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

Krzysztof Cabała 210047
Kinga Wilczek 210063

6 czerwca 2016

Spis treści

Spis rysunków

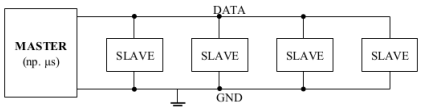
1 Założenia projektowe

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 do komunikacji między dwoma lub większą liczbą urządzeń przy wykorzystaniu zaledwie jednej lini danych, linii GND (konieczne odniesienie dla poprawnego rozpoznawania stanów logicznych) oraz zasialania Vcc. W ramach oszczędności przewodów ogranicza się połączenia do dwóch lini, wtedy układ zasilany jest pasożytniczo z lini danych. Przykładowe połacznie przedstawia schemat:



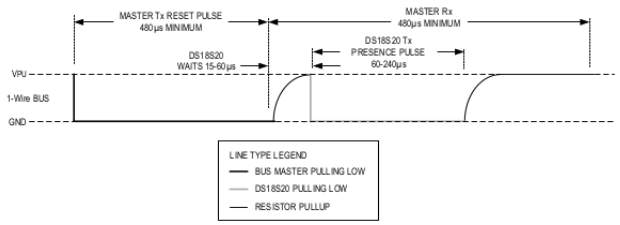
Rysunek 1: Przykładowe połączenie urządzeń

Wyróżnia się urządzania typu Master (najczęsciej mikrokontroler) oraz Slave (peryferia).

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 480μs. Następnie master oczekuje na sygnał obecności innego urządzenia na linii. Następuje wówczas zwolnienie magistrali, co powoduje podciągnięcie jej do stanu wysokiego przez rezystor pull-up. Urządzenie slave wykrywa wówczas narastające zbocze lini i po upływie 15μs-60μs sygnalizuje swoją obecność poprzez wymuszenie stanu niskiego na okres 60μs-240μs.



Rysunek 2: Diagram czasowy dla procedury inicjalizacji

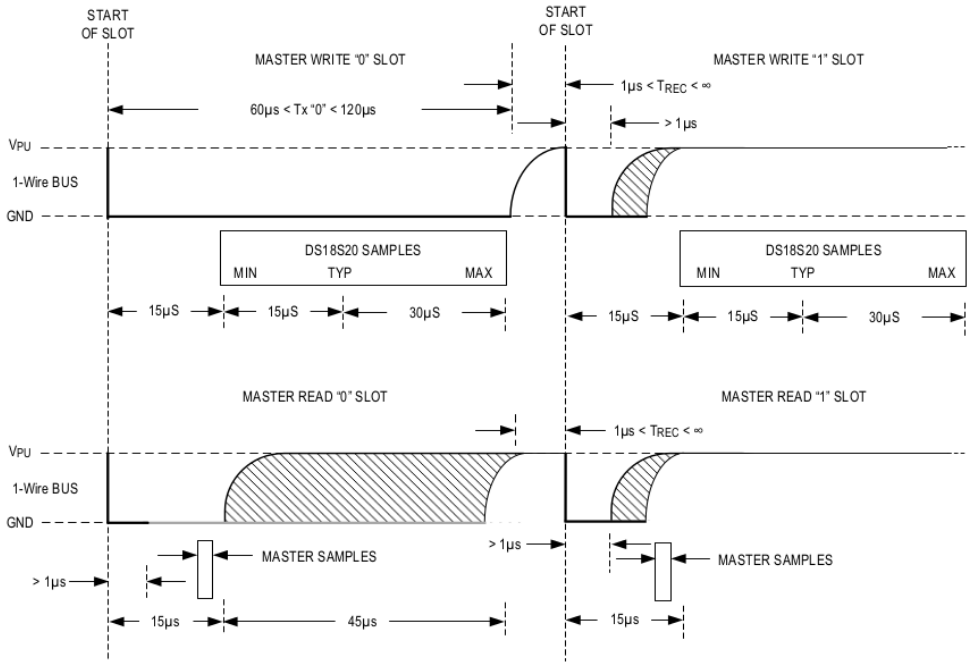
2.2.2 Zapis bitu

Operacje zapisu bitu realizowane są w ściśle określonych slotach czasowych. Długość jednego slotu wynosić zwykle 60μs. Próbkowanie dokonywane jest mniej więcej w środku slotu celem uodpornienia na błędy. Pomiędzy kolejnymi operacjami wymagana jest przynajmniej 1μs odstępu.

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 15μs, ale nie krótszym niż 1μs zwolnić magistralęm tak aby w momencie próbkowania (po 30μs od zbocza opadającego) była w stanie wysokim.

2.2.3 Odczyt bitu

Odczyt bitu również wymaga 60μs slotu oraz 1μs przerwy. Odczyt rozpoczyna się wymuszeniem przez master stanu niskiego na linii danych na czas nie krótszy niż 1μs i zwolnienie 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 maigistrala 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. Po poprawnej inicjalizacji czujnika DS18S20 ich zawartość odpowiada temperaturze +85°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 w punktu wyniku. 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).

