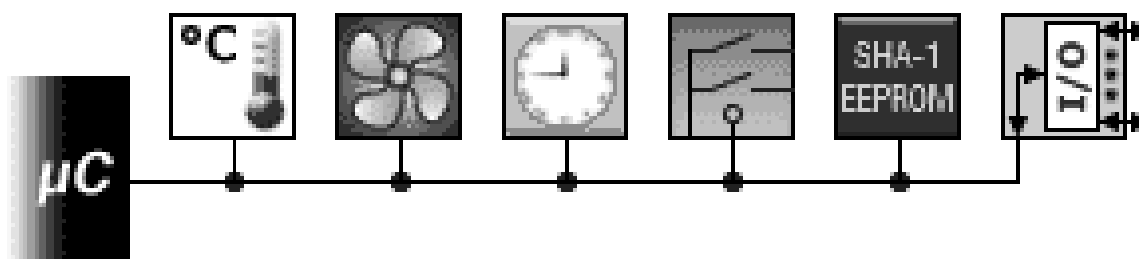


OBSŁUGA INTERFEJSU 1-WIRE NA PRZYKŁADZIE DS18B20

wydanie drugie

*Opracowanie zawiera treści różnych publikacji takich jak:
książki, datasheety, strony internetowe*



Cezary Klimasz

Kraków 2010

cezary.klimasz@gmail.com

Spis treści

1. Przedstawienie standardu 1-wire	str. 3
2. Dokumentacja układu DS18B20	str. 4
3. Obsługa termometru DS18B20 w języku C	str. 19
4. Podsumowanie	str. 24
5. Bibliografia	str. 24
DODATEK – Podstawowe operacje bitowe	str. 25

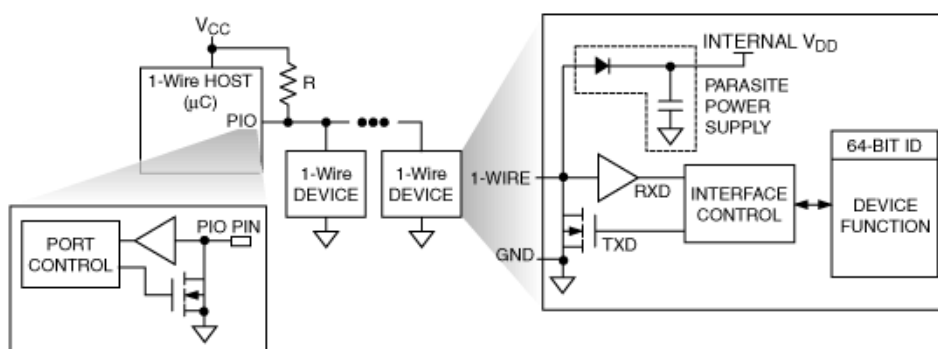
1. Przedstawienie standardu

WPROWADZENIE

1-Wire jest rodzajem interfejsu elektronicznego jak również i protokołu komunikacyjnego pomiędzy dwoma (lub więcej) urządzeniami. Jego nazwa wywodzi się stąd, że do całkowitej komunikacji używana jest tylko jedna linia danych. Dodatkowo, odbiornik może być zasilany bezpośrednio z linii danych, wykorzystując zasilanie pasożytnicze, co jest ogromną zaletą tego interfejsu. Układ *1-Wire* wyposażony jest bowiem w kondensator o pojemności 800 pF , który jest ładowany bezpośrednio z linii danych - następnie energia w nim zgromadzona używana jest do jego zasilania.

Połączenie *1-Wire* zostało opracowane przez firmę *Dallas Semiconductor* (obecnie *Maxim*). Umożliwia ono stosunkowo niewielką przepustowość transmisji danych - standardowo 16 kbps (w trybie *overdrive* maksymalnie do 142 kbps).

Interfejs *1-Wire* jest podobny do interfejsu **I²C**, ale z uwagi na pojedynczą linię komunikacyjną jest zarówno wolniejszym, jak i tańszym rozwiązaniem. *1-Wire* jest zazwyczaj używany do komunikacji pomiędzy niewielkimi urządzeniami, takimi jak: termometry cyfrowe, instrumenty metrologiczne, sterowniki ładowania akumulatorów, zamki elektroniczne typu *iButton*, itd.



Rysunek. Podłączenie urządzeń w protokole 1-Wire

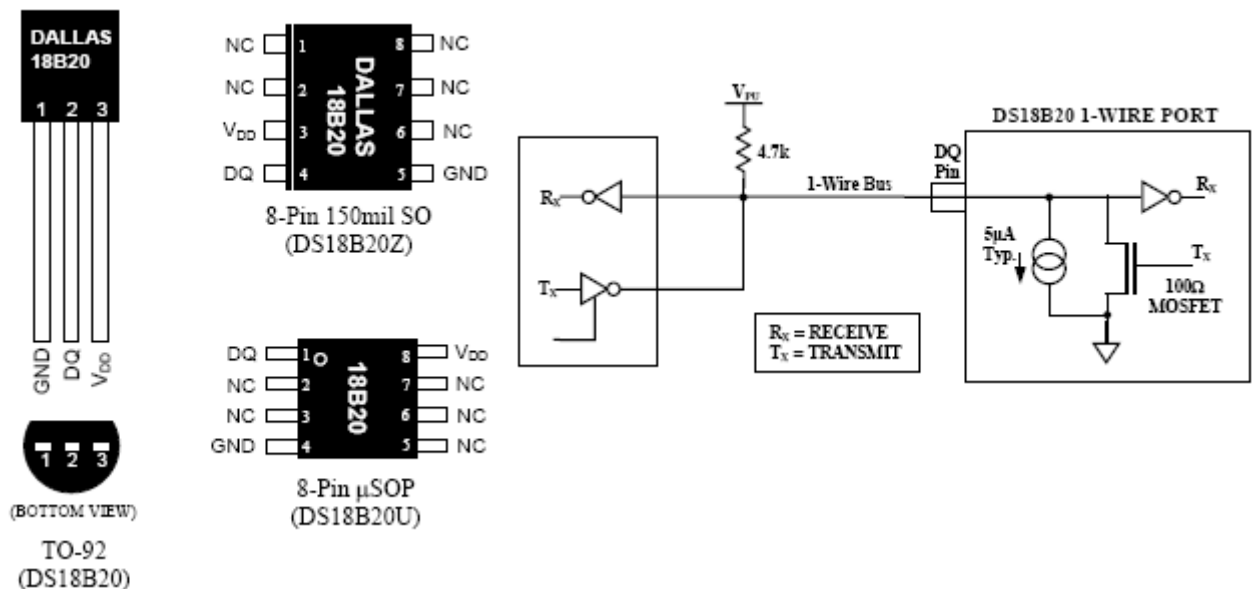
2. DOKUMENTACJA UKŁADU DS18B20



Układ **DS18B20** jest cyfrowym termometrem o programowalnej rozdzielczości. Jego podstawowe cechy to:

- o komunikacja za pomocą interfejsu **1-Wire**,
- o każdy odbiornik posiada **unikalny 64 bitowy kod** umieszczony w wewnętrznej pamięci ROM układu,
- o maksymalnie uproszczony sposób odczytu temperatury,
- o nie potrzebuje żadnych zewnętrznych komponentów,
- o może być zasilany z linii danych,
- o zasilanie od **+3V** do **+5,5V**,
- o dokładność **0,5°C** dla zakresu **-10°C ÷ 85°C**,
- o możliwość ustawienia **rozdzielczości od 9 do 12 bitów**,
- o konwersja 12 bitowego słowa – **max. 750ms**.

