

Implementacja interfejsu 1wire w VHDL przy użyciu Spartan-3E  
oraz DS18S20

Krzysztof Cabała 210047  
Kinga Wilczek 210063

# Spis treści

<b>Spis rysunków</b>	<b>3</b>
<b>1 Założenia projektowe</b>	<b>4</b>
<b>2 Wstęp teoretyczny</b>	<b>4</b>
2.1 Interfejs . . . . .	4
2.2 Komunikacja . . . . .	4
2.2.1 Inicjalizacja i reset . . . . .	4
2.2.2 Zapis bitu . . . . .	4
2.2.3 Odczyt bitu . . . . .	5
2.3 Konwersja i odczyt temperatury . . . . .	5
2.4 Double dabble . . . . .	5
<b>3 Implementacja</b>	<b>6</b>
3.1 Wykorzystanie układu Spartan 3e . . . . .	6
3.2 Schemat główny . . . . .	6
3.3 Termometer . . . . .	8
3.3.1 Controller . . . . .	9
3.3.2 ByteModule . . . . .	12
3.3.3 BusController . . . . .	17
3.4 Interpreter . . . . .	20
3.5 Double dabble . . . . .	20
3.6 LCD . . . . .	22
3.7 Pozostałe moduły . . . . .	23
<b>4 Obsługa</b>	<b>23</b>
<b>5 Podsumowanie</b>	<b>24</b>
5.1 Ocena projektu . . . . .	24
5.2 Możliwości rozbudowy . . . . .	24
<b>Literatura</b>	<b>24</b>

## **Spis rysunków**

1	Przykładowe połączenie urządzeń . . . . .	4
2	Diagram czasowy dla procedury inicjalizacji . . . . .	4
3	Diagramy czasowe dla procedury zapisu i odczytu . . . . .	5
4	Wykorzystanie zasobów układu Spartan 3e . . . . .	6
5	Schemat ogólny projektu . . . . .	7
6	Moduł Termometer . . . . .	8
7	Moduł Controller . . . . .	9
8	Maszyna stanów Controller . . . . .	10
9	Symulacja automatu Controller . . . . .	11
10	Moduł ByteModule . . . . .	12
11	Maszyna stanów BusController . . . . .	13
12	Symulacja zapisu bajtu . . . . .	14
13	Symulacja zapisu bajtu - szczegóły . . . . .	15
14	Symulacja odczytu bajtu . . . . .	16
15	Moduł BusController . . . . .	17
16	Maszyna stanów BusController . . . . .	18
17	Symbol IOBuf . . . . .	18
18	Symulacja odczytu bajtu . . . . .	19
19	Moduł Interpreter . . . . .	20
20	Moduł Double dabble . . . . .	20
21	Symulacja modułu double dabble . . . . .	22
22	Moduł LCD . . . . .	22
23	Maszyna stanów LCD . . . . .	23
24	Zrealizowany układ . . . . .	24

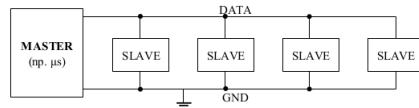
# 1 Założenia projektowe

Celem projektu było przygotowanie układu obsługującego sensor Dallas DS18S20 na platformie Xilinx Spartan-3E. W tym celu zaimplementowano obsługę magistrali One wire oraz interpretacje wyniku. Efektem działania układu jest wyświetlona wartość temperatury na wyświetlaczu LCD zgodnym ze standardem HD44780.

## 2 Wstęp teoretyczny

### 2.1 Interfejs

Interfejs 1wire jest interfejsem opracowanym przez firmę Dallas Semiconductor [1] do komunikacji między dwoma lub większą liczbą urządzeń przy wykorzystaniu zaledwie jednej lini danych, linii GND (konieczne odniesienie dla poprawnego rozpoznawania stanów logicznych) oraz zasilania Vcc. W ramach oszczędności przewodów ogranicza się połączenia do dwóch linii, wtedy układ zasilany jest pasożytniczo z linią danych. Przykładowe połączenie przedstawia schemat:



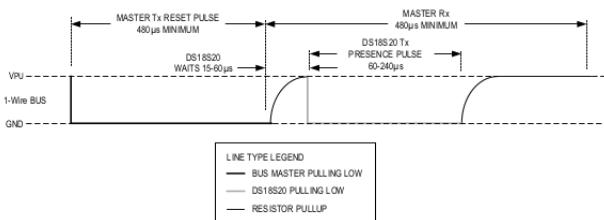
Rysunek 1: Przykładowe połączenie urządzeń

Wyróżnia się urządżania typu Master (najczęściej mikrokontroler) oraz Slave (periferia).

### 2.2 Komunikacja

#### 2.2.1 Inicjalizacja i reset

Każda próba komunikacji urządzeń master i slave musi zacząć się od sekwencji składającej się z sygnału reset, wysyłanego przez master, po którym następuje sygnał obecności układu slave. Sygnał reset to wymuszony stan 0 trwający przynajmniej  $480\mu s$ . Następnie master oczekuje na sygnał obecności innego urządzenia na linii. Następuje wówczas zwolnienie magistrali, co powoduje podciagnięcie jej do stanu wysokiego przez rezystor pull-up. Urządzenie slave wykrywa wówczas narastające zbocze linii i po upływie  $15\mu s$ - $60\mu s$  sygnalizuje swoją obecność poprzez wymuszenie stanu niskiego na okres  $60\mu s$ - $240\mu s$ .



Rysunek 2: Diagram czasowy dla procedury inicjalizacji

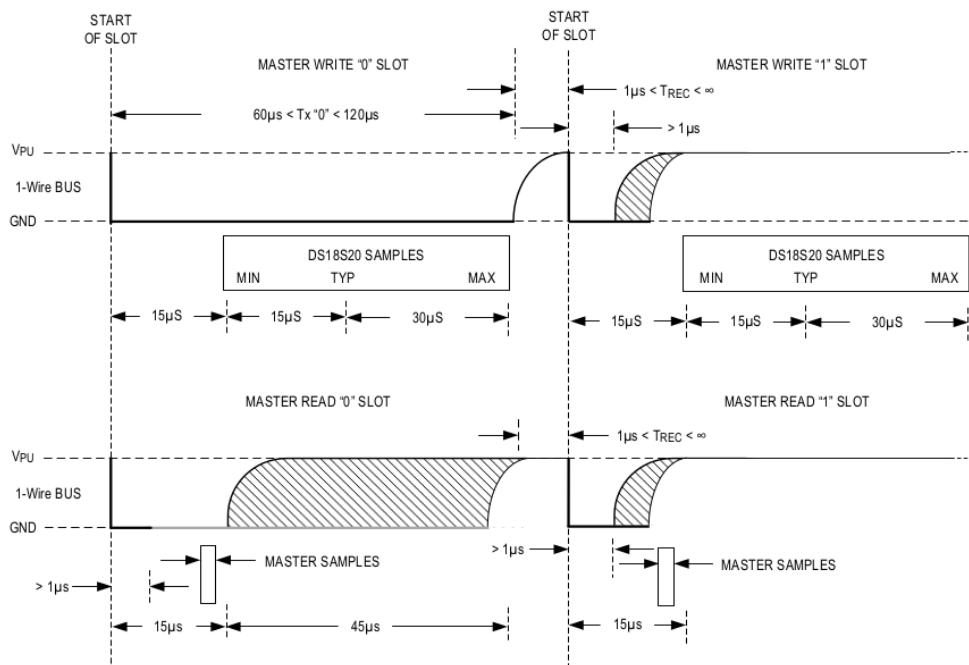
#### 2.2.2 Zapis bitu

Operacje zapisu bitu realizowane są w ścisłe określonych slotach czasowych. Długość jednego slotu wynosi zwykle  $60\mu s$ . Próbkowanie dokonywane jest mniej więcej w środku slotu celem uodpornienia na błędy. Pomiędzy kolejnymi operacjami wymagana jest przynajmniej  $1\mu s$  odstęp.

