POLITECHNIKA WROCŁAWSKA

WYDZIAŁ ELEKTRONIKI

PROJEKT Z UKŁADÓW CYFROWYCH I SYSTEMÓW WBUDOWANYCH 2

**Moduł RS232 z kolejką FIFO**

Termin zajęć: Piątek TP, 12:15–15:15

Autorzy: Prowadzący zajęcia:

Michał Tomaszewicz 235568 mgr inż. Antoni Sterna

Wojciech Czarnecki 235714

Wrocław, 2019 r.

Spis treści

[1. Wprowadzenie 3](#__RefHeading___Toc105_1390933273)

[1.1 Temat projektu 3](#__RefHeading___Toc291_205456578)

[1.2 Realizacja na sprzęcie 3](#__RefHeading___Toc293_205456578)

[1.3 Teoria 3](#__RefHeading___Toc293_205456578)

[1.3.1. Moduł RS232 3](#_Toc413365356)

[1.3.2. Kolejka FIFO 4](#_Toc413365357)

[1.4 Opis kodu VHDL 5](#__RefHeading___Toc291_205456578)

[2. Hierarchia źródeł 6](#__RefHeading___Toc107_1390933273)

[3. Opis modułów w kodzie VHDL 6](#__RefHeading___Toc572_1390933273)

[3.1 Moduł RS232\_TX 7](#__RefHeading___Toc291_205456578)

[3.1.1. Sygnały wejściowe i wyjściowe 7](#_Toc413365356)

[3.1.2. Sygnały wewnętrzne 7](#_Toc413365357)

[3.1.3. Transmisja danych 8](#_Toc413365358)

[3.2 Moduł rec\_fifo 9](#__RefHeading___Toc293_205456578)

[3.2.1. Sygnały wejściowe i wyjściowe 9](#_Toc413365356)

[3.2.2. Sygnały wewnętrzne 9](#_Toc413365357)

[3.2.3. Procedura fifo\_clear 10](#_Toc413365358)

[3.2.4. Procedura fifo\_push\_data 11](#_Toc413365358)

[3.2.5. Procedura fifo\_pop\_data 11](#_Toc413365358)

[3.2.6. Odbiór danych 12](#_Toc413365358)

[3.3 Implementacja pliku GenIO.ucf 14](#__RefHeading___Toc293_205456578)

[4. Symulacje modułów 15](#__RefHeading___Toc289_205456578)

[4.1 Symulacja modułu RS232\_TX 15](#__RefHeading___Toc291_205456578)

[4.2 Symulacja modułu FIFO 15](#__RefHeading___Toc293_205456578)

[4.3 Symulacja modułu rec\_fifo 16](#__RefHeading___Toc293_205456578)

[5. Implementacja na sprzęcie 17](#__RefHeading___Toc299_205456578)

[5.1 Użycie zasobów 17](#__RefHeading___Toc291_205456578)

[5.2 Zaprogramowanie układu i użycie na sprzęcie 18](#__RefHeading___Toc293_205456578)

[6. Podsumowanie 21](#__RefHeading___Toc301_205456578)

[6.1 Wnioski 21](#__RefHeading___Toc291_205456578)

[6.2 Dalszy rozwój projektu 21](#__RefHeading___Toc293_205456578)

[7. Bibliografia 21](#__RefHeading___Toc105_1390933273)

**1. Wprowadzenie**

1.1 Temat projektu

Tematem naszego projektu była własna realizacja modułu RS232 – nadajnika i odbiornika, a następnie połączenie odbiornika z kolejką FIFO. Wstępnie napisaliśmy trzy moduły: odbiornika, nadajnika i kolejki FIFO. Później połączyliśmy w kodzie odbiornik i kolejkę FIFO, a na końcowym schemacie całego układu dodaliśmy do tego modułu symbol nadajnika i pokrętła (*Rotary Encoder*).

1.2 Realizacja na sprzęcie

Obydwa moduły, nadajnika i odbiornika z kolejką FIFO, zostały napisane w języku VHDL na układ logiczny typu FPGA z rodziny Xilinx Spartan 3E FPGA Starter Kit Board, model XC3S500E. Wykorzystaliśmy również moduł *RotaryEnc*, który został stworzony przez dr. inż. Jarosława Sugiera, do obsługi pokrętła na płytce. Po wgraniu pliku programowalnego na płytkę (*rec\_fifo\_schemat.bit*), do używania naszego modułu wykorzystujemy klawiaturę podłączoną do komputera oraz pokrętło (*Rotary Encoder*) i diody do wyświetlania binarnej wartości kodu ASCII bajtu na płytce. Za pomocą klawiatury wprowadzamy do komputera bajt znaku, powodujemy jego transmisję i wrzucenie do kolejki, a następnie kręcąc pokrętłem w lewo odbieramy ten bajt, a kręcąc w prawo powodujemy jego odbiór, wyrzucenie z kolejki i wyświetlenie na diodach LED jego wartości binarnej.

1.3 Teoria

1.3.1 Moduł RS232

RS-232 (ang. Recommended Standard 232) odnosi się do protokołu transmisji szeregowej (ang. serial transmission), w której bity są przesyłane kolejno jeden po drugim. Już od ponad 20 lat port szeregowy RS-232 jest stosowany w komputerach PC do podłączania różnych urządzeń - najczęściej modemów i drukarek. Obecnie staje się już przestarzały i jest systematycznie wypierany przez USB.

1.3.2 Kolejka FIFO

Kolejka to sekwencyjna struktura danych o własności, że element zapisany jako pierwszy jest również odczytywany jako pierwszy. Taka struktura w literaturze informatycznej nosi nazwę FIFO (ang. First In First Out). Kolejkę możemy sobie wyobrazić jako tubę – elementy wstawiamy do tuby z jednej strony, po czym przesuwają się one wewnątrz i wychodzą z drugiej strony w tej samej kolejności, w jakiej zostały do tuby włożone. Dla kolejki są zdefiniowane następujące operacje:

* Sprawdzenie, czy kolejka jest pusta – operacja **empty** zwraca wartość ‘1’, jeśli kolejka nie zawiera żadnego elementu, w przeciwnym razie zwraca wartość ‘0’.
* Sprawdzenie, czy kolejka jest pełna – operacja **full** zwraca wartość ‘1’, jeśli jest cała zapełniona, w przeciwnym razie zwraca wartość ‘0’.
* Zapis elementu na koniec kolejki – operacja **push** dopisuje nowy element na koniec elementów przechowywanych w kolejce.
* Usunięcie elementu z kolejki – operacja **pop** usuwa z kolejki pierwszy element.

Naszą kolejkę FIFO zrealizowaliśmy w postaci tablicy. W takim przypadku musimy założyć z góry maksymalny rozmiar kolejki, my wybraliśmy rozmiar 10 bajtów. Zmienne opisujące naszą kolejkę:

* *t\_fifo* - tablica, w której będzie tworzona kolejka. Tablica ma rozmiar 10 elementów, indeksy rozpoczynają się od 0.
* *fifo\_in* – zmienna zapisująca dane do kolejki
* *fifo\_out* – zmienna usuwająca dane z kolejki
* *fifo\_cnt* – licznik przechodzący po kolejnych elementach kolejki

Kolejkę tworzą kolejne elementy o indeksach rozpoczynających się od *fifo\_in*. Na początku *fifo\_in* wskazuje pierwszy element tablicy o indeksie 0. Jeśli dodamy element do kolejki, to dopisanie elementu zwiększa licznik *fifo\_cnt* o 1. Odczyt elementu z kolejki polega na przetworzeniu elementu o indeksie *fifo\_out*. Usunięcie elementu z to zwiększenie o 1 indeksu *fifo\_out.* Jeśli po zwiększeniu *fifo\_out* wskazuje poza ostatni element tablicy, to *fifo\_out* należy wyzerować, a kolejka znów rozpocznie się od początku tablicy. Licznik *fifo\_cnt* zawsze zmniejszamy o 1.

