POLITECHNIKA WROCŁAWSKA

Wydział Elektroniki

Układy cyfrowe i systemy wbudowane 2

Projekt

Opracowano:	28 maja 2020
Grupa:	czw. 11:10 TP
Autorzy:	Wojciech Śliwa 241296
	Paweł Sajewicz 241314
Prowadzący:	dr inż. Jarosław Sugier

Spis treści

1. Wstęp

1.1. Cel i zakres projektu

Zaprojektowanie układu cyfrowego na płytę Spartan3E odczytującego pomiary z akcelerometru i wyświetlającego je na wyświetlaczu LCD.

1.2. Opis sprzętu

Rodzina programowalnych macierzy bramek Spartan-3E (FPGA)[1] została specjalnie zaprojektowana w celu zaspokojenia potrzeb dużych, wrażliwych na koszty aplikacji elektronicznych dla konsumentów. W porównaniu do poprzedniej rodziny (Spartan-3) Spartan-3E cechuje się większą ilością logiki na I/O, znacznie zmniejszając koszt na komórkę logiczną. Nowe funkcje poprawiają wydajność systemu i zmniejszają koszty konfiguracji. Te ulepszenia Spartan-3E FPGA, w połączeniu z zaawansowaną technologią 90 nm, zapewniają większą funkcjonalność i przepustowość.

Płyta Spartan-3E[2] pozwala na obsługę układów FPGA. Jest wyposażona w układ programowalny CPLD firmy Xilinx oraz moduł XC3S500E z wyprowadzeniami I/O, złącza portu JTAG, generator kwarcowy sygnału zegarowego, diody LED, klawisze, wyświetlacz LCD i wiele innych elementów.

ADXL345[3] to niewielki akcelerometr, czyli czujnik do pomiaru przyspieszeń w trzech osiach, z wysokiej rozdzielczości (13-bitów) pomiarem w zakresie ± 16 g. Cyfrowe dane wyjściowe są dostępne poprzez interfejs cyfrowy SPI (3- lub 4-przewodowy) lub I2C. Urządzenie mierzy przyspieszenie statyczne grawitacji, a także dynamiczne przyspieszenie wynikające z ruchu lub uderzenia. Jego wysoka rozdzielczość (3,9 mg / LSB) umożliwia pomiar zmian nachylenia mniejszych niż 1,0 °.

1.3. Podstawowe informacje

Projekt wykorzystuje szeregową, multi-master-multi-slave magistralę I^2C[4], do przesyłu danych z i do akcelerometru.

Z CS połączonym wysoko do $V_{DD\ I/O}$, ADXL345 znajduje się w trybie I^2C, wymagającym prostego 2-przewodowego podłączenia. Przy spełnieniu odpowiednich parametrów obsługiwane są tryby przesyłania danych: standardowy (100 kHz) i szybki (400 kHz). Obsługiwane są jedno- lub wielo-bajtowe operacje odczytu i zapisu danych. Przy wysokim pinie ALT ADDRESS 7-bitowy adres I^2C dla urządzenia to 0x1D, poprzedzający bit R/W. Przekłada się to na 0x3A dla zapisu i 0x3B dla odczytu. Alternatywny adres I^2C, 0x53 (poprzedzający bit R/W) można wybrać poprzez uziemienie styku ALT ADDRESS (Pin 12), co przekłada się na 0xA6 dla zapisu i 0xA7 dla odczytu.

Przez brak wewnętrznych rezystorów dla nieużywanych styków, nie ma domyślnego stanu dla styku CS lub ALT ADDRESS, jeśli pozostaną swobodne lub niepodłączone. Dlatego też podczas korzystania z I^2C wymagane jest, aby pin CS był podłączony do $V_{DD\ I/O}$, a pin ALT ADDRESS do $V_{DD\ I/O}$ lub GND.

Kluczowe dla działania akcelerometru są rejestry danych, przedstawione w tabeli 1[5]. W projekcie korzystamy z kilku z nich: ID urządzenia – adres 0x00; kontrola funkcji oszczędzania energii – adres 0x2D; kontrola włączania przerwań – adres 0x2E; kontrola szybkości transmisji i trybu zasilania, odpowiadająca za prędkość przesyłania kolejnych pomiarów – adres 0x2C i adres, pod którym zaczynają się rejestry do zapisywania wartości pomiarów – 0x32.

Address		Name	Type	Reset Value	Description
Hex	Dec				
0x00	0	DEVID	R	11100101	Device ID
0x01 to 0x1C	1 to 28	Reserved			Reserved; do not access
0x1D	29	THRESH_TAP	R/$\overline{W}$	00000000	Tap threshold
0x1E	30	OFSX	R/$\overline{W}$	00000000	X-axis offset
0x1F	31	OFSY	R/$\overline{W}$	00000000	Y-axis offset
0x20	32	OFSZ	R/$\overline{W}$	00000000	Z-axis offset
0x21	33	DUR	R/$\overline{W}$	00000000	Tap duration
0x22	34	Latent	R/$\overline{W}$	00000000	Tap latency
0x23	35	Window	R/$\overline{W}$	00000000	Tap window
0x24	36	THRESH_ACT	R/$\overline{W}$	00000000	Activity threshold
0x25	37	THRESH_INACT	R/$\overline{W}$	00000000	Inactivity threshold
0x26	38	TIME_INACT	R/$\overline{W}$	00000000	Inactivity time
0x27	39	ACT_INACT_CTL	R/$\overline{W}$	00000000	Axis enable control for activity and inactivity detection
0x28	40	THRESH_FF	R/$\overline{W}$	00000000	Free-fall threshold
0x29	41	TIME_FF	R/$\overline{W}$	00000000	Free-fall time
0x2A	42	TAP_AXES	R/$\overline{W}$	00000000	Axis control for single tap/double tap
0x2B	43	ACT_TAP_STATUS	R	00000000	Source of single tap/double tap
0x2C	44	BW_RATE	R/$\overline{W}$	00001010	Data rate and power mode control
0x2D	45	POWER_CTL	R/$\overline{W}$	00000000	Power-saving features control
0x2E	46	INT_ENABLE	R/$\overline{W}$	00000000	Interrupt enable control
0x2F	47	INT_MAP	R/$\overline{W}$	00000000	Interrupt mapping control
0x30	48	INT_SOURCE	R	00000010	Source of interrupts
0x31	49	DATA_FORMAT	R/$\overline{W}$	00000000	Data format control
0x32	50	DATAX0	R	00000000	X-Axis Data 0
0x33	51	DATAX1	R	00000000	X-Axis Data 1
0x34	52	DATAY0	R	00000000	Y-Axis Data 0
0x35	53	DATAY1	R	00000000	Y-Axis Data 1
0x36	54	DATAZ0	R	00000000	Z-Axis Data 0
0x37	55	DATAZ1	R	00000000	Z-Axis Data 1
0x38	56	FIFO_CTL	R/$\overline{W}$	00000000	FIFO control
0x39	57	FIFO_STATUS	R	00000000	FIFO status

Tabela 1. Mapa rejestrów akcelerometru.