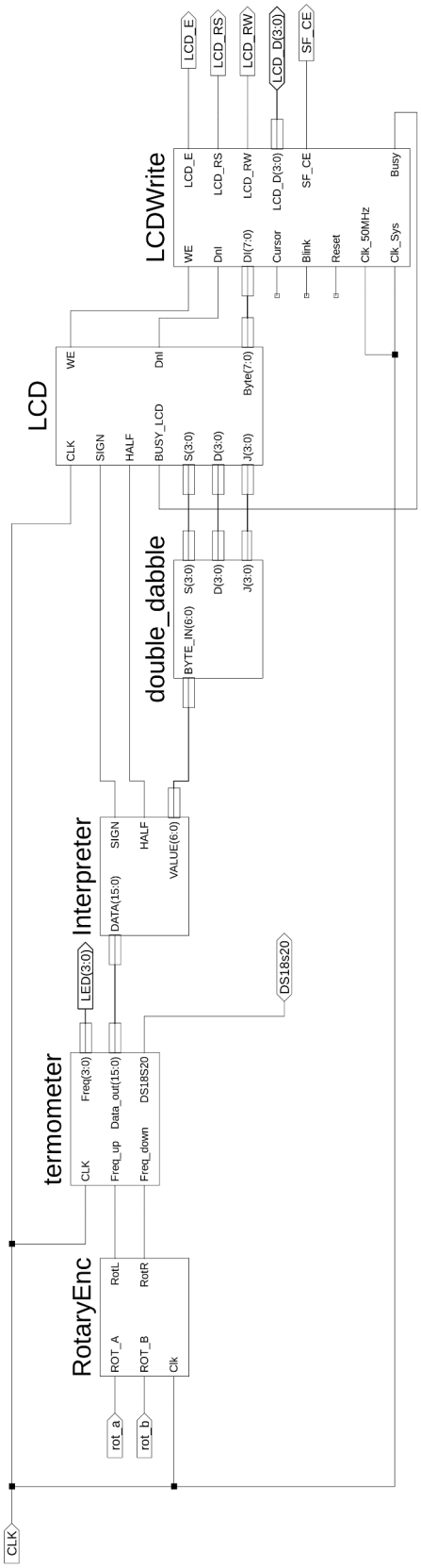
Algorytm polega na wykonaniu n iteracji (w zależności od długości ciągu bitów). Początkowo wynikowy kod BCD jest zainicjalizowany jako ciąg zer podzielonych na grupy po 4 bity. Podczas każdej iteracji wykonywane jest przesunięcie o jeden bit w lewo, a na koniec doklejany jest jeden bit ciągu wejściowego. 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 Schemat główny

Funkcjonalność projektu opiera się na 6 podstawowych modułach widocznych na rysunku 4:

- 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.

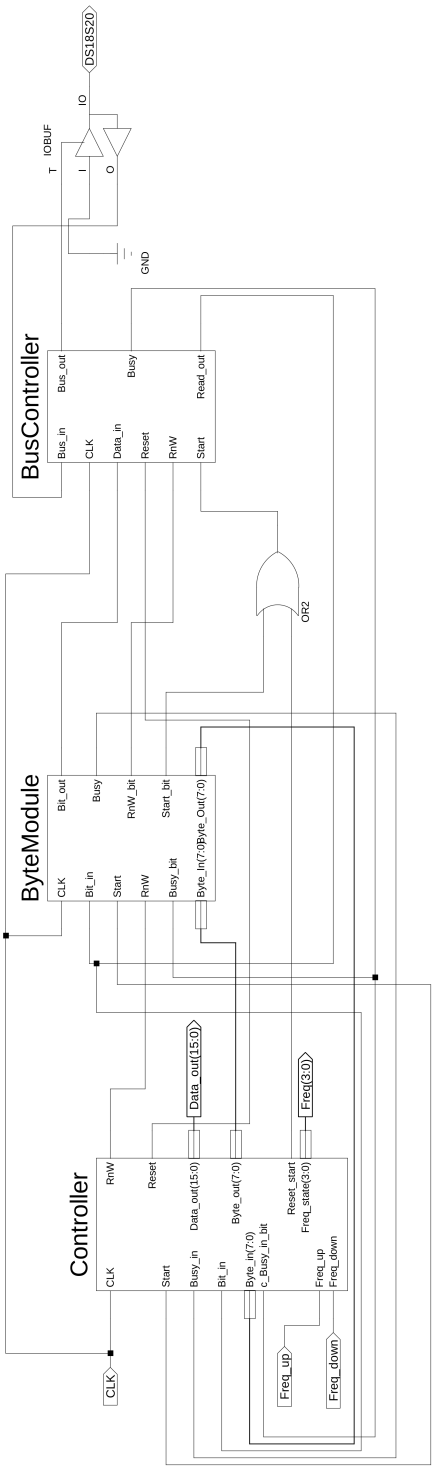


Rysunek 4: Schemat ogólny projektu

3.2 Termometer

Moduł Termometer jest najbardziej rozbudowanym modułem składającym się z 4 układów (rysunek 5):

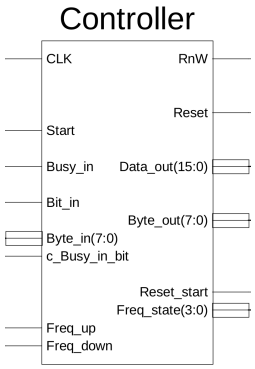
- Controller - obsługa sekwencji komunikacji i konwersji temperatury;
- ByteModule - obsługa komunikacji na poziomie bajtu (instrukcji);
- BusController - obsługa magistrali Onewire;
- IOBuf - ustala kierunek komunikacji.