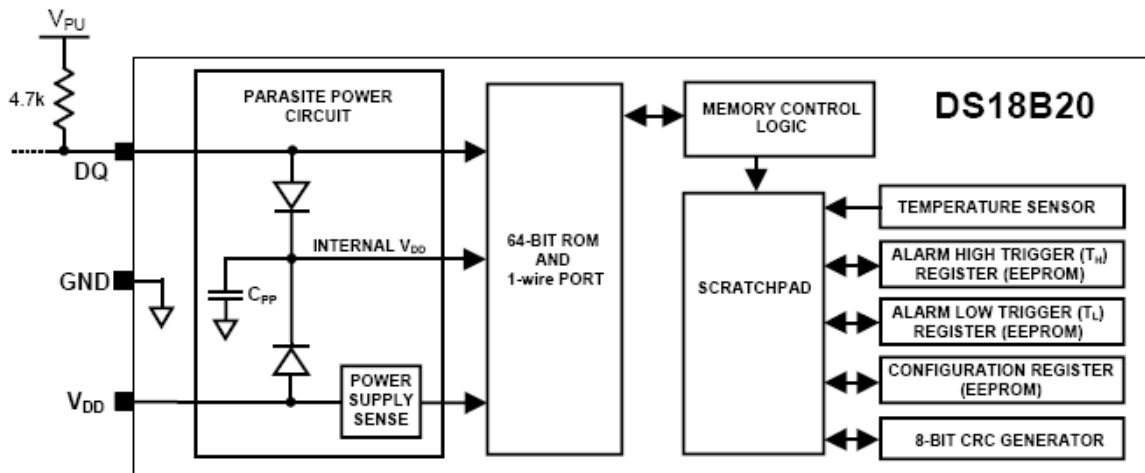
PIN ASSIGNMENT



Rysunek. Wyprowadzenia układu DS18B20 i sposób podłączenia

Rysunek poniżej przedstawia budowę logiczną układu *DS18B20*. Pamięć ROM 64-bitowa zawiera unikalny adres układu. **Scratchpad** (notatnik) zawiera 2 bajtowy rejestr z wartością temperatury dostarczoną z czujnika. W dodatku, *scratchpad* zapewnia dostęp do jedno-bajtowego rejestru porównań (T_H i T_L) oraz do jedno-bajtowego rejestru konfiguracyjnego.

Rejestr konfiguracyjny pozwala na ustawienie rozdzielczości konwersji temperatury (9, 10, 11 lub 12 bitów). T_H , T_L oraz rejestry konfiguracyjne są typu **EEPROM**, dlatego ich zawartość nie zmienia się po wyłączeniu zasilania.



Rysunek. Budowa wewnętrzna termometru DS18B20

Układ **DS18B20** korzysta z magistrali oraz protokołu *1-Wire* umożliwiającego komunikację przy użyciu tylko jednej linii sygnałowej. Linia ta potrzebuje **rezystora podciągającego**¹ (*pullup*). Dla takiej magistrali, mikrokontroler rozpoznaje i adresuje urządzenia używając unikalnych 64-bitowych kodów (tzw. **kod ROM**). Każde urządzenie ma swój własny unikalny kod, dlatego ilość urządzeń możliwych do zaadresowania w wirtualnej magistrali jest praktycznie nieograniczona. Jedną z cech układu *DS18B20* jest możliwość obsługi bez zewnętrznego źródła zasilania. Mówi się tutaj o tzw. **zasilaniu pasożytniczym**.

POMIAR TEMPERATURY

Podstawową zaletą układu *DS18B20* jest możliwość cyfrowego pomiaru temperatury. Rozdzielczość czujnika jest konfigurowalna przez użytkownika: 9, 10, 11 lub 12 bitów, odpowiadające odpowiednio dokładnościom 0.5°C , 0.25°C , 0.125°C , $0,0625^{\circ}\text{C}$. Ustawieniem domyślnym jest przetwarzanie *12-bitowe*. Aby zainicjować tryb pomiaru temperatury i przeprowadzić konwersję AC, układ master musi wysłać polecenie **Convert T (0x44)**. Po konwersji, wynik jest przechowywany w dwu-bajtowym rejestrze w pamięci *scratchpada*, zaś

¹ Dla zasilania układu DS18B20 napięciem +5V sugerowana wartość rezystora podciągającego to 4.7kΩ

układ *slave* odpowiada na to przechodząc w stan bezczynności. Jeśli *DS18B20* jest zasilany z zewnętrznego źródła, wtedy po konwersji *master* może zapytać o *read time slots*, zaś czujnik powinien odpowiedzieć, przez wysłanie zera (kiedy jest w trakcie konwersji) lub jedynki (kiedy konwersja się zakończyła). W trybie pracy pasożytniczym nie jest możliwa taka opcja.

Wyjściowa temperatura jest wyskalowana w **stopniach Celcjusza**. Dane o temperaturze przechowywane są jako *16-bitowe* rozszerzone o znak, dwie uzupełniające się liczby w rejestrze *temperature register*.

TEMPERATURE REGISTER FORMAT

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
LS Byte	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
MS Byte	S	S	S	S	S	2^6	2^5	2^4

Rysunek. Zawartość rejestru temperatury

Bit znaku **S** sygnalizuje, czy temperatura jest dodatnia ($S=0$), czy ujemna ($S=1$). Jeśli *DS18B20* jest ustawiony na 12-bitową rozdzielczość, wszystkie bity w rejestrze temperatury zawierają dane. Dla *11-bitowego* trybu, bit 0 jest niezdefiniowany. Dla *10-bitowej* rozdzielczości bity 1, 0 są niezdefiniowane, zaś dla *9-bitowej*, bity: 2, 1, 0. Poniżej znajduje się tabela z różnymi temperaturami (rozdzielczość 12-bitów) i wartościami im przypisanymi w rejestrach temperatury.

TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+125°C	0000 0111 1101 0000	07D0h
+85°C*	0000 0101 0101 0000	0550h
+25.0625°C	0000 0001 1001 0001	0191h
+10.125°C	0000 0000 1010 0010	00A2h
+0.5°C	0000 0000 0000 1000	0008h
0°C	0000 0000 0000 0000	0000h
-0.5°C	1111 1111 1111 1000	FFF8h
-10.125°C	1111 1111 0101 1110	FF5Eh
-25.0625°C	1111 1110 0110 1111	FE6Fh
-55°C	1111 1100 1001 0000	FC90h

Rysunek. Temperatury i odpowiadające im wartości liczbowe

PORÓWNANIE TEMPERATURY, USTAWIENIA ALARMU

Po wykonaniu przez układ konwersji temperatury, wartość temperatury jest porównywana z wartością zdefiniowaną przez użytkownika poprzez ustawienie wartości w rejestrze 1-bajtowym T_H i T_L . Bit znaku **S** sygnalizuje znak wartości. Jeśli wartość jest dodatnia ($S=0$), jeśli zaś ujemna ($S=1$). Rejestr ten jest komórką pamięci typu **EEPROM**, dlatego ustawienia nie są kasowane po odłączeniu zasilania.

T_H AND T_L REGISTER FORMAT

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
S	2^6	2^5	2^5	2^5	2^2	2^1	2^0

Rysunek. Zawartość rejestrów triggera alarmu T_H i T_L

Tylko **zakres od jedenastego do czwartego bitu** z rejestru **Temperature Register** jest używanych przy porównaniu. Jeśli mierzona temperatura jest niższa niż zadeklarowane T_L lub wyższa niż T_H ustawiana jest **flaga alarmu**. Flaga ta jest aktualizowana po każdym pomiarze. *Master* może sprawdzić status flagi alarmu przez wysłanie polecenia **Alarm Search (0xEC)**.

64-BITOWY KOD

Każdy układ *DS18B20* posiada **unikalny 64-bitowy numer** umieszczony w pamięci *ROM*. Osiem najmłodszych bitów dla tego typu układów to kod **0x28** (identyfikator *Family Code*). Następne 48 bitów zawiera unikalny numer. Osiem najstarszych bitów zawiera **CRC** obliczone z pierwszych 56 bitów kodu.

