na **rysunku 28**.

Zobaczymy teraz jak wykonać obsługę kolejnego układu – czujnika wilgotności HTS221 dołączonego do tej samej magistrali I²C.

Master	START	Adres Slave		Adres rejestru		zapisywane dane		STOP
Slave			ACK		ACK		ACK	

Rysunek 30. Sekwencja zapisania rejestru HTS221

Master	START	AdresSlave R/W=0		adres rejestru		START	Adres Slave R/W=1			NACK	STOP
Slave			ACK		ACK			ACK	odczytane dane		

Rysunek 31. Sekwencja odczytania rejestru HTS221

Listing 17. Odczytanie, konwersja i wyświetlenie temperatury mierzonej przez STLM75

```
ssp_err_t STLMTempRead(uint8_t x, uint8_t y)
{
    ssp_err_t   err;
    uint8_t data_write[6];
    uint8_t data_read[6];
    OpenI2CSTLM75();//otwarcie kanału I2C drivera STLM75
    data_write[0] = 0;//rejestr temperatury
    stlm_data_tx=0;
    err=R_RIIC_MasterWrite(STM75.p_ctrl,data_write,1,true);//
bez bitu stopu
    if(err!=SSP_SUCCESS) while(1);
    while(stlm_data_tx==0);
    stlm_data_rx=0;
    err=R_RIIC_MasterRead(STM75.p_ctrl,data_read,2,false);//z
bitem stopu
    if(err!=SSP_SUCCESS) while(1);
    while(stlm_data_rx==0);
    // Wyliczenie temperatury w stopniach Celsjusza
    int tempval = (int)((int)data_read[0] << 8) | data_read[1];
    tempval >>= 7;
    if (tempval <= 256) TempCelsiusDisplay[0] = '+';
    else TempCelsiusDisplay[0] = '-';
    tempval = 512 - tempval;
}

// wartośc po przecinku z dokładnością do 0,5stopnia
if (tempval & 0x01) TempCelsiusDisplay[5] = 0x05 + 0x30;
else TempCelsiusDisplay[5] = 0x00 + 0x30;
// wartośc całkowita temperatury
tempval >>= 1;
TempCelsiusDisplay[1] = (tempval / 100) + 0x30;
if(TempCelsiusDisplay[1]==0x30)
{
    TempCelsiusDisplay[1]=TempCelsiusDisplay[0];
    TempCelsiusDisplay[0]=' ';}
    TempCelsiusDisplay[2] = ((tempval % 100) / 10) + 0x30;
    TempCelsiusDisplay[3] = ((tempval % 100) % 10) + 0x30;
    //wyświetlenie na ekranie
    DispTxtRAM(x,y, TempCelsiusDisplay,16,1);
    CloseI2CSTLM75();//zamkniecie kanału I2C drivera STLM75
}
```

Listing 18. Pomiar temperatury

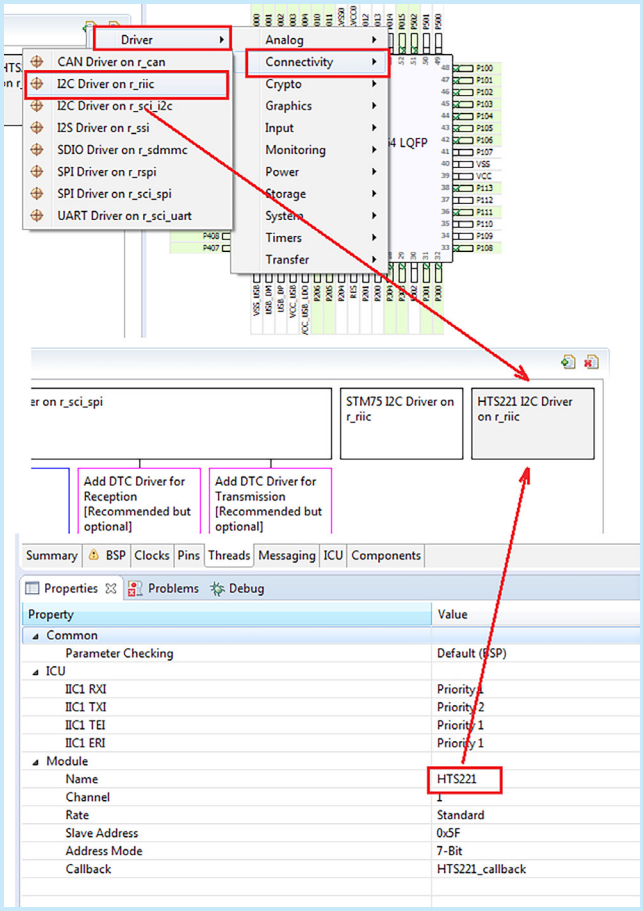
```
InitOled();
DispTxt(24,0, „STLM75 T=", 16,1);
while(1)
{
    STLMTempRead(20,16);
    RefreshRAM();}
    while(1);
    ...
```