Rysunek 5: Moduł Termometer

3.2.1 Controller

Moduł Controller (rysunek 6) odpowiada za przeprowadzenie poprawnej sekwencji operacji wymaganych do zmierzenia i odczytu temperatury.



Rysunek 6: 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 - Flaga zajętości BusController;

Wyjścia:

- Start - Sygnał start dla ByteModule;
- Data_out - Odebrany ciag reprezentujący temperaturę (2 bajty);
- Byte_out - Dane (kod instrukcji) dla ByteContorllera;
- Reser - Sygnał reset;
- Freq_state - Wektor reprezentujący aktualną częstotliwość próbkowania;
- Reset_start - Sygnał start dla BusControllera.

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

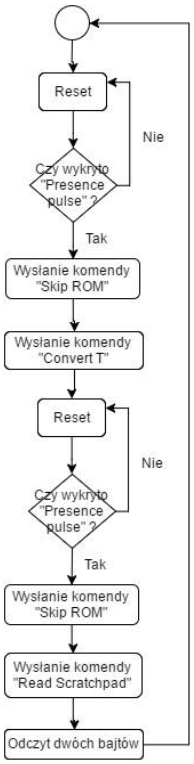
```
freq_process: process (CLK, freq_up, freq_down)
begin
        if rising_edge(clk) then
                if freq_up = '1' then
                        frequence <= frequence(2 downto 0) & frequence (3);
                elsif freq_down = '1' then
                        frequence <= frequence (0) & frequence(3 downto 1);
```

```
                end if;
        end if;
end process;
```

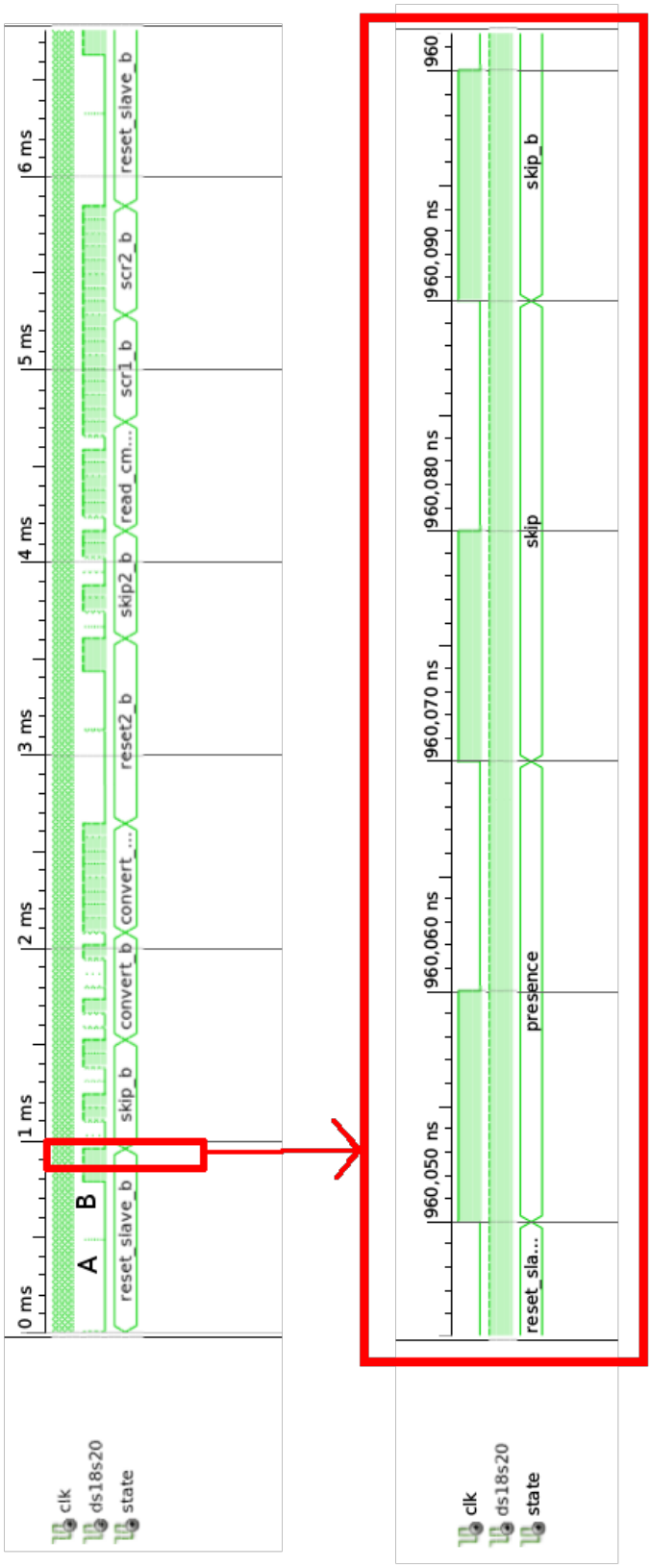
Moduł jest automatem skończonym (fsm). Zmiana stanów odbywa się w procesie state_service.