1.4 Opis kodu VHDL

W pierwszej wersji napisaliśmy trzy moduły, a w wersji końcowej utworzyliśmy dwa moduły:

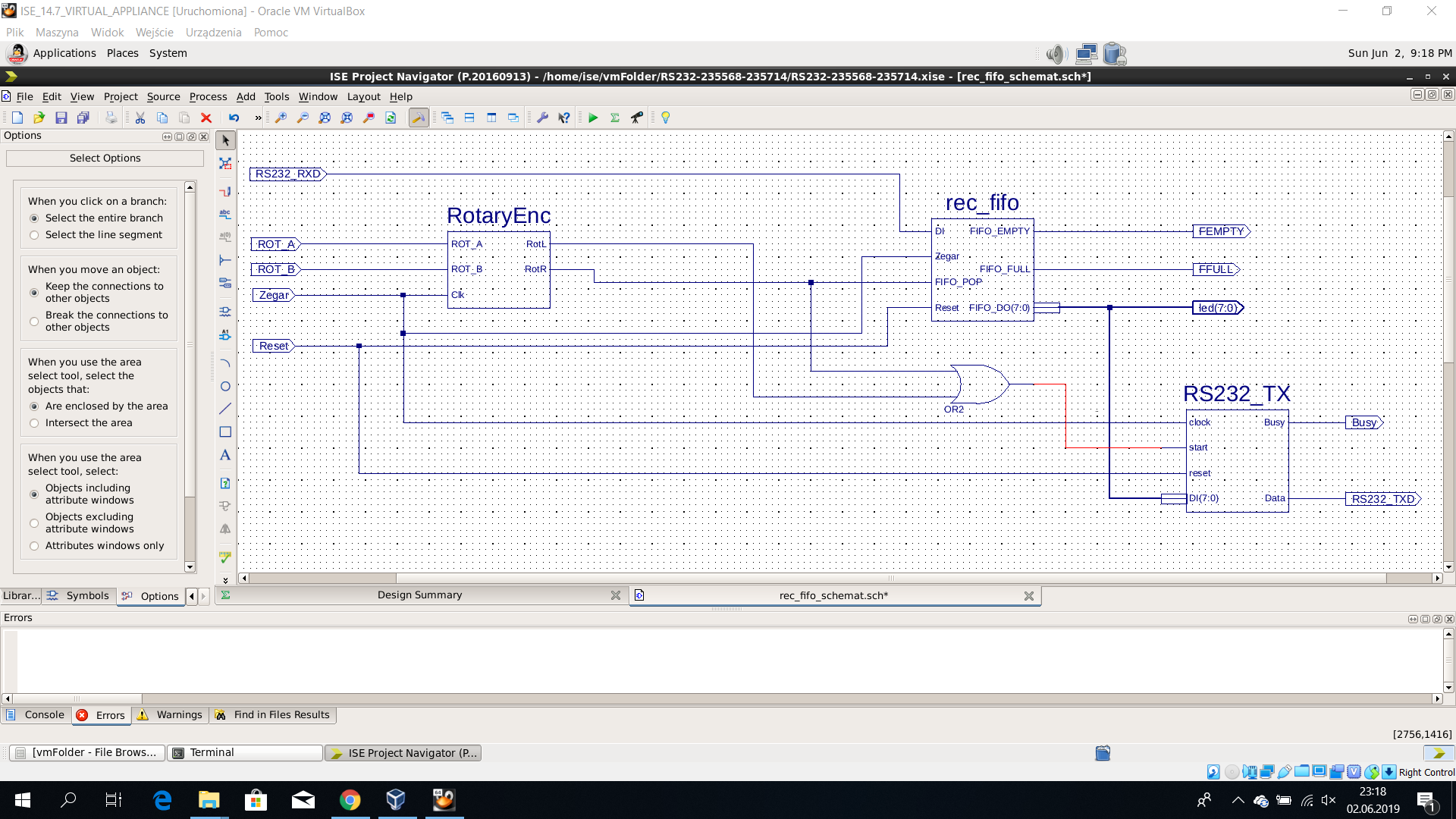
* nadajnik (*RS232\_TX.vhd*) i jego testbench (*RS232\_TX\_tb.vhd*)
* odbiornik połączony z kolejką FIFO (*rec\_fifo.vhd*) i jego testbench (*rec\_fifo\_tb.vhd*)

Również stworzyliśmy następujące pliki:

* plik schematu układu (*rec\_fifo\_schemat.sch*)
* plik *GenIO.ucf*

**2. Hierarchia źródeł**

Naszym głównym źródłem w projekcie jest plik schematu (*rec\_fifo\_schemat.sch*), który wygląda następująco:



Jak widać na powyższym schemacie, projekt składa się z trzech modułów. Moduły *rec\_fifo* i *RS\_232\_TX* napisaliśmy sami, natomiast do obsługi pokrętła wykorzystaliśmy moduł dr. Sugiera. Moduł *RS232\_TX* odpowiada za transmisję danych, a *rec\_fifo* za ich odbieranie i dodawanie/wyrzucanie z kolejki FIFO. Dodatkowo poza schematem i modułami w projekcie znajduje się plik *GenIO.ucf*, który mapuje wejścia i wyjścia z naszego kodu na wejścia i wyjścia płytki. Sygnały wejściowe i wyjściowe są w miarę łatwo zrozumiałe patrząc na schemat:

* *Zegar* – główny zegar
* *Reset* – sygnał z dolnego przycisku, który resetuje cały układ
* Dodatkowe sygnały wejściowe dla rec\_fifo to FIFO\_POP i DI (DataIn)
* Sygnały wyjściowe dla rec\_fifo to FIFO\_EMPTY i FIFO\_FULL sygnalizujące, czy coś znajduje się w kolejce i czy kolejka jest pełna
* Sygnał FIFO\_DO(7:0) wyprowadzający dane na diody
* Dodatkowe sygnały wejściowe dla RS232\_TX to start, który powoduje rozpoczęcie transmisji i DI(7:0), czyli przesyłający bajt danych
* Sygnały wyjściowe dla RS232\_TX to Busy i Data

Sygnały wewnętrzne to sygnały pochodzące z *Rotary Encodera*.

**3. Opis modułów w kodzie VHDL**

3.1 Moduł RS232\_TX

3.1.1 Sygnały wejściowe i wyjściowe

|  |  |
| --- | --- |
| entity RS232\_TX is  port (  clock : in std\_logic;  start : in std\_logic;  reset : in std\_logic;  DI : in std\_logic\_vector(7 downto 0);  Busy : out std\_logic;  Data : out std\_logic  );  end RS232\_TX; | Sygnał zegara  Sygnał startu transmisji  Sygnał resetu  Sygnał danych (1 bajt)  Sygnał Busy  Sygnał Data |

3.1.2 Sygnały wewnętrzne

|  |  |
| --- | --- |
| architecture Behavioral of RS232\_TX is    type state\_type is (A, B, C, D);  subtype t\_counter is integer range 0 to 433;  subtype t\_index is integer range 0 to 7;  subtype t\_byte is std\_logic\_vector(7 downto 0);  constant ticks : t\_counter := 433; --434 - 1  signal index : t\_index;  signal licznik : t\_counter;  signal state : state\_type;  signal DI\_buff : t\_byte;    begin | Układ został zaimplementowany jako maszyna stanów.  Stała *ticks*, której wartość jest równa 433 (liczy się od 0), oznacza ilość taktów zegara, jaka musi upłynąć na każdy transmitowany bit.  Prędkość transmisji RS232 jest równa 115200 bps. Częstotliwość taktowania zegara układu Spartan 3E wynosi 50 MHz. Stąd (50.000.000/115.200 = 434) na każdy bit przypada 434 cykli zegara. |

