IP Core Generator (wersja 1)

Treść zadania

Należy rozbudować projekt z ćwiczenia „Monitor portu RS-232” w następujący sposób:
System powinien buforować dane odbierane z portu RS232, a następnie po zebraniu 18 znaków lub wcześniej, jeżeli wykryty został kod 13 (klawisz enter), wydrukować je z powrotem do portu RS232 w formie „pseudograficznej” (używając jako „piksela” znaku o takim samym kodzie jak drukowany). Znaki o kodzie 13 nie są drukowane. Oprócz tego projekt powinien buforować znaki przychodzące z wejścia RS232 podczas drukowania linii (kiedy nie można ich na bieżąco drukować). Długość bufora: 64 znaki. Przekroczenie bufora powinno być sygnalizowane zapaleniem się diody LD0. Parametry portu RS232 oraz funkcja monitorowania odebranych znaków (przed buforowaniem) powinny pozostać tak jak w ćwiczeniu „Monitor portu RS-232”. W tym ćwiczeniu nie korzystamy z sygnału „reset”. Proszę zamiast tego stosować wartość początkową sygnałów i zmiennych.

Poniżej podano przykład wydruku kilku linii:

```
        bbb                 ddd            fff
         bb                  dd           ff ff
         bb                  dd           ff  f
 aaaa    bbbbb    cccc     dddd   eeee    ff
    aa   bb  bb  cc  cc   dd dd  ee  ee  fffff
 aaaaa   bb  bb  cc       dd  dd  eeeeee   ff
aa  aa   bb  bb  cc       dd  dd  ee       ff
aa  aa   bbb bb  cc  cc   dd  dd  ee  ee   ff
 aaa aa bbb bb    cccc    ddd dd  eeee   ffff
```

```
   AA   BBBBBB     CCC  DDDDD    EEEEEE  FFFFFF   GGGG   HH  HH
  AAAA   BB  BB   CC CC  DD DD    EE  E   FF  F  GGG GG  HH  HH
 AA  AA  BB  BB  CC   C  DD  DD   EE      FF     GG      HH  HH
 AA  AA  BB  BB  CC      DD  DD   EE E    FF F   GG      HH  HH
 AA  AA  BBBBB   CC      DD  DD   EEEE    FFFF   GG      HHHHHH
 AAAAAA  BB  BB  CC      DD  DD   EE E    FF F   GG GGG  HH  HH
 AA  AA  BB  BB  CC   C  DD  DD   EE      FF     GG  GG  HH  HH
 AA  AA  BB  BB   CC CC  DD DD    EE  E   FF     GGG GG  HH  HH
 AA  AA BBBBBB     CCC  DDDDD    EEEEEE  FFFF     GGGG   HH  HH
```

```
  11     2222    3333      44   555555    666   777777
 111    22  22  33  33    444   55       66     77  77
1111        22      33   4444   55      66          77
  11        22      33  44 44   55      66         77
  11       22    3333  44  44   55555   66666      77
  11      22        33 4444444      55  66  66    77
  11     22         33     44       55  66  66    77
  11    22  22  33  33     44   55  55  66  66    77
 1111   222222   3333     4444   5555    6666     77
```

Znaki o kodach poniżej 32 i powyżej 127 powinny być drukowane przy użyciu gwiazdek jako „pikseli”. Nie należy wprowadzać kontrolnej funkcji tych znaków (z wyjątkiem kodu 13, który oznacza koniec linii). Kod 13 nie powinien być drukowany. Przykład linii zawierającej znaki o kodach 0-31:

```
 ****     *****  *****   *****                    ***
**  **      *** *     * *******  ** **     *     *****
**  **     ** * * * * * ** * ** *******   ***    *****
**  **     **  * *     * *******  *******  *****    ***
 ****    ****   *     * ******* ******* ******* *******
  **    **  **  * *** * **   ** *******  *****  *******
******  **  **  *  *  * *** ***  *****    ***    *****
  **    **  **  *     * *******   ***      *       *
  **     ****    *****   *****     *              *****
```

Wejścia i wyjścia układu:
clk_i - zegar 100MHz,
RXD_i - wejście danych RS232,
TXD_o - wyjście danych RS232,
ld0 – wyjście sygnalizacji przepełnienia bufora FIFO (LED LD0),
led7_an_o – wyjście sterujące anodami wyświetlaczy LED,
led7_seg_o – wyjście sterujące segmentami wyświetlaczy LED.