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

Kolejne stany maszyny pokazuje Rysunek 7. 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 wykonwywane są kolejne komendy są jednotaktowe, podczas gdy oczekiwanie na ich wykonanie zajmuje stosunkowo dużo więcej czasu. Zauważyć to można na symulacji pokazanej na rysunku 8



Rysunek 7: Maszyna stanów Controller

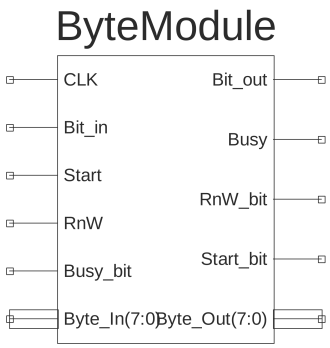


Rysunek 8: Symulacja automatu Controller

Zaznaczona na symulacji pozyjca 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.

3.2.2 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 9: Moduł ByteModule

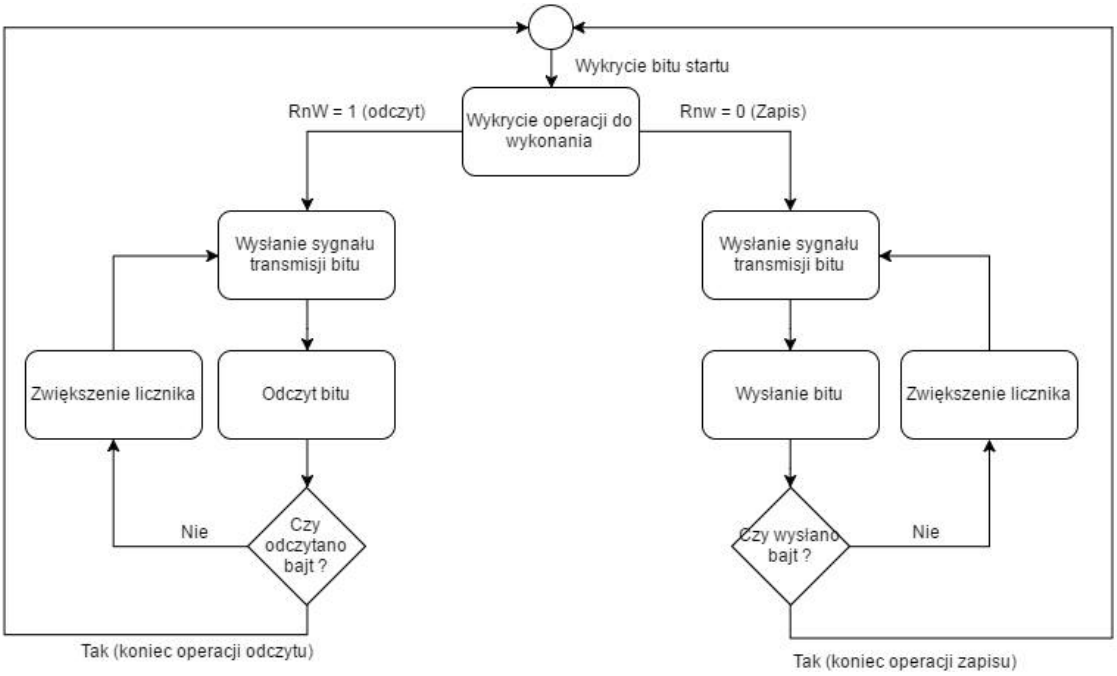
Opis wyprowadzeń:
Wejścia:

- Bit_in - Bit wchodzący podczas operacji odczytu;
- Start - Jednobitowy 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 równiez jest maszyną stanów. Jej uproszczony schemat przedstawia rysunek 10. Szczegóły implementacji można zobaczyć na symulacji przedstawionej na rysunkach 11 - **??**,