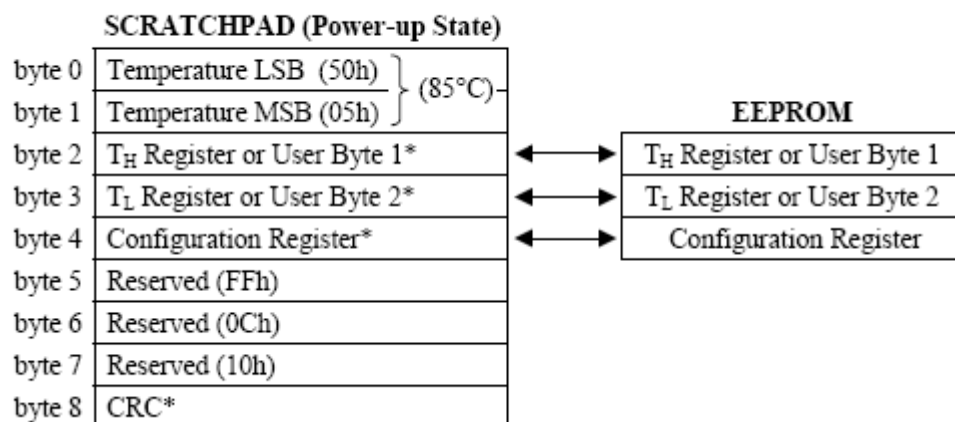
64-BIT LASERED ROM CODE

8-BIT CRC		48-BIT SERIAL NUMBER				8-BIT FAMILY CODE (28h)	
MSB	LSB	MSB	LSB	MSB	LSB	MSB	LSB

Rysunek. Numer identyfikacyjny układu *DS18B20*

PAMIĘĆ UKŁADU

DS18B20 MEMORY MAP



Rysunek. Mapa pamięci układu DS18B20

Powyżej znajduje się schemat przedstawiający pamięć układu *DS18B20*. Widoczny jest podział na pamięć ulotną i trwałą. W pamięci trwałej (*EEPROM*) znajdują się rejestry konfiguracyjne oraz T_H i T_L . Jeśli nie są używane funkcje alarmu, rejestry T_H i T_L mogą służyć jako ogólna pamięć.

Bajty 0 i 1 w *scratchpadzie* zawierają rejestry pomiaru temperatury. Bajty te służą wyłącznie do odczytu. **Bajty 2, 3** zapewniają dostęp do rejestrów T_H i T_L . **Bajt 4** zawiera informacje o konfiguracjach. **Bajty 5, 6, 7** są zarezerwowane do wewnętrznego użycia przez urządzenie i nie mogą być kasowane. Odczyt ich powoduje zwrócenie wartości jeden. **Bajt 8** w *scratchpadzie* służy wyłącznie do odczytu i zawiera wyliczone **CRC** z bajtów od 0 do 7 ze *scratchpadu*.

Dane zapisywane są do bajtów **2, 3, 4** wymagają użycia komendy **Write ScratchPad (0x4E)**. Dane muszą być wysłane do *DS18B20* rozpoczynając od najmłodszego bitu od bajtu 2. Aby zweryfikować poprawność danych możliwe jest odczytanie *scratchpadu* **Read Scratchpad (0xBE)** po zapisaniu danych. Kiedy odczytywany jest *scratchpad*, dane przesyłane są magistralą począwszy od najmłodszego bitu od bajtu nr 0. Aby przesłać dane konfiguracyjne oraz T_H , T_L ze *scratchpadu* do pamięci *EEPROM master* musi użyć polecenia **Copy Scratchpad (0x48)**. Dane z *EEPROM* mogą być również skopiowane do *scratchpadu*.

korzystając z polecenia **Recall E² (0xB8)**. Po wysłaniu tego żądania, *master* może zarządzać *Read Time Slots*, *DS18B20* zasygnalizuje wtedy swój status. Jeśli procedura *Recall* jest wykonywana zwróci zero, jeśli zakończyła się wtedy jedynką.

CONFIGURATION REGISTER

Czwarty bajt ze *scratchpada* zawiera rejestr konfiguracji, którego organizację przedstawiono poniżej.

CONFIGURATION REGISTER							
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
0	R1	R0	1	1	1	1	1

Rysunek. Rejestr konfiguracyjny układu *DS18B20*

Możliwe jest ustawienie rozdzielczości pomiaru używając bitów **R0** i **R1** w tym rejestrze. Ustawienia domyślne zakładają $R0=1$, $R1=1$ (**12 bitowa rozdzielczość**). Trzeba zaznaczyć, że istnieje zależność pomiędzy rozdzielczością a czasem konwersji. Bit 7 oraz bity 0..4 są zarezerwowane do wewnętrznego użytku przez urządzenie i nie powinny być kasowane. Poniżej przedstawiono możliwe ustawienia bitów *R0*, *R1*, wraz z szybkością konwersji.

THERMOMETER RESOLUTION CONFIGURATION

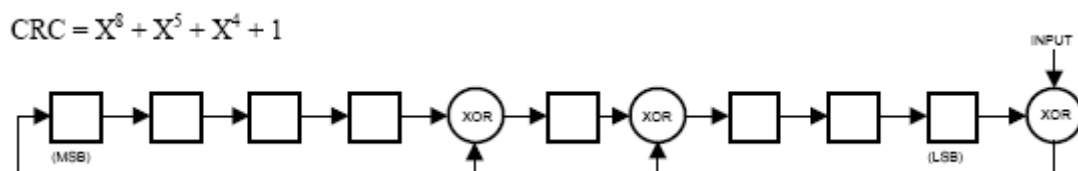
R1	R0	Resolution	Max Conversion Time	
0	0	9-bit	93.75 ms	($t_{CONV}/8$)
0	1	10-bit	187.5 ms	($t_{CONV}/4$)
1	0	11-bit	375 ms	($t_{CONV}/2$)
1	1	12-bit	750 ms	(t_{CONV})

Rysunek. Ustawienia rejestrów rozdzielczości *R1* i *R0*

GENERATOR CRC

Bajty **CRC** (*Cyclic Redudancy Check*) dostarczane są wraz z 64-bitowym kodem oraz w dziewiątym bajcie z pamięci *scratchpada*. CRC z pamięci *ROM* wyznaczany jest z pierwszych 56-bitów kodu i jest zawarte w najstarszym bajcie *ROM*. Z kolei CRC *scratchpada* liczone jest z danych umieszczonych w nim, oczywiście wraz ze zmianą zawartości *scratchpada* zmienia się również CRC. Bajt CRC dostarczany jest do układu mastera, celem sprawdzenia

poprawności transmisji. Układ nadrzędny po otrzymaniu wiadomości z bajtem *CRC*, oblicza *CRC* tej wiadomości i porównuje z dostarczonym *CRC*. Poniżej znajduje się sposób liczenia wielomianu *CRC*:



Rysunek. Wyznaczanie sumy *CRC*

Master oblicza *CRC* korzystając z zamieszczonego schematu. Obwód ten zawiera rejestry przesuwne oraz bramki **XOR**. Zaczynając od najmłodszego bitu z kodu *ROM* lub najmłodszego bitu od bajtu nr 0 ze *scratchpada*, po jednym bicie następuje przesuwanie do rejestrów. Po przesunięciu 56-stego bitu z *ROM* lub najstarszego bitu z bajtu 7 ze *scratchpada*, generator wielomianu zatrzymuje przeliczanie *CRC*. Następne 8 bitów kodu *ROM* lub *CRC scratchpada* powinno być wysłane do *mastera*. Jeśli przeliczenie *CRC* nastąpiło prawidłowo, rejestr przesuwny powinien zawierać zera.