Zapis rozpoczyna się wysterowaniem linii danych przez master na poziom niski. Zapis 0 wymaga utrzymania jej w tym stanie przez cały slot. Zapis 1 jest nieco bardziej skomplikowany. Master musi w czasie nie dłuższym niż  $15\mu s$ , ale nie krótszym niż  $1\mu s$  zwolnić magistralę tak aby w momencie próbkowania (po  $30\mu s$  od zbocza opadającego) była w stanie wysokim.

### 2.2.3 Odczyt bitu

Odczyt bitu również wymaga  $60\mu s$  slotu oraz  $1\mu s$  przerwy. Odczyt rozpoczyna się wymuszeniem przez master stanu niskiego na linii danych na czas nie krótszy niż  $1\mu s$  i zwolnieniem jej (powrót do stanu wysokiego). Po tym sygnale sterowanie linią przejmuje urządzenie slave, wysyłające bit 0 lub 1. Slave po wykryciu zbocza opadającego wymusza stan niski (dla 0) lub utrzymuje wysoki (dla 1) linii danych. Sygnał musi być wtedy spróbkowany przez master. Przed upłynięciem czasu końca slotu maistrala zostaje zwolniona przez slave, co powoduje jej powrót do stanu wysokiego.



Rysunek 3: Diagramy czasowe dla procedury zapisu i odczytu

## 2.3 Konwersja i odczyt temperatury

Aktualny odczyt temperatury zapisany jest w dwóch pierwszych bajtach pamięci Scratchpad [5]. Po poprawnej inicjalizacji czujnika DS18S20 ich zawartość odpowiada temperaturze  $+85^{\circ}\text{C}$ . W celu zmierzenia aktualnej temperatury należy wysłać do termometru komendę konwersji, zresetować układ i odczytać pamięć. W pamięci urządzenia jest 9 bajtów wyniku. Tylko dwa pierwsze są istotne z punktu widzenia wyniku pomiaru. Pierwszy uzyskany bajt określa znak odczytu. Natomiast ostatni bit drugiego bajtu stanowi o wystąpieniu cyfry po przecinku. Pozostałe bity wyniku to wartość odczytanej temperatury. Wartość ta następnie podlega konwersji na kod BCD za pomocą algorytmu double dabble.

## 2.4 Double dabble

Double dabble jest powszechnie wykorzystywanym algorytmem do konwersji liczb binarnych na kod BCD (Binary-Coded Decimal - system dziesiętny zakodowany dwójkowo) [7].

Algorytm polega na wykonaniu n iteracji (w zależności od długości ciągu bitów). Początkowo wynikowy kod BCD jest zainicjalizowany jako ciąg zer podzielonych na grupy po 4 bity. Podczas każdej iteracji wykonywane jest przesunięcie o jeden bit w lewo, a na koniec doklejany jest jeden bit ciągu wejściowego. Jeżeli przed kolejnym przesunięciem wartość w jednej z grup w kodzie BCD jest wyższa niż 4 to następuje dodanie binarnej 3. Po wykonaniu odpowiedniej ilości iteracji algorytm kończy swoje działanie. Ostatecznie wynikowy ciąg jest podzielony po cztery bity, które odpowiadają kolejnym cyfrom wyniku.

### 3 Implementacja

#### 3.1 Wykorzystanie układu Spartan 3e

Implementacji układu dokonano na platformie Spartan 3e [4]. Raport wyfenerowany przez oprogramowanie (rysunek 4) Xilinx ISE pokazuje zużycie zasobów na poziomie 6% co daje możliwości dalszej rozbudowy projektu.

Device Utilization Summary					[ - ]
Logic Utilization	Used	Available	Utilization	Note(s)	
Number of Slice Flip Flops	261	9,312	2%		
Number of 4 input LUTs	365	9,312	3%		
Number of occupied Slices	287	4,656	6%		
Number of Slices containing only related logic	287	287	100%		
Number of Slices containing unrelated logic	0	287	0%		
Total Number of 4 input LUTs	408	9,312	4%		
Number used as logic	363				
Number used as a route-thru	43				
Number used as Shift registers	2				
Number of bonded IOBs	16	232	6%		
Number of BUFGMUXs	1	24	4%		
Average Fanout of Non-Clock Nets	3.01				

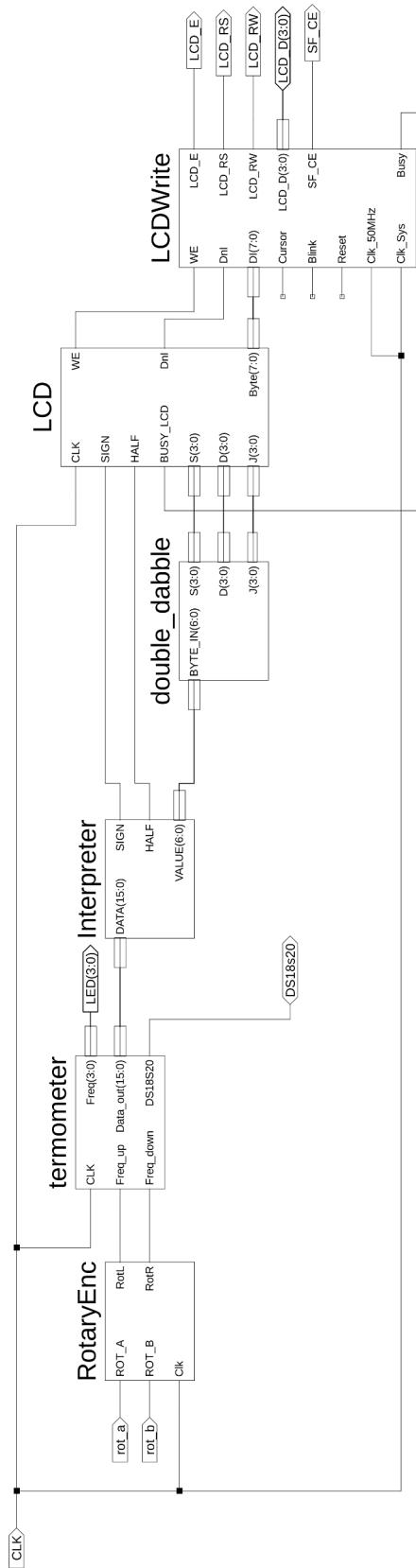
Rysunek 4: Wykorzystanie zasobów układu Spartan 3e

Raport czasowy określa maksymalną częstotliwość pracy układu na poziomie 57.071MHz, co gwarantuje jego poprawne zachowanie przy wbudowanym taktowaniu 50MHz.

#### 3.2 Schemat główny

Funkcjonalność projektu opiera się na 6 podstawowych modułach widocznych na rysunku 5:

- Termometer - obsługa sensora DS18S20;
- RotaryEnc- sterowanie częstotliwością pomiaru;
- Interpreter - interpretacja 2 bajtowego wyniku;
- double\_dabble - zamiana liczby binarnej bez znaku na kod BCD;
- LCD - generowanie zawartości wyświetlacza LCD;
- LCDWrite - sterownik wyświetlacza.

