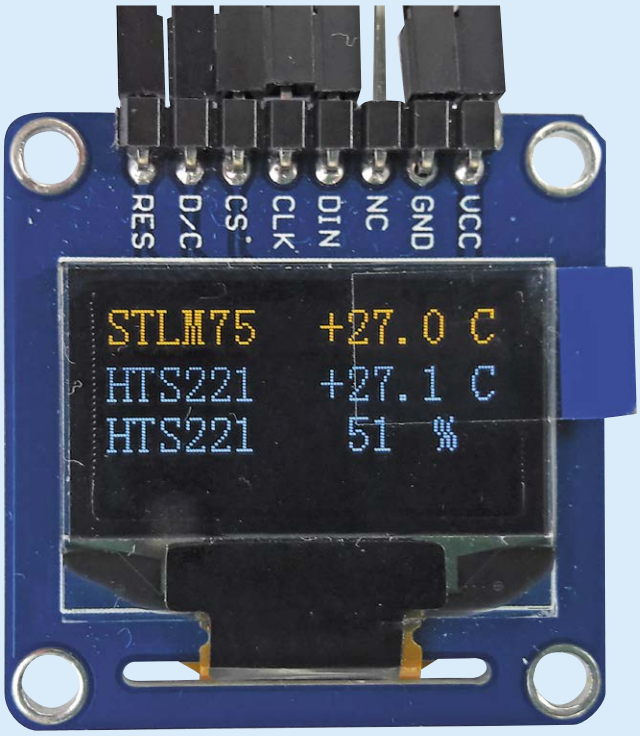
Renesas Synergy – interfejsy szeregowe (2)

Moduł Arrow Aris EDGE jest sprzętowo zgodny z systemem Arduino. Wykorzystamy płytkę rozszerzeń Kamami Nucleo zgodną ze standardem Arduino R3 zawierającą: cyfrowy termometr STLM75, cyfrowy ciśnieniomierz LPS331, cyfrowy higrometr HTS221 oraz miernik natężenia światła, 3-kolorową diodę LED i miniaturowy joystick. Wszystkie cyfrowe układy pomiarowe wyposażono w interfejs I²C.

Fotografia 16. Płytka rozszerzeń połączona z Arrow Aris EDGE

Co prawda płytka rozszerzenia była projektowana dla modułu Nucleo z mikrokontrolerami STM32, ale sprzętowa zgodność z Arduino powinna zapewnić możliwość współpracy z naszym modułem. Jak już wspomniałem, wszystkie sensory umieszczone na rozszerzeniu mają wbudowany interfejs szeregowy I²C. To zdecydowanie upraszcza połączenia sprzętowe, bo będziemy potrzebowali tylko dwóch linii: SDA i SCL. Na **rysunku 17** pokazano wyprowadzenia płytki rozszerzeń i wyprowadzenia modułu Arrow. Jak widać, nie powinno być problemu z połączeniem,

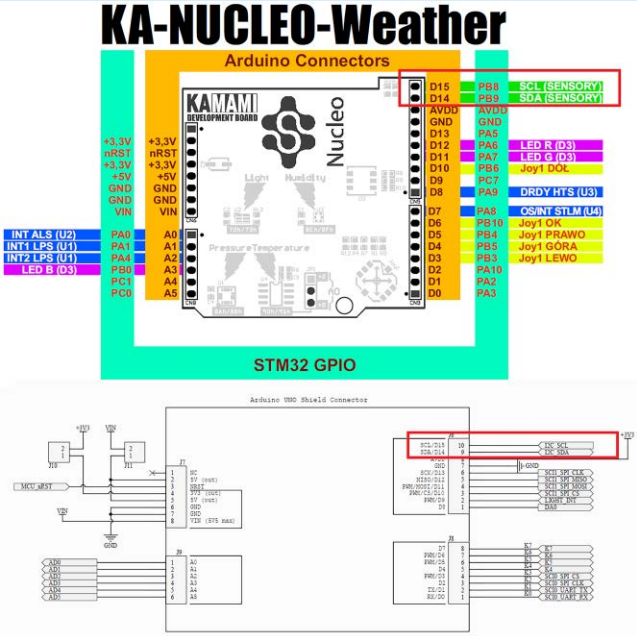


bo sygnały z SDA i SCL z obu płytek są na tych samych wyprowadzeniach złącza.

