POLITECHNIKA WROCŁAWSKA

Wydział Elektroniki

Układy cyfrowe i systemy wbudowane  
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 3](#_Toc41507020)

[1.1. Cel i zakres projektu 3](#_Toc41507021)

[1.2. Opis sprzętu 3](#_Toc41507022)

[1.3. Podstawowe informacje 3](#_Toc41507023)

[2. Przedstawienie układu 5](#_Toc41507024)

[2.1. Struktura projektu 5](#_Toc41507025)

[2.2. Opis modułów 7](#_Toc41507029)

[2.2.1. Moduł główny – adxl345\_vga 7](#_Toc41507030)

[2.2.2. Moduł adxl345\_driver 8](#_Toc41507036)

[2.2.3. Moduł adxl345\_i2c 12](#_Toc41507037)

[2.2.4. Moduł vga\_txt\_driver 14](#_Toc41507038)

[3. Implementacja 19](#_Toc41507039)

[3.1. Raporty 19](#_Toc41507040)

[3.2. Podręcznik użytkownika urządzenia 19](#_Toc41507041)

[4. Podsumowanie 19](#_Toc41507042)

[4.1. Ocena krytyczna efektu 19](#_Toc41507043)

[4.2. Ocena pracy 19](#_Toc41507044)

[4.3. Możliwy kierunek rozbudowy układu 19](#_Toc41507045)

[5. Literatura 20](#_Toc41507046)

# Wstęp

## Cel i zakres projektu

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

## 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 °.

## Podstawowe informacje

Projekt wykorzystuje szeregową, multi-master-multi-slave magistralę I²C[4], do przesyłu danych z i do akcelerometru.

Z CS połączonym wysoko do VDD I/O, ADXL345 znajduje się w trybie I2C, 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 I2C dla urządzenia to 0x1D, poprzedzający bit R/W. Przekłada się to na 0x3A dla zapisu i 0x3B dla odczytu. Alternatywny adres I2C, 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 I2C wymagane jest, aby pin CS był podłączony do VDD I/O, a pin ALT ADDRESS do VDD 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.

Obraz zawierający zrzut ekranu

Opis wygenerowany automatycznie

Tabela 1. Mapa rejestrówakcelerometru.

# Przedstawienie układu

## Struktura projektu

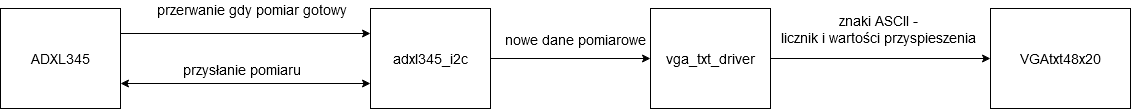
Definicja głównego układu zawarta jest w pliku *adxl345\_vga.sch.* Układa składa się z trzech modułów:

* *adxl345\_i2c* – (plik *adxl345\_i2c.sch*) odpowiada za komunikację za pośrednictwem protokołu I2C 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 przyspieszeniu w osiach z akcelerometru. Następnie dane w module *vga\_txt\_driver* konwertowane są na liczby heksadecymalne kodowane w ASCII. Kolejne cyfry przekazywane są do modułu *VGAtxt48x20*.

