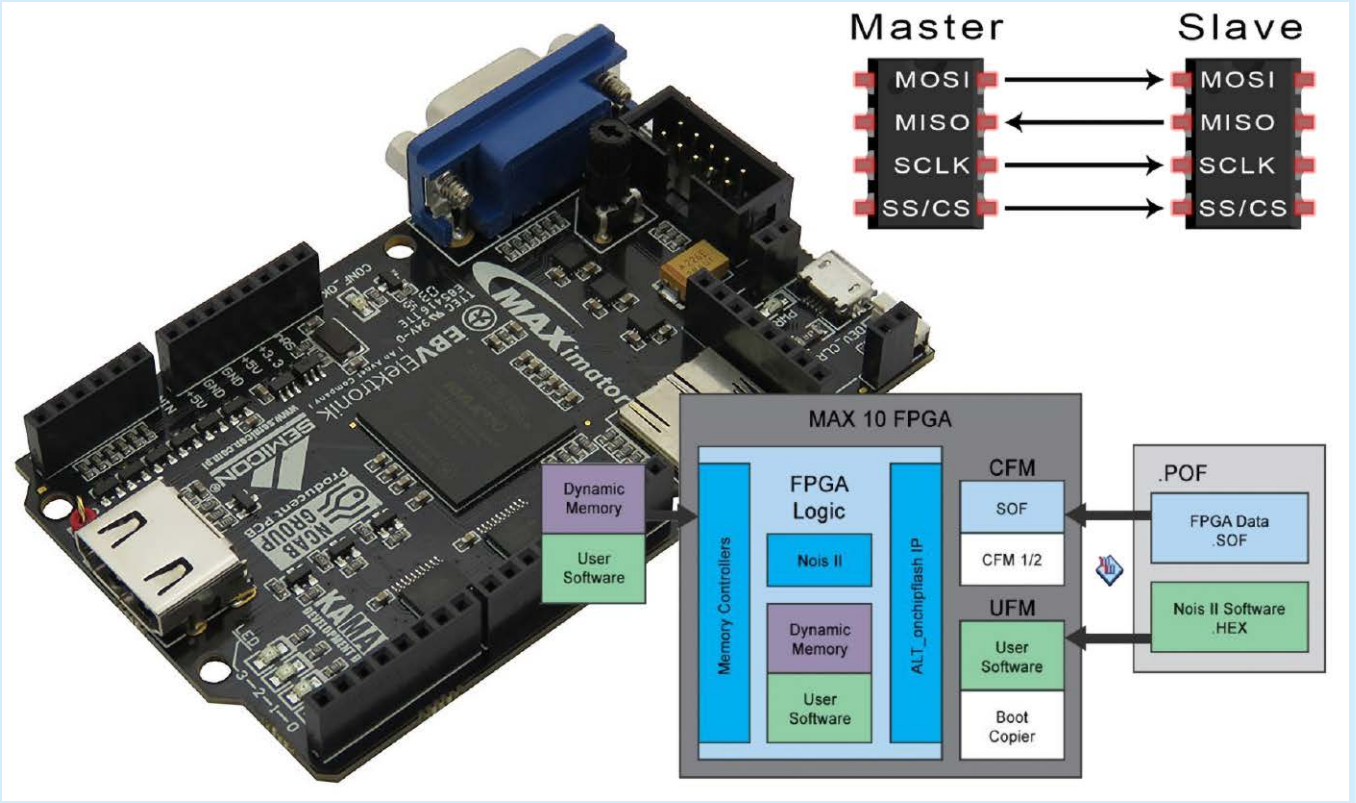


NIOS II na maXimatorze, czyli mikroprocesor w układzie FPGA (6)

SPI – nawiązujemy łączność z czujnikami

Jednymi z najbardziej popularnych i najczęściej stosowanych interfejsów służących do komunikacji pomiędzy układami scalonymi umieszczonymi na jednej płytce drukowanej lub w obrębie tego samego urządzenia są I²C oraz SPI. W czasie tego spotkania zapoznamy się z implementacją interfejsu SPI, zaś kolejną część poświęcimy interfejsowi I²C.

Typowo, wspomniane we wstępie interfejsy są stosowane na dystansach nieprzekraczających kilkunastu centymetrów. Za ich pomocą możemy przyłączyć wiele dodatkowych elementów, takich jak np. pamięci, sensory (akcelerometry, sensory atmosferyczne, karty pamięci i inne), przetworniki A/C oraz C/A, ekspandery portów IO i wiele, wiele innych. W skrócie – bez znajomości i umiejętności implementacji tych interfejsów w naszym systemie pozbawiamy się możliwości używania wielu układów, nieodzownych w bardziej zaawansowanych projektach.

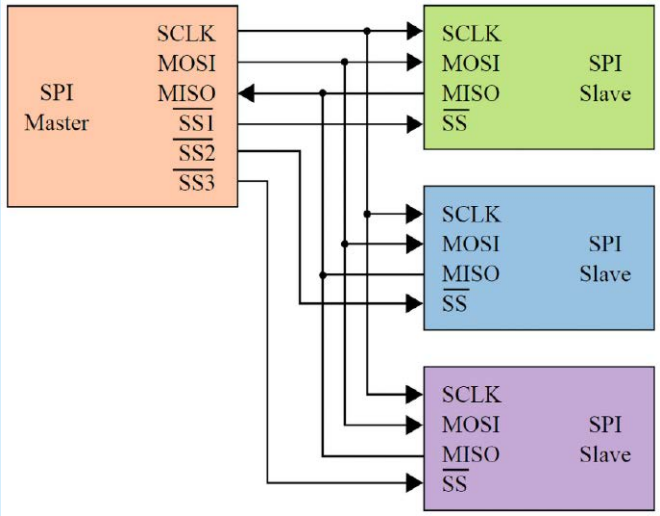
Krótkie wprowadzenie

INTERFEJS SPI wykorzystuje 3 główne linie oraz linie dodatkowe służące do wyboru układu, z którym jest prowadzona komunikacja (**rysunek 1**).