Interfejs I²C

Jeżeli popatrzymy na schemat Arrow Aris EDGE, to widzimy, że wyprowadzenia linii SDA są dołączone do portu P206, a linii SCL do portu P205. W wypadku SPI mieliśmy swobodę wyboru interfejsu sterującego, bo połączenia były wykonywane za pomocą przewodów. Tu jest inaczej. Sztywne połączenie elektryczne determinuje wybór interfejsu i nie możemy wybrać SCI pracującego w trybie I²C. Linie P205 i P206 są przypisane do natywnego interfejsu IIC1 i musi on być użyty do obsługi modułu.

Schemat blokowy modułu IIC jest pokazany na **rysunku 18**. Jak widać, jest on dość skomplikowany. Generalnie, konfiguracja i obsługa interfejsu I²C jest jedną z trudniejszych, jeśli porównać ją do innych interfejsów szeregowych. Wynika to z dużych możliwości przy minimalnych wymaganiach sprzętowych. Moduły komunikacyjne I²C mogą pracować w trybie master lub slave. Możliwe jest dołączanie do jednej magistrali kilku układów master. Wymagane jest wtedy stosowanie mechanizmu arbitrażu kolizji na magistrali. W dokumentacji mikrokontrolera jest zawarty szeroki opis modułu z rejestrami konfiguracyjnymi, przebiegami czasowymi i algorytmami obsługi transmisji. Zapoznanie się z tymi wszystkimi informacjami i użycie ich do zaprogramowania transmisji jest zadaniem bardzo czasochłonnym. Poza tym wiem z doświadczenia, że nawet w obrębie tej samej rodziny mikrokontrolerów producent potrafi zmienić budowę i sposób programowania modułu komunikacyjnego. Trzeba wtedy posiadaną wiedzę na nowo weryfikować. Pokażę, jak tego uniknąć, stosując



Rysunek 17. Wyprowadzenia modułu rozszerzenia i schemat wyprowadzeń modułu Arrow

driver warstwy HAL biblioteki SSP oraz konfigurator środowiska e2studio. Zobaczymy, że konfiguracja interfejsu jest łatwa, a jego obsługa niezbyt skomplikowana.

Driver modułu IIC

Podobnie jak w wypadku interfejsu SPI, trzeba w zakładce Threads HAL/Commons dodać driver *I2C Driver on r_iic*, jak to zostało pokazane na **rysunku 19**. Po dodaniu trzeba driver (interfejs IIC) skonfigurować poprzez:

- Odblokowanie i nadanie priorytetów przerwań używanych przez driver.
- Nadanie modułowi nazwy wcielenia.
- Wybranie numeru kanału. Tu będzie to kanał 1 z powodów opisanych wcześniej.
- Ustawienie adresu urządzenia slave.
- Wybranie prędkości transmisji (100 kb/s, 400 kb/s lub 1 Mb/s).
- Wybranie trybu adresowania 7-bitowego lub 10-bitowego (w naszym wypadku będzie to adresowanie 7-bitowe).
- Włączenie powiadamiania *callback* poprzez nadanie mu nazwy. W odróżnieniu od interfejsu SPI, *callback* nie jest koniecznie wymagany. Kiedy nie zdefiniujemy nazwy funkcji *callback*, to funkcje drivera stają się funkcjami blokującymi i czekają na zakończenie transmisji. W testowych programach *callback* jest zdefiniowany i używany.

Jak widać, konfiguracja jest łatwa i zajmuje mało czasu. Po kliknięciu na *Generate Project Content* konfigurator wygeneruje pliki z konfiguracjami i funkcjami drivera IIC oraz umieści je w projekcie.