3.1.3 Transmisja danych

|  |  |
| --- | --- |
| proces\_2 : process(reset, clock)  begin    if reset = '1' then  Busy <= '0';  Data <= '1';  index <= 0;  state <= A;  elsif rising\_edge(clock) then    case state is  when A =>  if start = '1' then  licznik <= 0;  index <= 0;    Busy <= '1';  Data <= '0';  DI\_buff <= DI;  state <= B;  end if;    when B =>  if licznik >= ticks then  licznik <= 0;  state <= C;  else  licznik <= licznik + 1;  end if;    when C =>  Data <= DI\_buff(index);    if licznik >= ticks then  licznik <= 0;    if index < 7 then  index <= index + 1;  else  index <= 0;  Data <= '1';  state <= D;  end if;  else  licznik <= licznik + 1;  end if;    when D => --stop  if licznik >= ticks then  licznik <= 0;  state <= A;  else  licznik <= licznik + 1;  end if;    end case;  end if;  end process proces\_2; | Czas trwania cyklu zegara wynosi 1/50.000.000 sekundy = 20 ns. Stąd czas trwania 1 bitu wynosi 8680 ns.  Po otrzymaniu sygnału *Reset* maszyna wraca do stanu A, w którym oczekuje na sygnał rozpoczynający transmisję – S*tart*.  Gdy sygnał S*tart* zmienia wartość na 1, to sygnał wyjściowy *Busy* zmienia wartość na 1, Data na 0 i następuje przejście do stanu B.  W stanie B należy odczekać 434 cykle zegara z wyjściową linią *Data* ustawioną na 0. W transmisji RS232 jest to tzw. bit startu i oznacza, że kolejny bit będzie już bitem danych.  W stanie C transmitowanych jest 8 bitów danych. Warto wspomnieć, że bity podawane są na linię wyjściową w kolejności od najmłodszego do najstarszego. Nadal każdy bit zajmuje 434 cykle zegara.  W zaimplementowanym transmiterze nie występują bity parzystości – stąd po bitach danych występuje bit stopu.  Stan D jest tzw. bitem stopu – na linię wyjściową *Data*  wyprowadzana jest wartość 1 przez czas trwania bitu. |

3.2 Moduł rec\_fifo

3.2.1 Sygnały wejściowe i wyjściowe

|  |  |
| --- | --- |
| entity rec\_fifo is  Port (  DI : in STD\_LOGIC;  Zegar : in STD\_LOGIC;  FIFO\_EMPTY : out STD\_LOGIC;  FIFO\_FULL : out STD\_LOGIC;  FIFO\_POP : in STD\_LOGIC;  FIFO\_DO : out STD\_LOGIC\_VECTOR(7 downto 0);  Reset : in STD\_LOGIC);  end rec\_fifo; | Sygnał DataIn  Sygnał zegara  Sygnał FIFO\_EMPTY  Sygnał FIFO\_FULL  Sygnał FIFO\_POP  Sygnał danych (1 bajt)  Sygnał Reset |

3.2.2 Sygnały wewnętrzne

|  |  |
| --- | --- |
| architecture Behavioral of rec\_fifo is  type state\_type is (A, B, C, D);    subtype t\_counter is integer range 0 to 433;  subtype t\_index is integer range 0 to 7;  subtype t\_data is std\_logic\_vector(7 downto 0);  type t\_fifo is array(natural range<>) of t\_data;  constant FIFO\_LEN : positive := 10;  constant ticks : t\_counter := 433; --434 - 1    signal fifo : t\_fifo(0 to FIFO\_LEN - 1);  signal fifo\_in : natural range 0 to FIFO\_LEN - 1;  signal fifo\_out : natural range 0 to FIFO\_LEN - 1;  signal fifo\_cnt : natural range 0 to FIFO\_LEN;  signal state : state\_type;  signal index : t\_index;  signal licznik : t\_counter;  signal prev : std\_logic := '1';  signal tmp : std\_logic := '0';  signal bufor : t\_data;  signal r : std\_logic := '0';  signal DI\_l : std\_logic := '1'; | Ten moduł, podobnie jak nadajnik, zaimplementowano w postaci maszyny stanów.  Kolejka fifo (*first in first out*) funkcjonuje jako dwuwymiarowa tablica – bufor w którym zmieści się 10 ośmiocyfrowych wektorów binarnych (bajtów). |

3.2.3 Procedura fifo\_clear

procedure fifo\_clear is

begin

fifo\_in <= 0;

fifo\_out <= 0;

fifo\_cnt <= 0;

for i in 0 to FIFO\_LEN - 1 loop

fifo(i) <= (others => '0');

end loop;

end procedure fifo\_clear;

Procedura *fifo\_clear* odpowiada za wyczyszczenie kolejki z danych po wciśnięciu guzika ‘reset’. Zerowane są też znaczniki *fifo\_out* oraz *fifo\_in,* które w kolejności wskazują na pierwszy i ostatni element w kolejce.

3.2.4 Procedura fifo\_push\_data

procedure fifo\_push\_data (data : in t\_data) is

begin

if fifo\_cnt < FIFO\_LEN then

fifo(fifo\_in) <= data;

fifo\_cnt <= fifo\_cnt + 1;

if fifo\_in = FIFO\_LEN - 1 then

fifo\_in <= 0;

else

fifo\_in <= fifo\_in + 1;

end if;

end if;

end procedure fifo\_push\_data;

Procedura *fifo\_push\_data* odpowiada za dodawanie do kolejki kolejnych bajtów. Warunkiem dodania kolejnego bajtu danych do kolejki jest wolne miejsce w kolejce (licznik elementów *fifo\_cnt* < maksymalnej długości kolejki). Podczas umieszczania danych w kolejce zwiększany jest licznik oraz indeks *fifo\_in* oznaczający pozycję następnego elementu który zostanie dodany jako kolejny.

3.2.5 Procedura fifo\_pop\_data

procedure fifo\_pop\_data (data : out t\_data) is

begin

if fifo\_cnt > 0 then

data := fifo(fifo\_out);

fifo\_cnt <= fifo\_cnt - 1;

if fifo\_out = FIFO\_LEN - 1 then

fifo\_out <= 0;

else

fifo\_out <= fifo\_out + 1;

end if;

end if;

end procedure fifo\_pop\_data;

Procedura *fifo\_pop\_data* odpowiada za usuwanie z kolejki kolejnych bajtów. Wskaźnik *fifo\_out* przechowuje pozycję w kolejce, która zostanie usunięta po wykonaniu tej procedury. Warunkiem pobrania elementu z kolejki jest niepusty stan kolejki.

3.2.6 Odbiór danychHŚ

|  |  |
| --- | --- |
| variable temp : t\_data;  begin  if Reset = '1' then  DI\_l <= '1';  index <= 0;  state <= A;    fifo\_clear;  elsif rising\_edge(Zegar) then  DI\_l <= DI;  prev <= DI\_l;    if FIFO\_POP = '1' then  fifo\_pop\_data (temp);  end if;    case state is  when A =>  licznik <= 0;  index <= 0;    if DI\_l = '0' and prev = '1' then  state <= B;  end if;    when B =>    if licznik >= ticks / 2 then  if DI\_l = '1' then  state <= A;  else  licznik <= 0;  state <= C;  end if;  else  licznik <= licznik + 1;  end if;    when C =>    if licznik >= ticks then  bufor(index) <= DI;  if index >= 7 then  licznik <= 0;  state <= D;  else  index <= index + 1;  licznik <= 0;  end if;  else  licznik <= licznik + 1;  end if;    when D =>    if licznik >= ticks then  state <= A;  fifo\_push\_data (bufor);  else  licznik <= licznik + 1;  end if;    end case;  end if;  end process proces;    FIFO\_EMPTY <= '1' when fifo\_cnt = 0 else '0';  FIFO\_FULL <= '1' when fifo\_cnt = FIFO\_LEN else '0';  FIFO\_DO <= fifo(fifo\_out);    end Behavioral; | Ustawienie stanu A po resecie. Stan domyślny maszyny – zawartość kolejki wraz ze znacznikami jest czyszczona.  W odbiorniku RS232, który zaimplementowano, wprowadzono mechanizm niwelujący negatywne skutki zjawiska metastabilności przerzutników.  Wykrycie zmiany linii wejściowej z wartości 1 na 0 odbywa się za pośrednictwem przerzutników – nie jest odczytywana bezpośrednio z linii danych.  Stan B – odbieranie bitu startu. Wartość linii wejściowej próbkowana jest w połowie czasu bitu – po 4340 ns. Ma to na celu zignorowanie możliwych oscylacji sygnału na granicach bitu.  W stanie C odczytywane są z linii wejściowej bity danych.  Ponownie, jak w przypadku transmisji, nie wykorzystano bitów parzystości.  W stanie D – odbierany jest bit stopu. Odebrany wektor bitów danych, jeśli to możliwe, umieszczany jest w kolejce.  Wysłanie sygnałów FIFO\_EMTY i FIFO\_FULL, gdy kolejka jest pusta/pełna. |

3.3 Implementacja pliku GenIO.ucf

Aby zaimplementować nasz układ na płytce, przygotowaliśmy odpowiedni plik .ucf.