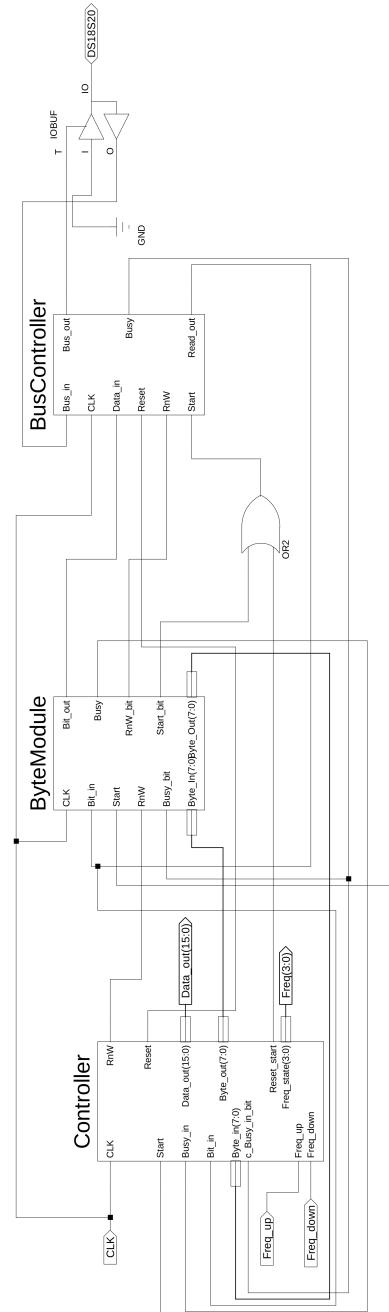


Rysunek 5: Schemat ogólny projektu

### 3.3 Termometer

Moduł Termometer jest najbardziej rozbudowanym modelem składającym się z 4 układów (rysunek 6):

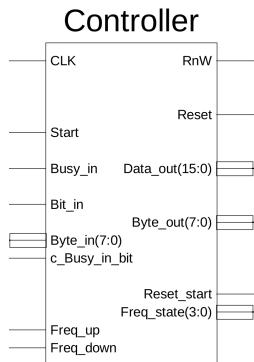
- Controller - obsługa sekwencji komunikacji i konwersji temperatury;
- ByteModule - obsługa komunikacji na poziomie bajtu (instrukcji);
- BusController - obsługa magistrali Onewire;
- IOBuf - ustala kierunek komunikacji.



Rysunek 6: Moduł Termometer

### 3.3.1 Controller

Moduł Controller (rysunek 7) odpowiada za przeprowadzenie poprawnej sekwencji operacji wymaganych do zmierzenia i odczytu temperatury.



Rysunek 7: Moduł Controller

Opis wyprowadzeń:

Wejścia

- CLK - Zegar;
- Freq\_up - Impuls zwiększenia częstotliwości próbkowania;
- Freq\_down - Impul zmniejszenia częstotliwości próbkowania;
- Byte\_in - Bajt odebrany przez ByteModule;
- Bit\_in - Bit odeberany przez BusController;
- Busy\_in - Flaga zajętości ByteModule;
- c\_Busy\_in\_bit - Flaga zajętości BusController;

Wyjścia:

- Start - Sygnał start dla ByteModule;
- Data\_out - Odebrany ciąg reprezentujący temperaturę (2 bajty);
- Byte\_out - Dane (kod instrukcji) dla ByteControllera;
- Reset - Sygnał reset;
- Freq\_state - Wektor reprezentujący aktualną częstotliwość próbkowania;
- Reset\_start - Sygnał start dla BusControllera.

W celu ograniczenia częstotliwości odczytu (co powoduje nadmierne nagrzewanie się czujnika) dodano możliwość sterowania odstępem między pomiarami. Moduł umożliwia odczyt co 4, 2, 1 sekundę lub ciągły. Zmiana odbywa się w procesie freq\_process.

```
freq_process: process (CLK, freq_up, freq_down)
begin
    if rising_edge(clk) then
```

```

    if freq_up = '1' then
        frequence <= frequence(2 downto 0) & frequence(3);
    elsif freq_down = '1' then
        frequence <= frequence(0) & frequence(3 downto 1);

    end if;
end if;
end process;

```

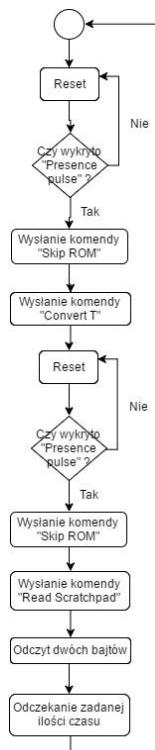
Moduł jest automatem skończonym (fsm). Zmiana stanów odbywa się w procesie state\_service.

```

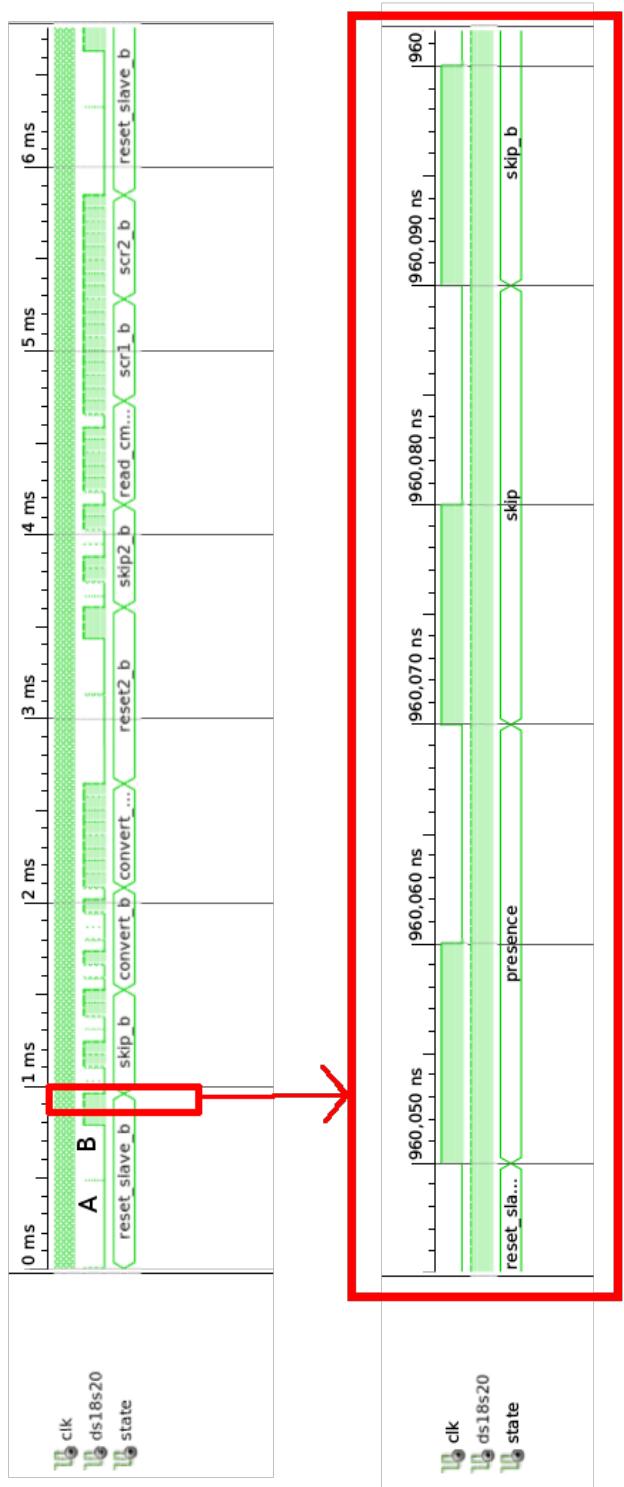
state_service: process (CLK, next_state)
begin
    if (rising_edge (CLK)) then
        state <= next_state;
    end if;
end process;