2. Przedstawienie układu

2.1. Struktura projektu

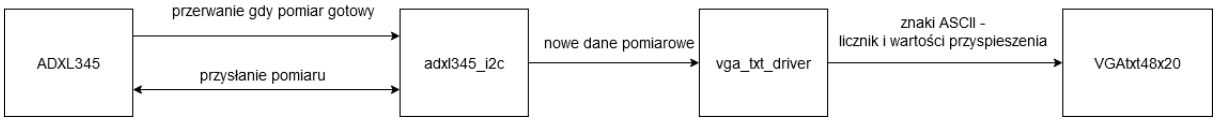
Definicja głównego układu zawarta jest w pliku *adxl345_vga.sch.* Układa składa się z trzech modułów:

- *adxl345_i2c* – (plik *adxl345_i2c.sch*) odpowiada za komunikację za pośrednictwem protokołu I^2C z akcelerometrem ADXL345.
- *vga_txt_driver* – (plik *vga_txt_driver.sch*) steruje modułem VGAtxt48x20.
- *VGAtxt48x20* – (moduł zewnętrzny) sterownik wyświetlacza VGA w trybie tekstowym.

Układ w obecnej formie pobiera z ustaloną częstotliwością informacje o przyśpieszeniu we wszystkich osiach z akcelerometru. Następnie dane w module *vga_txt_driver* konwertowane są na liczby heksadecymalne kodowane w ASCII. Kolejne znaki przekazywane są do modułu *VGAtxt48x20*.

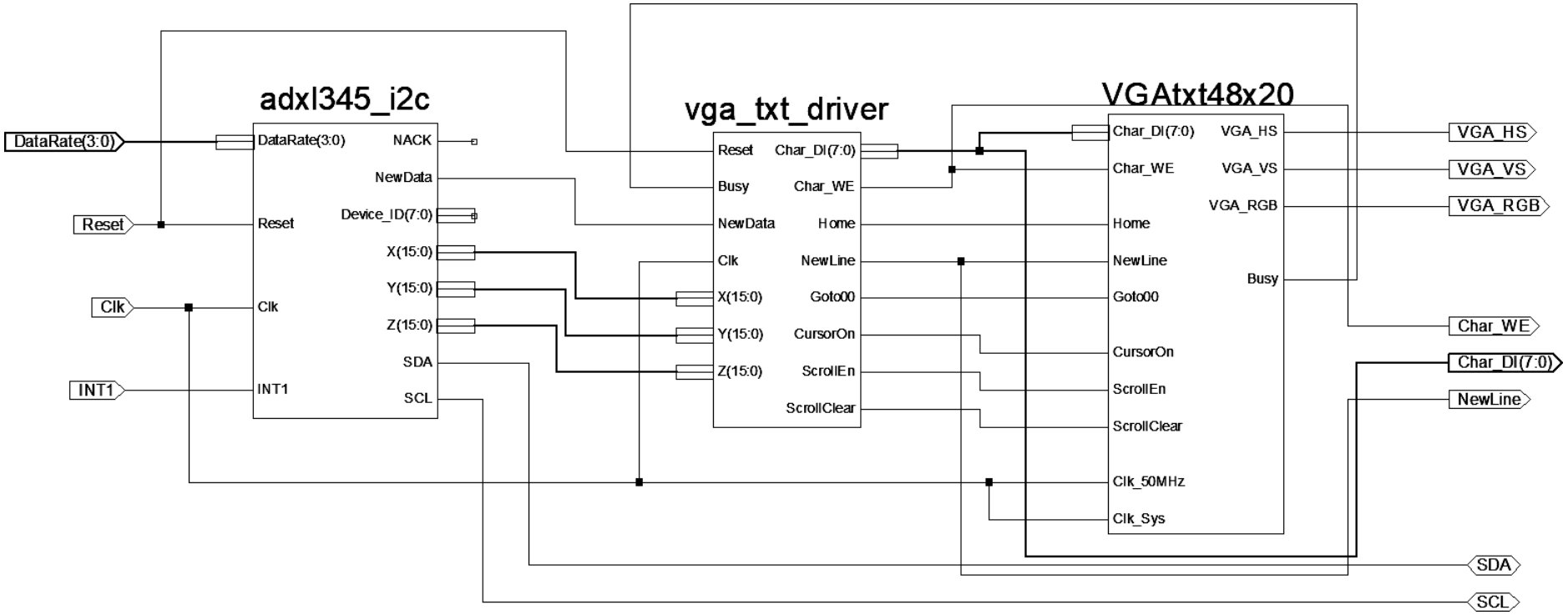
Każdy następny pomiar wyświetlany jest w nowej linii. Dodatkowo są one numerowane.

Format zapisu jest następujący: *NNN:XXXX YYYY ZZZZ.* Gdzie N to cyfra numeru, A X, Y i Z to kolejne cyfry pomiaru w danej osi.



Rysunek 1. Diagram przepływu danych

W projekcie zawarty jest również działający moduł sterowania wyświetlaczem VGA w trybie graficznym, w rozdzielczości 800 na 600. Jednak nie został on nigdzie wykorzystany.



Rysunek 2. Schemat szczytow

2.2. Opis modułów

2.2.1. Moduł główny – adxl345_vga

Funkcja

Łączy w jeden układ moduły składowe.

Lista wejść/wyjść

Nazwa	**Rodzaj**	**Typ**
Clk	wejście	std_logic
Reset	wejście	std_logic
DataRate	wejście	std_logic_vector(3:0)
SDA	dwukierunkowe	-
SCL	dwukierunkowe	-
VGA_HS	wyjście	-
VGA_VS	wyjście	-
VGA_RGB	wyjście	-

Tabela 2. List wyprowadzeń adxl345_vga

Symulacja

Moduł zawiera złożony plik testowy, za pośrednictwem którego można ustawić pożądaną wartość parametru *DataRate*.