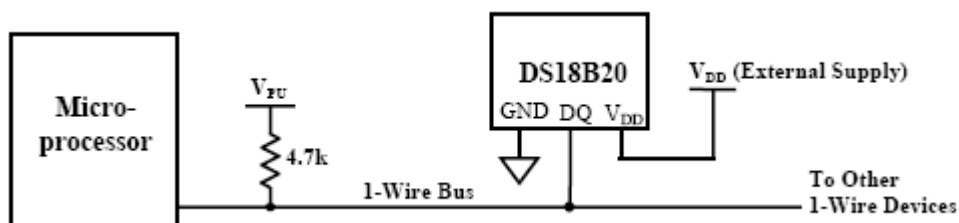
MAGISTRALA 1-WIRE

Magistrala *1-Wire* używana jest w wypadku jednego układu nadrzędnego i wielu urządzeń podrzędnych. Układ *DS18B20* jest zawsze układem podrzędnym (*Slave*). Wszystkie dane i komendy przesyłane są od najmłodszego do najstarszego bitu.

KONFIGURACJA SPRZĘTOWA

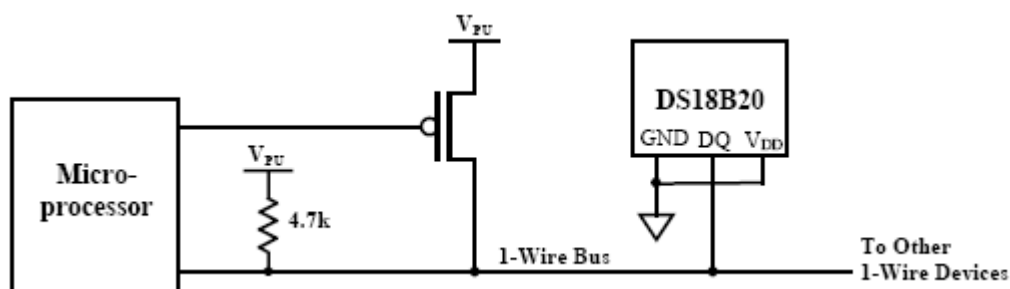
Magistrala *1-Wire* definiowana jest jako jednoprzewodowa. Wszystkie urządzenia komunikują się z linią danych przez **otwarty dren** lub **trójstanowy port**. Takie założenia pozwalają na stwierdzenie kiedy nie są transmitowane dane magistralą, tym samym kiedy magistrala może być dostępna do użytku przez inne urządzenie. Port danych układu *DS18B20* (**DQ**) jest typu **otwarty dren**. Magistrala *1-Wire* potrzebuje zewnętrznego podciągnięcia

rezystorem do zasilania (ok. $5k\Omega$). Poniżej przedstawiono rysunek połączenia fizycznego układu *DS18B20* do mikrokontrolera.



Rysunek. Zasilanie układu *DS18B20*

Powyższe połączenie zakłada doprowadzenie zasilania poprzez podanie bezpośrednio napięcia na pin V_{DD} układu. Jak już wcześniej wspomniano możliwe jest również zasilanie pasożytnicze (z linii danych). Poniżej widczony schemat takiego połączenia.



Rysunek. Zasilanie pasożytnicze układu *DS18B20*

TRANSMISJA DANYCH

Wymiana danych pomiędzy układem nadrzędnym a układem *DS18B20* odbywa się w następującej kolejności: **inicjalizacja, komendy ROM, funkcje**. Należy przestrzegać tej kolejności, bo w przeciwnym wypadku układ może nie odpowiadać na wysłane informacje. Wyjątkami od tych reguł są komendy **Search ROM** oraz **Alarm SEARCH**. Po użyciu tych komend, *master* musi wrócić do kroku pierwszego podanej sekwencji.

INICJALIZACJA

Każda transmisja na magistrali musi zacząć się sekwencją inicjalizującą. Sekwencja ta zawiera impuls **RESET PULSE** wysłany przez układ nadrzędny a następnie impuls wysłany przez układ podrzędny **PRESENCE PULSE**.

ROM COMMANDS

Po wykryciu przez układ nadrzędny **PRESENCE PULSE**, możliwa jest obsługa komend *ROM*. Poniżej przedstawione są możliwe komendy.

SEARCH ROM (0xF0)

W momencie kiedy załączane jest zasilanie, układ *Master* musi zidentyfikować kody *ROM* wszystkich urządzeń podrzędnych na magistrali. Umożliwia to określenie ilości i rodzajów układów *Slave*. Jeśli na magistrali znajduje się tylko jeden układ podrzędny, możliwe jest pominięcie wykonywania *Search ROM*, można skorzystać z komendy *Read ROM*.

READ ROM (0x33)

Komenda ta może być użyta tylko w wypadku istnienia jednego układu *Slave* na magistrali *1-Wire*. Wywołanie jej powoduje odczyt kodu *ROM* z układu podrzędnego.

MATCH ROM (0x55)

Komenda ta wywoływana jest celem zaadresowania układu podrzędnego. Na to wywołanie powinien odpowiedzieć jedynie układ podrzędny o kodzie wysyłanym *ROM*. Ewentualne, inne układy podrzędne powinny czekać na *RESET PULSE*.

SKIP ROM (0xCC)

Układ nadrzędny może używać tej komendy po to aby zaadresować wszystkie urządzenia na magistrali równocześnie, z pominięciem wysyłania jakiegokolwiek kodu *ROM*. Za przykład, można podać sytuację w której *Master* chce aby we wszystkich układach *DS18B20* znajdujących się na magistrali włączona została konwersja temperatura – zatem należy użyć komendy *Skip ROM*, po którym następuje komenda *Convert T (0x44)*.

ALARM SEARCH (0xEC)

Działanie tej komendy jest identyczne z komendą *Search ROM*, tyle, że w tym wypadku na wywołanie odpowiadają tylko układy *Slave* z ustawioną flagą alarmu. Komenda ta pozwala układowi Master ustalić, który z układów *DS18B20* na magistrali w czasie konwersji temperatury napotkał na stan alarmu. Po każdym cyklu *Alarm Search*, układ *Master* musi powrócić do pierwszego kroku sekwencji obsługi.

FUNCTION COMMANDS

Po wysłaniu na magistralę komendy *ROM* celem zaadresowania układu *DS18B20*, *Master* może wydać jedną z komend funkcji. Komendy te pozwalają układowi nadrzędnemu zapisywać i odczytywać pamięć *scratchpada*, inicjować konwersje temperatury oraz ustawiać odpowiedni tryb zasilania układu. Poniższa tabela ilustruje wszystkie te funkcje.

DS18B20 FUNCTION COMMAND SET

Command	Description	Protocol	1-Wire Bus Activity After Command is Issued	Notes
TEMPERATURE CONVERSION COMMANDS				
Convert T	Initiates temperature conversion.	44h	DS18B20 transmits conversion status to master (not applicable for parasite-powered DS18B20s).	1
MEMORY COMMANDS				
Read Scratchpad	Reads the entire scratchpad including the CRC byte.	BEh	DS18B20 transmits up to 9 data bytes to master.	2
Write Scratchpad	Writes data into scratchpad bytes 2, 3, and 4 (T_H , T_L , and configuration registers).	4Eh	Master transmits 3 data bytes to DS18B20.	3
Copy Scratchpad	Copies T_H , T_L , and configuration register data from the scratchpad to EEPROM.	48h	None	1
Recall E ¹	Recalls T_H , T_L , and configuration register data from EEPROM to the scratchpad.	B8h	DS18B20 transmits recall status to master.	
Read Power Supply	Signals DS18B20 power supply mode to the master.	B4h	DS18B20 transmits supply status to master.	