Przed użyciem funkcji generujących ruch na magistrali trzeba wywołać procedurę otwarcia interfejsu.

```
ssp_err_t  R_RIIC_MasterOpen (i2c_ctrl_t *const
           p_ctrl, i2c_cfg_t const *const
           p_cfg)
```

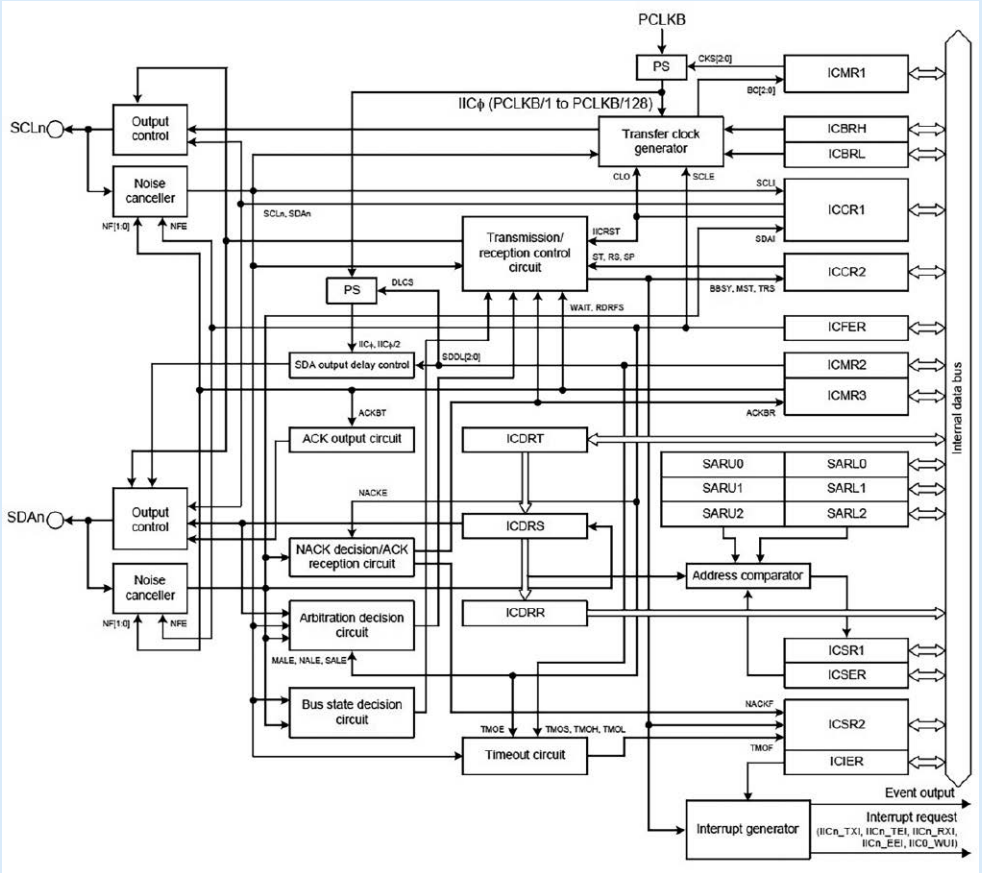
Jej argumentami są struktury STM75_ctrl_t i STM75_cfg_t wygenerowane przez konfigurator. Funkcja Open sprawdza poprawność parametrów i ewentualnie generuje informacje o błędach, włącza zasilanie modułu IIC oraz inicjalizuje rejestry konfiguracyjne. W naszym wypadku wywołanie funkcji otwarcia kanału drivera będzie wyglądać następująco:

```
ssp_err_t  err;
err=R_RIIC_MasterOpen(STM75.p_ctrl,
STM75.p_cfg);
```