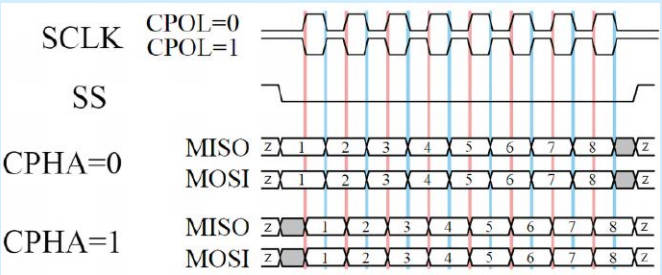
W tej komunikacji wyróżniamy 2 typy układów: nadrzędny (*master*) oraz układy podrzędne (*slave*). Układ *master* jest odpowiedzialny za rozpoczynanie komunikacji i sterowanie nią, zaś układy *slave* mogą jedynie reagować na jego „polecenia". Linie wykorzystywane w magistrali tej to (w nawiasach podano także inne popularnie stosowane nazwy tych sygnałów, jednakże czasem producenci stosują jeszcze inną nomenklaturę niż ta podana poniżej):

1. SCLK (SCK, CLK…) – sygnał zegarowy, który jest generowany przez układ nadrzędny, synchronizuje moment transmisji kolejnych bitów.
2. MOSI (SI, DIN…) – wyjście danych z układu nadrzędnego i wejście do układu podrzędnego (ang. *Master Output Slave Input*), za którego pomocą są transmitowane dane w momencie wystąpienia odpowiedniego zbocza sygnału SCLK.
3. MISO (SO, DOUT…) – wejście danych do układu nadrzędnego i wyjście z układu podrzędnego (ang. *Master Input Slave Output*).

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



Rysunek 1. Przykład podłączenia 3 układów SPI do urządzenia nadrzędnego. Źródło: wikipedia: Cburnett



Rysunek 2. Przebiegi na liniach magistrali SPI przy różnych konfiguracjach dla transmisji 8 bitów danych. Opracowano na podstawie: wikipedia: Cburnett

4. SS (CS…) – linia wyboru układu. Ponieważ interfejs SPI w układach *slave* najczęściej nie obsługuje adresowania układów, więc istnieje konieczność, aby wybrać układ, z którym chcemy się komunikować. Dokonujemy tego aktywując, czyli najczęściej ustawiając w stan niski linii CS połączonej z konkretnym układem.

Jak wynika z powyższego, dane są transmitowane za pomocą linii MOSI do układu *slave*, zaś za pomocą linii MISO w przeciwnym kierunku. Dane transmitowane są bit-po-bicie „w takt" zegara SCLK, zaś transmisja ma miejsce tylko między układem *master* i (zazwyczaj) jednym układem *slave*. Oprócz samego połączenia, czeka nas jeszcze jedna pułapka – mianowicie tryb pracy magistrali SPI.

Na **rysunku 2** przedstawiono wszystkie możliwe konfiguracje. Pierwszą opcją jest *CPOL*, czyli polaryzacja sygnału zegarowego – jak widzimy może ona przyjmować dwie wartości – albo sygnał zegarowy jest na poziomie niskim w stanie bezczynności, albo jest wtedy na poziomie wysokim. Drugi parametr (*CPHA*) definiuje fazę sygnału zegarowego względem danych – albo dane są pobierane (są stabilne = nie zmieniają się) przy pierwszym zboczu zegara (*leading edge, CPHA=0*), albo przy drugim zboczu (*trailing edge, CPHA=1*). Informacje o tym, który tryb wybrać powinniśmy pozyskać z dokumentacji układu, który chcemy przyłączyć. Czasem producenci stosują różne oznaczenia (np. *Mode* lub *CKE* zamiast *CPHA* – patrz **rysunek 3**), lub w ogóle nie podają takich informacji *explicite*. Wtedy zostaje nam jedynie przeanalizować prezentowane w dokumentacji przebiegi i korzystając ze „ściągi" w postaci rys. 2 ustalić interesujące nas parametry. To chyba największa pułapka interfejsu SPI, gdyż czasem błędne ustawienie tych parametrów jest trudne do wykrycia i daje niejednoznaczne błędy w transmisji (np. wszystko działa, a jedynie gubimy 1 bit z każdego bajta).

O zaletach i wadach SPI w porównaniu z UART i I²C porozmawiamy w czasie kolejnego spotkania, gdy poznamy ostatni z tych protokołów.

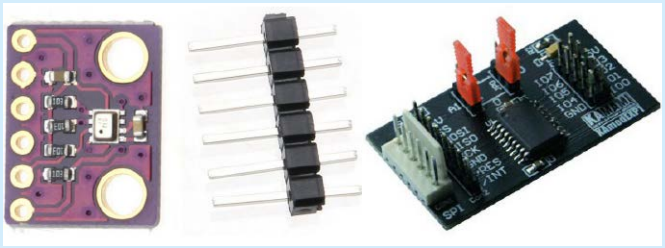
Przykładowe układy

W celu praktycznego sprawdzenia działania magistral warto zaopatrzyć się w jakieś układy je wspierające. W swoich testach wykorzystałem: miernik ciśnienia atmosferycznego i temperatury BMP280 (zamontowany na wygodnym module) oraz ekspander GPIO MCP23S08 (dostępny w obudowie PDIP, łatwej do montażu na płytce stykowej lub w formie modułów) – **rysunek 4**.

Mode	CPOL	CPHA	CKE
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	0

Rysunek 3. Możliwe tryby pracy magistrali SPI

Układ BMP280 posiada możliwość współpracy zarówno z interfejsami SPI, jak i I²C, zaś układ MCP23S08 z interfejsem SPI ma swojego brata – układ MCP23008 z interfejsem I²C. Dzięki temu w czasie kolejnego spotkania poświęconego tej drugiej magistrali, będziemy pracować już z układami, które znamy.



Rysunek 4. Moduł BMP280 oraz układ MCP23S08 na module KAmodEXP1

Oczywiście pierwsze, co powinniśmy zrobić po otrzymaniu tych układów, to zajrzeć do noty katalogowej samego układu, jak i uzyskać jak największej ilości informacji o płytce, na której jest zamontowany nasz układ. Po pierwsze, wartym zwrócenia uwagi jest fakt, że układ BMP280 pracuje w standardzie napięciowym 3,3 V, zatem możemy go na naszej płytce dołączyć w zasadzie tylko do wyprowadzeń „analogowych", czyli *ANIN0…ANIN5*. Te piny mogą akceptować sygnały cyfrowe w standardzie 5 V, jednak z powodu braku bufora dwukierunkowego na wyjściu podają sygnały w standardzie 3,3 V, co idealnie odpowiada naszym potrzebom. Układy MCP23x08 mogą z kolei pracować w szerokim zakresie napięcia, więc możemy je zasilić napięciem 5 V i dołączyć do pinów układu wyposażonych w bufor.

GY-BMP280

W użytym w przykładzie module czujnik BMP280 wyposażono jedynie w kilka elementów pasywnych, istotnych głównie dla magistrali I²C. Sygnały i linie zasilania na module wymieniono w **tabeli 1**.

Aby moduł pracował w trybie magistrali SPI, należy po włączeniu zasilania jednorazowo doprowadzić na pin CS poziom niski – spowoduje to wyłączenie komunikacji po I²C aż do momentu wyłączenia zasilania układu. Realizacją tego zadania zajmiemy się nieco później.