Częstotliwość pomiarów [Hz]	**Wartość DataRate**
3200	1111
1600	1110
800	1101
400	1100
200	1011
100	1010
50	1001
25	1000
12,5	0111
6,25	0110

Tabela 3. Możliwe częstotliwości pomiarów

Odpowiada on za częstotliwość wykonywania pomiarów. Plik testowy zawiera proces symulujący działanie układu I^2C slave, który melduje wykonywane operacje w terminalu tekstowym. W konsoli wyświetlany jest również ciąg znaków, który powinien pojawić się na ekranie monitora.

Poniżej znajduje się przykładowy zapis z symulacji. Slave informuje o otrzymaniu adresu 0x3B. Jest to adres rejestrów z wartościami pomiaru przyspieszenia. Następnie do urządzenia master przesłane zostaje 6 bajtów danych. W linii 9 widać ciąg znaków, który w fizycznym układzie byłby wypisany na wyświetlaczu.

```
[I2C 10051.3701 us] START condition
[I2C 10071.9901 us] address byte: 3B
[I2C 10097.6251 us] byte transmitted: EA with positive ACK
[I2C 10120.1251 us] byte transmitted: EB with positive ACK
[I2C 10142.6251 us] byte transmitted: EC with positive ACK
[I2C 10165.1251 us] byte transmitted: ED with positive ACK
[I2C 10187.6251 us] byte transmitted: EE with positive ACK
[I2C 10210.1251 us] byte transmitted: EF, NACK in response
001:EBEA EDEC EFEE
```

2.2.2. Moduł adxl345_driver

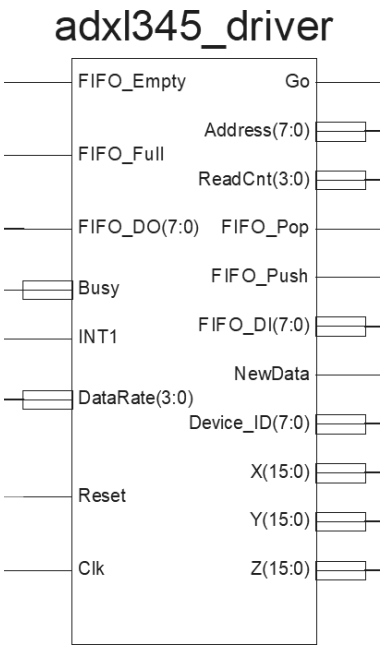
Funkcja

Moduł jest sterownikiem akcelerometru *ADXL345*. Został stworzony do współpracy z *I2C_Master*. Dlatego też ich wyprowadzenia w większości się pokrywają.
Układ po uruchomieniu pobiera z urządzenia typu slave numer identyfikacyjny, a następnie przeprowadza jego konfiguracje poprzez:

- ustawienie częstotliwości pomiarów,
- włączeniu trybu pracy ciągłej,
- włączenie przerwań na wejściu INT1, w momencie wykonania pomiaru.

Po skonfigurowaniu moduł oczekuje na przerwanie i gdy ono nastąpi pobiera wynik pomiaru i wystawia go na wyjściach X, Y oraz Z. Dodatkowo NewData zostanie na jeden impuls zegarowy ustawione w stan wysoki.

Symbol



Rysunek 3. Symbol modułu adxl345_driver

Lista wejść/wyjść

Nazwa	**Rodzaj**	**Typ**	**Komentarz**
Clk	wejście	std_logic	-
Reset	wejście	std_logic	-
DataRate	wejście	std_logic_vector(3:0)	częstotliwość pomiarów
INT1	wejście	std_logic	przerwanie z ADXL345, impuls występuje , gdy pojawi się nowy pomiar
Busy	wejście	std_logic	wejście jest w stanie wysokim, gdy trwa transmisja (I2C_Master)
FIFO_DO	wejście	std_logic_vector(7:0)	bajt z początku kolejki (I2C_Master)
FIFO_Full	wejście	std_logic	flaga statusu kolejki (I2C_Master)
FIFO_Empty	wejście	std_logic	flaga statusu kolejki (I2C_Master)
Go	wyjście	std_logic	impuls startu transmisji
Address	wyjście	std_logic_vector(7:0)	adres odczytu lub zapisu slave
ReadCnt	wyjście	std_logic_vector(3:0)	liczba bajtów, które mają być odczytane
FIFO_Pop	wyjście	std_logic	sygnał pobrania bajtu z kolejki
FIFO_Push	wyjście	std_logic	sygnał wysłania bajtu do kolejki
FIFO_DI	wyjście	std_logic_vector(7:0)	bajt, który ma zostać dodany do kolejki
NewData	wyjście	std_logic	stan wysoki oznacza, że na wyjściach X, Y, Z są nowe dane pomiarowe
Device_ID	wyjście	std_logic_vector(7:0)	Identyfikator urządzenia slave
X	wyjście	std_logic_vector(15:0)	pomiar przyspieszenia w osi X
Y	wyjście	std_logic_vector(15:0)	pomiar przyspieszenia w osi Y
Z	wyjście	std_logic_vector(15:0)	pomiar przyspieszenia w osi Z

Tabela 4. Lista wyprowadzeń modułu adxl345_driver

Organizacja modułu

Maszyna stanów modułu jest zobrazowana na poniższym rysunku. Napisy występujące przy niektórych krawędziach, informują o warunku wymaganym do zmiany stanu.

Rysunek 4. Graf maszyny stanów modułu adxl345_driver

Moduł zawiera również trzy procesy, aktywowane gdy wystąpią konkretne stany. Są to:

- *store_device_id* – zapisuje identyfikator urządzenia do bufora.

```
1. store_device_id : process(Clk, state, next_state)
2. begin
3.     if rising_edge(Clk) then
4.         if state = read_id then
5.             device_id_register <= FIFO_DO;
6.         end if;
7.     end if;
8. end process store_device_id;
```

- *store_data* – zapisuje kolejne (według licznika *byte_count*) bajty danych pomiarowych do buforów.

```
1. store_data : process(Clk, state, next_state)
2. begin
3.     if rising_edge(Clk) then
4.         if state = read_data then
5.             case byte_count is
6.                 when 0 =>
7.                     data_x_register(7 downto 0) <= FIFO_DO;
8.                 when 1 =>
9.                     data_x_register(15 downto 8) <= FIFO_DO;
10.                when 2 =>
11.                    data_y_register(7 downto 0) <= FIFO_DO;
12.                when 3 =>
13.                    data_y_register(15 downto 8) <= FIFO_DO;
14.                when 4 =>
15.                    data_z_register(7 downto 0) <= FIFO_DO;
16.                when 5 =>
17.                    data_z_register(15 downto 8) <= FIFO_DO;
18.            end case;
19.        end if;
20.    end if;
21. end process store_data;
```