Pomiar wilgotność względnej powietrza jest wykorzystywany w stacjach meteorologicznych, urządzeniach AGD (np. lodówkach) czujnikach wentylatorów łazienkowych, sterownikach klimatyzacji, układów automatyki domowej itp. Umieszczony na płytce czujnik HTS221 mierzy wilgotność w zakresie od 20% do 80% rH z dokładnością ±4,5%. Pomiar może być wykonywany z częstotliwością od 1 pomiaru na sekundę do 25 pomiarów na 2 sekundy. Oprócz pomiaru wilgotności czujnik mierzy temperaturę w zakresie -40…+120°C z dokładnością ±0,5°C w zakresie +15…+40°C oraz ±1°C w zakresie 0…+60°C. Schemat blokowy układu pokazano na **rysunku 29**.

Podobnie jak dla innych czujników, konfiguracja parametrów pracy i odczytywanie mierzonych wartości odbywa się przez zapisywanie i odczytywanie rejestrów wewnętrznych. W tym celu będziemy potrzebowali dwóch funkcji: odczytania zawartości rejestru o podanym adresie i zapisania rejestru o podanym adresie.

Zapisanie rejestru rozpoczyna się od wysłania przez mikrokontroler sekwencji START, a po niej adresu Slave 0x5F. Potwierdzenia adresu przez HTS221 pozwala na wysłanie przez mikrokontroler adresu rejestru, a po nim zapisywanej danej (**rysunek 30**). Odczytanie rejestru rozpoczyna się od wysłania sekwencji START i adresu Slave z bitem R/W=0 (zapis) i bajt adresu rejestru. Potem jest wysyłana powtórna sekwencja START, adres Slave z bitem R/W=1, a po nim odczytywany jest jeden bajt zawartości zaadresowanego rejestru (**rysunek 31**). Podobnie jak w przypadku termometru STML75 będziemy potrzebowali dodania i konfiguracji funkcji drivera IIC.

Konfiguracja jest podobna do drivera dla STLM75. Co oczywiste, inny jest adres Slave (0x5f) i inne nazwy kanału oraz funkcji callback. Do komunikacji z układem będziemy potrzebowali funkcji: otwarcia i zamknięcia kanału IIC, konfiguracji układu, funkcji zapisania i odczytywania rejestrów (**rysunek 32**). Funkcje otwarcia i zamknięcia kanału IIC pokazano na **listingu 19**. Sterowanie układem, oraz wyniki pomiaru i zawartości rejestrów kalibracyjnych



Rysunek 32. dodanie drivera IIC do obsługi układu HTS221

Listing 19. Funkcje otwarcia i zamknięcia kanału IIC

```
ssp_err_t OpenI2CHTS221(void)
{
    ssp_err_t   err;
    err=R_RIIC_MasterOpen(HTS221.p_ctrl, HTS221.p_cfg);
    return(err);
}

ssp_err_t CloseI2CHTS221(void)
{
    ssp_err_t   err;
    err=R_RIIC_MasterClose(HTS221.p_ctrl, HTS221.p_cfg);
    return(err);
}
```

są umieszczone w adresowanych rejestrach. Wykaz tych rejestrów pokazano na **rysunku 33**. Pomiary wilgotności i temperatury mogą być wykonywane na żądanie, po wysłaniu polecenia (one shot) lub w sposób ciągły z określoną częstotliwością. Do programowania częstotliwości pomiarów jest przeznaczony rejestr CTRL_REG1 o adresie 0x20, pokazany na **rysunku 34**. Bit PD trego rejestru jest przeznaczony do wprowadzania układu w stan obniżonego poboru mocy, a bit BDU określa czy rejestr z danymi wyjściowymi wilgotności i temperatury ma być odświeżany automatycznie po wykonaniu pomiarów, czy tez po odczytaniu przez mikrokontroler 8 starszych bitów wynikupoprzedniego pomiaru. Po właczeniu zasilania bit PD jest wyzerowany i żeby można było wykonywac pomiary trzeba do PD wpisać „1".