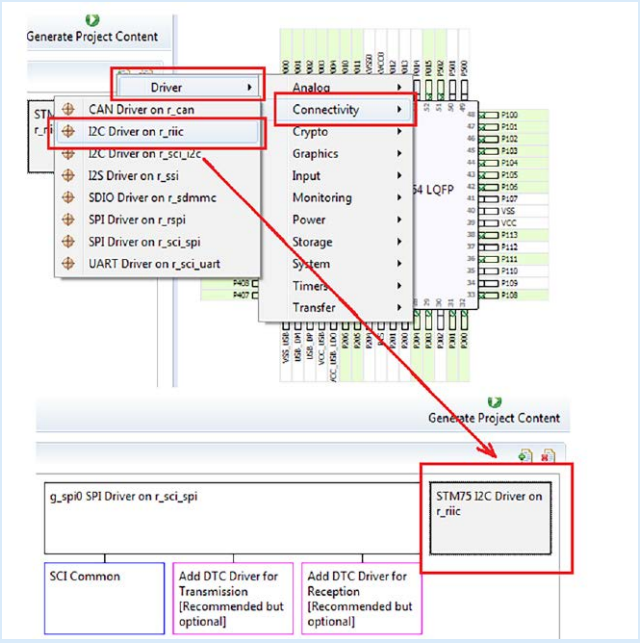
Wszystkie funkcje drivera zwracają informacje o statusie wykonania funkcji *ssp_err_t*. Status *SSP_SUCCESS* oznacza prawidłowe wykonanie funkcji, *SSP_ERR_IN_USE* próbę otwarcia drivera już otwartego i SSP_ERR_INVALID_RATE – nie można ustawić żądanej prędkości transmisji.

Do komunikowania się z układami peryferyjnymi w trybie Master potrzebne będą tylko 2 procedury: zapisu danych na magistralę i odczytu danych z magistrali. Zapis danych na magistralę realizuje procedura R_IIC_MasterWrite:

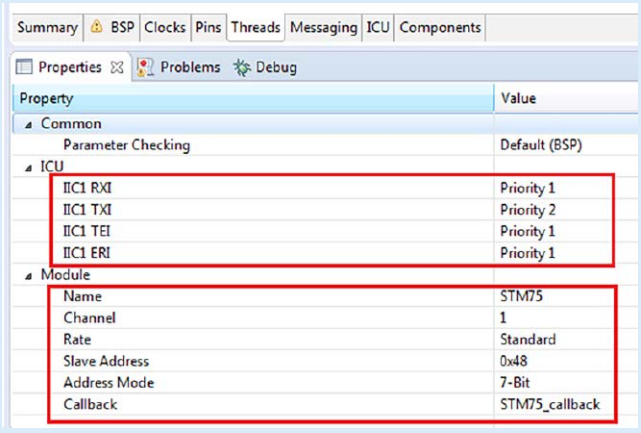
```
ssp_err_t  R_RIIC_MasterWrite (i2c_
           ctrl_t *const p_ctrl, uint8_t
           *const p_src, uint32_t const bytes,
           bool const restart).
```



Rysunek 18. Schemat blokowy modułu IIC



Rysunek 19. Dodanie drivera IIC



Rysunek 20. Konfiguracja drivera IIC

Jej argumentami są:

- Wskaźnik na strukturę sterującą *i2c_ctrl_c*.
- Wskaźnik na bufor z danymi do wysłania.
- Liczba bitów do przesłania.
- Znacznik bitowy *restart*.

Jak się łatwo domyślić, wywołanie procedury musi zainicjować sekwencję START, a potem jest wysyłany adres slave ustawiony w konfiguracji z bitem R/W=0 i kolejne dane zapisane w buforze w liczbie określonej przez zmienną *bytes*. Funkcja zapisu na magistralę, podobnie jak opisywana funkcja odczytu z magistrali, nie ma jawnie ustawianego w argumencie adresu slave urządzenia peryferyjnego. Ten adres jest zapisany w strukturze konfiguracyjnej i ustawiany w oknie *Properties* w polu *Slave Address* (**rysunek 20**). Taki sposób adresowania slave powoduje, że dla każdego z urządzeń dołączonych do magistrali I²C trzeba zdefiniować osobny driver.