```

Kolejne stany maszyny pokazuje Rysunek 8. Po każdym ze stanów (poza presence) występuje dodatkowy stan oczekiwania na zwolnienie flagi busy (zajętości kontrolerów niższego rzędu). Stany te zostały pominięte na schemacie ze względu na jego czytelność. Należy zaznaczyć, że stany w których wykonywane są kolejne komendy są jednotaktowe, podczas gdy oczekiwanie na ich wykonanie zajmuje stosunkowo dużo więcej czasu. Zauważać to można na symulacji pokazanej na rysunku 9



Rysunek 8: Maszyna stanów Controller



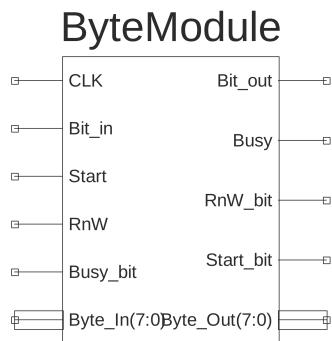
Rysunek 9: Symulacja automatu Controller

Zaznaczona na symulacji pozycja A pokazuje moment zwolenia magistrali przez master (stan wysokiej impedancji). Zaraz po jej zwolnieniu kontrolę przejmuje slave, który wymuszając stan niski sygnalizuje swoją obecność (moment B). Przybliżony fragment (czerwona ramka) obrazuje przejścia stanów pomiędzy

reset\_slave\_b a skip\_b. Widać, że stany presence oraz skip są jednotaktowe.

### 3.3.2 ByteModule

ByteModule został zaimplementowany w celu obsługi dwukierunkowej komunikacji pomiędzy master, a slave. W zależności od kierunku transmitowanego bajtu wykonywane są na nim różne operacje. W przypadku zapisu danych bajt wejściowy jest rozdzielany na pojedyncze bity, które następnie kolejno są przesyłane do urządzenia. Natomiast operacja odczytu polega na formowaniu pojedynczych bitów wejściowych w jeden bajt wyjściowy.



Rysunek 10: Moduł ByteModule

Opis wyprowadzeń:

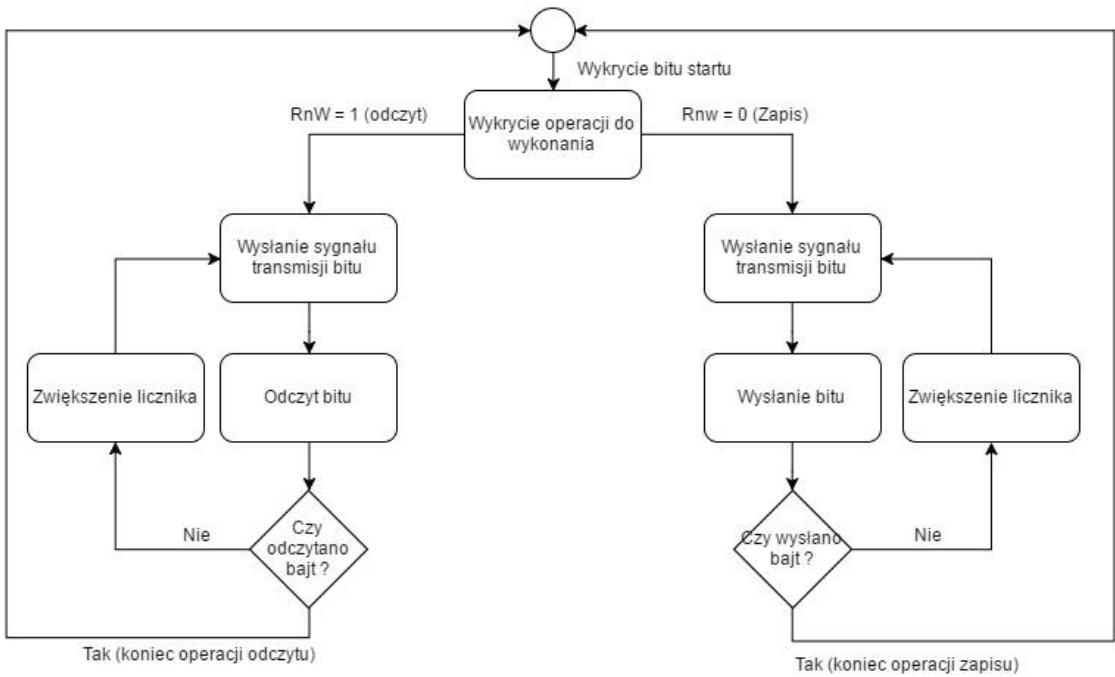
Wejścia:

- Bit\_in - Bit wchodzący podczas operacji odczytu;
- Start - Jednabitowy sygnał rozpoczęcia operacji;
- RnW - Flaga informująca o typie wykonywanej operacji. Jeżeli ustawiona w stan wysoki wykonywany jest odczyt;
- Busy\_bit - Flaga zajętości BusController;
- Byte\_In(7:0) - Wejściowy bajt przy operacji zapisu;

Wyjścia:

- Byte\_Out(7:0) - Wyjściowy bajt przy operacji odczytu;
- Start\_bit - Sygnał start dla modułu;
- RnW\_bit - Bit określający rodzaj operacji wykonywanej przez BusController;
- Busy - Bit zajętości ByteModule;
- Bit\_out - Kolejny bit zapisywany przez BusController;

Ten moduł również jest maszyną stanów. Jej uproszczony schemat przedstawia rysunek 11. Szczegóły implementacji można zobaczyć na symulacji przedstawionej na rysunkach 12 - 14,