Rysunek 10: Maszyna stanów BusController

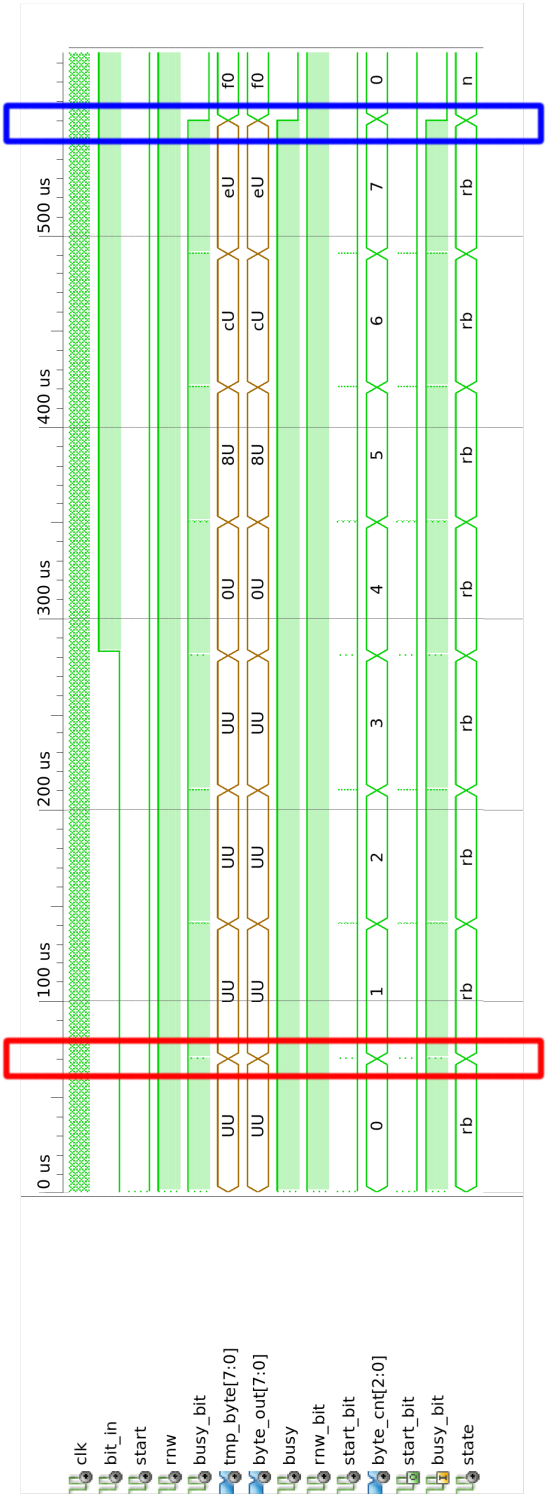
Cykl odczytu rozpoczyna stan rs w którym ustawiany jest bit RnW na wartość 1 oraz wysyłany jest jendotaktowy sygnał start_bit. Następnie w stanie rb następuje oczekiwanie na sygnalizujację zakończenia operacji przez BusController (flaga Busy_bit). W stanie re w rejestrze tmp_byte zatrzaskiwana jest odczytana wartość kolejnego bitu. Wartość ta jest współbieżnie przepsiwana 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:

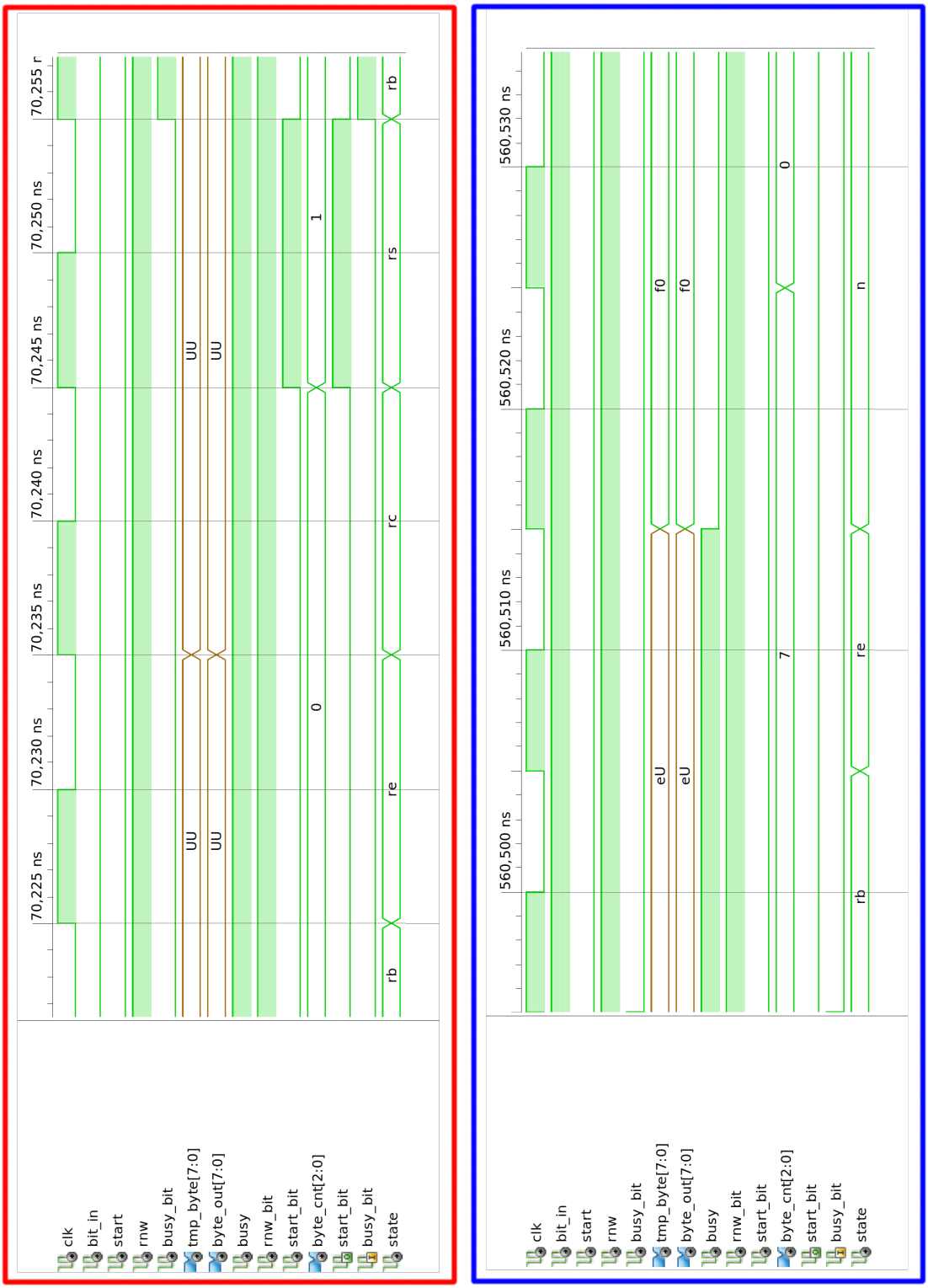
```
byte_reading: process (CLK, state)
begin
if rising_edge(CLK) then
        if state = re then
                tmp_byte <= Bit_in & tmp_byte(7 downto 1);
        end if;
end if;
end process;
```