- *count_bytes* – inkrementuje wspomniany w poprzednim punkcie licznik bajtów. Proces odpowiada również za zerowanie licznika w momencie jego przepełnienia lub gdy wystąpi sygnał *Reset*.

```
1. count_bytes : process(Clk)
2. begin
3.     if rising_edge(Clk) then
4.         if Reset = '1' then
5.             byte_count <= 0;
6.         end if;
7.         if state = pop_data then
8.             if byte_count = 5 then
9.                 byte_count <= 0;
10.            else
11.                byte_count <= byte_count + 1;
12.            end if;
13.        end if;
14.    end if;
15. end process count_bytes;
```

Symulacja

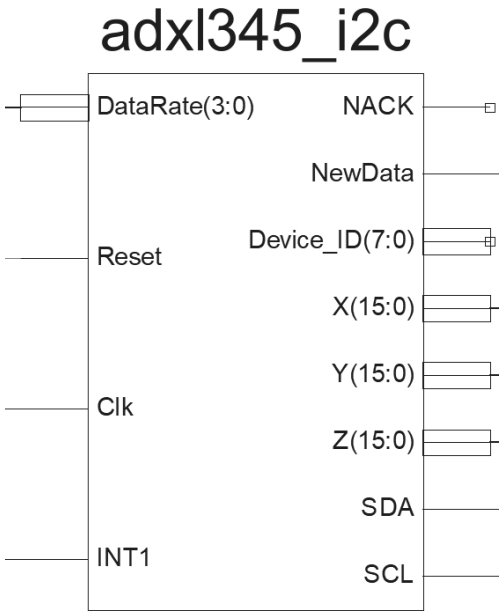
Symulacja działania układu znajduje się w punkcie dotyczącym modułu *adxl345_i2c*. Wynika to z faktu, że *adxl345_driver* może być właściwie przetestowany jedynie w połączeniu z *I2C_Master*.

2.2.3. Moduł adxl345_i2c

Funkcja

Moduł łączy ze sobą sterownik *adxl345_driver* z *I2C_Master*.

Symbol



Rysunek 5. Symbol modułu adxl345_i2c

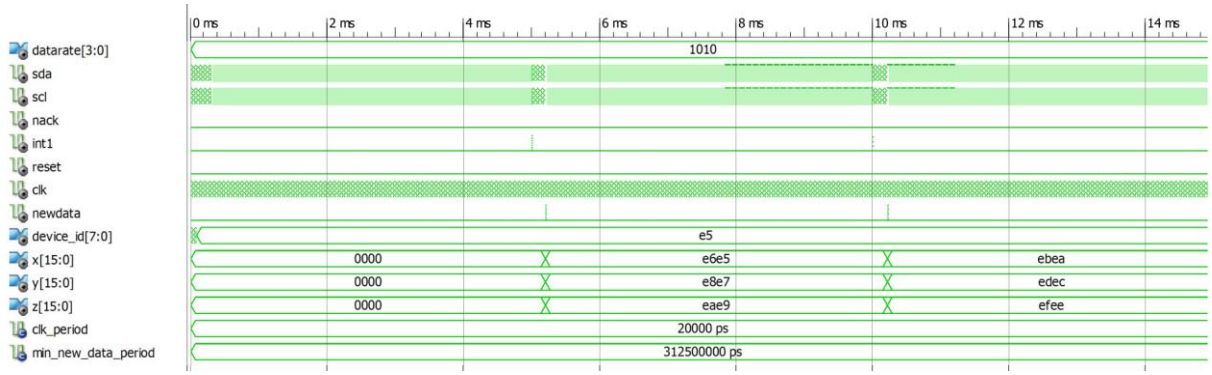
Lista wejść/wyjść

Nazwa	**Rodzaj**	**Typ**	**Komentarz**
Clk	wejście	std_logic	-
Reset	wejście	std_logic	-
DataRate	wejście	std_logic_vector(3:0)	częstotliwość pomiarów
INT1	wejście	std_logic	przerwanie z ADXL345, impuls występuje , gdy pojawi się nowy pomiar
NACK	wyjście	std_logic	sygnał błędu transmisji (I2C_Master)
NewData	wyjście	std_logic	stan wysoki oznacza, że na wyjściach X, Y, Z są nowe dane pomiarowe
Device_ID	wyjście	std_logic_vector(7:0)	Identyfikator urządzenia slave
X	wyjście	std_logic_vector(15:0)	pomiar przyspieszenia w osi X
Y	wyjście	std_logic_vector(15:0)	pomiar przyspieszenia w osi Y
Z	wyjście	std_logic_vector(15:0)	pomiar przyspieszenia w osi Z
SDA	dwukierunkowe	-	(I2C_Master)
SCL	dwukierunkowe	-	(I2C_Master)

Tabela 3. Lista wyprowadzeń modułu adxl345_i2c

Symulacja

Poniższy zrzut ekranu z symulator należy porównać z logami procesu urządzenia slave.