Wyzerowanie bitów ODR1 i ODR2 wprowadza układ w tryb pomiaru na żądanie. Żeby wyzwolić taki pomiar trzeba ustawić bit ONE_SHOT w rejestrze CTRL_REG2, jak pokazano na **rysunku 35**. Po wyzwoleniu pomiaru (również w trybie pomiaru ciągłego) trzeba testować czy wynik pomiaru został zapisany do rejestrów wyjściowych. Żeby to zrobić trzeba odczytać zawartość rejestru statusowego STATUS_REG. Ustawienie bitu H_DA (bit b1 STATUS_REG) oznacza, że wynik pomiaru wilgotności został zapisany do rejestrów wyjściowych HUMIDITY_OUT_L i HUMIDITY_OUT_H, a ustawienie bitu T_DA (bit b0 STATUS_REG) oznacza, że wynik pomiaru temperatury został wpisany

Rejestr	ADRES hex	POR	Funkcja
WHO_AM_I	0F	0xBC	ID układu
AV_CONF	10	0x7A	Rejestr konfiguracji rozdzielczości
CTRL_REG1	20	0	Rejestr kontrolny 1
CTRL_REG2	21	0	Rejestr kontrolny 2
CTRL_REG3	22	0	Rejestr kontrolny 3
STATUS_REG	27	0	Rejestr statusu
HUMIDITY_OUT_L	28	0	Młodsza cześć rejestru mierzonej wilgotności
HUMIDITY_OUT_H	29	0	Starsza cześć rejestru mierzonej wilgotności
TEMP_OUT_L	2A	0	Najstarsza cześć rejestru mierzonego ciśnienia
TEMP_OUT_H	2B	0	Najmłodsza cześć rejestru mierzonej temperatury

Rysunek 33. Wykaz rejestrów sterujących i rejestrów wyniku pomiaru

Listing 20. Zapisanie danej do rejestru o adresie w argumencie addr

```
//addr - adres rejestru
//data - dabe do zpisania
ssp_err_t HTS221WriteReg(uint8_t addr, uint8_t data)
{
    ssp_err_t   err;
    uint8_t buf[2];
    buf[0]=addr;//adres rejestru
    buf[1]=data;//dana do zapisania
    hts_data_tx=0;
    err=R_RIIC_MasterWrite(HTS221.p_ctrl,buf,2,false);
    if(err!=SSP_SUCCESS) while(1);
    while(hts_data_tx==0); return(err);
}
```

Listing 21. Odczytywanie danych z rejestru o adresie w argumencie addr

```
uint8_t HTS221ReadReg(uint8_t addr)
{
    ssp_err_t   err;
    uint8_t data_write[3];
    uint8_t data_read[3];
    data_write[0] = addr;//adres rejestru
    hts_data_tx=0;
    err=R_RIIC_MasterWrite(HTS221.p_ctrl,data_write,1,true); //
bez bitu stopu
    if(err!=SSP_SUCCESS) while(1);
    while(hts_data_tx==0); hts_data_rx=0;
    err=R_RIIC_MasterRead(HTS221.p_ctrl,data_read,1,false);//z
bitem stopu
    if(err!=SSP_SUCCESS) while(1);
    while(hts_data_rx==0); return(data_read[0]);
}
```

B7	B6	B5	B4	B3	B2	B1	B0
PD	Res	res	res	res	BDU	ODR1	ODR2

PD Power Down PD=0 power down, PD=1 tryb aktywny
BDU – odświeżanie rejestru wyjściowego BDU=0 dane odświeżane ciągle, BDU=1 dane odświeżane po odczycie rejestrów ciśnienia i temperatury
ODR1:ODR1 – częstotliwość odczytywania danych wyjściowych
00 – pomiar na żądanie One Shot
01 – 1Hz
10- 7Hz
11- 12,5Hz

Rysunek 34. Rejestr CTRL_REG1

do rejestrów pomiaru temperatury. Operacje zapisywania i odczytywania rejestrów są pokazane na **listingach 20** i **21**.