# soldered 50MHz Clock.

NET "Zegar" LOC = "C9" | IOSTANDARD = LVTTL | PERIOD = 20.0ns HIGH 50%;

(...)

# Push-buttons (Press = Hi)

NET "Reset" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN;

(...)

# Rotary encoder

NET "ROT\_A" LOC = "K18" | IOSTANDARD = LVTTL | PULLUP;

NET "ROT\_B" LOC = "G18" | IOSTANDARD = LVTTL | PULLUP;

(...)

# Simple LEDs (Hi = On)

NET "led<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;

NET "led<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;

NET "led<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;

NET "led<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;

NET "led<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;

NET "led<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;

NET "led<6>" LOC = "E9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;

NET "led<7>" LOC = "F9" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 4;

(...)

# RS-232 Serial Port: DCE

NET "RS232\_RXD" LOC = "R7" | IOSTANDARD = LVTTL ;

NET "RS232\_TXD" LOC = "M14" | IOSTANDARD = LVTTL | DRIVE = 8 | SLEW = SLOW ;

(...)

W pliku .ucf wykorzystujemy następujące komponenty:

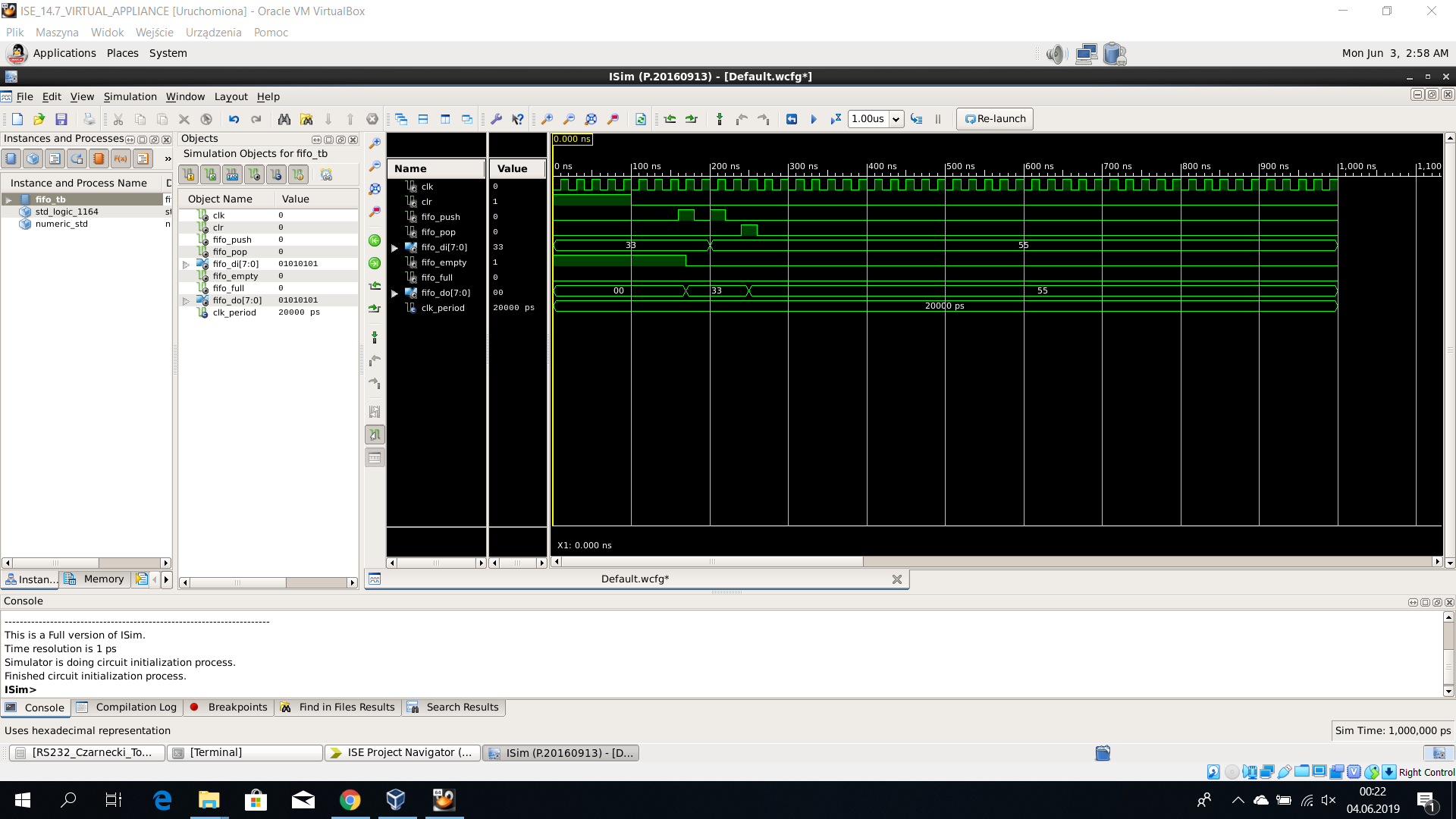
* zegar o częstotliwości 50 MHz
* dolny guzik jako reset układu
* pokrętło do odbierania i wyrzucania poszczególnych bajtów z kolejki
* diody LED do wyświetlania binarnej wartości bajtu
* port DCE do transmisji i odbierania danych

**4. Symulacje modułów**

4.1 Symulacja modułu RS232\_TX



Na powyższej symulacji widzimy transmisję dwóch bajtów o wartościach szesnastkowych 4A i 27. Sygnał ‘start’ pojawia się, gdy rozpoczyna się transmisja bajtu. Sygnał ‘reset’ ustawiliśmy w symulacji po 46 us, ponownie ustawiany jest na wartość ‘0’ po 14 us.

4.2 Symulacja modułu FIFO

Na powyższym zdjęciu znajduje się symulacja samej kolejki FIFO, jeszcze niepołączonej z modułem odbiornika RS232. Na początku wartości sygnałów FIFO\_PUSH i FIFO\_POP mają przypisane wartości ‘0’. Reset ustawiony jest na wartość ‘1’ przez 100 ns, a następnie po trzech cyklach zegara rozpoczynamy dodawanie do kolejki bajtu o wartości szesnastkowej ‘33’. Następnie dodajemy do kolejki bajt o wartości szesnastkowej ‘55’. Następnie usuwamy z kolejki bajt o wartości ‘33’.