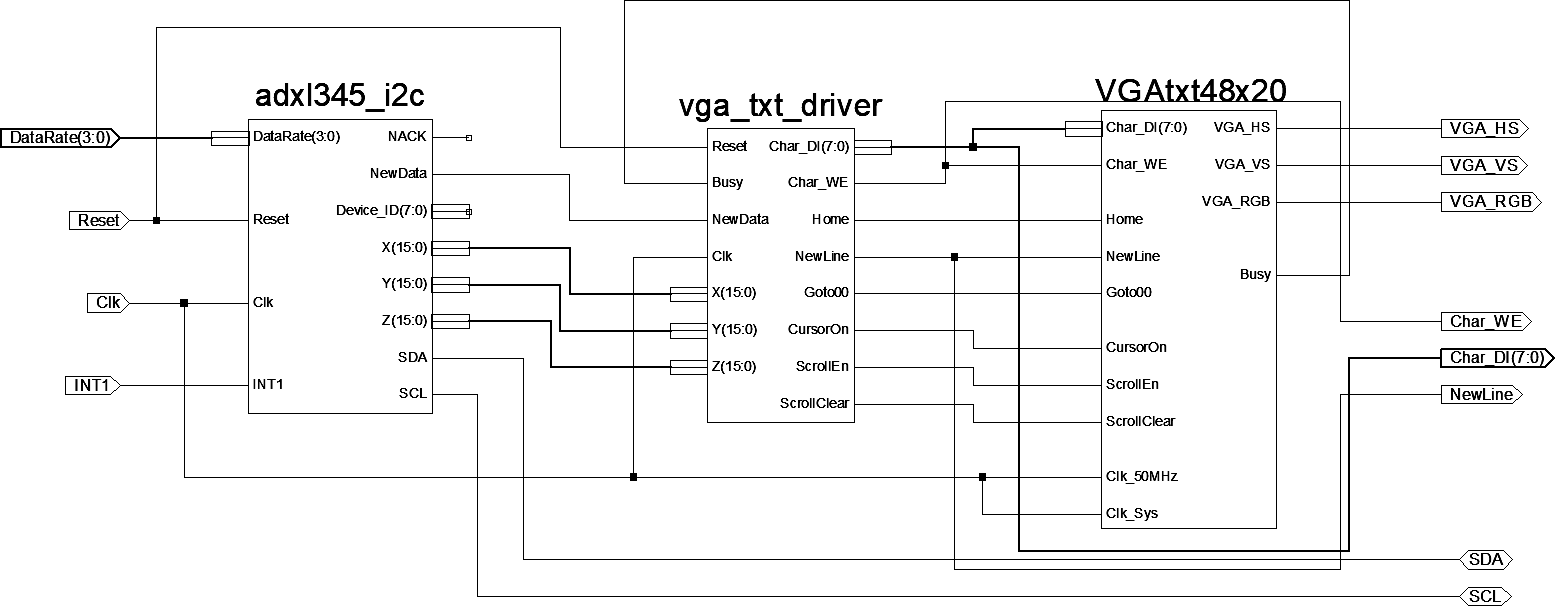
Każdy kolejny pomiar wyświetlany jest w nowej linii. Dodatkowo są one numerowane.

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



## Opis modułów

### 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 | - |

Symulacja  
Moduł zawiera złożony plik testowy, za którego pośrednictwem można ustawić wartość 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 1. Możliwe częstotliwości pomiarów

Odpowiada on za częstotliwość wykonywania pomiarów. Plik testowy zawiera proces symulujący działanie układu I2C slave, który melduje wykonywane operacje w terminalu tekstowym. W konsoli wyświetlany jest również ciąg znaków, który powinien pojawić się na ekranie VGA.

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 przysłane jest 6 bajtów danych. W linii 9 widać linie, która w fizycznym układzie zostałaby wypisana 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



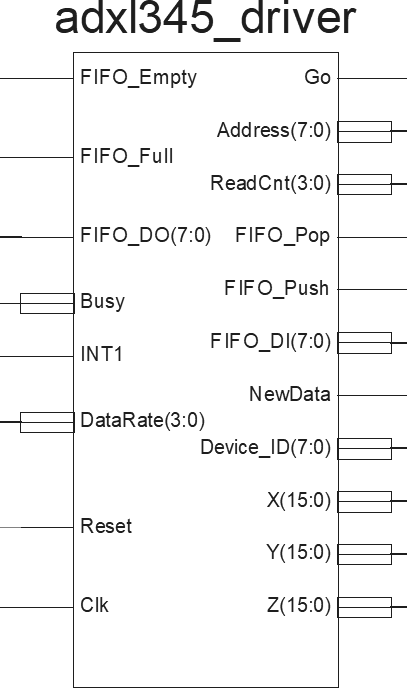
### Moduł adxl345\_driver

Funkcja  
Moduł jest sterownikiem akcelerometru ADXL345. Został stworzony do współpracy z modułem 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