MCP23S08

Układ ekspandera portów ma – oprócz wyprowadzeń interfejsu SPI oraz 8 linii wejścia/wyjścia (GPx) – także 2 linie adresowe (A0, A1),

Tabela 1. Sygnały i linie zasilania modułu GY-BMP280

Moduł GY-BMP280	Połączenie	maXimator
VCC	+3.3 V	+3.3 V
GND	GND	GND
SCL	SCLK	ANIN3 (D1)
SDA	MOSI	ANIN2 (C1)
CSB	CS	ANIN4 (E1)
SDO	MISO	ANIN1 (B2)

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

Tabela 2. Sposób podłączenia linii KAmodEXP2 do maXimatora

Układ MCP23S08	Moduł KAmodEXP1	Połączenie	maXimator
VDD	V+	+5V	+5V
VSS	GND	GND	GND
SCK	SCK	SCLK	D2 (J16)
SI	MOSI	MOSI	D3 (H15)
SO	MISO	MISO	D4 (H16)
CS	SS	CS	D1 (J15)
A0	A0*	GND*	GND
A1	A1*	GND*	GND
RESET	/RES*	+5V*	+5V
INT	/INT	-	-
GPx	IOx	...	...

pin zerowania oraz przerwania (**rysunek 5**). Linie adresowe pozwalają na dołączenie wielu układów tego typu do magistrali SPI z wykorzystaniem tylko 1 pinu CS, jeśli zostanie włączony odpowiedni tryb (ustawienie bitu HAEN w rejestrze IOCON). Pin przerwania zapewnia sygnalizację pewnych zdarzeń, które jest wstanie wykryć nasz układ (np. zmiana stanu jakiegoś pinu). W przypadku chęci wykorzystania takiej funkcjonalności w naszym systemie NIOS II wykorzystalibyśmy zewnętrzne przerwania (tak, mowa o tych przerwaniach, które zupełnie nie nadawały się do obsługi bezpośrednio podpiętych przycisków).

Jeśli skorzystamy z układu montowanego na płytce stykowej, to musimy zadbać o połączenie wszystkich sygnałów wymienionych w **tabeli 2**. W przypadku stosowania modułu stan linii A0 i A1 ustalamy za pomocą zworek (gdzie oczywiście 0 to GND, a 1 to połączenie z V+), zaś sygnał reset powinien być na wspomnianym module podciągnięty do V+ za pomocą rezystora (patrz sygnały oznaczone *).

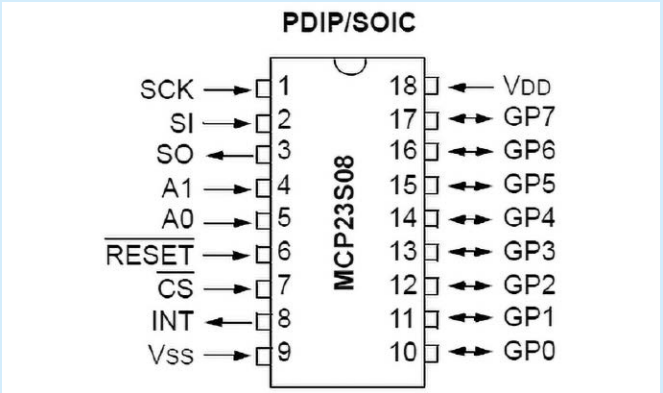
Do linii GP0…GP3 dołączymy przyciski zwierające do masy (możemy też użyć przewodu), zaś do GP4…GP7 diody świecące wraz z odpowiednimi rezystorami (zwykle w granicach 330 Ω).

Dodajemy moduł SPI do naszego systemu

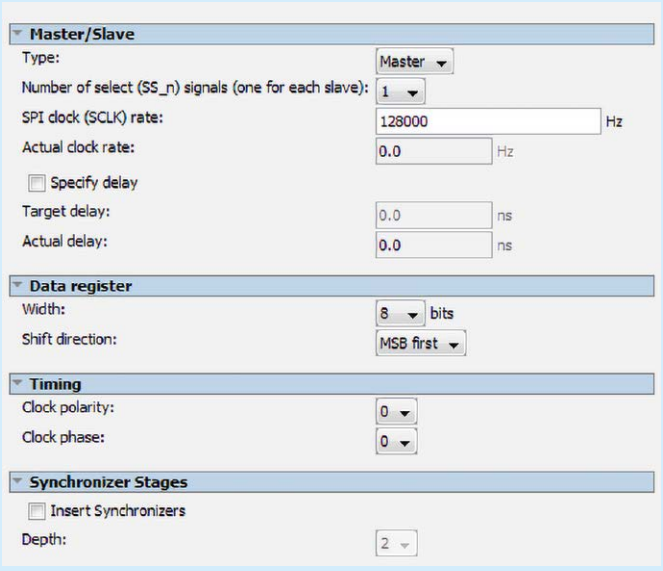
Aby dodać moduł SPI do naszego systemu otwieramy projekt systemu w Platform Designer (**rysunek 6**) i odnajdujemy moduł *SPI (3 Wire Serial)*. W sumie dodamy do naszego projektu dwa moduły:

1. Pierwszy nazwiemy: *SPI*, będzie on miał 1 wyjście CS. Będzie on wyprowadzony na linie ANIN, dzięki czemu będzie pracował w standardzi3 3,3 V – posłuży do obsługi układu BMP280.
2. Drugi nazwiemy: *SPI_5V*, będzie on miał 2 wyjścia CS. Będzie on wyprowadzony na linie cyfrowe (przez konwerter napięć) i posłuży do dołączenia układów MCP23S08 zasilonych napięciem 5 V.

Podobnie jak poprzednio omówmy krótko możliwe ustawienia.



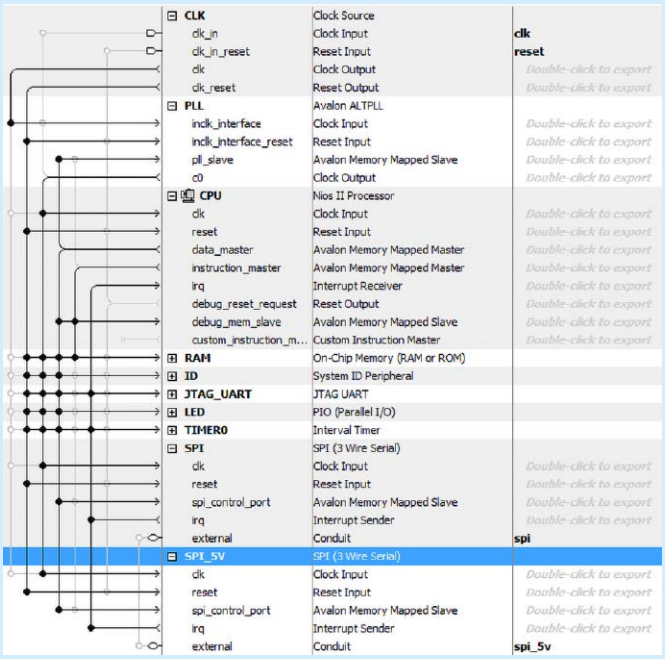
Rysunek 5. Wyprowadzenia układu MCP23S08