Rysunek 6. Symulacja działania adxl345_i2c

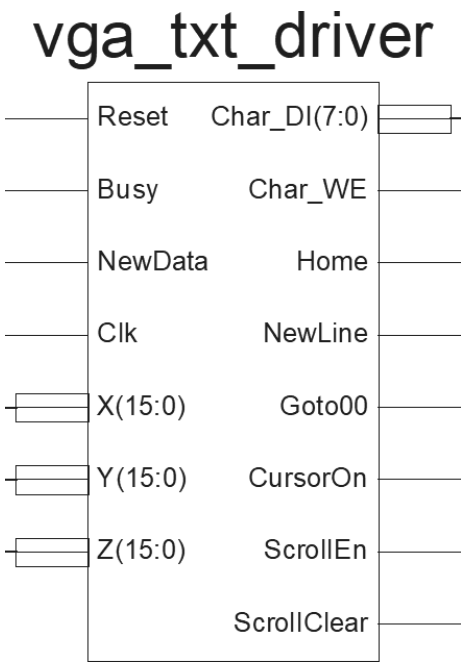
```
1. [I2C 21.9301 us] address byte: 3A       - adres zapisu
2. [I2C 44.4301 us] byte received: 00      - adres rejestru ID
3. [I2C 71.9701 us] address byte: 3B       - adres odczytu
4. [I2C 97.6051 us] byte transmitted: E5, NACK in response - ID
5. [I2C 101.4901 us] START condition
6. [I2C 122.1101 us] address byte: 3A      - adres zapisu
7. [I2C 144.6101 us] byte received: 2D     - adres rejestru POWER_CTL
8. [I2C 167.1101 us] byte received: 08     - nowa wartość rejestru
9. [I2C 172.1101 us] STOP condition
10.  [I2C 174.0701 us] START condition
11.  [I2C 194.6901 us] address byte: 3A    - adres zapisu
12.  [I2C 217.1901 us] byte received: 2C   - adres rejestru BW_RATE
13.  [I2C 239.6901 us] byte received: 0A   - nowa wartość rejestru
14.  [I2C 244.6901 us] STOP condition
15.  [I2C 246.6501 us] START condition
16.  [I2C 267.2701 us] address byte: 3A    - adres zapisu
17.  [I2C 289.7701 us] byte received: 2E   - adres rejestru INT_ENABLE
18.  [I2C 312.2701 us] byte received: 80   - nowa wartość rejestru
19.  [I2C 317.2701 us] STOP condition
20.  [I2C 5001.3101 us] START condition
21.  [I2C 5021.9301 us] address byte: 3A   - adres zapisu
22.  [I2C 5044.4301 us] byte received: 32  - adres rejestru danych pom.
23.  [I2C 5049.4301 us] STOP condition
24.  [I2C 5051.3501 us] START condition
25.  [I2C 5071.9701 us] address byte: 3B   - adres odczytu
26.  [I2C 5097.6051 us] byte transmitted: E5 with positive ACK - X0
27.  [I2C 5120.1051 us] byte transmitted: E6 with positive ACK - X1
28.  [I2C 5142.6051 us] byte transmitted: E7 with positive ACK - Y0
29.  [I2C 5165.1051 us] byte transmitted: E8 with positive ACK - Y1
30.  [I2C 5187.6051 us] byte transmitted: E9 with positive ACK - Z0
31.  [I2C 5210.1051 us] byte transmitted: EA, NACK in response - Z1
32.  [I2C 10001.3301 us] START condition
33.  [I2C 10021.9501 us] address byte: 3A  - adres zapisu
34.  [I2C 10044.4501 us] byte received: 32 - adres rejestru danych pom.
35.  [I2C 10049.4501 us] STOP condition
36.  [I2C 10051.3701 us] START condition
37.  [I2C 10071.9901 us] address byte: 3B  - adres odczytu
38.  [I2C 10097.6251 us] byte transmitted: EA with positive ACK - X0
39.  [I2C 10120.1251 us] byte transmitted: EB with positive ACK - X1
40.  [I2C 10142.6251 us] byte transmitted: EC with positive ACK - Y0
41.  [I2C 10165.1251 us] byte transmitted: ED with positive ACK - Y1
42.  [I2C 10187.6251 us] byte transmitted: EE with positive ACK - Z0
43.  [I2C 10210.1251 us] byte transmitted: EF, NACK in response - Z1
```

2.2.4. Moduł vga_txt_driver

Funkcja

Moduł pobiera dane pomiarowe z *adxl345_i2c*. Posiada wewnętrzny liczniki odebranych pomiarów. Następnie, zarówno licznik jaki i dane, są zamieniane na liczby w systemie szesnastkowym i kodowane w ASCII. Kolejne znaki ciągu są przekazywane do *VGAtxt48x20*. Przesłanie ciągu kończone jest jednotaktowym impulsem NewLine, który powoduje przejście kursora do nowej linii. Jeżeli wszystkie wiersze ekranu zostaną zapisane, kursor przejdzie do pozycji początkowej.

Symbol



Rysunek 7. Symbol modułu vga_txt_driver

Lista wejść/wyjść

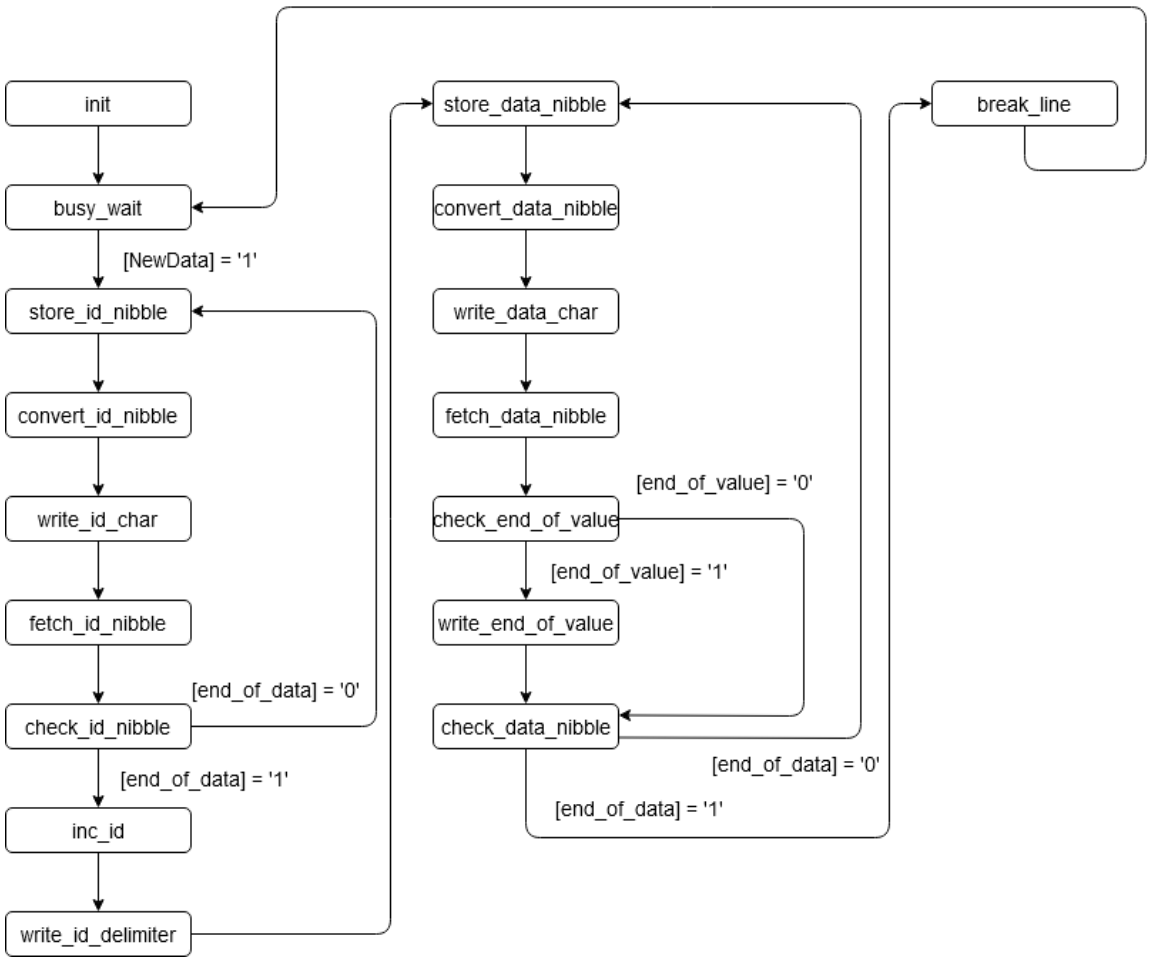
Nazwa	**Rodzaj**	**Typ**	**Komentarz**
Reset	wejście	std_logic	-
Busy	wejście	std_logic	stan wysoki oznacz, że moduł VGAtxt48x20 jest zajęty np. czyszczeniem linii
NewData	wejście	std_logic	stan wysoki oznacza, że na wejściach X, Y, Z są nowe dane pomiarowe
Clk	wejście	std_logic	-
X	wejście	std_logic_vector(15:0)	pomiar przyspieszenia w osi X
Y	wejście	std_logic_vector(15:0)	pomiar przyspieszenia w osi Y
Z	wejście	std_logic_vector(15:0)	pomiar przyspieszenia w osi Z
Char_DI	wyjście	std_logic_ vector(7:0)	znak, który ma zastać wypisany
Char_WE	wyjście	std_logic	impuls na tym wejściu powoduje zapis znaku podanego na Char_DI