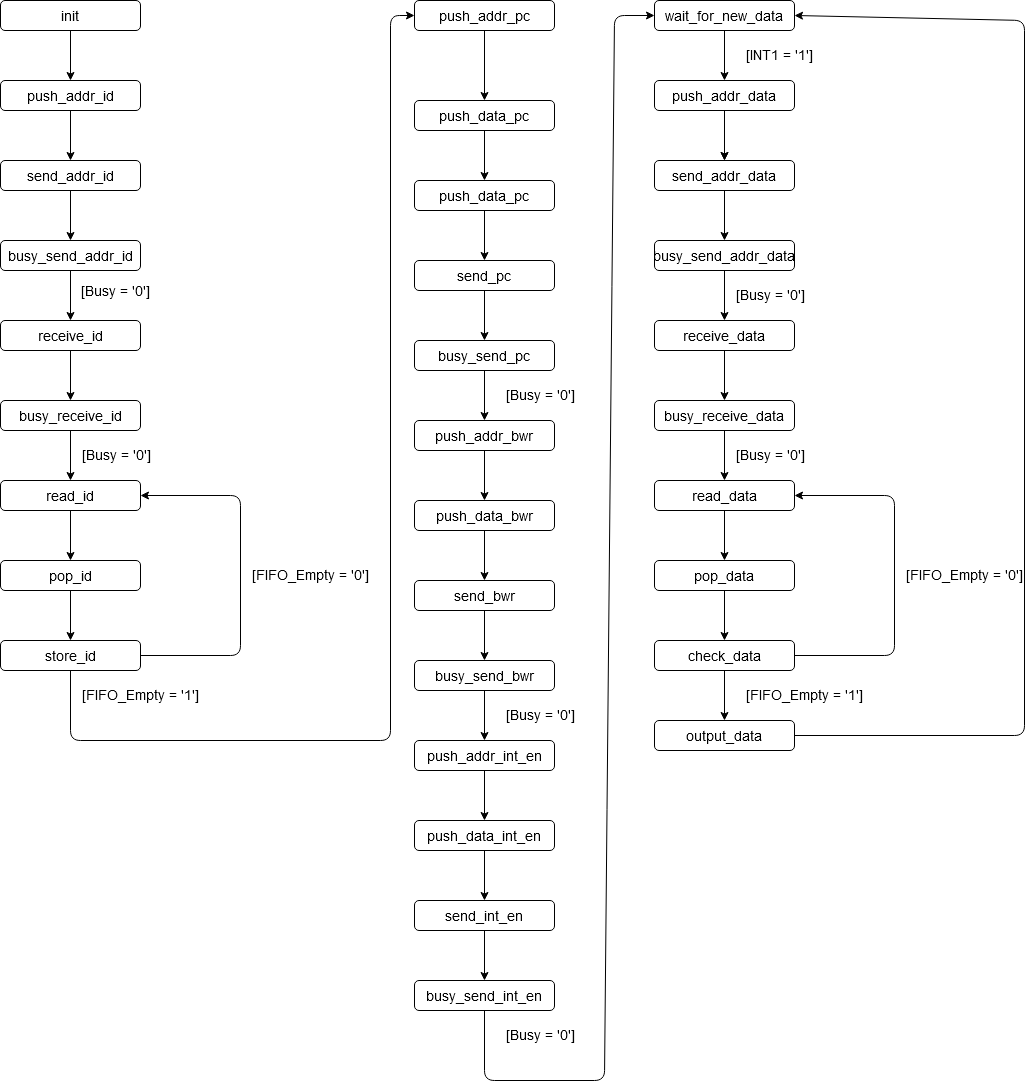
Lista wejść/wyjść

|  |  |  |  |
| --- | --- | --- | --- |
| **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 2. 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 przepełnienia lub wystąpienia sygnału *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;

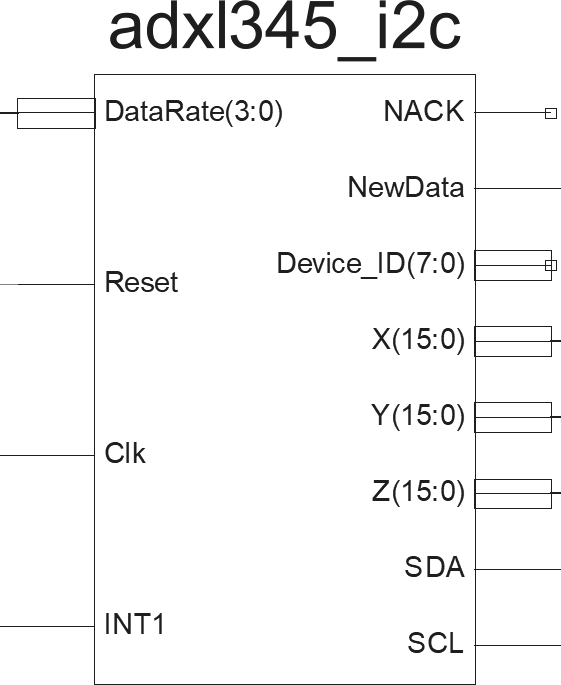
Symulacja

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.