Rysunek. Komendy układu DS18B20**CONVERT T (0x44)**

Komenda ta inicjalizuje pojedynczą konwersję temperatury. Po zakończeniu konwersji, jej rezultat jest przechowywany w dwu-bajtowym rejestrze w pamięci

scratchpada, zaś układ *DS18B20* przechodzi w stan bezczynności. Jeśli urządzenie jest obsługiwane w trybie zasilania pasożytniczego, pomiędzy przedziałem czasu $10\mu s$, po wydaniu tej komendy, *Master* musi aktywować podciągnięcie magistrali. Jeśli z kolei *DS18B20* zasilany jest z zewnętrznego źródła, *Master* może wysłać *read time slots*. Jeśli odbierze od układu *Slave* wartość zero oznacza to, że układ jest w trakcie konwersji, zaś jeśli odbierze jeden to układ zakończył konwersję.

WRITE SCRATCHPAD (0x4E)

Komenda ta pozwala zapisać 3 bajty danych do układu *DS18B20*. Pierwszy bajt zapisywany jest do rejestru T_H (bajt nr 2 w *scratchpadzie*), drugi do rejestru T_L (bajt nr 3 w *scratchpadzie*), zaś trzeci do rejestru konfiguracyjnego. Dane muszą być przesłane od najmłodszego do najstarszego bitu. Wszystkie 3 bajty muszą być zapisane przed wydaniem przez układ master polecenia *reset*. W innym wypadku dane mogą zostać uszkodzone.

READ SCRATCHPAD (0xBE)

Komenda ta pozwala układowi *Master* odczytać dane umieszczone w pamięci *scratchpada*. Transmisja danych rozpoczyna się od najmłodszego bitu, od bajtu nr 0 i kontynuowana jest aż do 9-tego bajtu. Układ *Master* może wysłać żądanie *reset* aby zakończyć odczyt w dowolnym momencie.

COPY SCRATCHPAD (0x48)

Komenda ta pozwala na skopiowanie danych z rejestrów T_H , T_L oraz konfiguracyjnego (bajty 2, 3 i 4) do pamięci *EEPROM*. Jeśli urządzenie używa zasilania pasożytniczego należy zadbać o to, aby w przeciągu $10\mu s$, układ *Master* podciągnął magistralę.

RECALL E² (0xB8)

Komenda ta wywołuje alarmy wartości T_H , T_L oraz konfiguracyjne z pamięci *EEPROM* i umiejscawia je na pozycjach 2, 3, 4 w pamięci *scratchpada*. *Master* może wysłać polecenie *Read Time Slots*, celem sprawdzenia postępu komendy *RECALL*. Jeśli zostanie zwrócona wartość zero, oznacza to, że wykonywanie funkcji trwa, zaś jeden, że funkcja

została wykonana. Operacja *RECALL* jest wykonywana automatycznie po podłączeniu zasilania do układu.

READ POWER SUPPLY (0xB4)

Układ *Master* może zażądać (poprzez wysłanie *Read Time Slots*) odpowiedzi, który układ *DS18B20* umieszczony na magistrali korzysta z zasilania pasożytniczego. Podczas *Read Time Slot*, układy zasilane pasożytniczo powinny ustawić magistralę w poziom niski, zaś zasilane zewnętrznie powinny podciągnąć ją do stanu wysokiego.

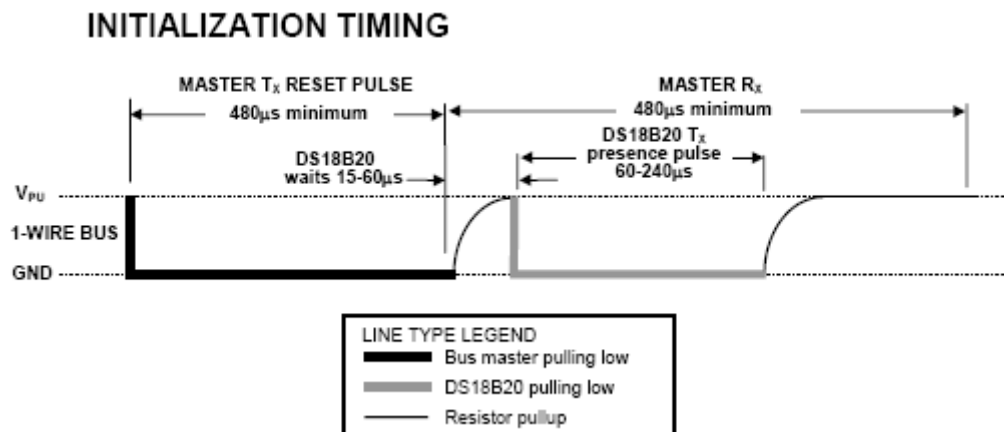
PROTOKÓŁ 1-WIRE

DS18B20 używa protokołu komunikacyjnego 1-Wire. Protokół definiuje rodzaje sygnałów takich jak: **RESET PULSE**, **PRESENCE PULSE**, **zapis 0**, **zapis 1**, **odczyt 0**, **odczyt 1**. *Master* magistrali wysyła wszystkie te sygnały oprócz *PRESENCE PULSE*.

INICJALIZACJA: RESET PULSE, PRESENCE PULSE

Każda transmisja rozpoczyna się sekwencją inicjalizacyjną zawierającą impuls resetu **RESET PULSE** wysyłany z *Mastera*, po którym następuje impuls „przedstawienia się” **PRESENCE PULSE** wysyłany przez układ *DS18B20*. W momencie kiedy *DS18B20* wysyła *PRESENCE PULSE* w odpowiedzi na *reset*, sygnalizuje to układowi *Master*, że *DS* jest gotowy do obsługi.

Podczas sekwencji inicjalizacji, *Master* wysyła reset przez ustawienie linii danych w stan niski przez minimum $480\mu s$. Po tym zabiegu *Master* zwalnia magistralę i przechodzi w tryb odbiornika. Kiedy magistrala jest zwolniona, rezystor $5k\Omega$ podciąga linie *1-Wire* do stanu wysokiego. Kiedy *DS18B20* wykryje narastające zbocze, odczeka $15 - 60\mu s$ a następnie wyśle impuls *PRESENCE PULSE*, poprzez ustawienie magistrali w stanie niskim przez $60 - 240\mu s$. Poniższy schemat przedstawia sposób inicjalizacji.



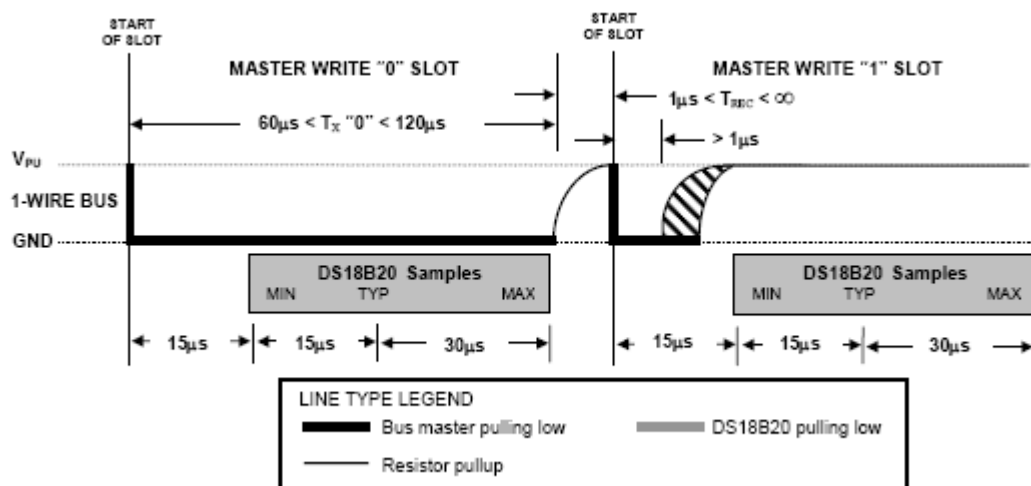
Rysunek. Zależności czasowe podczas inicjalizacji transmisji

PRZEDZIAŁY CZASOWE ZAPIS/ODCZYT

Master zapisuje dane do układu *DS18B20* podczas trwania tzw. **Write Time Slots**, odczyt z kolei następuje w czasie trwania **Read Time Slots**. Podczas każdego takiego przedziału czasowego jeden bit danych jest wysyłany lub odbierany przez magistralę.