Home	wyjście	std_logic	impuls spowoduje przejście do początku wiersza (VGAtxt48x20)
NewLine	wyjście	std_logic	impuls spowoduje przejście do nowej linii (VGAtxt48x20)
Goto00	wyjście	std_logic	impuls spowoduje przejście do początku ekranu (VGAtxt48x20)
CursorOn	wyjście	std_logic	włączenie/wyłączenie kursora – domyślnie w stanie wysokim (VGAtxt48x20)
ScrollEn	wyjście	std_logic	włączenie/wyłączenie przewijania ekranu – domyślnie w stanie niskim (VGAtxt48x20)
ScrollClear	wyjście	std_logic	włączenie/wyłączenie czyszczenia linii po przewinięciu – domyślnie w stanie niskim (VGAtxt48x20)

Tabela 4. Lista wyprowadzeń modułu adxl345_i2c

Organizacja modułu

Maszyna stanów modułu jest zobrazowana na poniższym rysunku. Napisy występujące przy niektórych krawędziach, informują o warunku wymaganym do zmiany stanu.



Rysunek 8. Graf maszyny stanów modułu vga_txt_driver

Moduł zawiera również cztery procesy, aktywowane gdy wystąpią konkretne stany. Są to:

- *store_nibble* – zapisanie półbajtu aktualnie przetwarzanej danej (może to być licznik lub jedna ze współrzędnych) do bufora. Zawartość tego bufora zostanie w innym procesie skonwertowana na znak ASCII.
  Sygnał *measurements_cnt_vector* jest pomocniczym sygnałem, który przechowuje licznik pomiarów skonwertowany z typu integer do std_logic_vector.

```
1. store_nibble : process(Clk, state, next_state)
2. begin
3.     if rising_edge(Clk) then
4.         if state = store_id_nibble then
5.             case char_count is
6.                 when 0 =>
7.                     current_data_nibble <= measurements_cnt_vector(11
    downto 8);
8.                 when 1 =>
9.                     current_data_nibble <= measurements_cnt_vector(7
   downto 4);
10.                 when 2 =>
11.                     current_data_nibble <= measurements_cnt_vector(
   3 downto 0);
12.                 when others =>
13.                     current_data_nibble <= X"0";
14.             end case;
15.         elsif state = store_data_nibble then
16.             case char_count is
17.                 when 0 =>
18.                     current_data_nibble <= X(15 downto 12);
19.                 when 1 =>
20.                     current_data_nibble <= X(11 downto 8);
21.                 when 2 =>
22.                     current_data_nibble <= X(7 downto 4);
23.                 when 3 =>
24.                     current_data_nibble <= X(3 downto 0);
25.                 -- (...) analogicznie dla Y oraz Z
26.                 when others =>
27.                     current_data_nibble <= X"0";
28.             end case;
29.         end if;
30.     end if;
31. end process store_nibble;
```

- *convert_hex_to_char* – proces, który koduje aktualnie przetwarzany półbajt danej w ASCII. Znak wyjściowy otrzymywany jest przez konkatenacje 0x3 z półbajtem, jeżeli jego wartość jest mniejsza od 10. Jest to równoważne z dodanie do niego 0x30, czyli cyfry ‘0’ w ASCII.
  W przypadku, gdy półbajt jest większy, bądź równy 10, należy dodać do niego 0x37. Czyli kod znaku ‘A’ pomniejszony o 10.
  Aktualny znak zostaje zapisany w buforze *current_character*.

```
1. convert_hex_to_char : process(Clk, state, next_state)
2. begin
3.     if rising_edge(Clk) then
4.         if state = convert_id_nibble or state = convert_data_nibble t
   hen
5.             if to_integer(unsigned(current_data_nibble)) < 10 then
6.                 current_character <= X"3" & current_data_nibble;
7.             else
8.                 current_character <= std_logic_vector(X"37" + unsigne
   d(current_data_nibble));
9.             end if;
10.          end if;
11.      end if;
12.  end process convert_hex_to_char;
```

- *count_character* – proces zlicza przetworzone półbajty/znaki. Licznik jest zerowany jeżeli:
    - Osiągnie wartość 2 – w przypadku, gdy zliczane są znaki numeru pomiaru. Resetowi towarzyszy jednotaktowy impuls sygnału *end_of_data*.
    - Osiągnie wartość 11 – gdy zliczane są znaki danych pomiarowych. Resetowi towarzyszy jednotaktowy impuls sygnału *end_of_data*.
    - Wystąpi stan wysoki na wejściu *Reset*.