### Moduł adxl345\_i2c

Funkcja  
Moduł łączy ze sobą sterownik adxl345\_driver z I2C\_Master.

Symbol



Rysunek 5. Symbol modułu adxl345\_i2c

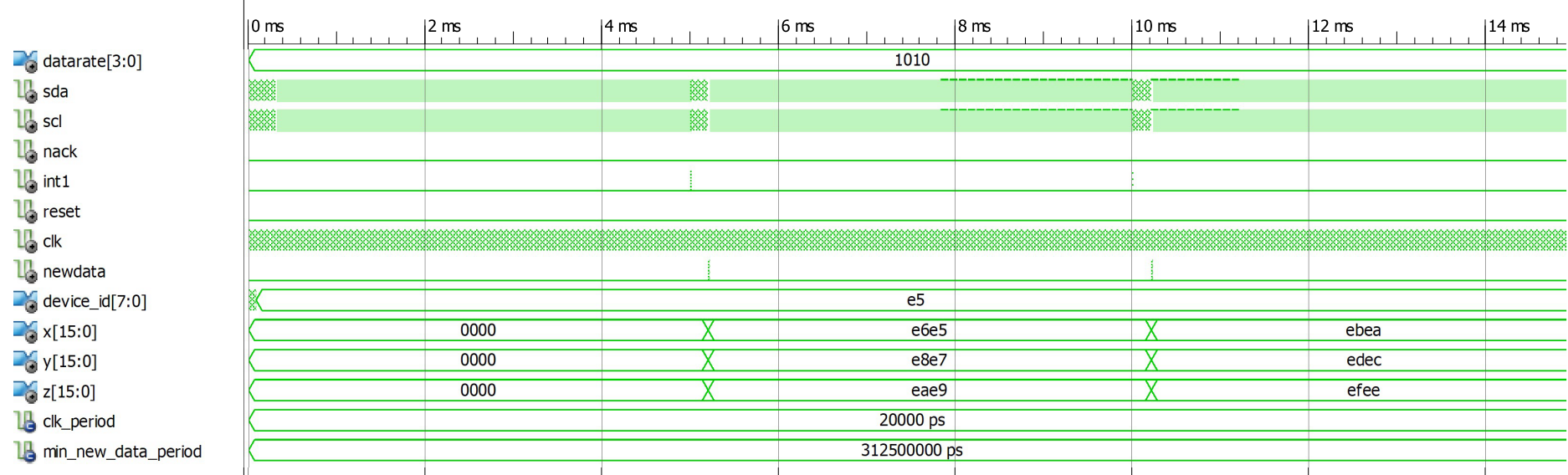
Lista wejść/wyjść

|  |  |  |  |
| --- | --- | --- | --- |
| **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.



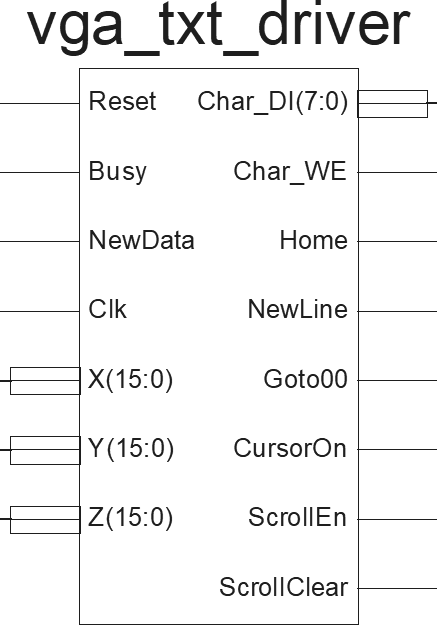
Rysunek 6. Symulacja działania adxl345\_i2c