Rysunek 6. Konfiguracja modułu SPI w Platform Designer

1. *Type* – możemy wybrać, czy dodany do układu moduł ma pełnić rolę *master* czy *slave*. W naszym wypadku wybieramy oczywiście pierwszą z opcji.
2. *Number of…* – wybieramy liczbę wyprowadzeń CS naszego modułu. Dzięki temu moduł będzie posiadał od razu odpowiednią liczbę wyprowadzeń do sterowania wyborem układu (u nas 1 dla pierwszego modułu, oraz 2 dla kolejnego).
3. *SPI clock…* – ustalamy częstotliwość sygnału SCLK – powinniśmy ją dobrać tak, aby była zgodna z parametrami najwolniejszego układu. Dla naszego projektu wybierzmy śmiesznie małą prędkość 100 000 Hz (100 kHz). Po doprowadzeniu do modułu sygnału zegarowego (co zaraz zrobimy w Platform Designer) w szarym polu poniżej pojawi się rzeczywista częstotliwość, którą uda się osiągnąć stosując dostępne w module podzielniki.
4. *Specify delay* – zaznaczenie tej opcji pozwala na ustawienie opóźnienia pomiędzy ustawieniem pinu CS a rozpoczęciem transmisji w trybie automatycznym – nasz moduł SPI domyślnie sam aktywuje odpowiednią linię CS przed transmisją jednej ramki danych, po czym automatycznie ją dezaktywuje.
5. *Data register* – opcje tu dostępne pozwalają na zmianę ilości bitów transmitowanych jako jedna ramka danych (czyli ilości bitów, które jednorazowo mieszczą się w rejestrze modułu, zatem mogą być transmitowane bez opóźnień i konieczności ingerencji ze strony procesora). Druga opcja pozwala wybrać czy jako pierwszy ma być nadawany najbardziej czy najmniej znaczący bit z rejestru, czyli pozwala na odwrócenie kolejności bitów w danych. Opcje te zostawiamy bez zmian, czyli 8 bitów oraz najbardziej znaczący bit (*MSB*) nadawany jako pierwszy.
6. *Timing* – ta grupa opcji pozwala ustawić omawiane już wcześniej parametry CPOL i CPHA. Oba wykorzystane przez nas układy mogą pracować w trybie 0, zatem taki wybieramy, ale nie wierzcie na słowo i sami sprawdźcie tę informację w notach katalogowych.
7. *Synchronizer…* – ta opcja umożliwia dodanie na wejściach układu dodatkowych bloków synchronizacyjnych, o czym pisałem już w wypadku układu UART. Opcję tę pozostawiamy bez zmian.

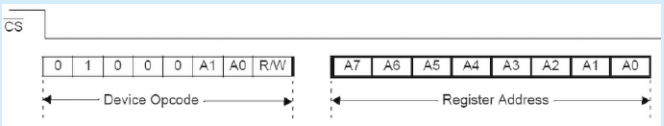
Po dokonaniu tych ustawień pozostało nam w zasadzie tylko przyłączenie odpowiednich sygnałów (wraz z przerwaniem) w Platform Designer (**rysunek 7**) oraz „wyeksportowanie" połączeń zewnętrznych. Usuńmy (lub wyłączmy za pomocą zaznaczeń w kolumnie



Rysunek 7. Widok systemu po dodaniu modułu SPI

in	clk_clk	Input	PIN_L3	3.3-V LVTTL
out	led_export[3]	Output	PIN_R16	3.3-V LVTTL
out	led_export[2]	Output	PIN_P16	3.3-V LVTTL
out	led_export[1]	Output	PIN_N16	3.3-V LVTTL
out	led_export[0]	Output	PIN_M16	3.3-V LVTTL
in	reset_reset_n	Input	PIN_B10	3.3-V LVTTL
in	spi_5v_MISO	Input	PIN_H16	3.3-V LVTTL
out	spi_5v_MOSI	Output	PIN_H15	3.3-V LVTTL
out	spi_5v_SCLK	Output	PIN_J16	3.3-V LVTTL
out	spi_5v_SS_n[1]	Output	PIN_J15	3.3-V LVTTL
out	spi_5v_SS_n[0]	Output	PIN_L16	3.3-V LVTTL
in	spi_MISO	Input	PIN_B2	3.3-V LVTTL
out	spi_MOSI	Output	PIN_C1	3.3-V LVTTL
out	spi_SCLK	Output	PIN_D1	3.3-V LVTTL
out	spi_SS_n	Output	PIN_E1	3.3-V LVTTL

Rysunek 8. Przypisanie pinów w projekcie



Rysunek 9. Sposób rozpoczynania komunikacji z układem MCP23S08

Dobre praktyki – czyli jak przeciwdziałać problemom.

W przypadku podpinania układów zewnętrznych do jakiegokolwiek systemu zawsze powinniśmy zakładać, że nie znamy stanu w jakim się znajdują (np. zawartości rejestrów konfiguracyjnych). Dlatego, aby uchronić się przed nieprzyjemnymi niespodziankami powinniśmy w pisanych przez nas programach uwzględniać:

1. Sprzętowe zresetowanie układów zależnych (podrzędnych, *slave*), np. MCP23x08 ma taką możliwość.
2. Programowe zresetowanie układów, np. BMP280 posiada rejestr umożliwiający wywołanie programowego resetu.
3. Jeśli żadne z powyższych nie jest możliwe lub akceptowalne (np. chcemy zaoszczędzić 1 pin układu i nie podłączymy sygnału reset) to powinniśmy zapisać wszystkie rejestry układu, które mają znaczenie dla jego pracy, a także przewidzieć wszystkie możliwe sytuacje (np. to że układ MCP23S08 może mieć już aktywowany tryb adresowania).

Zanim przyłączymy jakikolwiek układ (czy inne urządzenie/moduł) warto sprawdzić, jaki program aktualnie pracuje w układzie FPGA – w skrajnym wypadku podłączenie bez przemyślenia może doprowadzić do uszkodzenia (np. dojdzie do zwarcia sygnałów wyjściowych). Warto pamiętać także, że po utracie zasilania zacznie w układzie pracować program zapisany w pamięci Flash układu FPGA, którą powinniśmy programować tylko w naprawdę uzasadnionych przypadkach. Warto więc na czas eksperymentów wgrać do pamięci Flash konfigurację, która ustawi wszystkie dostępne wyprowadzenia jako wejścia – uchroni nas to przed niespodziankami – a następnie operować już tylko na pamięci RAM. Wtedy w razie utraty zasilania wszystkie układy będą bezpieczne.