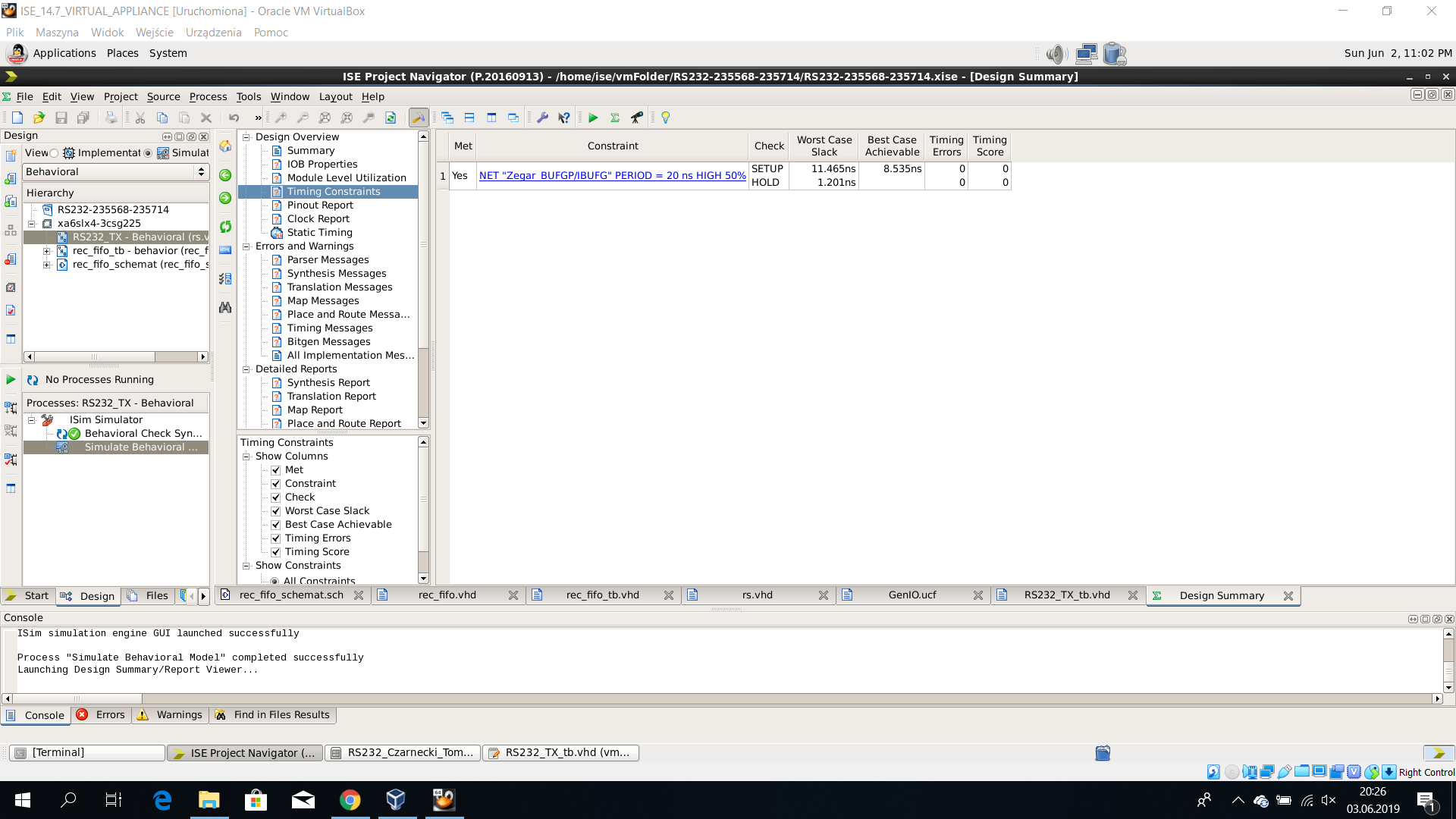
4.3 Symulacja modułu rec\_fifo

Na powyższym zdjęciu znajduje się symulacja odbiornika połączonego z kolejką FIFO. Dodaliśmy do niej również procedurę *sim\_rx byte*, aby nie sterować w symulacji linią DI w odbiorniku ręcznie. Odbieramy bajt o wartości szesnastkowej ‘30’. Następnie kolejny bajt o wartości szesnastkowej ‘41’. Następnie usuwamy z kolejki obydwa bajty, przez co sygnał FIFO\_EMPTY przyjmuje wartość ‘1’.