Każda sekwencja transferu na magistrali musi zakończyć się sekwencją STOP, jednak często zdarza się, że odczyt danych jest poprzedzony zapisem. Tak jest na przykład podczas odczytywania danych z pamięci EEPROM. Najpierw są wysyłane bajty adresu, a po nich sekwencja odczytania porcji danych. W takim wypadku po zapisaniu adresu nie wystawia się sekwencji STOP, tylko ponowny START. Żeby to było możliwe, wprowadzono argument **restart**. Kiedy ma on wartość false, to funkcja wysyła na magistralę sekwencję STOP, a kiedy ma wartość true, to sekwencja nie jest wysyłana. Jak to działa, zobaczymy za chwilę przy okazji obsługi termometru STLM75.

Do odczytywania danych na magistrali jest przeznaczona procedura R_IIC_MasterRead

```
ssp_err_t  R_RIIC_MasterRead (i2c_ctrl_t *const
           p_ctrl, uint8_t *const p_dest,
           uint32_t const bytes, bool const
           restart)
```

z następującymi argumentami:

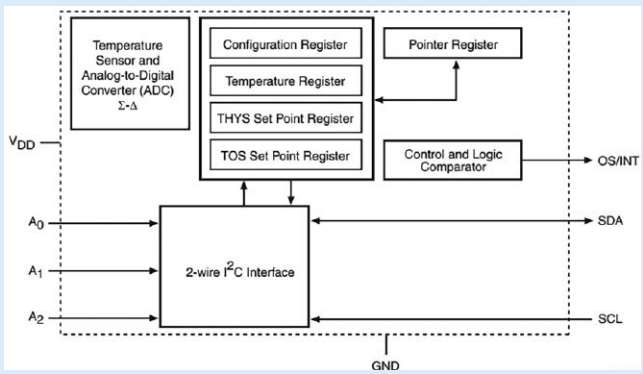
- Wskaźnik na strukturę sterującą *i2c_ctrl_c*.
- Wskaźnik na bufor z danymi odbieranymi.
- Liczba bitów do odebrania.
- Znacznik bitowy *restart*.

Procedura rozpoczyna działanie od wysłania sekwencji START (REPEAT START). Po niej jest wysyłany adres slave z bitem R/W=1. Aby odczytać dane z urządzenia slave, moduł wysyła kolejne cykle zegarowe. Odczytywanie kończy się lub nie sekwencją STOP zależnie od wartości zapisanej w argumencie *restart*, dokładnie tak samo, jak w wypadku funkcji zapisu danych na magistralę.

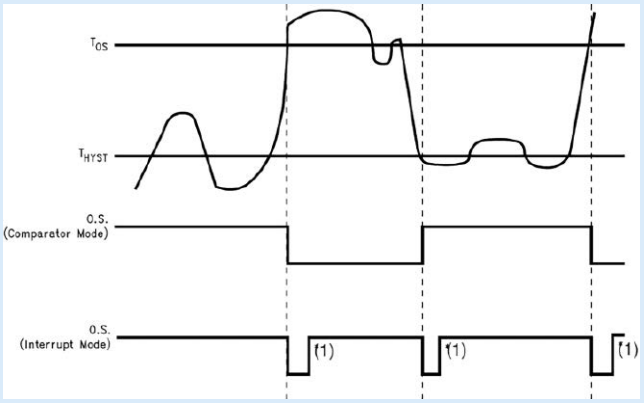
Praktyczne wykorzystanie drivera IIC rozpoczniemy od obsługi cyfrowego termometru STLM75 umieszczonego na płytce Nucleo. STLM75 mierzy temperaturę z rozdzielczością 0,5°C w zakresie –55…+125°C. To zakres pozwalający na pomiar w większości typowych zastosowań. Niepewność pomiaru dla całego zakresu pomiarowego wynosi ±3°C. Może to oznaczać, że pomimo wyniku pomiaru temperatury +13,5°C, w rzeczywistości w najgorszym wypadku może ona mieć wartość +10,5°C lub +16,5°C. W wypadku temperatury „pokojowej" taki pomiar praktycznie nie ma sensu. Jednak warto zauważyć, że jest to maksymalna odchyłka ±3°C w całym zakresie pomiarowym. Termometr jest tak skalibrowany, że typowa niepewność pomiaru dla temperatury otoczenia nie powinna być gorsza niż ±0,5°C.