WRITE TIME SLOTS

Wyróżnia się dwa rodzaje *Write Time Slots*: **Write 1 Time Slots** oraz **Write 0 Time Slots**. Master używa *Write 1 Time Slot*, kiedy zapisuje logiczną jedynkę do układu *DS18B20*, zaś *Write 0 Time Slot* w momencie zapisywania zera logicznego. Wszystkie te przedziały czasowe powinny trwać minimum $60\mu s$ z minimalnym okresem $1\mu s$ pomiędzy indywidualnymi *Write Slots*.



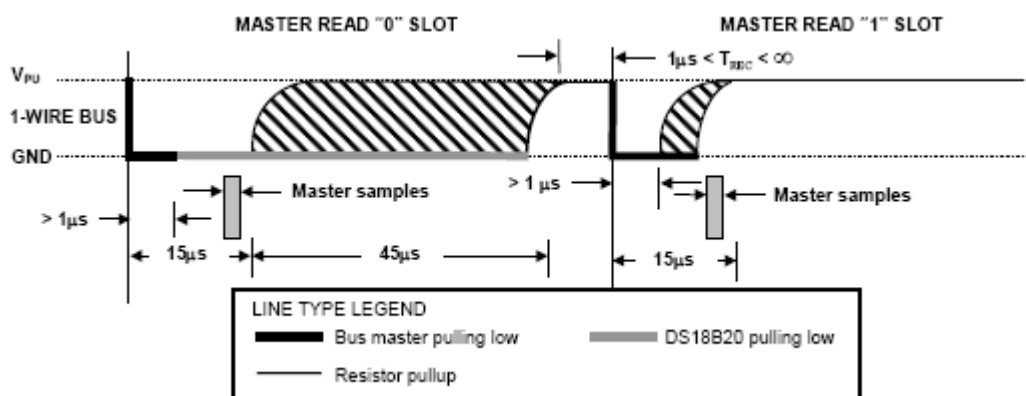
Rysunek. Zależności czasowe podczas wysyłania danych na magistralę 1-Wire

Z kolei zapisać jeden do układu *DS18B20*, po ustawieniu magistrali w stan niski, *Master* powinien zwolnić magistralę w przeciągu $15\mu s$. Kiedy magistrala jest zwolniona, rezystor podciąga linie danych do stanu wysokiego. Z kolei aby wygenerować *Write 0 Time Slot*, po ustawieniu magistrali w stan niski, *Master* powinien kontynuować trzymanie magistrali w stanie niskim, przez okres najmniej $60\mu s$.

Po tym jak *Master* zainicjuje *Write Time Slot*, *DS18B20* sprawdza poziom magistrali, oknami czasowymi o szerokości $15\mu s - 60\mu s$. Jeśli poziom magistrali jest wysoki w momencie próbkowania, wtedy zapisywana jest jedynka do układu *DS18B20*. Z kolei jeśli poziom magistrali jest niski, zapisywane jest zero do układu.

READ TIME SLOTS

Układ *DS18B20* może wysyłać dane do układu *Master* tylko w momencie, kiedy układ *Master* wygeneruje *Read Time Slot*. Należy zaznaczyć, że *Master* musi wygenerować *Read Time Slot* natychmiast po wydaniu instrukcji *Read Scratchpad* ($0xBE$), *Read Power Supply* ($0xB4$), *Convert* ($0x44$), *Recall E^2* ($0xB8$).

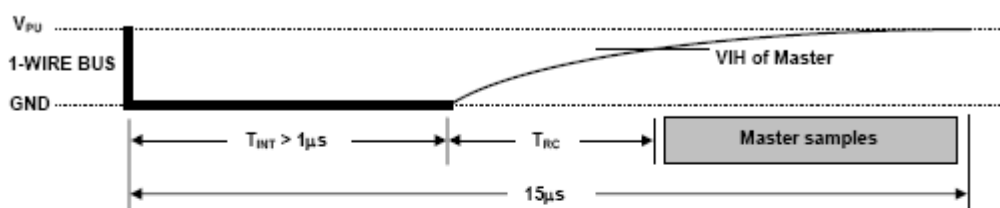


Rysunek. Odczyt danych z magistralii 1-Wire

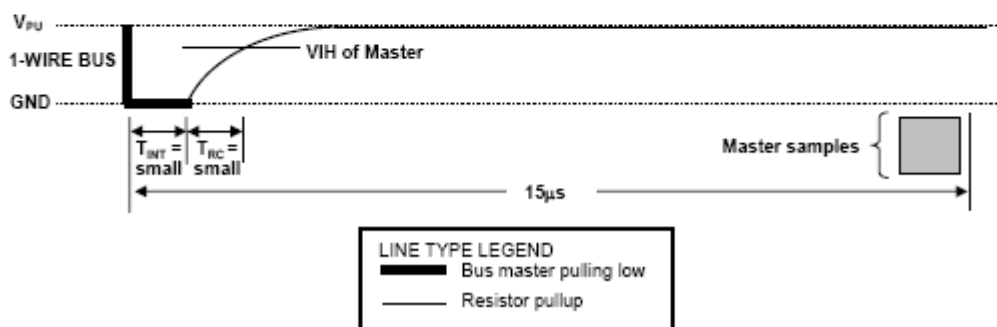
Każdy *Read Time Slot* musi trwać co najmniej $60\mu s$ z przerwami minimum $1\mu s$ pomiędzy poszczególnymi *Read Time Slots*. Takie okno czasowe jest inicjowane przez układ *Master*, przez ustawienie magistrali 1-Wire w stan niski przez minimum $1\mu s$ a następnie zwolnienie magistrali. Po inicjacji układ *DS18B20* zaczyna generować dane na magistralę (zera lub jedynki). Zera transmitowane są przez ustawienie magistrali w stanie niskim, zaś jedynki

poprzez podciągnięcie jej do stanu wysokiego. Kiedy transmitowane jest zero, *DS18B20* zwalnia magistralę po zakończeniu *time slot'a*, a następnie magistrala jest podciągana w stan wysoki przez rezystor podciągający. Dane wyjściowe są dostępne przez $15\mu\text{s}$ po zboczu opadającym, które inicjowało *read time slot*. Trzeba dodać, że *Master* musi zwolnić magistralę i próbować jej stan na przestrzeni tych $15\mu\text{s}$ od czasu wygenerowania *read time slot*. Poniżej przedstawione są szczegółowe oraz polecane sposoby inicjowania i odczytu danych przez układ *Master*.

DETAILED MASTER READ 1 TIMING



RECOMMENDED MASTER READ 1 TIMING



Rysunek. Zależności czasowe podczas odczytu danych z magistralii 1-Wire

Z dokumentacji układu *DS18B20* zaczerpnięto przykład obsługi termometru. W przykładzie tym układ zasilany jest napięciem pasywnym. *Master* inicjuje konwersję temperatury, odczytuje pamięć *scratchpada* i oblicza *CRC* aby zweryfikować odebrane dane.

Tabela na kolejnej stronie przedstawia jeden z możliwych sposobów odczytu temperatury. Trzeba zaznaczyć, że jest to kompletny kod do odczytu pamięci *scratchpada*. Nie zawsze konieczny jest odczyt jak w tym wypadku wszystkich 9 bajtów oraz odczytywania kodu termometru. Kiedy mamy do czynienia tylko z jednym termometrem na magistrali, możemy ominąć odczytywanie kodu ROM poprzez wysłanie funkcji *SKIP ROM*.