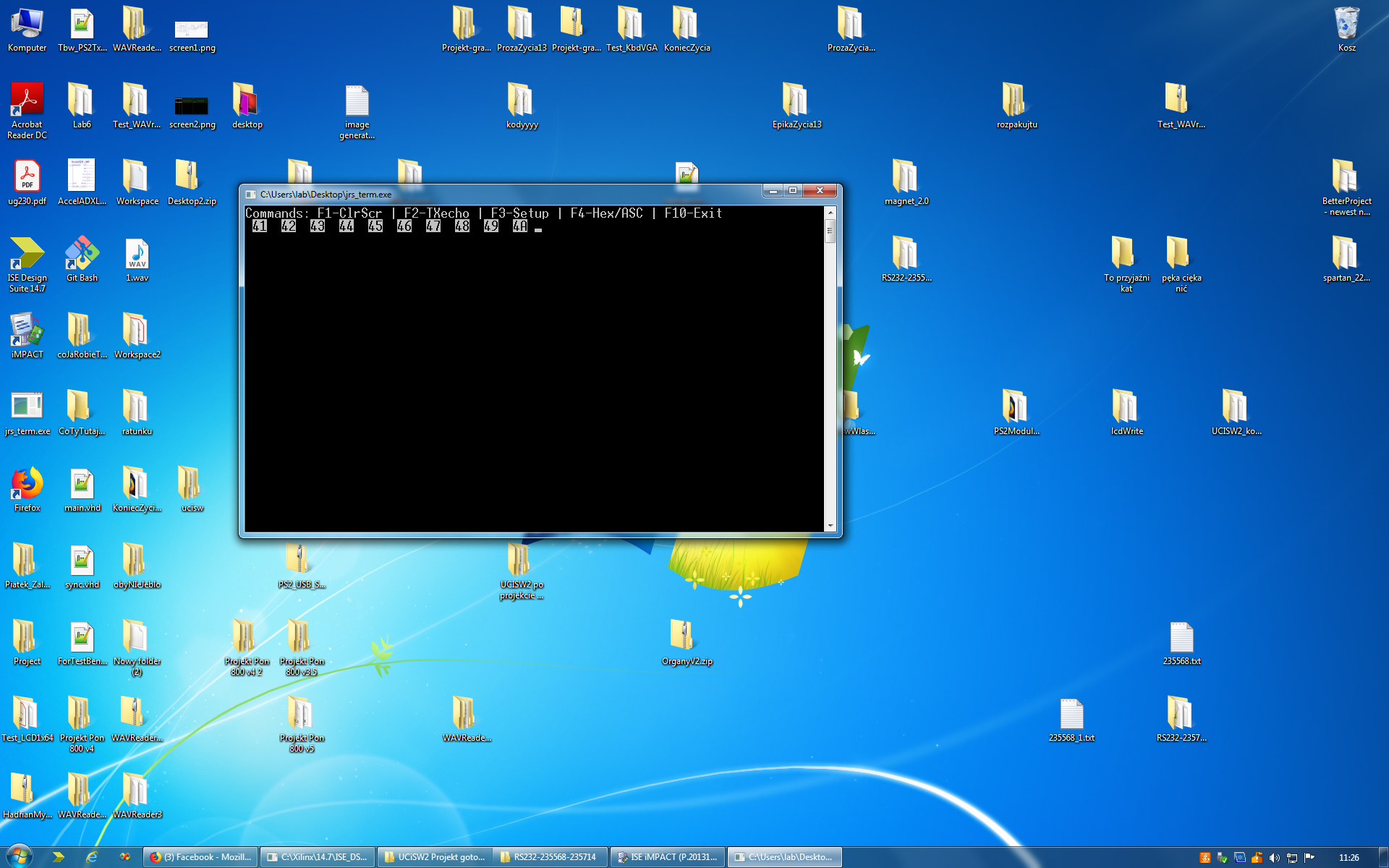
**5. Implementacja na sprzęcie**

5.1 Użycie zasobów

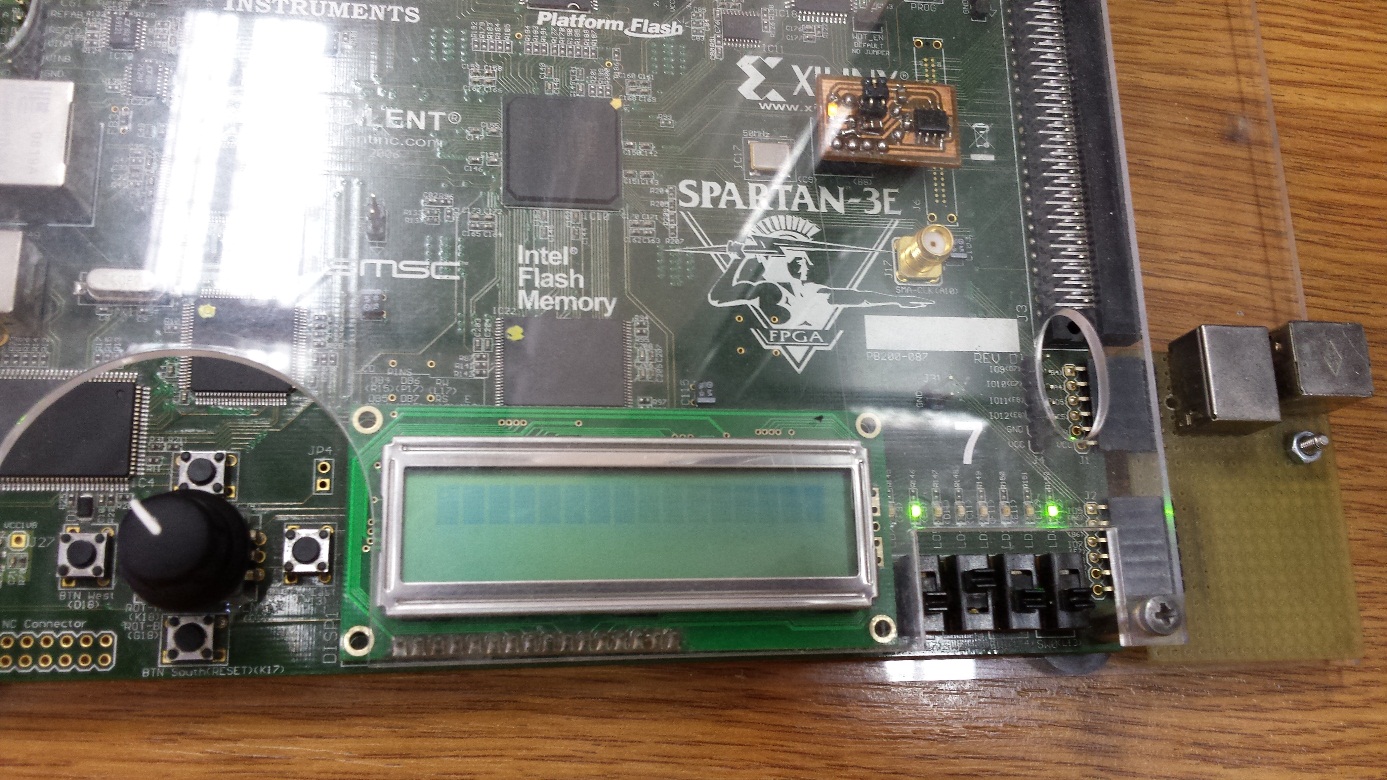
Użycie konkretnych zasobów układu prezentuje poniższa tabela (*Design Summary*).

Nasz program jest taktowany zegarem 50MHz, czyli posiada okres 20ns. Z uwagi na „timing constraint”, nasz rzeczywisty okres wynosi 8.535ns, co przekłada się na zegar rzeczywisty 117.16MHz.

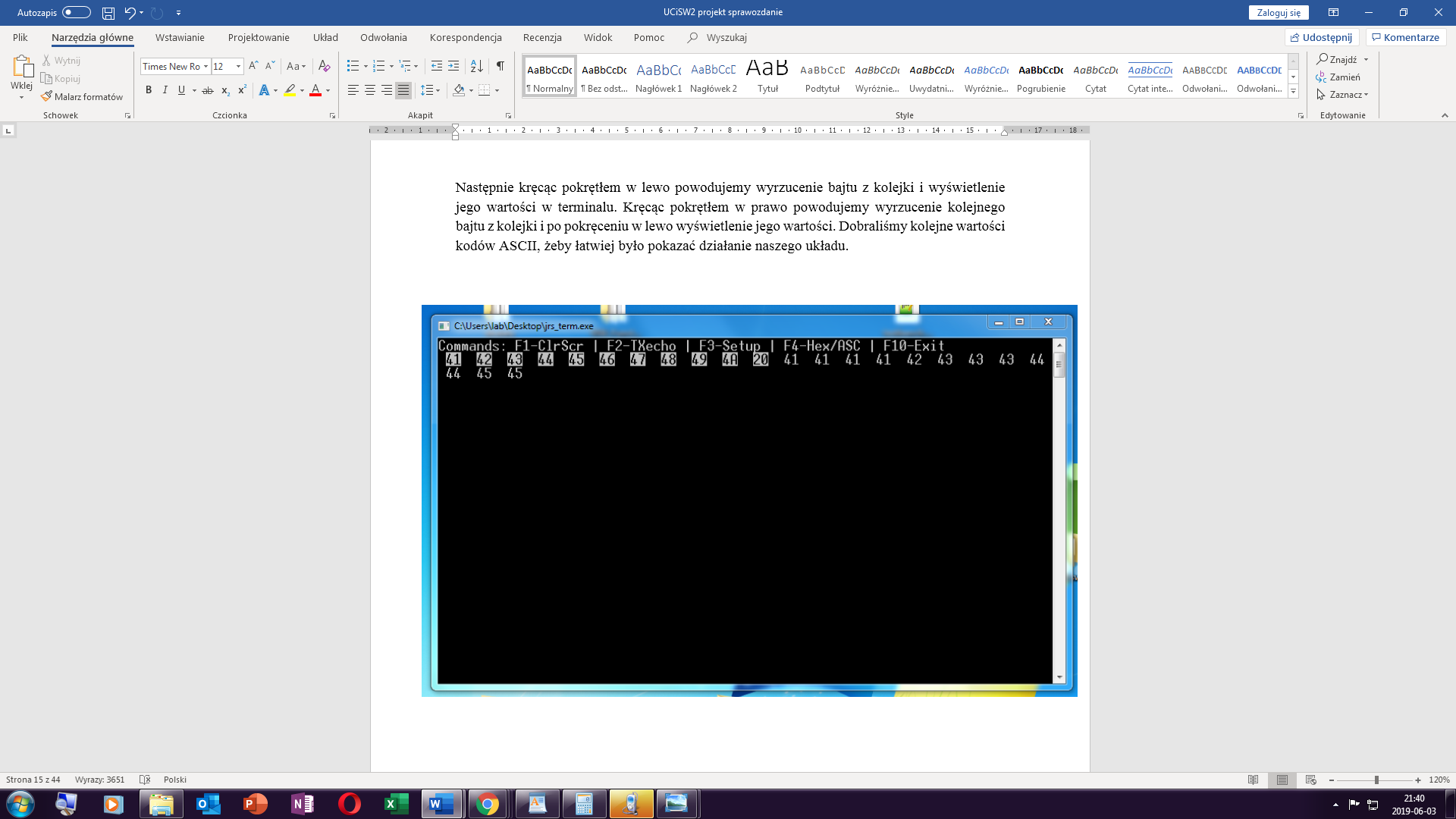
5.2 Zaprogramowanie układu i użycie na sprzęcie

Aby nasz projekt w pełni działał na sprzęcie, nie musieliśmy podłączać do płytki Spartan 3E dodatkowych urządzeń, wystarczyła nam jedynie płytka połączona z komputerem. Po sprawdzeniu, czy jest ona poprawnie połączona z komputerem, kompilujemy projekt („*Generate Programming File*”) i wgrywamy gotowy plik (*rec\_fifo\_schemat.*bit) poprzez iMPACT. Wgrywamy tylko plik *.bit* i pomijamy wszystkie inne wyskakujące okienka. Po kliknięciu prawym przyciskiem i wybraniu opcji „*Program*” na sprzęcie nie powinno się nic wyświetlać, np. na diodach LED. Dopiero po wprowadzeniu bajtu do komputera, na płytce zapalają się diody LED oznaczające jego wartość binarną. Na komputerze w terminalu pojawiają się za to wartości heksadecymalne. W naszym przypadku kolejka ma rozmiar dziesięciu bajtów, więc wprowadziliśmy do niej taką samą ilość danych.