Use) nieużywane elementy (SEGMENT, SW, DISPLAY), standardowo przypiszmy adresy bazowe oraz numery przerwań, nadajmy modułowi SPI stosowną nazwę i możemy generować nasz projekt.

Po tej czynności wykonujemy standardowo – *Analysys & Synthesis*, po czym uzupełniamy przypisanie pinów, wcześniej usuwając w *Pin Planner* przypisania związane z usuniętymi modułami (SEGMENT, SW, DISPLAY – **rysunek 8**) i dokonujemy pełnej kompilacji projektu, po czym wgrywamy go do układu FPGA.

Czas na poważną rozmowę „w języku SPI”

Teraz już najwyższy czas, aby znów wrócić do środowiska *Eclipse* (i wygenerować odpowiedni projekt oprogramowania wraz z BSP) w celu napisania stosownego oprogramowania.

MCP23S08

Na początek dołączmy do naszego procesora jeden 8-bitowy ekspander wejścia/wyjścia. Łączymy go z naszą płytką zgodnie z wcześniej

Register Name	Address (hex)	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	POR/RST value
IODIR	00	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0	1111 1111
IPOL	01	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0	0000 0000
GPINTEN	02	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2	GPINT1	GPINT0	0000 0000
DEFVAL	03	DEF7	DEF6	DEF5	DEF4	DEF3	DEF2	DEF1	DEF0	0000 0000
INTCON	04	IOC7	IOC6	IOC5	IOC4	IOC3	IOC2	IOC1	IOC0	0000 0000
IOCON	05	—	—	SREAD	DISSLW	HAEN*	ODR	INTPOL	—	--00 000-
GPPU	06	PU7	PU6	PU5	PU4	PU3	PU2	PU1	PU0	0000 0000
INTF	07	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INTO	0000 0000
INTCAP	08	ICP7	ICP6	ICP5	ICP4	ICP3	ICP2	ICP1	ICP0	0000 0000
GPIO	09	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0	0000 0000
OLAT	0A	OL7	OL6	OL5	OL4	OL3	OL2	OL1	OL0	0000 0000

* Not used on the MCP23008.

Rysunek 10. Rejestry układów MCP23X08

wspomnianą tabelą. Połączenie nie powinno stwarzać nam znaczących trudności, więc przyjrzyjmy się nocie katalogowej i metodzie rozpoczynania „rozmowy" z naszym bohaterem.

Na początku, po wymuszeniu na linii CS odpowiedniego poziomu logicznego, wysyłamy adres układu (*Device Opcode*) – **rysunek 9**. Zawiera on część stałą, następnie dwa bity (A1…A0), które po włączeniu odpowiedniej funkcji w rejestrze kontrolnym są definiowane za pomocą poziomów odpowiednich wyprowadzeń układu (w przeciwnym razie niezależnie od poziomu tych pinów przyjmują one wartość 00), a na końcu znajduje się bit sygnalizujący kierunek transmisji (1 dla odczytu, 0 dla zapisu). Kolejny nadawany bajt to adres rejestru w układzie. Po nim albo wysyłamy bajt, który ma zostać do danego rejestru zapisany, albo nadajemy dowolny bajt, w czasie którego układ odsyła nam odpowiedź.

Spójrzmy na rejestry, które mają układy z serii MCP23X08 (**rysunek 10**). Te, które nas szczególnie interesują, to:

- IODIR – definiuje kierunek każdego z pinów: 1 to wejście, zaś 0 to wyjście.
- GPPU – kontroluje podciąganie do zasilania (*pull-up*) dla każdego z pinów (1 powoduje włączenie podciągania). Ma sens jedynie dla pinów skonfigurowanych jako wejścia.
- IOCON – rejestr konfiguracyjny, interesujące dla nas na chwile obecną jest ustawienie bitu HAEN, w celu możliwości korzystania z pinów adresowych układu, o czym przekonamy się później.
- GPIO – rejestr zapisu i odczytu stanu pinów. Przy zapisie ustawia stan na wyjściach (dla pinów skonfigurowanych jako wejścia nie ma to znaczenia), zaś przy odczycie odczytuje stan wszystkich pinów (zarówno wejść jak i wyjść.

Aby uporządkować funkcje związane z obsługą ekspanderów, najlepiej utworzyć trzy pliki. Dwa nagłówkowe, z czego jeden zawierać będzie definicje adresów rejestrów, a drugi prototypy funkcji, oraz plik źródłowy, zawierający ciała wszystkich funkcji. Na poniższym listingu przedstawiono fragment pliku źródłowego, który definiuje funkcje zapewniające dostęp do rejestrów naszego układu, oraz umożliwiające adresowanie układów za pomocą linii CS, oraz podając adres układu na magistrali SPI (**listing 1**).

Najważniejsze jednak jest tutaj zauważenie i omówienie jak działa dostarczona przez producenta funkcja do obsługi transmisji po SPI. Nie jest ona może idealna, ale spełnia swoją rolę w większości typowych sytuacji (w wypadku nietypowych sytuacji polecam podejrzeć źródło tej funkcji i napisać na jej podstawie własną, dostosowaną do potrzeb).

Na początku definiujemy 2 bufory (2 tablice). Następnie w danej funkcji do jednej z tablic zapisujemy dane, które chcemy wysłać za pomocą interfejsu SPI i wywołujemy funkcję *alt_avalon_spi_command*. Jej kolejne argumenty mają następujące znaczenie:

1. Adres bazowy modułu SPI, który wykorzystujemy.
2. Numer pinu CS, który należy aktywować podczas transmisji.
3. Liczba bajtów do nadania z tablicy…
4. …oraz wskaźnik na tę tablicę.