Rysunek 11: Maszyna stanów BusController

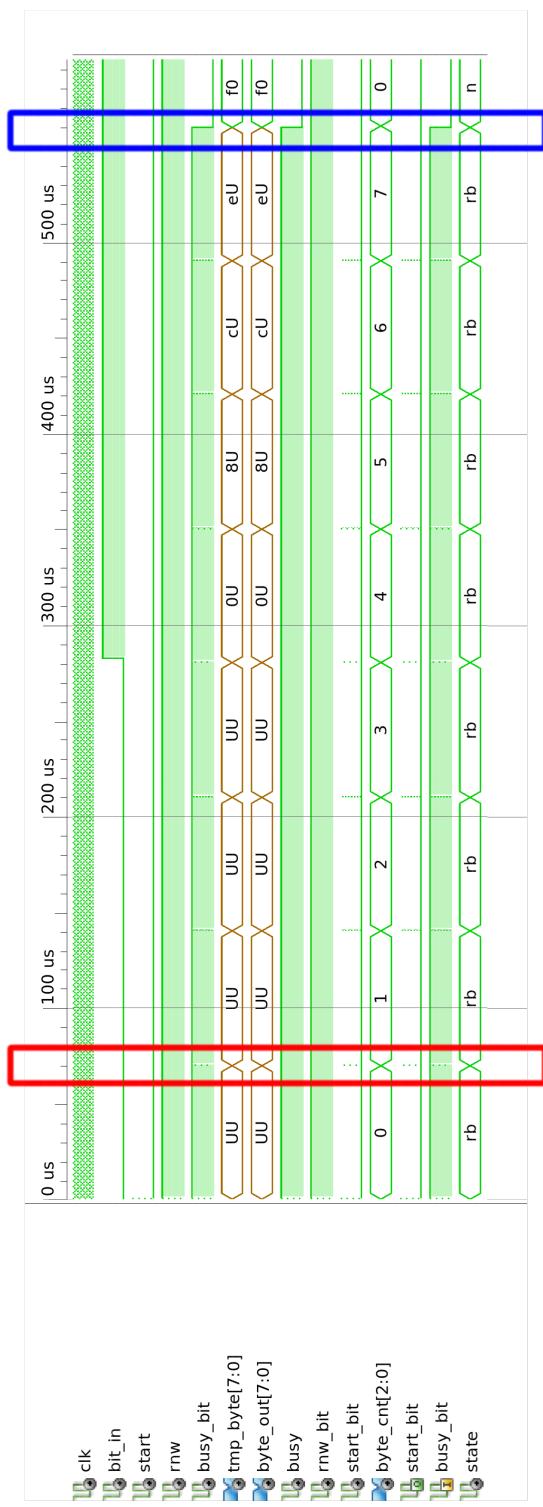
Cykl odczytu rozpoczyna stan rs w którym ustawiany jest bit RnW na wartość 1 oraz wysyłany jest jendotaktowy sygnał start\_bit. Następnie w stanie rb następuje oczekiwanie na sygnalizującą zakończenia operacji przez BusController (flaga Busy\_bit). W stanie re w rejestrze tmp\_byte zatrzaszkowana jest odczytana wartość kolejnego bitu. Wartość ta jest współbieżnie przepisywana na wyjście byte\_read. W zależności od ilości odczytanych bitów następuje przejście do stanu rc zwiększania zmiennej counter lub stanu n oznaczającego oczekiwanie na następną operację.

Za zapamiętywanie kolejnych bitów odpowiedzialny jest proces byte\_reading:

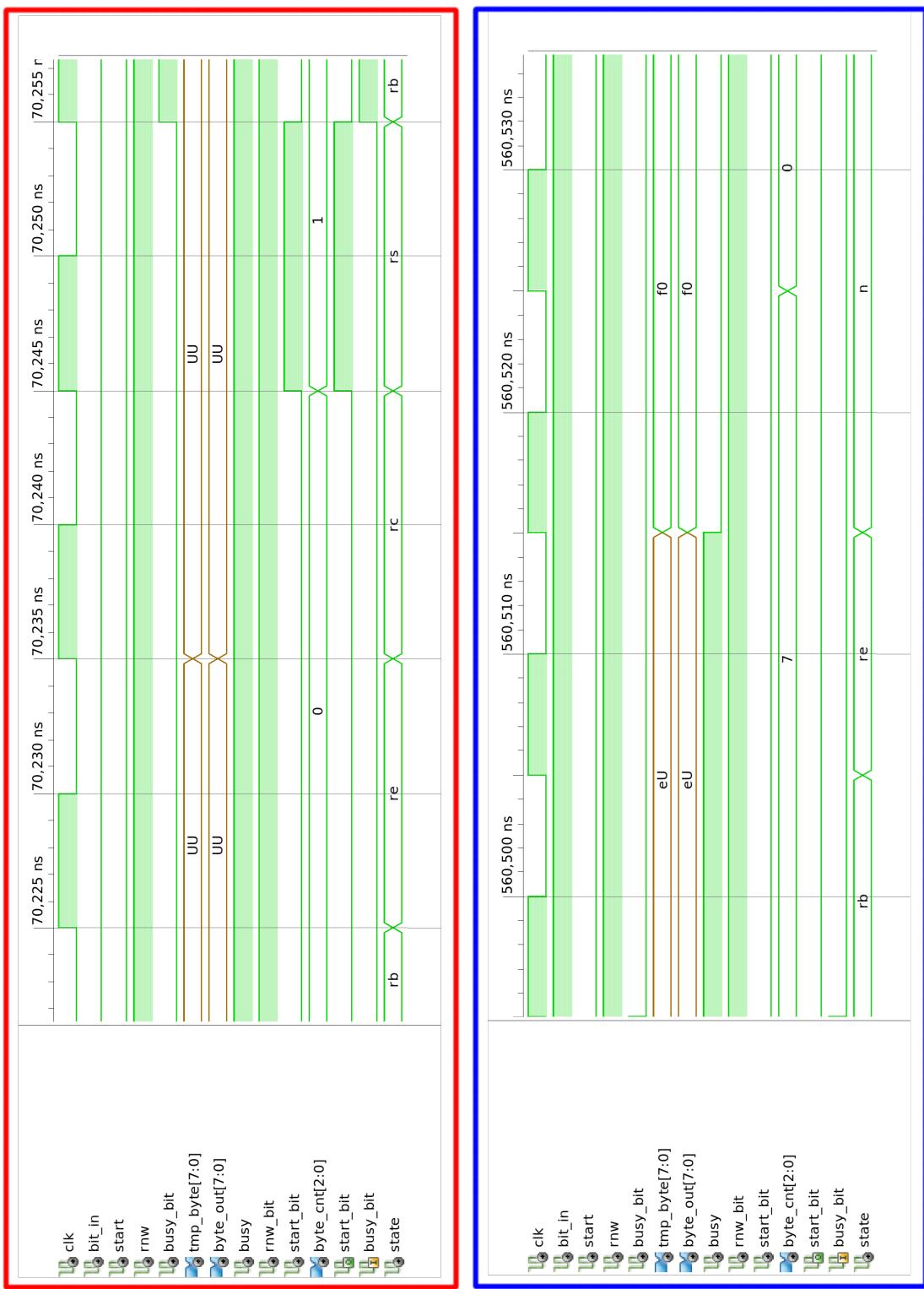
```
byte_reading: process (CLK, state)
begin
if rising_edge(CLK) then
    if state = re then
        tmp_byte <= Bit_in & tmp_byte(7 downto 1);
    end if;
end if;
end process;
```

Symulacja z rysunku 12 pokazuje przebieg całej procedury. Zaznaczone czerwoną i niebieską ramką fragmenty można zobaczyć w powiększeniu na rysunku 13. Widać na nich odpowiednio przejścia między odczytem kolejnych bitów oraz odczytem ostatniego bitu.