MASTER MODE	DATA (LSB FIRST)	COMMENTS
TX	Reset	Master issues reset pulse.
RX	Presence	DS18B20s respond with presence pulse.
TX	55h	Master issues Match ROM command.
TX	64-bit ROM code	Master sends DS18B20 ROM code.
TX	44h	Master issues Convert T command.
TX	DQ line held high by strong pullup	Master applies strong pullup to DQ for the duration of the conversion (t_{conv}).
TX	Reset	Master issues reset pulse.
RX	Presence	DS18B20s respond with presence pulse.
TX	55h	Master issues Match ROM command.
TX	64-bit ROM code	Master sends DS18B20 ROM code.
TX	BEh	Master issues Read Scratchpad command.
RX	9 data bytes	Master reads entire scratchpad including CRC. The master then recalculates the CRC of the first eight data bytes from the scratchpad and compares the calculated CRC with the read CRC (byte 9). If they match, the master continues; if not, the read operation is repeated.

Rysunek. Przykład komunikacji z czujnikiem DS18B20

Odczyt temperatury możemy zakończyć w momencie odczytania dwóch pierwszych bajtów – które w pamięci *scratchpada* zawierają 16 bitową informację o temperaturze.

4. Obsługa termometru DS18B20 w języku C

Podstawowe zagadnienia obsługi termometru cyfrowego *DS18B20* bazują na opisie katalogowym tego układu. Wysyłanie i odbieranie danych odbywa się za pomocą ustawiania stanu magistrali lub detekcji tego stanu przez układ *Master*. Układ *DS18B20* pracuje tylko w trybie *Slave*. Implementacja programowa obsługi tego układu opiera się o stworzenie odpowiednich procedur ustawiających linie danych w stan wysoki i niski w odpowiednich przedziałach czasowych, jak i detekcje tych stanów (nasłuchiwanie przez *Mastera*).

Program pokazujący możliwości układu *DS18B20* będzie miał za zadanie w sposób najprostszy wyświetlić temperaturę. Założono, że układ będzie zasilany zewnętrznym. Środowisko programowania to *WinAVR* wykorzystujące kompilator *GCC*. Poniżej przedstawiono ustawienia bibliotek, makr i zmiennych globalnych.

(1)

```

//biblioteki
#include <avr/io.h>
#include "lcd.h"

//definicje
#define PIN_1WIRE      0
#define PORT_1WIRE     PIND

#define OUT_1WIRE_LOW  PORT_1WIRE&=~(1<<PIN_1WIRE);
#define OUT_1WIRE_HIGH PORT_1WIRE|=1<<PIN_1WIRE;
#define DIR_1WIRE_IN   DDRD&=~(1<<PIN_1WIRE);
#define DIR_1WIRE_OUT  DDRD|=1<<PIN_1WIRE;

char cStringBuffer[8]; //zmienna potrzebna przy konwersji temperatury

```

Biblioteka *lcd.h* jest to biblioteka z pakietu *rklibavr*. Jeśli chodzi o makra, to założono, że linia danych będzie znajdowała się na **pinie 0 portu D**.

Poniższy kod przedstawia wysłanie impulsu **RESET** oraz oczekiwanie na impuls **PRESENCE**.

(2)

```

//Funkcja resetuje magistrale i oczekuje na impuls Presence
unsigned char uc1Wire_ResetPulse(void)
{
    unsigned char ucPresenceImpulse;

    OUT_1WIRE_LOW;           //ustaw port w stan niski
    DIR_1WIRE_OUT;           //kierunek pinu na: wyjście
    delayus(500);            //odczekanie 500us
    DIR_1WIRE_IN;            //kierunek pinu na: wejście
    delayus(30);             //oczekiwanie 30us

    if (bit_is_clear(PORT_1WIRE, PIN_1WIRE)) //sprawdzamy poziom linii
        ucPresenceImpulse=1;                //1-odebrano bit PRESENCE,
    else
        ucPresenceImpulse=0;                //0 - stan nieaktywności

    delayus(470);                //odczekanie przez mastera 470us

    if (bit_is_set(PORT_1WIRE, PIN_1WIRE)) //sprawdzamy poziom linii
        ucPresenceImpulse=1;
    else
        ucPresenceImpulse=0;

    return PRESENCE;             //zwracamy wartość do funkcji
}

```

Wszystkie czasy zawarte w powyższym kodzie wynikają z dokumentacji technicznej wysyłania i odbierania impulsu **RESET** i **PRESENCE**. Według dokumentacji układ Master powinien ustawić linie w stan niski na czas min. $480\mu s$, następnie ustawić w poziom wysoki i oczekiwać na reakcje ze strony układu termometru. Układ termometru w momencie wykrycia impulsu **RESET** powinien wysłać na magistralę swój impuls **PRESENCE**, polegający na ustawienie magistrali w stan niski przez $60\text{--}240\mu s$ a następnie podciągnięcie w stan wysoki.

Master ma zadanie wykryć tą sekwencję. W wypadku wykrycia zwraca do funkcji wartość jeden, w innym przypadku zero. Poniższa funkcja ilustruje sposób wysłania pojedynczego bitu do układu *DS18B20*.

(3)

```
//Funkcja wysyla na magistrale pojedynczy bit
void v1Wire_SendBit(char cBit)
{
    DIR_1WIRE_OUT;                //ustawienie kierunku pin na wy
    delayus(5);

    if(cBit==1)
        DIR_1WIRE_IN;            //zwolnienie magistralii, pin na we

    delayus(80);                  //przetrzymanie - wyslanie zera
    DIR_1WIRE_IN;
}
```

Wysłanie bitu polega na ustawieniu magistrali w stan niski, następnie jeśli po czasie min. $1\mu s$ zostanie ona ustawiona w stan wysoki, równoznaczne jest to z wysłaniem jedynki. Jeśli z kolei po wyczyszczeniu magistrali nie nastąpi w określonym czasie podciągnięcie jej do jedynki to uznaje się to za wysłanie zera logicznego. Czasy sprecyzowane są w opisie katalogowym. Poniżej zaprezentowano funkcję odbierającą pojedyncze bity z magistrali.

(4)

```
//Funkcja odbiera bit z magistrali
unsigned char uc1Wire_ReadBit(void)
{
    unsigned char ucBit;

    DIR_1WIRE_OUT;                //kierunek pinu na wy
    delayus(2);                   //odczekanie 2us
    DIR_1WIRE_IN;                 // zwolnienie magistrali
    delayus(15);                  // delay 15us

    if(bit_is_set(PORT_1WIRE, PIN_1WIRE))
        ucBit=1;                 //odbior wartosci "1"
    else
        ucBit=0;                 //odbior wartosci "0"

    return(ucBit);               //zwrocenie wartosci
}
```

Odczyt bitu z magistrali polega na wysłaniu *read time slot* a następnie sprawdzeniu zmian na magistrali. Funkcja zwraca jedynkę lub zero w zależności od stanu logicznego magistrali. Kiedy mamy już podstawowe funkcje obsługi układu *DS18B20* pozwalające na komunikację z nim, należy stworzyć funkcje ułatwiające operowanie na wysyłaniu i odbieraniu większej ilości

danych. Kolejną zaprezentowaną funkcją, będzie funkcja wysyłająca jeden bajt danych. Sposób jej realizacji opiera się na ośmiokrotnym wysłaniu pojedynczego bitu na magistralę z odpowiednimi odstępami czasowymi. Poniżej zaprezentowano tą funkcję.