Plik z ograniczeniami projektowymi: isp5a.xdc

Do realizacji projektu potrzebne będą dwa moduły funkcjonalne wygenerowane przez syntezer IP Core. Pamięć ROM posłuży do przechowywania kształtów znaków, zaś pamięć FIFO posłuży do implementacji 64-znakowego bufora FIFO buforującego port wejściowy RS-232. Definicje znaków 8x16 pikseli są dostępne w pliku: chargen.coe

Demonstracja syntezy IP core w Vivado (screencast)

Generacja modułu pamięci ROM za pomocą syntezera IP core w Vivado

- W okienku **Flow navigator** rozwijamy **PROJECT MANAGER** i klikamy w **IP Catalog**.

- Następnie rozwijamy **Memories & Storage Elements** i dalej rozwijamy **RAMs & ROMs & BRAM**.

- Klikamy podwójnie w **Block Memory Generator**.

- W polu **Component Name** wpisujemy nazwę, którą otrzyma generowany przez nas moduł, np.: **char_mem**

- W zakładce **Basic** zmieniamy **Memory Type** na **Single Port ROM**.

- W zakładce **Port A Options** zmieniamy **Port A Width** na **8** (definicja znaku ma w poziomie 8 pikseli), następnie zmieniamy **Port A Depth** na **4096** (jest 256 znaków po 16 bajtów każdy), następnie w polu **Enable Port Type** wybieramy **Always Enabled**. Wyłączamy opcję **Primitives Output Register**.

- W zakładce **Other Options** zaznaczamy **Load Init File**, a następnie za pomocą przycisku **Browse** znajdujemy plik chargen.coe z definicjami znaków i klikamy w **OK**. Pliku nie edytujemy.

- Znajdujący się w lewym górnym rogu przycisk **Documentation** otworzy szczegółową dokumentację generowanego komponentu.

- W zakładce **Summary** możemy znaleźć podsumowanie istotnych parametrów tworzonego komponentu. Szczególnie istotna jest informacja: **Total Port A Read Latency: 1 Clock Cycle(s)**

- Na zakończenie potwierdzamy ustawienia przyciskiem **OK** w prawym dolnym rogu i po chwili zatwierdzamy w okienku **Generate Output Products** przyciskiem **Generate** rozpoczęcie syntezy nowego komponentu.

- Jeszcze raz potwierdzamy w okienku **Generate Output Products** przyciskiem **OK**.

- Po zakończeniu generacji komponent zostanie dołączony do listy plików źródłowych projektu.

Poniżej podano sposób przechowywania kształtów znaków w pamięci ROM (na przykładzie literki R):

Adres	b7	b6	b5	b4	b3	b2	b1	b0
kod_ascii*16+0	0	0	0	0	0	0	0	0
kod_ascii*16+1	0	0	0	0	0	0	0	0
kod_ascii*16+2	**1**	**1**	**1**	**1**	**1**	**1**	0	0
kod_ascii*16+3	0	**1**	**1**	0	0	**1**	**1**	0
kod_ascii*16+4	0	**1**	**1**	0	0	**1**	**1**	0
kod_ascii*16+5	0	**1**	**1**	0	0	**1**	**1**	0
kod_ascii*16+6	0	**1**	**1**	**1**	**1**	**1**	**1**	0
kod_ascii*16+7	0	**1**	**1**	0	**1**	**1**	0	0
kod_ascii*16+8	0	**1**	**1**	0	0	**1**	**1**	0
kod_ascii*16+9	0	**1**	**1**	0	0	**1**	**1**	0
kod_ascii*16+10	**1**	**1**	**1**	**1**	0	**1**	**1**	0
kod_ascii*16+11	0	0	0	0	0	0	0	0
kod_ascii*16+12	0	0	0	0	0	0	0	0
kod_ascii*16+13	0	0	0	0	0	0	0	0
kod_ascii*16+14	0	0	0	0	0	0	0	0
kod_ascii*16+15	0	0	0	0	0	0	0	0

Adres – adres w pamięci ROM.

kod_ascii – kod ASCII wyświetlanego znaku.

Adres dla pamięci ROM najłatwiej jest wygenerować łącząc 8-bitowy wektor kodu znaku ASCII z 4-bitowym wektorem numeru linii w znaku (0-15) za pomocą operatora &. Należy pamiętać o tym, że po wydrukowaniu każdej linii ze znakami należy wysłać dwa znaki: CR (kod: 13) oraz LF (kod: 10) w celu przejścia do nowej linii.