Listing 1. Definiowanie funkcji zapewniających dostęp do rejestrów MCP23S08

```
uint8_t spiWriteData[3];
uint8_t spiReadData[1];

void MCP23X08writeReg(uint8_t cs, uint8_t addr, uint8_t reg, uint8_t val){
    spiWriteData[0] = MCP23x08_ADDR | MCP23x08_SPI_W | ((addr & 0b11)<<1);
    spiWriteData[1] = reg;
    spiWriteData[2] = val;
    alt_avalon_spi_command(SPI_5V_BASE, cs, 3, spiWriteData, 0, spiReadData, 0);
}

uint8_t MCP23X08readReg(uint8_t cs, uint8_t addr, uint8_t reg){
    spiWriteData[0] = MCP23x08_ADDR | MCP23x08_SPI_R | ((addr & 0b11)<<1);
    spiWriteData[1] = reg;
    alt_avalon_spi_command(SPI_5V_BASE, cs, 2, spiWriteData, 1, spiReadData, 0);
    return spiReadData[0];
}
```

Listing 2. Odczyt rejestru GPIO i wyświetlenie jego stanu

```
int main()
{
    MCP23X08writeReg(1, 0, MCP23x08_IODIR, 0x0F);
    MCP23X08writeReg(1, 0, MCP23x08_GPPU, 0x0F);
    while(1){
        uint8_t data = MCP23X08readReg(1, 0, MCP23x08_GPIO);
        IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, data);
        MCP23X08writeReg(1, 0, MCP23x08_GPIO, ~(data << 4));
    }
    return 0;
}
```

Listing 3. Modyfikacja umożliwiająca komunikowanie się z dwoma układami na magistrali SPI

```
...
    MCP23X08writeReg(1, 0, MCP23x08_IODIR, 0x0F);
    MCP23X08writeReg(1, 0, MCP23x08_GPPU, 0x0F);
    MCP23X08writeReg(0, 0, MCP23x08_IODIR, 0xFF);
    MCP23X08writeReg(0, 0, MCP23x08_GPPU, 0xFF);
    while(1){
        uint8_t data = MCP23X08readReg(1, 0, MCP23x08_GPIO);
        MCP23X08writeReg(1, 0, MCP23x08_GPIO, ~(data << 4));
        data = MCP23X08readReg(0, 0, MCP23x08_GPIO);
        IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, data);
    }
...
```

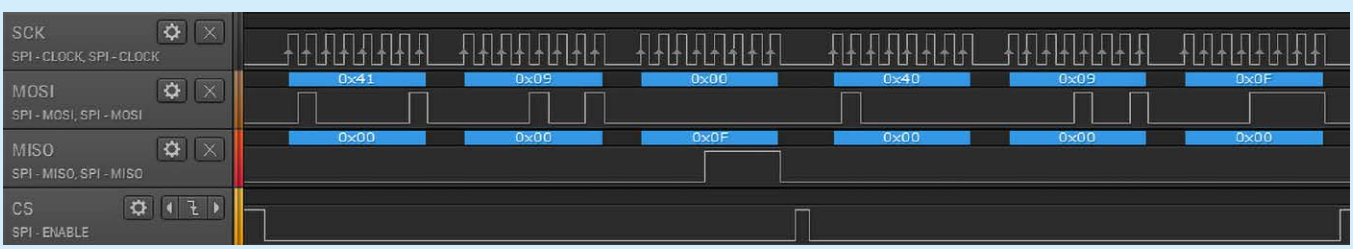
5. Liczba bajtów do odebrania…
6. …oraz wskaźnik na tablicę, w której te dane mają być zapisane.
7. Dodatkowe flagi, które umożliwiają wymuszenie aktywowania sygnału CS dla każdego bajta osobno lub pozostawienie linii CS aktywnej po wyjściu z tej funkcji.

Dziecinnie łatwe! Teraz kolej na praktyczne zastosowanie! W naszym programie musimy jedynie zdefiniować, które piny ekspandera wykorzystane będą jako wyjścia, a które jako wejścia oraz włączyć podciąganie do zasilania pinów wejściowych. Następnie, w pętli głównej odczytywać będziemy rejestr GPIO, a jego stan pokazywać na 4 diodach podłączonych do ekspandera oraz na 4 diodach na płytce maXimator, co pokazano na **listingu 2**.

Obserwacja transmisji realizowanej w czasie pętli *while* jest zaprezentowana na **rysunku 11**.

Co się tutaj dzieje? Wyjaśnijmy sobie tę sprawę od… początku:

1. Linia CS ustawiana jest w stan niski
2. Nadawany jest na linii MOSI bajt o wartości 0x41 – adres układu wraz ze wskazaniem odczytu
3. Nadawany jest adres rejestru 0x09 (rejestr GPIO)
4. Nadawany jest bajt o wartości 0x00, w czasie którego układ wysyła nam zawartość rejestru GPIO: 0x0F
5. Linia CS ustawiana jest w stan wysoki – powoduje to zakończenie transmisji i możliwość jej ponownego rozpoczęcia
6. Linia CS znów ustawiana jest w stan niski – zaczyna się kolejna transmisja



Rysunek 11. Przebiegi na magistrali SPI w czasie komunikacji z układem MCP23S08

7. Nadawany jest adres układu z ze wskazaniem zapisu (0x40)
8. Nadawany jest adres rejestru 0x09 (rejestr GPIO)
9. Nadawana jest wartość do zapisania do tego rejestru – w tym wypadku 0x0F. Niższe 4 bity są ignorowane, ponieważ te piny są wejściami, zaś pozostałe 4 oznaczają w tym wypadku, że żadna dioda nie zostanie zaświecona.

Listing 4. Włączenie używania pinów adresowych

```
...
    MCP23X08writeReg(1, 0, MCP23x08_IOCON, MCP23x08_IOCON_HAEN);
    MCP23X08writeReg(1, 0, MCP23x08_IODIR, 0x0F);
    MCP23X08writeReg(1, 0, MCP23x08_GPPU, 0x0F);
    MCP23X08writeReg(1, 1, MCP23x08_IODIR, 0xFF);
    MCP23X08writeReg(1, 1, MCP23x08_GPPU, 0xFF);
    while(1){
        uint8_t data = MCP23X08readReg(1, 0, MCP23x08_GPIO);
        MCP23X08writeReg(1, 0, MCP23x08_GPIO, ~(data << 4));
        data = MCP23X08readReg(1, 1, MCP23x08_GPIO);
        IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, data);
    }
...
```

Czas na więcej układów na magistrali

Ideą magistrali SPI jest to, że można do niej dołączyć wiele układów, które mogą być rozróżniane za pomocą linii CS. Warto więc tę możliwość przetestować! Weźmy drugi układ MCP23S08, podłączamy jego wszystkie linie równolegle z pierwszym, poza linią CS, którą tym razem podpinamy do wyjścia D0 (L16). Dzięki temu, w zależności od użytej linii CS, będziemy mogli komunikować się albo z jednym albo z drugim układem!

Listing 5. Sterowanie linią CS - wymuszenie trybu pracy SPI dla BMP280

```
//wybieram aktywowanie układu, którego pin CS podłączony jest do wyjścia [0]
IOWR_ALTERA_AVALON_SPI_SLAVE_SEL(SPI_BASE, 0b1);
//teraz aktywuję na ten układ na ~1ms
//pozwalam sobie na nadpisanie wartości rejestru,
//bo jestem pewny, że nie korzystam z innych jego bitów
IOWR_ALTERA_AVALON_SPI_CONTROL(SPI_BASE, ALTERA_AVALON_SPI_CONTROL_SSO_MSK);
ALT_USLEEP(1000);
IOWR_ALTERA_AVALON_SPI_CONTROL(SPI_BASE, 0);
```

W zasadzie, musimy jedynie dodać instrukcje wysyłające po SPI dane do drugiego układu – dotychczasowe instrukcje zostaną przez nowy układ zignorowane, gdyż jego linia CS nie będzie w czasie ich wysyłania aktywowana (**listing 3**).