(5)

```
//Funkcja wysyla bajt na magistrale
void v1Wire_SendByte(char ucByteValue)
{
    unsigned char ucCounter;           //zmienna licznikowa
    unsigned char ucValueToSend;       //wartosc do wyslania

    for (ucCounter=0; ucCounter<8; ucCounter++) //wysylanie bitow pojedynczo
    {
        ucValueToSend = ucByteValue>>ucCounter; // przesuniecie bitowe w prawo
        ucValueToSend &= 0x01;                 // skopiowanie bitu
        v1Wire_SendBit(ucValueToSend);         // wyslanie bitu na magistrale
    }
    delayus(100);                             //odczekanie 100us
}
```

Pętla *for()* odpowiada za ośmiokrotne wysłanie pojedynczego bitu danych. Operacje bitowe pozwalają na przesuwania bitów i wyodrębnienie właściwego do wysłania. Kolejną funkcją jest funkcja odbierająca bajt danych. Została zrealizowana analogicznie do funkcji wysyłającej bajt.

(6)

```
//Funkcja odbiera bajt z magistrali
unsigned char uv1Wire_ReadByte(void)
{
    unsigned char ucCounter;           //zmienna licznikowa
    unsigned char ucReadByte = 0;      //odczytana wartosc

    for (ucCounter=0;ucCounter<8;ucCounter++) //petla wykonywana 8 razy
    {
        if (uc1Wire_ReadBit())           //odczyt po jednym bicie
            ucReadByte|=0x01<<iCounter;
        delayus(15); //odczekanie 15us
    }

    return(ucReadByte);                 //zwrot wartosci
}
```

Powyższa funkcja nie wymaga komentarza. W dodatku na końcu opracowania znajdują się wytłumaczone podstawowe operacje bitowe wraz z przykładami.

Po stworzeniu funkcji wysyłających i odbierających pojedyncze bajty wystarczy analizując dokumentację oraz schemat blokowy obsługi *DS18B20* odebrać informację na temat temperatury.

Poniżej przedstawiono główną procedurę programu, w której zaimplementowano powyższy sposób komunikacji z termometrem.

```
(7)
//Główna funkcja programu
int main (void)
{
    unsigned char ucReset;
    char cTemperatureH=0, cTemperatureL=0;
    float fTemperature=0;                                     //otrzymana temperatura

    LCD_init();                                              //inicjacja wyświetlacza
    LCD_clear();                                             //wyczyszczenie wyświetlacza
    LCD_putstr_P(PSTR("1-wire"));                          //wyswietlenie napisu
    delayms(200);                                           //200ms zwłoki
    LCD_clear();                                             //wyczyszczenie wyświetlacza

    for(;;)                                                 //petla główna programu
    {
        ucReset=uc1Wire_ResetPulse();                      //impuls resetu

        if (ucReset==1)
        {
            v1Wire_SendByte(0xCC);                          //SKIP ROM
            v1Wire_SendByte(0x44);                          //CONVERT T
            delayms(750);                                    //odczekaj 750ms - czas konwersji
            ucReset=uc1Wire_ResetPulse();                    //wyslanie impulsu reset
            v1Wire_SendByte(0xCC);                          //SKIP ROM
            v1Wire_SendByte(0xBE);                          //READ SCRATCHPAD
            cTemperatureL=uv1Wire_ReadByte();                //odczytanie LSB
            cTemperatureH=uv1Wire_ReadByte();                //odczytanie MSB
            ucReset=uc1Wire_ResetPulse();                    //zwolnienie magistrali

            fTemperature=(float)(cTemperatureL+(cTemperatureH<<8))/16;

            dtostrf(fTemperature,1,1,cStringBuffer);         //konwersja float do stringa
            LCD_xy(0,0);                                     //ustawienie w pozycji 0,0
            LCD_putstr(cStringBuffer);                       //wyswietlenie temp
            delayms(200);                                    //nastepny pomiar za 200ms
        }
        else                                                 //jesli nie wykryto PRESENCE_PULSE
        {
            LCD_xy(0,0);                                     //pozycja 0,0 na LCD
            LCD_putstr_P(PSTR("cisza"));                     //napis "cisza"
        }
    }
}
```

Inicjację zawsze rozpoczyna **RESET PULSE**, w naszym wypadku odpowiada za to funkcja **uc1Wire_ResetPulse ()**. Zwraca ona wartość jeden w momencie odebrania **PRESENCE PULSE** z układu termometru. Kiedy mamy pewność, że na magistrali znajduje się układ termometru powinniśmy poprosić o kod *ROM* układu. Nie jest to konieczne w wypadku pojedynczego układu *Slave* na magistrali. Używamy zatem komendy **Skip ROM (0xCC)**. W kolejnym punkcie powinniśmy włączyć konwersję temperatury **ConverT (0x44)**. Konwersja trwa min. *750ms* dla rozdzielczości 12 bitowej (fabryczne ustawienia). Analizując schemat obsługi układu kolejnym krokiem jest wysłanie impulsu **RESET**, następnie komendy **READ SCRATCHPAD (0xBE)**. Po tej komendzie można zacząć odbierać bajty z pamięci

scratchpadu. Nas interesują tylko dwa pierwsze bajty, zawierające informację o temperaturze. Po ich odebraniu można zwolnić magistralę.

Przetworzenie odebranych na bajtów na system dziesiętkowy opiera się o podstawowe operacje arytmetyczne. W wypadku wyświetlenia temperatury jako zmiennej float (zmiennoprzecinkowa) użyto funkcji wbudowanej w bibliotekę *avrlibc*, mianowicie *dtostrf()*. Składnia tej funkcji jest następująca:

```
*char dtostrf (double _val, char _width, char _prec, char *_s);
```

W oparciu o powyższy program można stworzyć rozbudowaną aplikację umożliwiającą obsługę kilku układów termometrów na jednej linii danych. Należy wyłącznie wysyłać odpowiednie komendy na magistralę oraz postępować zgodnie ze schematem obsługi układu.

6. Podsumowanie

Układ **DS18B20** jest jednym z kamieni milowych współczesnej elektroniki. Zadziwia łatwością obsługi i możliwościami. Wynik pomiaru temperatury jest cyfrowy – łatwy do obróbki we współczesnych systemach mikroprocesorowych. Interfejs *1-Wire* rozszerza możliwości układu o wiele aspektów, ot choćby o realizację magistrali z wieloma układami *Slave*. Programowalna rozdzielczość pozwala na dosyć precyzyjne pomiary nawet z dokładnością $0,0625^{\circ}\text{C}$.

Wszystkie te zalety sprawiają, że jest to jeden z najpopularniejszych cyfrowych układów pomiaru temperatury o niskiej cenie do zastosowań różnego rodzaju.

7. Bibliografia

[1] Grębosz J.: *Symfonia C++*. Kraków 1999, Kallimach

[2] Opracowanie własne – *Programowanie mikrokontrolerów AVR w języku C*. Kraków 2007

[3] Dallas Semiconductor : *DS18B20 datasheet*.

[4] <http://wikipedia.org>

DODATEK – Podstawowe operacje bitowe

Operatory bitowe pozwalają nam na operacje na bitach typów całkowitych takich jak *char*, *int*, *long int*. W języku C mamy do dyspozycji 6 takich operatorów.

\sim - bitowa negacja

$|$ - bitowa alternatywa (OR)

$\&$ - bitowa koniunkcja (AND)

\wedge - bitowa różnica symetryczna (XOR)

\gg - bitowe przesunięcie w prawo

\ll - bitowe przesunięcie w lewo

$$a = 7|4 = (00000111)_{U2}|(00000100)_{U2} = (00000111)_{U2} = 7$$

$$a = 90\wedge(-24) = (01011010)_{U2}\wedge(11101000)_{U2} = (10110010)_{U2} = -78$$

$$a = 7\&4 = (00000111)_{U2}\&(00000100)_{U2} = (00000100)_{U2} = 4$$

$$a = 80\gg 2 = (01010000)_{U2}\gg 2 = (00010100)_{U2} = 20$$

$$a = 5\ll 4 = (00000101)_{U2}\ll 4 = (01010000)_{U2} = 80$$