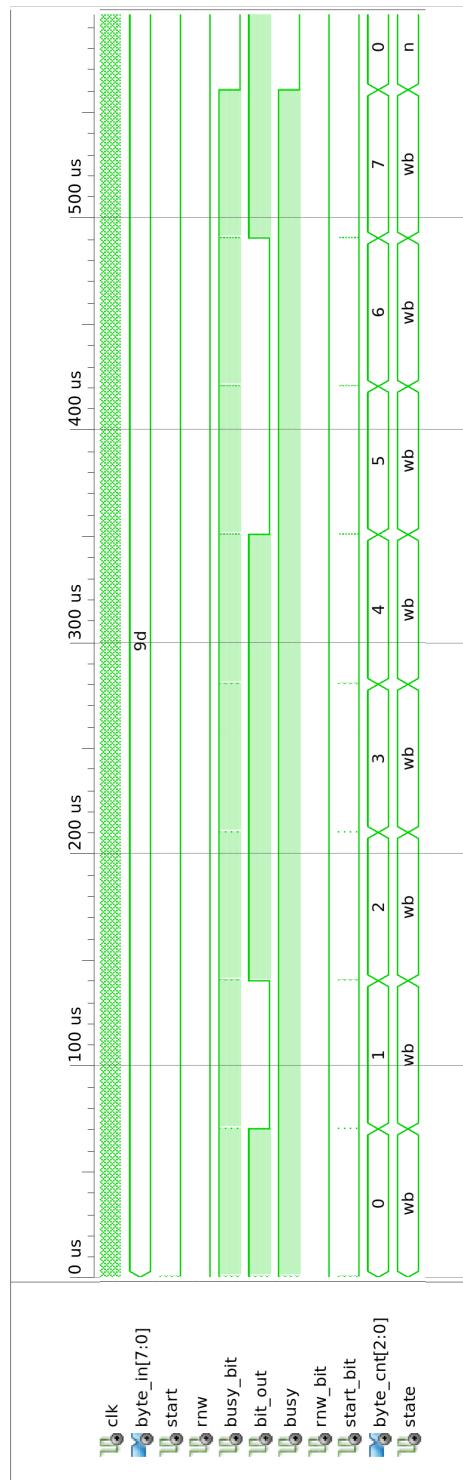
Procedura zapisu bajtu jest analogiczna do procedury odczytu. Wymienionym wcześniej stanom odpowiadają kolejno ws (RnW = 0), wb, we, wc. Wartości kolejnych bitów są odczytywane wprost z wektora byte\_in w zależności od wartości byte\_cnt. Symulację przedstawia rysunek 14



Rysunek 12: Symulacja zapisu bajtu



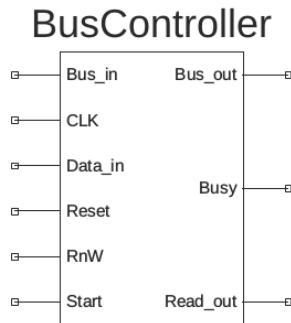
Rysunek 13: Symulacja zapisu bajtu - szczegółowy



Rysunek 14: Symulacja odczytu bajtu

### 3.3.3 BusController

Moduł BusController realizuje podstawowe operacje: reset, odczyt oraz zapis bitu.



Rysunek 15: Moduł BusController

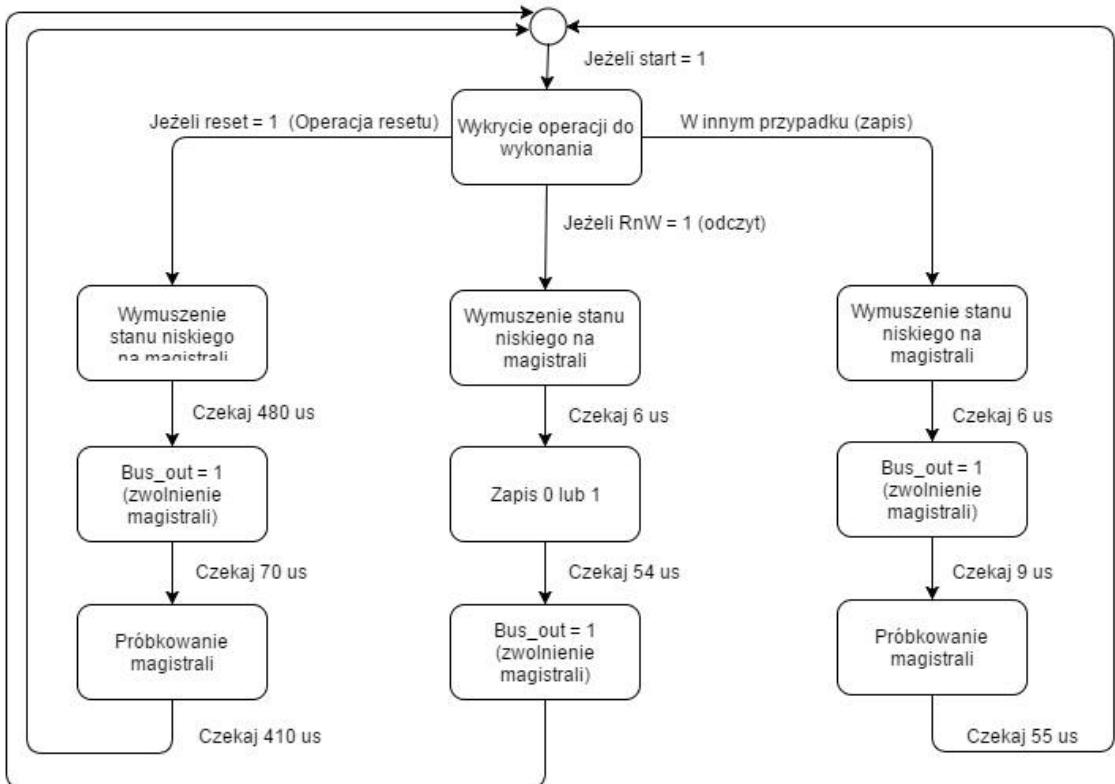
Opis wyprowadzeń: Wejścia:

- CLK - Wejście zegarowe;
- Bus\_in - Bit odczytu z termometru;
- Data\_id - Bit do zapisu z ByteModule;
- Reset - Jednobitowy sygnał resetu;
- RnW - Flaga informująca o typie wykonywanej operacji. Jeśli ustawiona w stan wysoki wykonywany jest odczyt;
- Start - Sygnal start dla BusController;

Opis Wyjść:

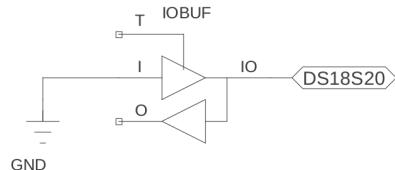
- Bus\_out - Zapis bitu do termometru;
- Busy - Bit zajętości BusController;
- Read\_out - Odczytany bit z termometru;

Poniżej przedstawiono schemat blokowy tego automatu (rysunek 16):