Symulacja z rysunku 11 pokazuje przebieg całej procedury. Zaznaczone czerowną i niebieską ramką fragmenty można zobaczyć w powiększeniu na rysunku 13. Widać na nich odpowiednio przejścia między odczytem kolejnych bitów oraz odczytem ostatniego bitu.

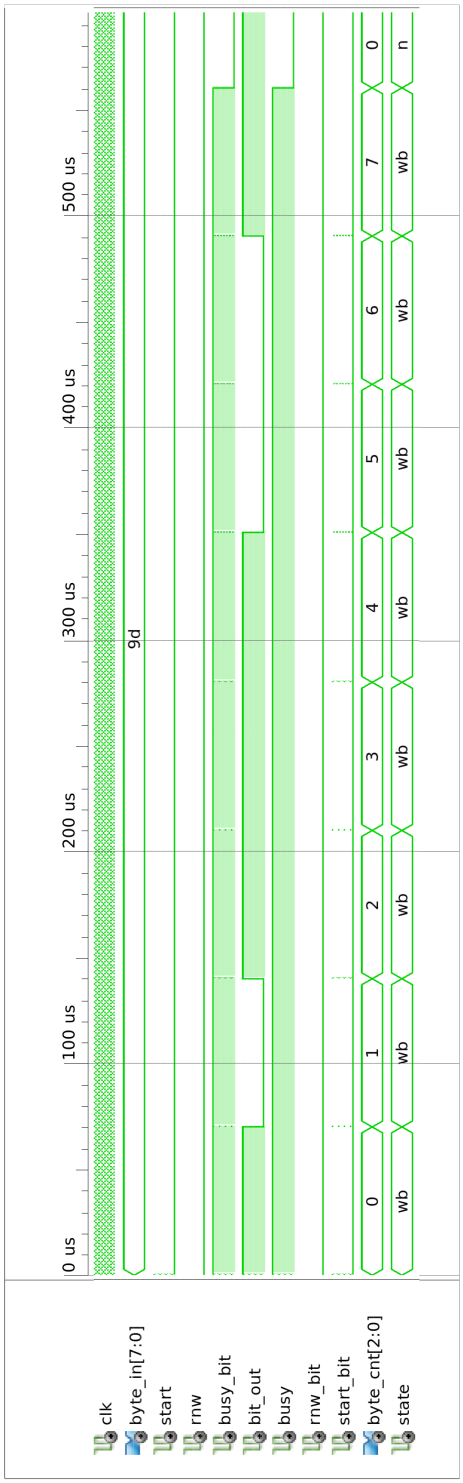
Procedura zapisu bajtu jest analogiczna do procedury odczytu. Wymienionym wcześniej stanom odpowiadają kolejno ws (RnW = 0), wb, we, wc. Wartości kolejnych bitów są odczytywane wprost z wektora byte_in w zależności od wartości byte_cnt. Symulację przedstawia rysunek **??**



Rysunek 11: Symulacja zapisu bajtu



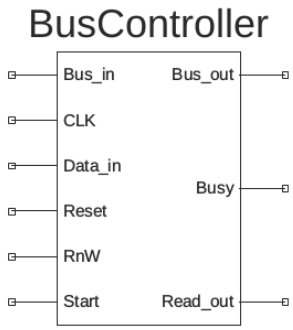
Rysunek 12: Symulacja zapisu bajtu - szczegóły