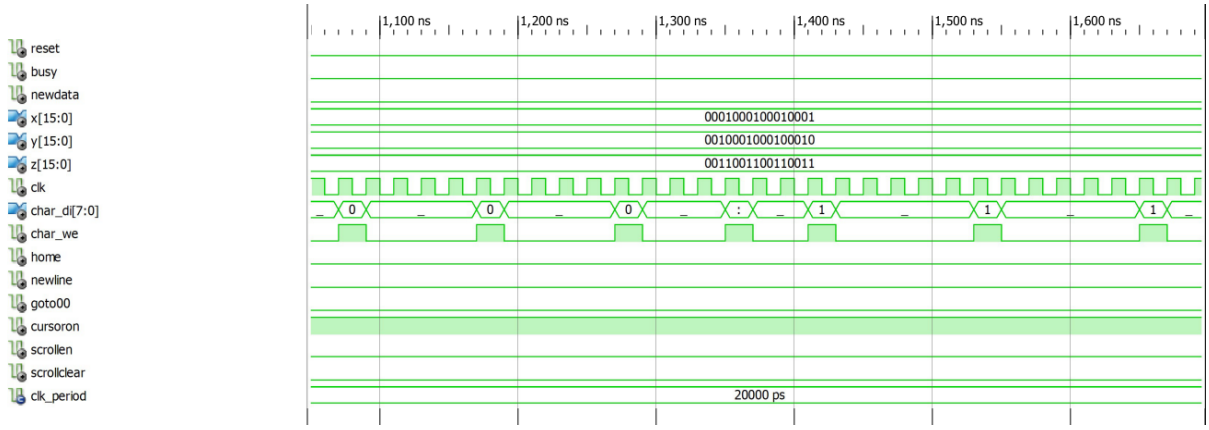
  Dodatkowo proces odpowiada, za wysłanie impulsu *end_of_value*, jeżeli zakończy się wypisywanie wartości przyspieszenia dla jednej współrzędnej. Sygnał ten jest potrzebny do wstawienia odstępów do ciągu znaków.

```
1. count_character : process(Clk, state, next_state)
2. begin
3.     if rising_edge(Clk) then
4.         if Reset = '1' then
5.          char_count <= 0;
6.         end if;
7.         if state = fetch_id_nibble then
8.             if char_count = 2 then
9.                 char_count <= 0;
10.                end_of_data <= '1';
11.            else
12.                char_count <= char_count + 1;
13.                end_of_data <= '0';
14.            end if;
15.         elsif state = fetch_data_nibble then
16.            if char_count = 11 then
17.                char_count <= 0;
18.                end_of_data <= '1';
19.            else
20.                char_count <= char_count + 1;
21.                end_of_data <= '0';
22.            end if;
23.
24.            if char_count = 3 or char_count = 7 or char_count = 11
   then
25.                end_of_value <= '1';
26.            else
27.                end_of_value <= '0';
28.            end if;
29.         end if;
30.     end if;
31. end process count_character;
```

- *increment_measurments_cnt* – standardowy proces licznika, z tym że inkrementacja następuje tylko w stanie *inc_ic.*

Symulacja

Procedura testowa modułu ustawia stałe wartości na wejściach *X*, *Y* oraz *Z* równe odpowiednio 0x1111, 0x2222, 0x3333. Następnie z określoną częstotliwością na wejście *NewData* podawany jest stan wysoki. Efekt działanie symulacji, można obserwować zarówno w postaci przebiegów czasowych jaki i w oknie konsoli symulatora:



Rysunek 9. Symulacja działania vga_txt_driver

W powyższym zrzucie ekranu, można zauważyć kolejne wypisywane znaki. Są to 000 – aktualny stan licznika, ‘:’ – oddziela numer pomiaru od danych oraz 111 – wartości przyspieszenia w osi X.

Gotowy ciąg wyświetli się w terminalu w następujący sposób:

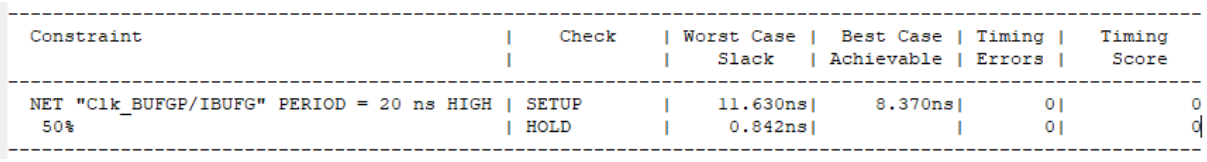
```
1. 000:1111 2222 3333
2. 001:1111 2222 3333
3. 002:1111 2222 3333
4. 003:1111 2222 3333
5. 004:1111 2222 3333
6. 005:1111 2222 3333
```

3. Implementacja

3.1. Raporty

3.1.1. Maksymalna szybkość pracy układu

Ograniczenie	Najgorszy przypadek	Najlepszy przypadek
NET "Clk_BUFGP/IBUFG" PERIOD = 20 ns HIGH 50%	11.630ns	8.370ns



Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
NET "Clk_BUFGP/IBUFG" PERIOD = 20 ns HIGH 50%	SETUP	11.630ns	8.370ns	0	0
	HOLD	0.842ns		0	0

Rysunek 10. Zrzut ekranu raportu zegara bezpośrednio z konsoli ISE

3.1.2. Rozmiar układu

Wykorzystanie elementów logicznych	Użytych	Dostępnych	Wykorzystanie
Number of Slice Flip Flops	270	9,312	2%
Number of 4 input LUTs	431	9,312	4%
Number of occupied Slices	315	4,656	6%
Number of Slices containing only related logic	315	315	100%
Number of Slices containing unrelated logic	0	315	0%
Total Number of 4 input LUTs	498	9,312	5%
Number used as logic	412		
Number used as a route-thru	67		
Number used for Dual Port RAMs	16		
Number used as Shift registers	3		
Number of bonded IOBs	22	232	9%
Number of RAMB16s	2	20	10%
Number of BUFGMUXs	1	24	4%
Average Fanout of Non-Clock Nets	3.41		