Konwersja napięcia proporcjonalnego do mierzonej temperatury jest wykonywana przez 9-bitowy przetwornik sigma-delta (**rysunek 21**). Wyprowadzenie OS/INT typu otwarty dren (wymaga podciągania do plusa zasilania) jest przeznaczone do sygnalizacji przekroczenia nastawionych progów temperatury. W trybie przerwania (INT) na tym wejściu wystąpią impulsy w momencie przekroczenia progów, a w trybie termostatu wyjście to zmienia stan w zależności od tego, czy temperatura jest niższa, czy wyższa niż progowa (**rysunek 22**). Temperatury progu Tos i histerezy Thys są programowane przez użytkownika przez zapisanie odpowiednich rejestrów. Z wyjściem OS/INT jest powiązany zaburzeń. Po przekroczeniu progu jest odliczana zaprogramowana liczba cykli czasowych (od 1 do 6) i po tym czasie jest ponownie sprawdzany warunek przekroczenia progu. Jeżeli próg jest przekroczony, to OS/INT zmienia stan, a jeżeli nie, to pozostaje w stanie niezmienionym. Pozwala to na eliminowanie ewentualnych zakłóceń.

Jak wspomniałem, termometr mierzy temperaturę z rozdzielczością 0,5°C w zakresie –55…+125°C. Wynik konwersji jest



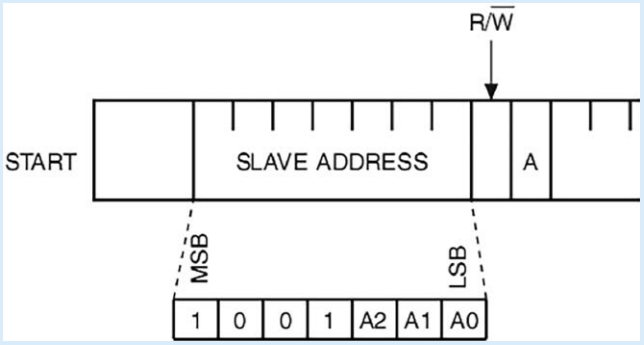
Rysunek 21. Schemat blokowy STLM75



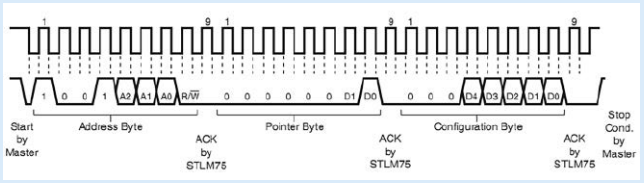
Rysunek 22. Tryby pracy wyjścia OS/INT

Temperatura	Cyfrowa wartość wyjściowa	
	binarnie	heksadecymalnie
+125°C	0 1111 1010	0xFA
+25°C	0 0011 0010	0x32
+0,5°C	0 0000 0001	0x01
0°C	0 0000 0000	0x00
–0,5°C	1 1111 1111	0x1FF
–25°C	1 1100 1110	0x1CE
–40°C	1 1011 0000	0x1B0
–55°C	1 1001 0010	0x192

Rysunek 23. Zależność pomiędzy temperaturą a cyfrową wartością wyjściową



Rysunek 24. Adresowanie STLM75



Rysunek 25. Sekwencja zapisu rejestru konfiguracyjnego

zapisywany na 9 bitach w kodzie U2. Dla temperatury dodatniej (najstarszy bit wyniku konwersji równy 0) 8 najstarszych bitów zawiera wartość części całkowitej wartości temperatury, a najmłodszy bit określa rozdzielczość 0,5°C. Dla temperatury ujemnej (najstarszy bit wyniku konwersji równy 1) trzeba wykonać konwersję polegającą na zanegowaniu wszystkich bitów i dodaniu 1. Potem określenie temperatury przebiega tak samo, jak temperatury dodatniej. Na **rysunku 23** pokazano zależność pomiędzy temperaturą i wartością odczytaną z rejestru termometru.