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

### 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

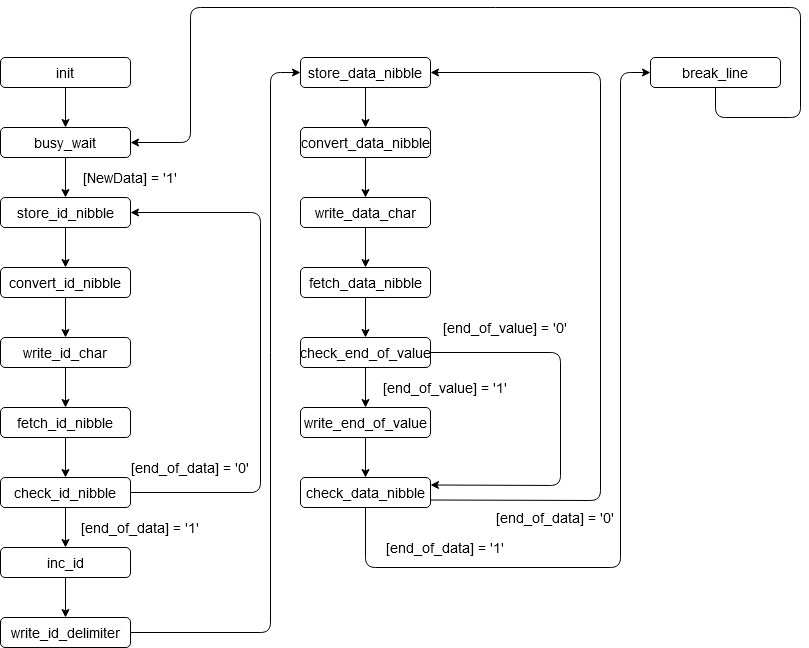
Lista wejść/wyjść

|  |  |  |  |
| --- | --- | --- | --- |
| **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 then
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" + unsigned(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.*

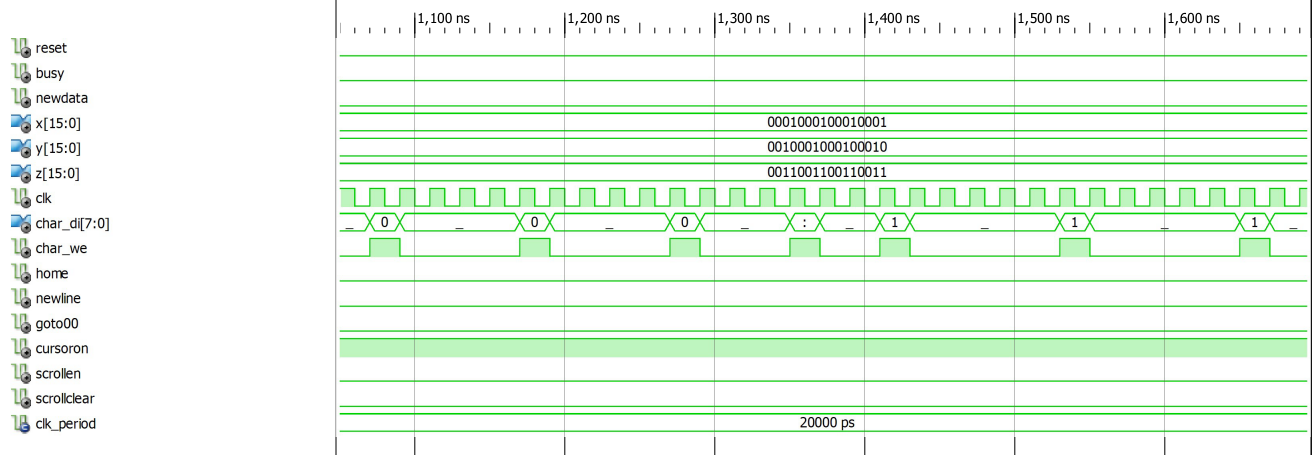
Dodatkowo proces odpowiada, za wysłanie impulsu *end\_of\_value*, jeżeli zakończy się wypisywanie wartości przyspieszenia dla jednej współrzędnej. Sygnał ten jest potrzebny do wstawienia odstępów do ciągu znaków.

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;
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

# Implementacja

## Raporty

## Podręcznik użytkownika urządzenia

# Podsumowanie

## Ocena krytyczna efektu

## Ocena pracy

Umiarkowane problemy sprawiło zaprojektowanie modułu obsługującego akcelero­metr. Pierwszy raz korzystaliśmy z protokołu I2C, 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.

## 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.

# 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>