W czasie testów czujnika skonfigurowałem układ do pracy z wyzwalaniem na żądanie i z ustawionym bitem BDU. Procedura inicjalizacyjną pokazano na **listingu 22**. W wielu czujnikach wartości odczytywane z wyjściowych rejestrów pomiarów trzeba przekonwertować z formatu U2, ewentualnie przeskalować lub dodać offset. W czujniku HTS221 jest inaczej. Pierwszą zasadniczą różnicą jest

B7	B6	B5	B4	B3	B2	B1	B0
BOOT	Res	res	res	res	res	HE-ATER	ONE_ SHOT

BOOT – ustawienie tego bitu powoduje przepisanie ustawień kalibracyjnych z wewnętrznej pamięci Flash do rejestrów kalibracyjnych
HEATER – ustawienie tego bitu włącza wewnętrzne podgrzewanie w celu usunięcia wilgoci z kondensacji pary wodnej znajdującej się na czujniku wilgotności. Wyzerowanie bitu powoduje wyłączenie podgrzewania.
ONE_SHOT – ustawienie tego bitu inicjuje jednokrotny pomiar wilgotności i temperatury. Po wykonaniu pomiaru ONE_SHOT jest sprzętowo zerowany

Rysunek 35 Rejestr CTRL_REG2

umieszczenie w pamięci układu szeregu dodatkowych rejestrów kalibracyjnych. Każdy układ w procesie produkcyjnym jest kalibrowany, a w jego nieulotnej pamięci Flash są zapisywane dane do kalibracji. W czasie sekwencji włączania zasilania układu dane kalibracyjne są przepisywane z pamięci Flash do rejestrów kalibracyjnych pokazanych na **rysunku 36**. W trakcie inicjalizacji układu mikrokontroler musi odczytać dane kalibracyjne, po to by je potem wykorzystać przy każdorazowym odczycie wilgotności i temperatury. Procedurę odczytu rejestrów kalibracyjnych pokazano na **listingu 23**. W wyniku działania tej funkcji są zapisane zmienne globalne pokazane na **listingu 24**. Te zmienne w połączeniu z 16-bitowymi danymi wyjściowymi będą służyły do wyliczenia mierzonej wartości. W wypadku temperatury wartości T0_cal, T1_cal, T0_degC i T1degC są współrzędnymi wyznaczającymi prostą kalibracji –pokazano to na **rysunku 37**.

Rejestr	Adres hex
H0_rH_x2	30
H1_rH_x2	31
T0_degC_x8	32
T1_degC_x8	33
T1/T0 msb	35
H0_T0_OUT	36
	37
H1_T1_OUT	3A
	3B
T0_OUT	3C
	3D
T1_OUT	3E
	3F

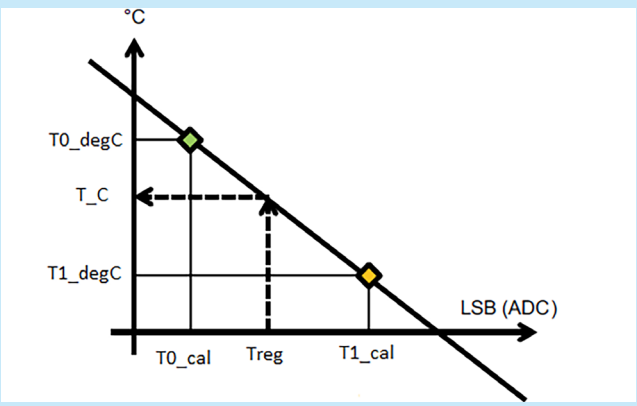
Rysunek 36. Rejestry kalibracyjne

Aby zmierzyć temperaturę należy:

- Wyzwolić pomiar.
- Poczekać na zakończenie pomiaru (odczytywanie rejestru statusu).
- Odczytać rejestry temperatury.
- Wyliczyć na podstawie odczytanych rejestrów i danych kalibracji mierzona temperaturę.
- Wyświetlić wynik na ekranie wyświetlacza.

Na **listingu 25** pokazano procedurę odczytywania rejestrów temperatury. Kompletna procedura inicjowania, odczytywania, konwersji wyświetlania wyniku jest pokazana na **listingu 26**. Do konwersji zmiennej typu float na znaki ASCII potrzebnej do wyświetlenia wyniku użyłem standardowej funkcji sprintf. W trakcie testów okazało się, że konwersja nie działa poprawnie. Zatrzymanie programu na pułapce programowej za wywołaniem sprintf oraz podejrzenie zawartości bufora str[] i zmiennej float Temperature wykazało, że znaki w buforze str zupełnie nie odpowiadają wartości zmiennej Temperature. Okazało się, że standardowe funkcje potrzebują do działania sterty. Projekt domyślnie ustawia rozmiar sterty na zero, zapewne po to, aby minimalizować użycie pamięci RAM.