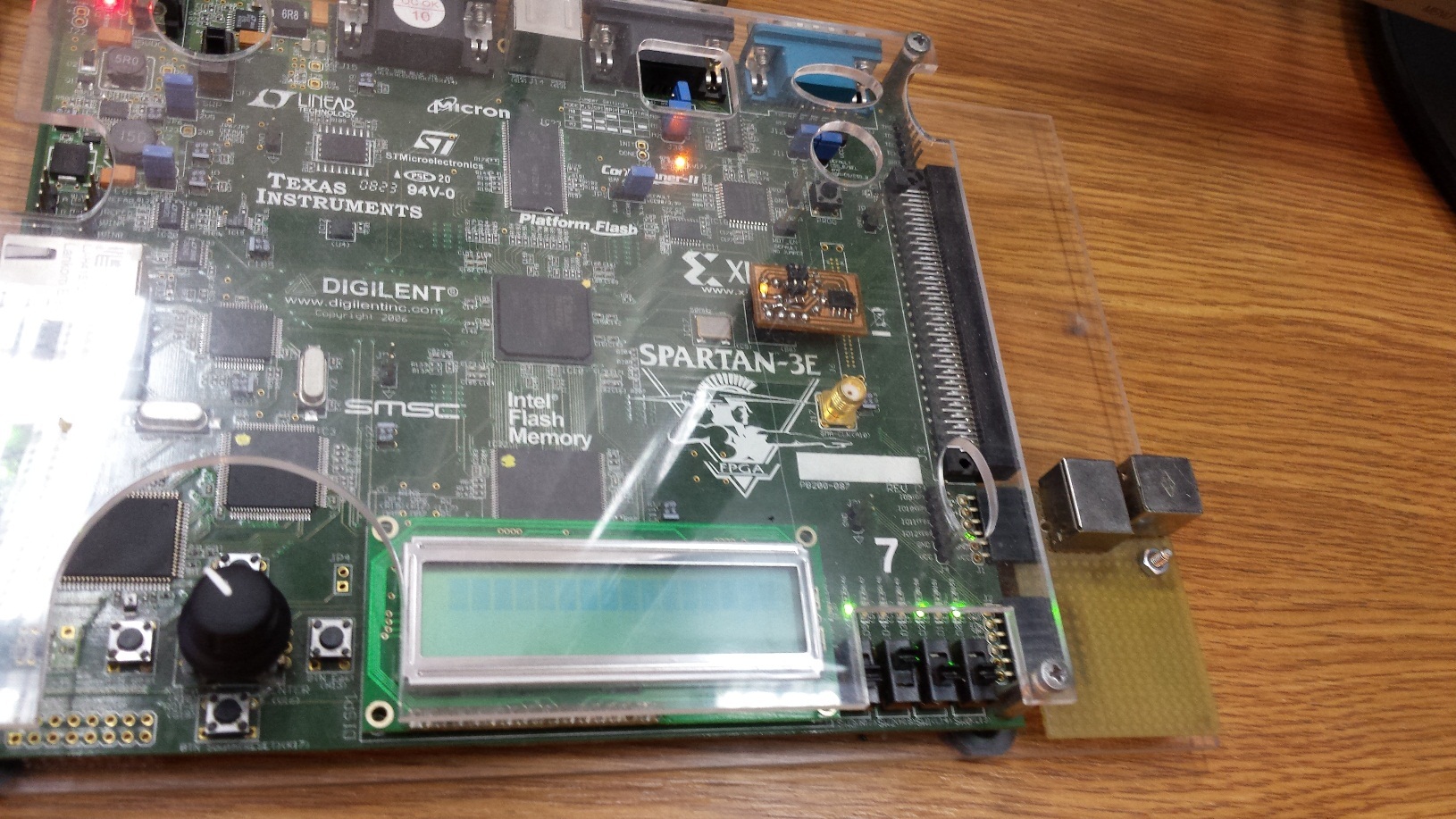
Po wprowadzeniu danych do kolejki wyświetlana jest na diodach wartość pierwszego wprowadzonego znaku w kodzie ASCII (w tym przypadku wprowadziliśmy literę A, binarnie 01000001, szesnastkowo 41).



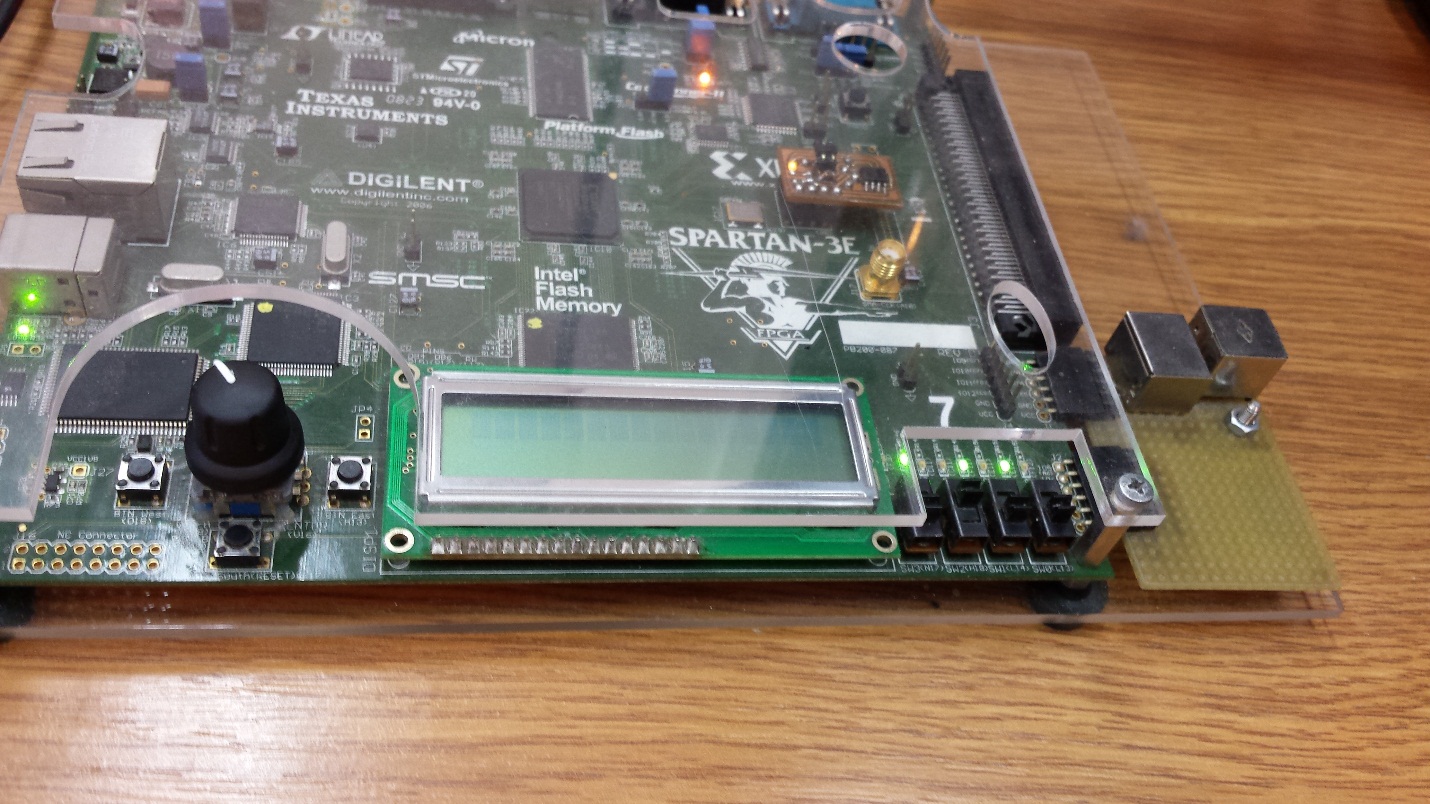
Następnie kręcąc pokrętłem w lewo powodujemy dalszy odbiór bajtu i wyświetlenie jego wartości w terminalu. Kręcąc pokrętłem w prawo powodujemy wyrzucenie kolejnego bajtu z kolejki i po pokręceniu w lewo wyświetlenie jego wartości. Specjalnie dobraliśmy kolejne wartości kodów ASCII, żeby łatwiej było pokazać działanie naszego układu.