Rysunek 13: Symulacja odczytu bajtu

3.2.3 BusController

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



Rysunek 14: Moduł BusController

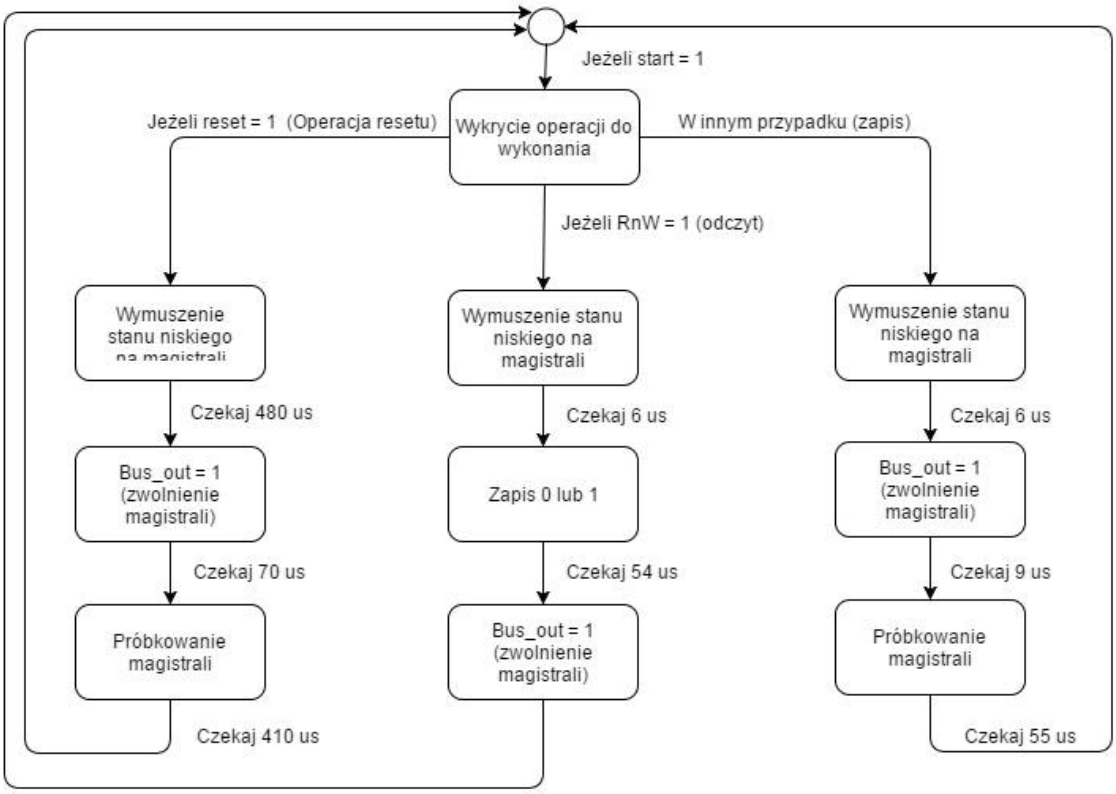
Opis wyprowadzeń: Wejścia:

- 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 - Sygnał start dla BusController;

Opis Wyjść:

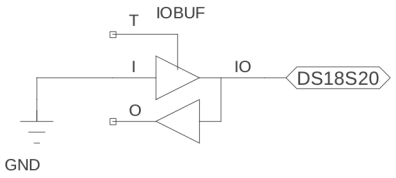
- Bus_out - Zapis bitu do termometru;
- Busy - Bit zajętości BusController;
- Read_out - Odczytany bit z termometru;

Poniżej przedstawiono schemat blokowy tego automatu (rysunek 15):