Rozmiar sterty jest ustawiany w zakładce BSP konfiguratora projektu (**rysunek 38**). W okienku jest również umieszczona informacja o tym, ze funkcje standardowe wymagają sterty o minimalnym rozmiarze 4 kB. Po zmianie ustawień funkcja sprintf zaczęła działać poprawnie.



Rysunek 37. Kalibracja i obliczanie temperatury w °C

Listing 22. Inicjalizacja Higrometru

```
//inicjalizacja układu HTS221
//wyzwalanie pomiaru na żądanie
//bit PDU - ustawiony
void HTS221Init(void)
{
    OpenI2CHTS221();
    HTS221WriteReg(0x20,0 );//CTRL_REG1 Power Down
    HTS221Cal();//odczytanie wspolczynnikow kalibracji
    HTS221WriteReg(0x10, 0x1b);//
    HTS221WriteReg(0x20, 0x84);//CTRL_REG1 Power On, One Shot,
Update after read
    CloseI2CHTS221();
}
```

Listing 23. Odczytanie rejestrów kalibracyjnych

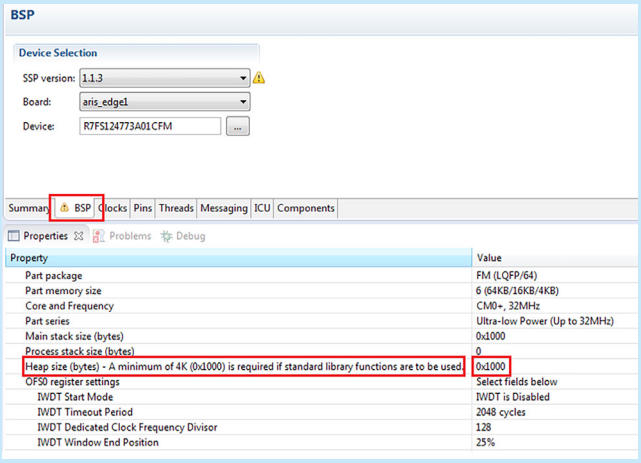
```
void HTS221Cal(void)
{
    int16_t T0,T1;
    struct
    {
        uint8_t H0_rH_x2;//addr 0x30
        uint8_t H1_rH_x2;//addr 0x31
        uint8_t  T0_degC_x8;//addr 0x32
        uint8_t  T1_degC_x8;//addr 0x33
        uint8_t  T1_T0_msb;//addr 0x35
        int16_t H0_T0_OUT;//addr 36, 37
        int16_t H1_T0_OUT;//addr 3a, 3b
        int16_t T0_OUT;//addr 3c, 3d
        int16_t T1_OUT;//addr 3e, 3f
    } cal;
    //współczynniki kalibracji dla wilgotnosci
    cal.H0_rH_x2=HTS221ReadReg(0x30);
    cal.H1_rH_x2=HTS221ReadReg(0x31);
    H0_rh=(float)cal.H0_rH_x2/2;
    H1_rh=(float)cal.H1_rH_x2/2;
    cal.H1_T0_OUT=HTS221ReadReg(0x3b);
    cal.H1_T0_OUT<<=8;
    cal.H1_T0_OUT|=HTS221ReadReg(0x3a);
    cal.H0_T0_OUT=HTS221ReadReg(0x37);
    cal.H0_T0_OUT<<=8;
    cal.H0_T0_OUT|=HTS221ReadReg(0x36);
    H0_cal=cal.H0_T0_OUT;
    H1_cal=cal.H1_T0_OUT;
    //wspolczynniki kalibracji dla temperatury
    cal.T0_degC_x8=HTS221ReadReg(0x32);
    cal.T1_T0_msb=HTS221ReadReg(0x35);
    T0=cal.T1_T0_msb&3;
    T0<<=8;
    T0=T0|cal.T0_degC_x8;
    T0_degC=(float)T0/8;
    cal.T1_degC_x8=HTS221ReadReg(0x33);
    cal.T1_T0_msb=HTS221ReadReg(0x35);
    T1=(cal.T1_T0_msb&0x0c);
    T1>>=2;
    T1<<=8;
    T1=T1|cal.T1_degC_x8;
    T1_degC=(float)T1/8;
    cal.T0_OUT=(int16_t)HTS221ReadReg(0x3d);
    cal.T0_OUT<<=8;
    cal.T0_OUT|=(int16_t)HTS221ReadReg(0x3c);
    cal.T1_OUT=(int16_t)HTS221ReadReg(0x3f);
    cal.T1_OUT<<=8;
    cal.T1_OUT|=(int16_t)HTS221ReadReg(0x3e);
     T0_cal=cal.T0_OUT;
     T1_cal=cal.T1_OUT;
}
```