Adres slave układu jest określany przez poziom logiczny na trzech wyprowadzeniach adresowych: A0, A1 i A2. Na **rysunku 24** pokazano sposób adresowania układu. Dla A0=A1=A2=0 adres dla zapisu danych wynosi 0x48 (R/W=0), a dla odczytu 0x49 (R/W=1). Taki

Rejestr konfiguracyjny został pokazany na rysunku 26

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	0	0	FT1	FT0	POL	M	SD

SD – Shutdown – ustawienie tego bitu włącza tryb oszczędzania energii
M – Termostat mode M=0 tryb termostatu, M=1 tryb przerwań (Interrupt)
POL – polaryzacja wyjścia OS/INT POL=0 aktywny stan niski, POL=1 aktywny stan wysoki
FT1:FT0 liczba cykli eliminacji zakłóceń

Rysunek 26. Rejestr konfiguracyjny

adres wpisałem w polu *adres slave* konfiguracji drivera IIC. Zapisanie danych do układu standardowo rozpoczyna się od wysłania sekwencji START, a następnie jest przesyłany adres *slave*. Jeżeli adres jest prawidłowy, to termometr potwierdza go w dziewiątym takcie zegara. Po potwierdzeniu adresu można wysyłać do układu dane.

STLM75 ma 4 rejestry: 16-bitowy rejestr mierzonej temperatury TEMP, 8-bitowy konfiguracji CONF, 16-bitowy histerezy temperatury Thys i 16-bitowy progu temperatury Tos. Poza rejestrem odczytywanej temperatury TEMP, który można tylko odczytywać, pozostałe rejestry można zapisywać i odczytywać. Każdy rejestr przed dostępem (zapisem lub odczytem) musi być zaadresowany. Pierwszą ważną sekwencją protokołu komunikacji z STLM75 jest zapisywanie bajtu rejestru konfiguracyjnego. Polega ona na:

- Wysłaniu sekwencji START.
- Wysłaniu adresu *slave* z bitem R/W=0 (i odebraniu potwierdzenia ACK).
- Wysłaniu bajtu z adresem rejestru=1.
- Wysłaniu bajta rejestru konfiguracyjnego.
- Wysłaniu sekwencji STOP.

Zostało to pokazane na **rysunku 25**.

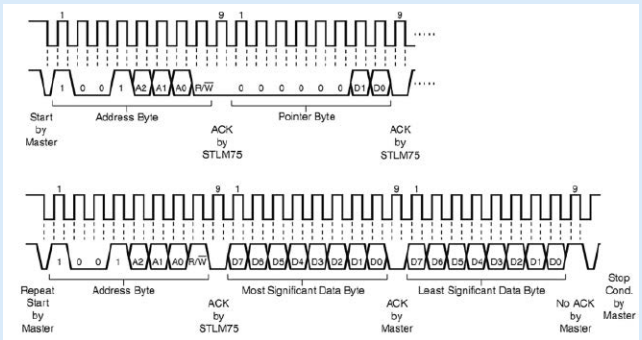
Domyślnie po włączeniu zasilania wszystkie bity rejestru konfiguracyjnego są wyzerowane. Rejestry temperatury Thys i Tos są zapisywane podobnie jak rejestr konfiguracyjny, tylko po wysłaniu bajtu z adresem (równego 2 lub 3) wysyłane są dwa kolejne bajty rejestru 16-bitowego.

Kolejną ważną sekwencją jest odczytywanie dowolnego rejestru polegającego na:

- Wysłaniu sekwencji START.
- Wysłaniu adresu *slave* z bitem R/W=0 (i odebraniu potwierdzenia ACK).
- Wysłaniu bajtu z adresem rejestru (dla odczytania rejestru temperatury wyzerowany).
- Wysłaniu powtórnej sekwencji START.
- Wysłaniu adresu *slave* z bitem R/W=1 (i odebraniu potwierdzenia ACK).
- Odczytaniu dwu bajtów rejestru.
- Wysłaniu sekwencji STOP.

Zostało to pokazane na **rysunku 27**.

Tomasz Jabłoński, EP



Rysunek 27. Sekwencja odczytywania rejestru STLM75