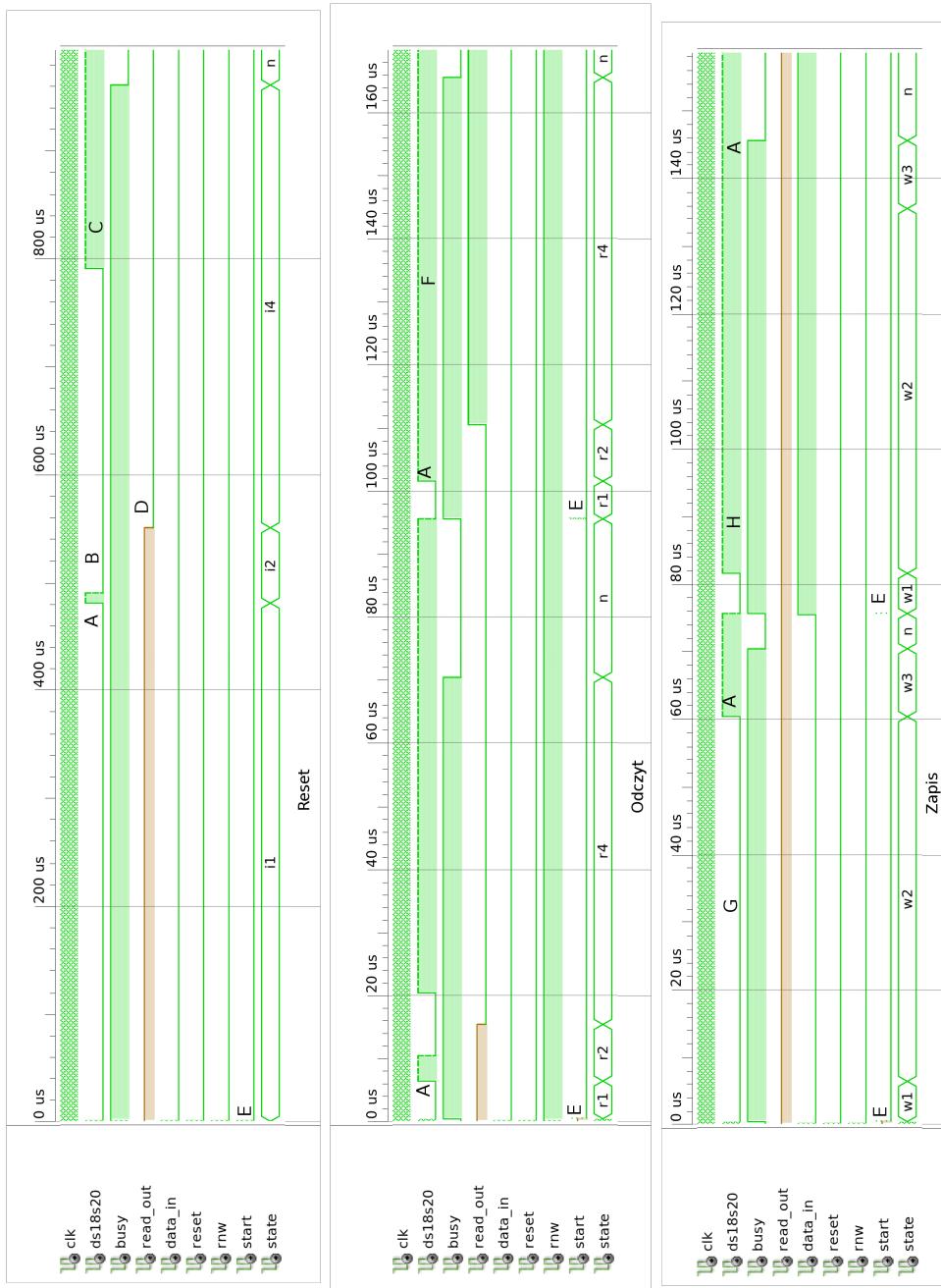


Rysunek 16: Maszyna stanów BusController

Poprawna obsługa magistrali wymaga jej zwalniania poprzez ustawienie w stan wysokiej impedancji (podciągnięcie do Vcc przez rezystor pull-up). Służy do tego element IOBuf. Podanie logicznego zera na wejściu T otwiera bufor wyjściowy i przekazuje sygnał podawany na pin I (GND - Logiczne 0). Logiczne 1 na wejściu T ustawia linie w stanie wysokiej impedancji (zwalnia magistralę). Pin O służy do odczytu stanu magistrali.



Rysunek 17: Symbol IOBuf



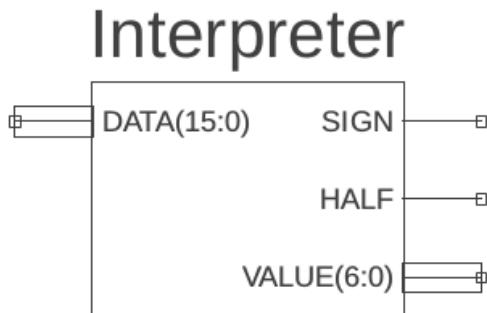
Rysunek 18: Symulacja odczytu bajtu

Na rysunku 18 przedstawiono charakterystykę pracy układu wraz z buforem IO. Zaznaczono charakterystyczne punkty:

- A Zwolnienie magistrali przez master;
- B Sygnał obecności slave;
- C Zwolnienie magistrali przez slave;
- D Sygnał obecności slave na wyjściu modułu;
- E Impuls start;

- F Magistrala w stanie wysokim - odczyt 1;
- G Master wymusza stan niski - zapis 0;
- H Master zwalnia magistralę przy zapisie - zapis 1;

### 3.4 Interpreter



Rysunek 19: Moduł Interpreter

Interpreter jest modułem kombinacyjnym, którego zadaniem jest odpowiedni podział wyniku odczytu. Jako wejście przyjmowane są dwa bajty. W wyniku konwersji pierwszy bajt zmieniany jest na znak wartości temperatury. Kolejne 7 bitów jest wartością temperatury. Ostatni bit określa wartość po przecinku.

Opis wyprowadzeń:

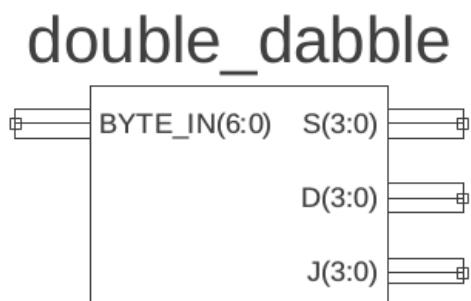
Wejścia:

- DATA(15:0) - dwa bajty wyniku uzyskanego z modułu thermometer;

Wyjścia:

- SIGN - bit znaku, gdzie stan wysoki oznacza wystąpienie wartości ujemnej;
- HALF - bit określający wartość po przecinku (0 lub 5). Jeżeli ustawiony na jeden wartość po przecinku wynosi 5;
- VALUE(6:0) - wektor wartości temperatury po konwersji;

### 3.5 Double dabble



Rysunek 20: Moduł Double dabble

Moduł ten realizuje konwersję uzyskanej wartości na kod BCD na podstawie wcześniej opisanego algorytmu double dabble. Wykorzystane do zaimplementowania tego mechanizmu zostały zmienne, z względu na zastosowanie pętli for, w wyniku czego powstała bardzo skomplikowana logika kombinacyjna.

Opis wprowadzeń:

Wejścia:

- BYTE\_IN(6:0) - wektor wejściowy konwertowany na BCD;

Wyjścia:

- S(3:0) - 4 bity cyfry setek;
- D(3:0) - 4 bity cyfry dziesiątek;
- J(3:0) - 4 bity cyfry jedności;

Algorytm double dabble realizowany jest w procesie bcd1:

```
bcd1: process(BYTE_IN)

    variable temp: STD.LOGIC_VECTOR (6 downto 0) ;
    variable jds: STD.LOGIC_VECTOR (11 downto 0):="000000000000" ;

begin
    temp := BYTE_IN;
    jds :=(others => '0');

    for i in 0 to 6 loop
        if jds(3 downto 0) > 4 then
            jds(3 downto 0) := jds(3 downto 0) + 3;
        end if;

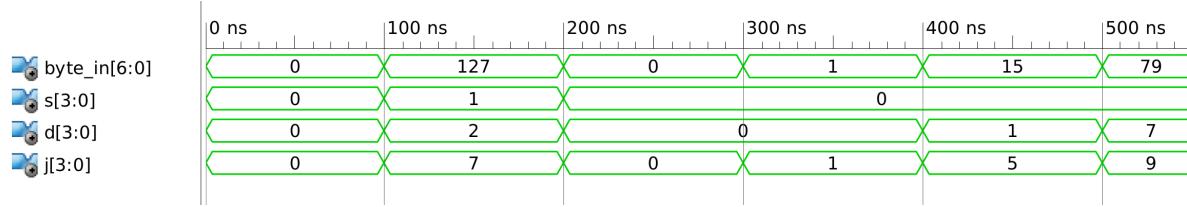
        if jds(7 downto 4) > 4 then
            jds(7 downto 4) := jds(7 downto 4) + 3;
        end if;
        jds := jds(10 downto 0) & temp(6);

        temp := temp(5 downto 0) & '0';

    end loop;

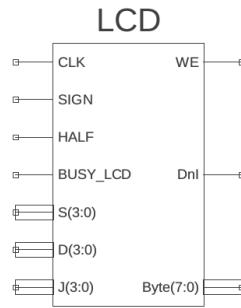
    J <= jds(3 downto 0);
    D <= jds(7 downto 4);
    S <= jds(11 downto 8);
end process bcd1;
```

Przykład działania pokazuje symulacja na rysunku 21:



Rysunek 21: Symulacja modułu double dabble

### 3.6 LCD



Rysunek 22: Moduł LCD

Moduł LCD opiera się na prostej maszynie stanów, która przesyła kolejne znaki ASCII do wyświetlacza kontrolowanego przez element LCDWrite. Ze względu na sporą ilość stanów i prostotę automatu zostanie przedstawiony jedynie uproszczony graf przejść (rysunek 23).