Listing 24. Zmienne kalibracji

```
/* Temperaturea w stopniach C dla kalibracji  */
float T0_degC, T1_degC;
/* Wyjsciowa wartosc temperatury dla kalibracji */
int16_t T0_cal, T1_cal;
/* Wilgotnosc dla kalibracji  */
float H0_rh, H1_rh;
/* Wyjsciowa wartosc wilgotnosci dla kalibracji */
int16_t H0_cal, H1_cal;
```

Listing 25. Odczytanie rejestru temperatury

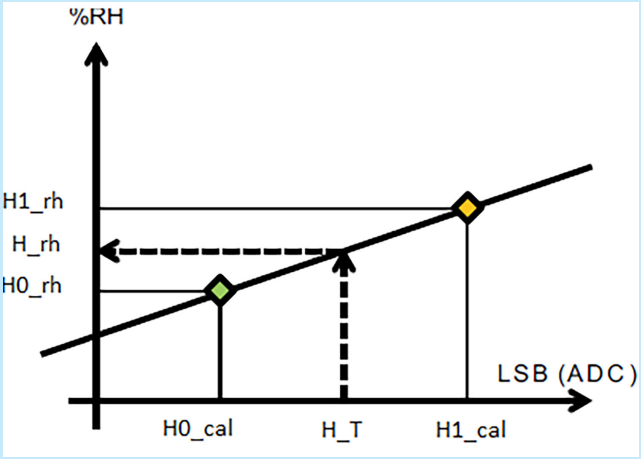
```
//odczytanie 16 bitowego rejestru temperatury
int16_t HTS221TempRead(void)
{
    int16_t  temp;
    temp=0;
    temp=HTS221ReadReg(0x2B);//TEMP_OUT_H
    temp=temp<<8;
    temp=temp|HTS221ReadReg(0x2A);//TEMP_OUT_L
    return(temp);
}
```



Rysunek 38. Definiowanie sterty w oknie konfiguratora

Listing 26. Odczytanie rejestru wilgotności

```
//odczytanie 16 bitowego rejestru wilgotnosci
uint16_t HTS221HumRead(void)
{
    uint16_t hum;
    hum=HTS221ReadReg(0x29);//HUMIDITY_OUT_H
    hum=hum<<8;
    hum=hum|HTS221ReadReg(0x28);//HUMIDITY_OUT_L
    return(hum);
}
```



Rysunek 39. Kalibracja i obliczanie wilgotności w %

Podobnie działają funkcje odczytywania, przeliczania i wyświetlania wilgotności. Wilgotność jest wyliczana z wyrażenia H_rh = (((H_T - H0_cal))/(H1_cal - H0_cal) * (H1_rh - H0_rh) + H0_rh (**rysunek 39**). Na **listingu 26** i **27** pokazano procedury odczytu rejestru wilgotności oraz kompletną procedurę inicjowania pomiaru, odczytania rejestru wilgotności, konwersji na podstawie danych kalibracyjnych i prezentacji wyniku. W końcowym teście, po zainicjowaniu układów STLM75

Listing 26. Odczytanie i wyświetlenie temperatury z czujnika HTS221

```
//odczytanie i wyswietlenie temperatury
void HTS221ReadTemp(char x, char y)
{
    int16_t Treg, tempt;
    float T_C,Temperature;
    char str[20];
    OpenI2CHTS221();
    //start pomiaru
    HTS221WriteReg(0x21,1);//start One shot
    while((HTS221ReadReg(0x27)&1)==0); //czekaj na dane pomiaru

    Treg=HTS221TempRead();//odczytaj rejestr temperatury
    //wylicz temperaturę
    //na podstawie danych kalibracji
    T_C = ((float)(Treg-T0_cal))/(T1_cal-T0_cal)*(T1_degC-T0_degC)+T0_degC;
    tempt = (int16_t)(T_C *100);
    Temperature = ((float)tempt)/100;
    //wyswietl temperature
    sprintf(str,"% +4.1f C",Temperature);
    DispTxtRAM(x,y, str,16,1);
    CloseI2CHTS221();
}
```