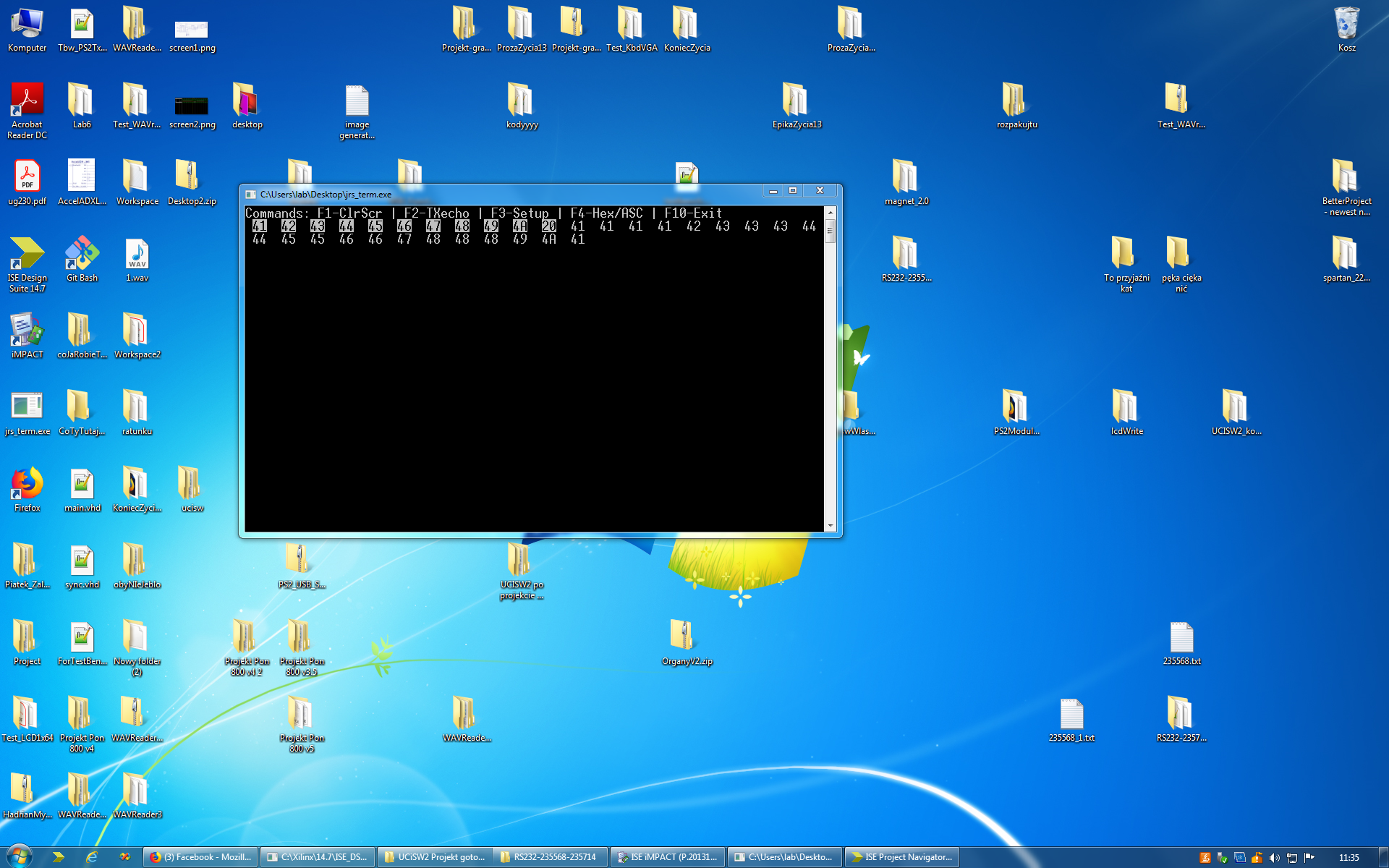
Na diodach LED obserwujemy wyświetlanie kolejnych bajtów z kolejki.



Po wyrzuceniu wszystkich bajtów z kolejki odbierany i wyświetlany jest bajt, który był dodany do kolejki jako pierwszy. W naszym przypadku literę A dodaliśmy jako pierwszą, więc jej kod ASCII będzie wyświetlany na diodach LED. Na poniższym zdjęciu obserwujemy wyświetlanie ostatniego bajtu z kolejki (litera J, kod ASCII hex 4A, bin 01001010).



Na terminalu widzimy wyświetlany bajt po opróżnieniu całej kolejki, który był dodany do niej jako pierwszy.



Dla przetestowania, czy sygnały FIFO\_FULL i FIFO\_EMPTY działają poprawnie, wykorzystaliśmy chwilowo diody led <5> i led <7> do wyświetlenia ich wartości. Wybraliśmy te dwie diody, gdyż na nich zawsze wyświetlana jest dla bitu wartość ‘0’. Po przetestowaniu na płytce okazało się, że wartości sygnałów są prawidłowe. W momencie dodania dziesięciu bajtów do kolejki zapala się dioda oznaczająca wartość ‘1’ dla sygnału FIFO\_FULL. Po jej opróżnieniu zapala się dioda oznaczająca wartość ‘1’ dla sygnału FIFO\_EMPTY. Gdybyśmy chcieli dodać więcej bajtów do kolejki niż jest w niej miejsca, to dodatkowe bajty nie są nadpisywane.

**6. Podsumowanie**

6.1 Wnioski

Praca nad projektem z naszej winy rozpoczęła się dopiero od połowy semestru. Naiwnie liczyliśmy to, że taki czas wystarczy na ukończenie prac. Dzięki uprzejmości Pana Prowadzącego po zajęciach zostawaliśmy dłużej, by nadrabiać zaległości. Ostatecznie nie starczyło czasu na modyfikację parametrów pracy modułu RS232 i znacznie czytelniejsze wyświetlanie stanu kolejki na wyświetlaczu LCD. Podczas prac projektowych wiele się nauczyliśmy zarówno z tworzenia modułów w języku VHDL, przeprowadzania symulacji, jak i zarządzania projektem. Ważną lekcją było docenienie czasu jaki przeznaczony jest na pracę w sali laboratorium 0.16, gdyż jest tylko wtedy możliwe jest testowanie oprogramowania na sprzęcie.

6.2 Dalszy rozwój projektu

* Wykorzystanie wyświetlacza LCD do pokazywania zawartości kolejki
* Wyświetlanie na diodach LED wartości sygnałów FIFO\_FULL i FIFO\_EMPTY. Gdy z diod LED przenieślibyśmy wyświetlanie aktualnie odbieranego bajtu, to na dwóch diodach moglibyśmy wyświetlić, czy w kolejce coś się znajduje i czy kolejka jest pełna.

**7. Bibliografia**

* https://www.xilinx.com/ - strona producenta
* https://www.xilinx.com/support/documentation/boards\_and\_kits/ug230.pdf - podręcznik użytkownika (Spartan-3E FPGA Starter Kit Board User Guide, User Guide, ug230.pdf)
* http://www.zsk.ict.pwr.wroc.pl/zsk\_ftp/fpga/ - strona z modułami dr. Jarosława Sugiera
* http://staff.iiar.pwr.wroc.pl/antoni.sterna/ - strona mgr. Antoniego Sterny
* https://pl.wikipedia.org/wiki/RS-232
* https://eduinf.waw.pl/inf/alg/002\_struct/0050.php
* https://eduinf.waw.pl/inf/alg/001\_search/0105.php