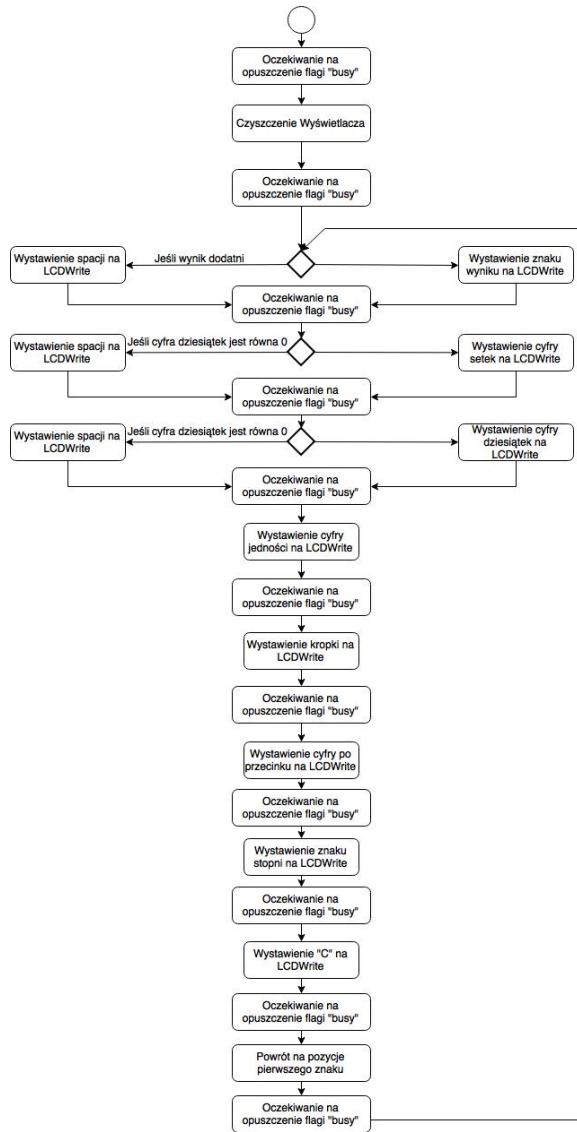
Opis wyprowadzeń:

Wejścia:

- CLK - wejście zegarowe;
- SIGN - bit określający znak wyniku;
- HALF - bit określający wartość po przecinku (0 lub 5);
- BUSY\_LCD - bit zajętości LCDWrite;
- S(3:0) - 4 bity cyfry setek;
- D(3:0) - 4 bity cyfry dziesiątek;
- J(3:0) - 4 bity cyfry jedności;

Wyjścia:

- WE - jednotaktowy impuls inicjalizujący;
- DnI - bit sterujący;
- Byte(7:0) - wektor wyjściowy znaku do wyświetlenia;



Rysunek 23: Maszyna stanów LCD

### 3.7 Pozostałe moduły

Moduły LCDWrite oraz RotaryEnc pochodzą z zasobów dr. Jarosława Sugiera, ich opis znajduje się w [2] oraz [3]

## 4 Obsługa

Gotowy układ przedstawia rysunek 24. Termometr należy podłączyć do złącza J4. Wynik pomiaru wyświetlony jest na wyświetlaczu LCD. Sterowanie częstotliwością pracy odbywa się przy użyciu enkodera znajdującego się po lewej stronie ekranu. Aktualnie wybraną częstotliwość pokazują diody LED po jego prawej stronie (zapalone skrajna prawa oznacza pomiar ciągły, 4 od prawej co 4 s).



Rysunek 24: Zrealizowany układ

## 5 Podsumowanie

### 5.1 Ocena projektu

W ramach projektu zostały zrealizowane wszystkie założenia ustalone przed przystąpieniem do pracy. Dodatkowo, już po realizacji zamierzeń, dodana została obsługa zmiany częstotliwości pomiaru. Wszystkie moduły napisane w ramach projektu były na bieżąco testowane przy użyciu symulatora ISim, co zapewnia poprawność jego pracy. Ostatecznie, układ może działać z częstotliwością około 57 MHz. Dość niska maksymalna częstotliwość wynika prawdopodobnie z algorytmu double dabble, którego implementacja wymaga złożonej logiki kombinacyjnej. Przed przystąpieniem do ewentualnej rozbudowy projektu należałoby ujednolicić konwencję nazewiczą stanów, sygnałów wewnętrznych oraz wyprowadzeń układów.

### 5.2 Możliwości rozbudowy

Cały projekt zajmuje około 6% zasobów dostępnych w układzie Spartan 3e. Daje to szerokie możliwości jego rozbudowy. Jedną z nich może być zwiększenie liczby przyłączonych czujników. Dodatkowo można go wzbogacić o obsługę komunikacji np. z komputerem PC z użyciem wybranego interfejsu. Innym rozszerzeniem może być dodanie obsługi pamięci (np. karta sd), która miałaby za zadanie zapisywać wyniki pomiarów. Tak zapisane wyniki mogłyby być reprezentowane na monitorze VGA w postaci wykresów.

## Literatura

- [1] Komunikacja 1-Wire [dostęp 23.05.2016]  
<https://www.maximintegrated.com/en/app-notes/index.mvp/id/126>
- [2] Opis modułu LCDWrite [dostęp 23.05.2016]  
[http://www.zsk.ict.pwr.wroc.pl/zsk\\_ftp/fpga/#\\_Toc286057186](http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga/#_Toc286057186)
- [3] Opis modułu RotaryEnc [dostęp 23.05.2016]  
[http://www.zsk.ict.pwr.wroc.pl/zsk\\_ftp/fpga/#\\_Toc286057181](http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga/#_Toc286057181)

- [4] Przewodnik układu FPGA Spartan 3e starter kit [dostęp 23.05.2016]  
[http://www.xilinx.com/support/documentation/boards\\_and\\_kits/ug230.pdf](http://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf)
- [5] Dokumentacja układu DS18S20 [dostęp 23.05.2016]  
<https://datasheets.maximintegrated.com/en/ds/DS18S20.pdf>
- [6] Dokumentacja biblioteki NUMERIC\_STD [dostęp 23.05.2016]  
[http://www.csee.umbc.edu/portal/help/VHDL/packages/numeric\\_std.vhd](http://www.csee.umbc.edu/portal/help/VHDL/packages/numeric_std.vhd)
- [7] Opis algorytmu Double dabble [dostęp 23.05.2016]  
[https://en.wikipedia.org/wiki/Double\\_dabble](https://en.wikipedia.org/wiki/Double_dabble)