Listing 27. Odczytanie i wyświetlenie wilgotności

```
//odczytanie i wyswietlenie wilgotnosci
void HTS221ReadHum(char x, char y)
{
    int16_t H_T, humt;
    float H_rh, Hum;
    char str[20];
    OpenI2CHTS221();
    //start pomiaru
    HTS221WriteReg(0x21,1);//start One shot
    while((HTS221ReadReg(0x27)&2)==0);
    H_T = HTS221HumRead();//odczytaj rejestr wilgotnosci
    H_rh = ((float)(H_T-H0_cal))/(H1_cal-H0_cal)*(H1_rh-H0_rh)+H0_rh;
    humt = (uint16_t)(H_rh * 100);
    Hum= ((float)humt)/100;
    sprintf(str,"% 4.0f %",Hum);
    DispTxtRAM(x,y, str,16,1);
    DispTxt(x=x*6, y,"%", 16,1);
    CloseI2CHTS221();
}
```

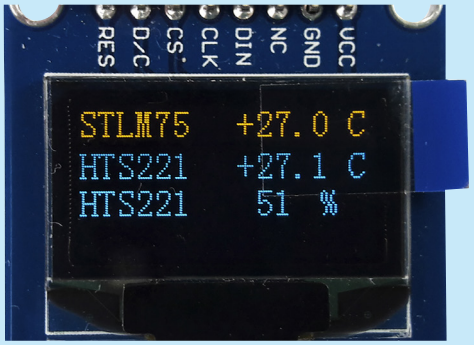
i HTS221, funkcje pobierania i wyświetlania danych z obu czujników są umieszczone w pętli (**listing 28**). Wynik działania tego programu pokazano na **rysunku 40**.

Podsumowanie

Przedstawione tu informacje mogą być przydatne przy konstruowaniu urządzeń w tym urządzeń IoT wykorzystujących interfejsy szeregowe mikrokontrolerów Renesas Synergy. Zastosowałem konfigurator środowiska e2studio oraz funkcje drivera warstwy HAL firmowej biblioteki SSP. Zakończenie transmisji było testowane przez mechanizm powiadamiania callback, również konfigurowanym z poziomu e2studio.

Konfigurowanie i programowanie interfejsów jest stosunkowo łatwe, ale w trakcie pracy nad programem okazało się, że wsparcie społeczności użytkowników jest znikome w porównaniu z bardziej znanymi rodzinami mikrokontrolerów innych producentów. Brak tego wsparcia oznacza, ze jest bardzo trudno znaleźć chociażby najprostsze przykłady pomagające w zrozumieniu działania nawet mniej skomplikowanych elementów programowania mikrokontrolerów Synergy. Najprawdopodobniej wynika to z tego, ze Synergy jest nowością na rynku i pewnie za jakiś czas to się zmieni. Mam nadzieję, że artykuł będzie pomocą w zrozumieniu idei konfigurowania i użycia najniższej warstwy HAL w programowaniu interfejsów szeregowych mikrokontrolerów Renesas Synergy.

Tomasz Jabłoński, EP



Rysunek 40. Pomiar temperatury i wilgotności

Listing 28. Testowanie czujników

```
void hal_entry(void)
{
    InitOled();//inicjowanie wyświetlacza OLED
    HTS221Init();//inicjowanie higrometru
    STLM75Config (2);//inicjowanie termometru
    DispTxt(0,0, „STLM75 „, 16,1);
    DispTxt(0,16, „HTS221 „, 16,1);
    DispTxt(0,32, „HTS221 „, 16,1);
    while(1)
    {
        STLMTempRead(60,0);   //odczytanie i wyświetlenie temperatury z STLM75
        HTS221ReadTemp(68,16); //odczytanie i wyświetlenie temperatury z HTS221
        HTS221ReadHum(60,32);//odczytanie i wyświetlenie wilgotności z STLM75
        RefreshRAM();
    }
    while(1);
}
```