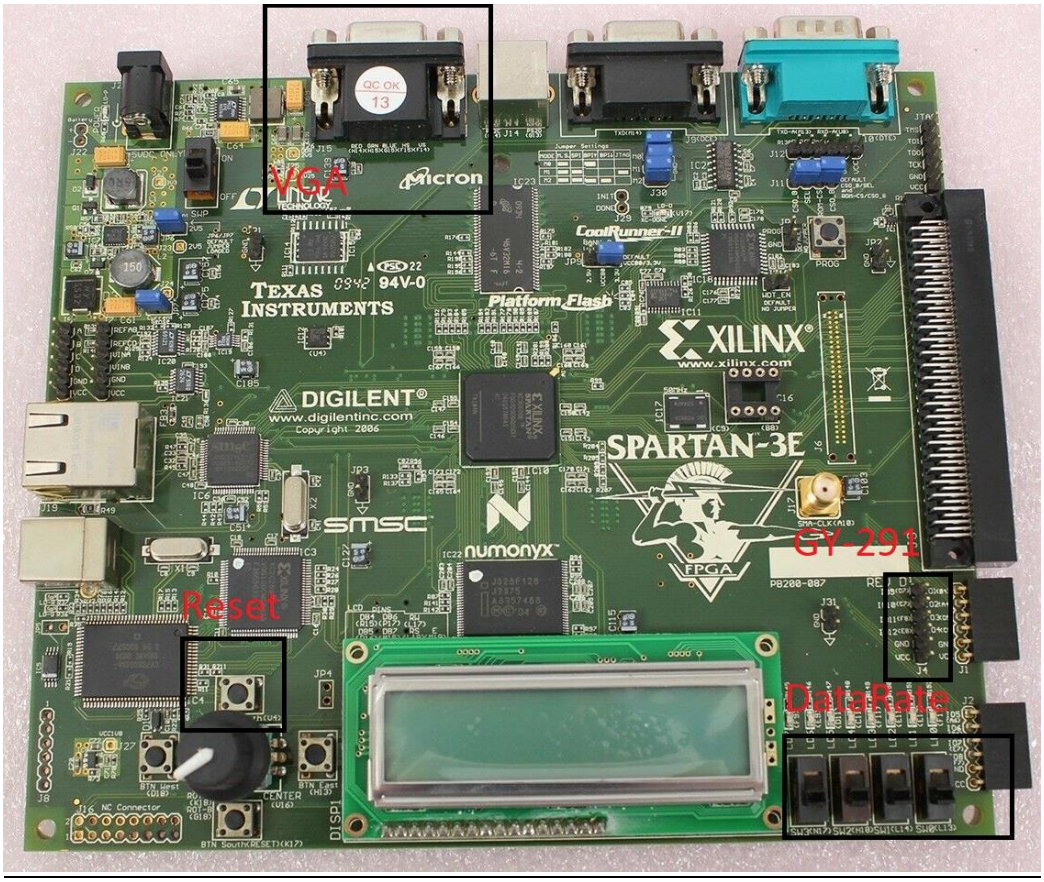
Tabela 5. Podsumowanie rozmiaru układu

3.2. Podręcznik użytkownika urządzenia

Wymagane peryferia

- płyta GY-291 z akcelerometrem ADXL345
- monitor z 15-pinowym złączem D-SUB wspierający standard VGA

Schemat podłączenia i obsługi



Monitor należy podłączyć kablem D-SUB do oznaczonego etykietą VGA gniazda. Płytkę GY-291 należy podłączyć do gniazda J4.

Przycisk znajdujący się nad enkoderem obrotowym, pełni funkcje przycisku reset. Przełączniki bistabilne w prawym dolnym rogu płytki służą do ustawienia częstotliwości wykonywania pomiarów przez akcelerometr.

Częstotliwość pomiarów [Hz]	**SW(3:0)**
3200	1111
1600	1110
800	1101
400	1100
200	1011
100	1010
50	1001
25	1000
12,5	0111
6,25	0110

Tabela 6. Możliwe częstotliwości pomiarów i odpowiadające ustawienia przełączników

Działanie układu

Po zaprogramowaniu, podłączeniu urządzeń peryferyjnych i uruchomieniu układ będzie w zadanych na przełącznikach odstępach czasu wypisywał na ekranie monitora, aktualną wartość przyspieszenia oddziaływującą na akcelerometr.

Format zapisu pomiaru jest następujący:

numer_pomiaru:przyśpieszenie_x przyśpieszenie_y przyśpieszenie_z

Gdzie wszystkie wartości są w systemie szesnastkowym. Licznik jest 3-cyfrowy, a pomiary przyspieszenia dla każdej osi 4-cyfrowe.

4. Podsumowanie

4.1. Ocena krytyczna efektu

Projekt w naszej opinii został wykonany dobrze. Wszystkie planowane funkcjonalności zostały zaimplementowane. Oczywiście organizacja i jakość kodu jest zgodna z naszymi umiejętnościami i stanem wiedzy i na pewno dałoby się napisać lepiej niektóre moduły. Na przykład poprzez zredukowanie ilości instrukcji *case when*.

Wszystkie moduły zostały dokładnie przetestowane w symulatorze i wydają się działać poprawnie.

Synteza układu wykonuje się bez żadnych błędów, jedynie z kilkoma mniej istotnymi ostrzeżeniami.

4.2. Ocena pracy

Umiarkowane problemy sprawiło zaprojektowanie modułu obsługującego akcelerometr. Pierwszy raz korzystaliśmy z protokołu I^2C, więc musieliśmy spędzić trochę czasu na czytaniu dokumentacji, by przyswoić niezbędną wiedzę.

Moduł akcelerometru posiada wiele sygnałów, zarówno wejściowych jak i wyjściowych. Konieczne jest również korzystanie z rejestrów, w których urządzenie zapisuje dane. W efekcie maszyna stanów musiała być skomplikowana i zaprojektowanie jej pochłonęło dużo czasu.

Nie obyło się również bez drobnych błędów. Układ sterujący terminalem do wyświetlania tekstu, modyfikuje wartość otrzymanego pomiaru, aby otrzymać odpowiedni znak ASCII. Początkowo to rozwiązanie nie działało poprawnie i układ zwracał niewłaściwe znaki. Rozwiązaniem okazało się dodanie do test bench'a instrukcji case, która zwraca odpowiedni znak w zależności od wartości parametru.

4.3. Możliwy kierunek rozbudowy układu

Pierwotnie projekt miał być prostą grą, polegającą na kierowaniu samochodem i unikaniu przeszkód. Dokładnie w tym kierunku można rozwinąć układ. Akcelerometr może służyć do sterowania pojazdem. Niestety moduł do wyświetlania tekstu prawdopodobnie nie znajdzie zastosowania i będzie go trzeba zastąpić modułem do wyświetlania grafiki za pomocą VGA. Projekt zawiera układ sterowania wyświetlaczem w trybie graficznym. Moduł ten został przetestowany na fizycznym sprzęcie i mógłby stanowić podstawę do dalszego rozwoju projektu.

5. Literatura

[1] „Spartan-3E FPGA Family Data Sheet”, https://www.xilinx.com/support/documentation/data_sheets/ds312.pdf

[2] „Spartan-3E FPGA Starter Kit Board User Guide”, https://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf

[3] „ADXL345 Data Sheet”, https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf

[4] „ADXL345 Data Sheet”, s. 18, https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf

[5] „ADXL345 Data Sheet”, s. 23, tabela 19, https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf

[6] Moduł „I2C_Master”, dr inż. Jarosław Sugier, http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga/#_Toc479592727

[7] Moduł „VGAtxt48x20”, dr inż. Jarosław Sugier, http://www.zsk.ict.pwr.wroc.pl/zsk_ftp/fpga/#_Toc479592716