Do programu z list. 2 dodaliśmy 4 linijki: na początku ustawiającą wszystkie piny nowego ekspandera jako wejścia z podciąganiem, a następnie, w pętli głównej, przepisujące stan jego wejść na diody LED umieszczone na płytce maXimatora. Banalnie łatwe, prawda?

Inne metody na kierowanie przesyłek SPI

Czasem różne układy udostępniają dodatkowe metody służące podziałowi danych między kilka „scalaków" podpiętych do magistrali SPI. Czasem producent umożliwia spięcie układów w tzw. *daisy-chain* – połączenie, w którym sygnał MISO jednego układu podłączamy do linii MOSI drugiego układu, zamiast łączyć z procesorem. Dzięki temu układy przekazują dane „w łańcuchu" między sobą, aż ostatni z nich dostarcza je do procesora.

Inną opcją oferowaną czasem przez producentów jest adresowanie układów na magistrali SPI. W tym trybie zwykle układ jako pierwszy (lub kilka pierwszych) bajt przyjmuje adres. Jeśli ten się zgadza układ zaczyna dalszą komunikację. W przeciwnym razie układ zachowuje się pasywnie do momentu zmiany stanu linii CS. Taką właśnie możliwość oferują przykładowo zastosowane układy MCP23S08.

W naszym wypadku modyfikacja programu będzie kosmetyczna – wystarczy zmienić sposób adresowania – zawsze używać tego samego pinu CS, natomiast podawać inne adresy układów. Oczywiście musimy to poprzedzić wysłaniem do wszystkich układów, czyli na domyślnym adresie 0, polecenia ustawienia wspomnianego wcześniej bitu konfiguracyjnego, który włącza używanie pinów adresowych – **listing 4**.

Co więcej – jeśli chcemy kilka układów skonfigurować w ten sam sposób, możemy takie konfiguracje wysłać przed „komendą HAEN", tym samym oszczędzając czas. Należy jednak być tu szczególnie ostrożnym, gdyż wywołanie w takiej chwili jakiegokolwiek odczytu spowoduje konflikt na magistrali.

Oprócz zmian w programie musimy dokonać kosmetycznych zmian w połączeniach. Po pierwsze zmieniamy połączenie pinu A0 drugiego układu, aby przyjął on stan wysoki, po drugie podpinamy piny CS obu układów równolegle do pinu D1 (J15). Viola, jeśli wszystko zrobiliśmy prawidłowo oba układy nadal powinny działać, a my zaoszczędziliśmy jeden pin.

BMP280

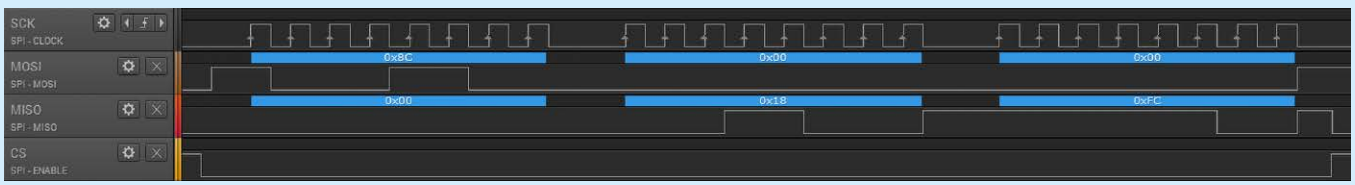
Na koniec dołączymy do naszego systemu nieco ciekawszy układ – cyfrowy czujnik ciśnienia i temperatury. Po podłączeniu go zgodnie z tabelą nieco wcześniej w artykule, powinniśmy zadbać, aby układ miał dezaktywowany

Listing 6. Odczyt 2-bajtowego rejestru

```
uint8_t spiWriteData[2];
uint8_t spiReadData[3];
//...
uint16_t BMP280Read16U(uint8_t address){
    spiWriteData[0] = address & BMP280_SPI_R;
    alt_avalon_spi_command(SPI_BASE, 0, 1, spiWriteData, 2, spiReadData, 0);
    return (((uint16_t)spiReadData[0])<<0) | (((uint16_t)spiReadData[1])<<8);
}
```

Register Name	Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset state
temp_xlsb	0xFC	temp_xlsb<7:4>				0	0	0	0	0x00
temp_lsb	0xFB	temp_lsb<7:0>								0x00
temp_msb	0xFA	temp_msb<7:0>								0x80
press_xlsb	0xF9	press_xlsb<7:4>				0	0	0	0	0x00
press_lsb	0xF8	press_lsb<7:0>								0x00
press_msb	0xF7	press_msb<7:0>								0x80
config	0xF5	t_sb[2:0]			filter[2:0]				spi3w_en[0]	0x00
ctrl_meas	0xF4	osrs_t[2:0]			osrs_p[2:0]			mode[1:0]		0x00
status	0xF3					measuring[0]			im_update[0]	0x00
reset	0xE0	reset[7:0]								0x00
id	0xD0	chip_id[7:0]								0x58
calib25...calib00	0xA1...0x88	calibration data								individual

Rysunek 12. Mapa rejestrów układu BMP280



Rysunek 13. Przebiegi podczas transmisji SPI

Listing 7. Obliczenie ciśnienia

```
int main()
{
    alt_putstr("Pomiar temperatury i cisnienia\r\n");
    BMP280SetSPI();
    BMP280Init();
    while(1){
        temp = BMP280ConvertT(BMP280ReadT());
        press = BMP280ConvertP(BMP280ReadP());
        printf("%07ld %07lu\r\n", temp, press);
        ALT_USLEEP(100000);
    }
    return 0;
}
```

Jak już wspomniałem dokonujemy tego w przypadku czujnika BMP280 za pomocą podania stanu niskiego na pin CS. Jak to osiągnąć? Najpierw musimy w rejestrze wyboru układu modułu SPI wskazać (za pomocą maski bitowej, a nie liczby odpowiadającej numerowi kolejnego układu), który pin CS chcemy sterować (nie mamy na tej magistrali SPI dużego wyboru). Następnie musimy polecić modułowi SPI ręczne aktywowanie tej linii, po czym dezaktywować ją (**listing 5**). W normalnym trybie moduł aktywuje linię CS wskazanego układu (lub wskazanych układów – możemy podać maskę wybierającą na raz większą liczbę układów) na czas pojedynczej transmisji (czyli u nas 8 bitów). Zwykle układy, z którymi będziemy „rozmawiać" wymagają aktywowania linii CS na dłuższy czas, stąd też przedstawiona powyżej metoda jest nieodzowna.