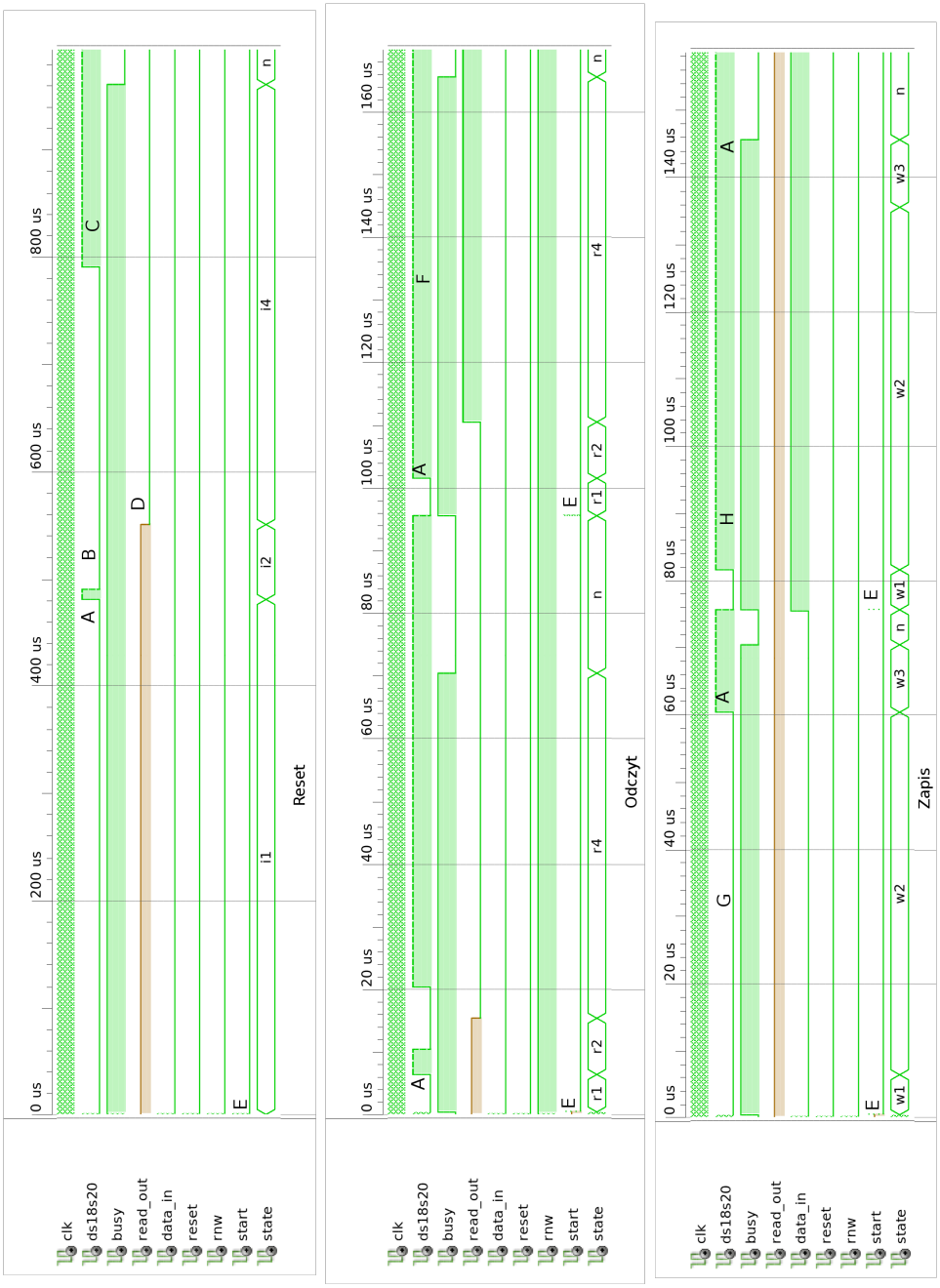


Rysunek 15: Maszyna stanów BusController

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 wejście T otwiera bufor wyjściowy i przekazuje sygnał podawany na pin I (GND - Logiczne 0). Logiczne 1 na wejściu T ustawia linie w stanie wysokiej impednacji (zwalnia magistralę). Pin O służy do odczytu stanu magistrali.



Rysunek 16: Symbol IOBuf



Rysunek 17: Symulacja odczytu bajtu

Na rysunku 17 przedstawiono charakterystykę pracy układu wraz z buforem IO. Zaznaczono charakterystyne punkty:

A Zwolnienie magistrali przez master;

B Sygnał obecności slave;

C Zwolnienie magistrali przez slave;

D Sygnał obecności slave na wyjściu modułu;

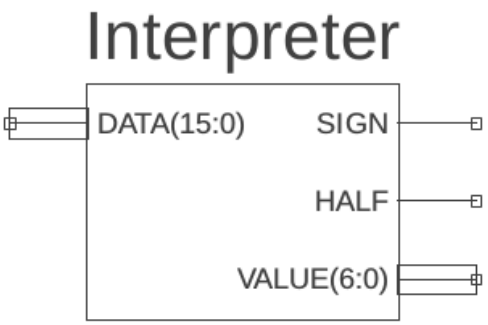
E Impuls start;

F Magistrala w stanie wysokim - odczyt 1;

G Master wymusza stan niski - zapis 0;

H Master zwalnia magistralę przy zapisie - zapis 1;

3.3 Interpreter



Rysunek 18: 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:

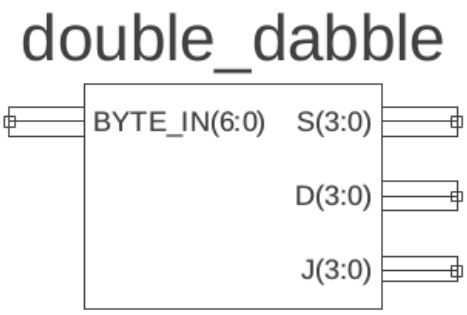
Wejścia:

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

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;

3.4 Double dabble



Rysunek 19: 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 wyprowadzeń:

Wejścia:

- BYTE_IN(6:0) - wektor wejściowy konwertowany na BCD;

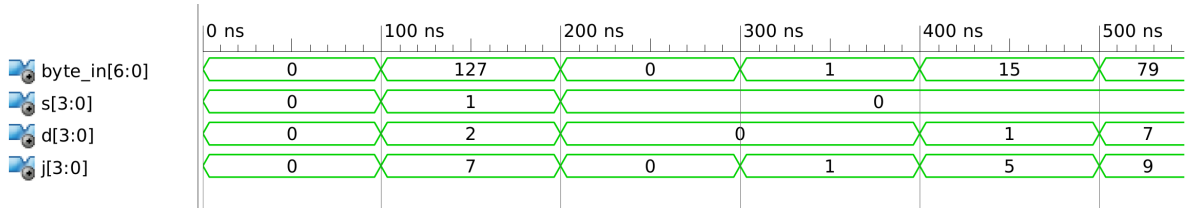
Wyjścia:

- 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;
```

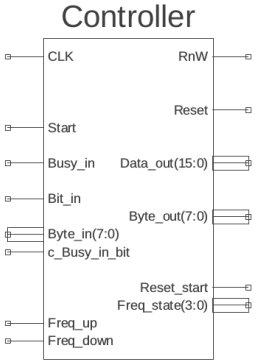
Przykład działania pokazje symulacja na rysunku 20:



Rysunek 20: Symulacja modułu double dabble

3.5 LCD

Moduł LCD opiera się na prostej maszynie stanów, która przesyła kolejne znaki ASCII do wyświetlacza kontrolowanego przez element LCDWrite.



Rysunek 21: Moduł LCD