I teraz poważne pytanie – co dalej? Rzecz jasna musimy poszperać w nocie katalogowej czujnika i przeczytać ją bardzo uważnie (**rysunek 12**)! Ponieważ powinniśmy poświęcić uwagę głównie obsłudze interfejsu SPI, pozwolę sobie lekturę noty katalogowej zostawić Wam na zadanie domowe, zaś teraz jedynie omówić te kwestie w telegraficznym skrócie.

Po pierwsze, należy zapisać odpowiednie wartości do rejestru *ctrl_meas*, aby wyprowadzić czujnik ze stanu uśpienia oraz włączyć wykonywanie pomiarów temperatury i ciśnienia. Następnie musimy odczytać dane kalibracyjne, które są konieczne do wyliczenia temperatury i ciśnienia, i możemy przystępować do odczytu danych. W komunikacji po interfejsie SPI z tym czujnikiem najpierw nadajemy 1 bajt – adres, do którego chcemy się „dostać", z tym, ze najstarszy bit (8) oznacza kierunek transmisji – 1 dla odczytu i 0 dla zapisu (zatem w celu zapisu do rejestru kontrolnego wysłalibyśmy wartość 0x74 zamiast 0xF4). Teraz albo przesyłamy dane do zapisu, albo wysyłamy jakiekolwiek dane, które są ignorowane i odczytujemy dane, które są zapisane pod wskazanym wcześniej adresem. Co ważne, układ po przesłaniu nam pojedynczego bajta danych automatycznie zwiększa adres odczytu o 1, więc możemy bez ponownego podawania adresu odczytać kolejne komórki (wykorzystamy to np. przy odczycie danych kalibracyjnych pogrupowanych po 2 bajty, czy przy odczycie danych o ciśnieniu i temperaturze)

Wszystkie potrzebne funkcje umieściłem w pliku *bmp280.c* (któremu towarzyszą 2 pliki nagłówkowe). Przykładowo odczyt 2 bajtowego rejestru może wyglądać tak, jak pokazano na listingu 6. Taka transmisja powoduje wygenerowanie przebiegów widocznych na **rysunku 13**. Co się tutaj dzieje?

1. Linia CS ustawiana jest zerowana – wybieramy układ czujnika ciśnienia do komunikacji.
2. Transmitowany jest adres (0x8C), który chcemy odczytać z pamięci (najstarszy bit ustawiony na 1 wskazuje na odczyt). Układ w tym czasie przesyła nam wartość 0 – czyli nic ;)
3. Transmitowany jest bajt o wartości 0, a w tym czasie układ wysyła nam dane spod adresu 0x8C (dane mają tu wartość 0x18).
4. Układ wewnętrznie inkrementuje adres po czym w kolejnym cyklu wysyła dane o wartości 0xFC spod adresu 0x8D.
5. Linia CS ustawiana jest w stan wysoki – koniec komunikacji

```
Pomiar temperatury i cisnienia
0002751 0993945
0002751 0993940
0002752 0993949
0002751 0993960
0002751 0993958
```

Rysunek 14. Wyniki pomiaru parametrów środowiskowych

Teraz właściwie cały trick polega jedynie na wykonywaniu podobnych transmisji we właściwy sposób oraz obliczanie (według dosyć karkołomnego wzoru) na podstawie odebranych danych temperatury oraz ciśnienia. Przypomnę tylko, gdyby ta informacja Wam umknęła, że aby prawidłowo wyliczyć ciśnienie należy wcześniej odczytać i obliczyć temperaturę – wzory podane w nocie przez producenta uwzględniają korektę odczytu czujnika w zależności od temperatury. Ostatecznie, ukrywając komunikację w kodzie naszej „biblioteki" główny program będzie wyglądał, jak na **listingu 7**.

Naszym oczom w konsoli *Nios II Console* (korzystamy z transmisji danych po JTAG) ukaże się obraz zamieszczony na **rysunku 14**. Oznacza to (zgodnie z opisami w pliku ze źródłami funkcji) temperaturę 27.51°C oraz ciśnienie 993.945 hPa. Jako zadanie domowe, jak się zapewne domyślacie, należy tak zmodyfikować wyświetlanie, aby uwzględniało ono kropkę oraz jednostkę.

W naszym programie są także 2 dodatkowe wady: po pierwsze stosujemy znów instrukcję *ALT_USLEEP*, zaś po drugie – nasza „biblioteka" działa tylko i wyłącznie z takim układem SPI, jak użyty w naszym projekcie. Odpowiedź na pierwszy zarzut jest krótka – potrzebujemy prostego opóźnienia do celów demonstracyjnych. Co do drugiego zarzutu – powinniście bez problemu zmodyfikować te pliki tak, aby w jednym miejscu móc zdefiniować adres bazowy modułu SPI z którego korzystają funkcje związane z BMP280 oraz numer wyprowadzenia CS, które jest połączone z tym układem. Ponadto warto by na podstawie noty katalogowej uzupełnić także plik z definicjami rejestrów (*bmp280_regs.h*), aby zawierał on dane pozwalające na dostęp do wszystkich zakamarków układu. Kolejnym usprawnieniem mógłby być odczyt ID układu BMP280, w celu weryfikacji poprawności podłączenia oraz jego programowy reset (o czym wspominałem w ramce powyżej). Ale te wszystkie dodatki zostawiam już Waszej kreatywności.

Podsumowanie

Na koniec muszę wspomnieć o paru kwestiach – po pierwsze – każdy układ może mieć zupełnie inny mechanizm prowadzenia komunikacji po SPI. Na tym polu producenci mają sporą dowolność i czasem stosują mniej lub bardziej intuicyjne protokoły. Nauka z tego faktu jest taka, że zawsze należy bardzo dokładnie czytać noty katalogowe (począwszy od poprawnego trybu SPI, a skończywszy na zawiłościach związanych z rejestrami).

Po drugie – czasem zdarza się, że w czasie transmisji znaczących danych w jednym kierunku, także w drugim kierunku dane są transmitowane – w takim wypadku konieczna byłaby albo modyfikacja omawianej przez nas funkcji, albo napisanie od podstaw własnej, w oparciu o działania na rejestrach modułu SPI.

Myślę, że udało mi się skutecznie zapoznać Was z tematyką magistrali SPI, a także przedstawić sposób jej implementacji w ekosystemie NIOS II. Zachęcam do dalszych eksperymentów oraz podjęcia próby obsłużenia innych układów. W czasie kolejnego spotkania zajmiemy kolejną ważną magistralą, jaką jest I²C.

Piotr Rzeszut, AGH

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