W celu ułatwienia dołączenia komponentu do hierarchii aktualnego projektu w okienku **Sources** można wybrać zakładkę **IP sources**. Wtedy wyświetlą się wygenerowane moduły IP. Po rozwinięciu wybranego modułu oraz rozwinięciu **Instantiation Template** można zobaczyć plik ***.vho** zawierający szkielet deklaracji komponentu oraz przykład jego użycia (tworzenia instancji). Możemy te elementy łatwo przekopiować do naszego pliku projektowego, w którym planujemy podłączyć wygenerowany komponent, a zaprojektowany komponent zajmie właściwe miejsce w hierarchii projektu.

Istotny fragment przykładowego pliku VHO:

```
-- The following code must appear in the VHDL architecture header:
------------- Begin Cut here for COMPONENT Declaration ------ COMP_TAG
component char_mem
        port (
        clka: IN std_logic;
        addra: IN std_logic_VECTOR(11 downto 0);
        douta: OUT std_logic_VECTOR(7 downto 0));
end component;
-- COMP_TAG_END ------ End COMPONENT Declaration ------------

-- The following code must appear in the VHDL architecture
-- body. Substitute your own instance name and net names.
------------- Begin Cut here for INSTANTIATION Template ----- INST_TAG
your_instance_name : char_mem
                port map (
                        clka => clka,
                        addra => addra,
                        douta => douta);
-- INST_TAG_END ------ End INSTANTIATION Template ------------
-- You must compile the wrapper file char_mem.vhd when simulating
-- the core, char_mem. When compiling the wrapper file, be sure to
-- reference the VHDL simulation library.
```

Tekst zaznaczony na czerwono należy skopiować do obszaru definicji sygnałów lokalnych. Tekst zaznaczony na niebiesko (po ustaleniu nazwy instancji komponentu oraz sygnałów podłączonych) należy skopiować do obszaru opisu systemu. Po dokonaniu powyższych modyfikacji komponent powinien zająć właściwe miejsce w hierarchii plików projektu.

Generacja modułu pamięci FIFO za pomocą syntezera IP core w Vivado

- W okienku **Flow navigator** rozwijamy **PROJECT MANAGER** i klikamy w **IP Catalog**.
- Następnie rozwijamy **Memories & Storage Elements** i dalej rozwijamy **FIFOs**.
- Klikamy podwójnie w **FIFO Generator**.
- W polu **Component Name** wpisujemy nazwę, którą otrzyma generowany przez nas moduł, np.: **fifo_mem**
- W zakładce **Native Ports** zmieniamy **Write Width** na **8** (znak przesyłany interfejsem RS232 ma długość 8 bitów), następnie zmieniamy **Write Depth** na **64** (zakładana pojemność FIFO) oraz sprawdzamy czy **Read Width** jest ustawione na **8** (jeżeli nie to poprawić). Następnie wyłączamy opcję **Reset Pin**.
- Znajdujący się w lewym górnym rogu przycisk **Documentation** otworzy szczegółową dokumentację generowanego komponentu.

- W zakładce **Summary** możemy znaleźć podsumowanie istotnych parametrów tworzonego komponentu. Szczególnie istotna jest informacja: **Read Latency (From Rising Edge of Read Clock): 1**

- Na zakończenie potwierdzamy ustawienia przyciskiem **OK** w prawym dolnym rogu i po chwili zatwierdzamy w okienku **Generate Output Products** przyciskiem **Generate** rozpoczęcie syntezy nowego komponentu.

- Jeszcze raz potwierdzamy w okienku **Generate Output Products** przyciskiem **OK**.

- Po zakończeniu generacji komponent zostanie dołączony do listy plików źródłowych projektu.

Podobnie jak w poprzednim przypadku należy teraz podłączyć wygenerowany komponent do projektu (można wykorzystać wygenerowany plik *.vho).

Weryfikacja układu

Należy zweryfikować układ praktycznie poprzez zaprogramowanie płytki testowej. W razie potrzeby należy wykonać symulację funkcjonalną (nie jest obowiązkowa). Weryfikacja praktyczna polega na sprawdzeniu działania systemu poprzez wyświetlenie kilku linii w trybie „pseudograficznym” oraz sprawdzeniu prawidłowości działania układu po przepełnieniu bufora FIFO. Testy, które **obowiązkowo** należy wykonać i przedstawić prowadzącemu laboratorium są wymienione w poniższej prezentacji (screencast). Znajdują się tam także informacje o typowych ostrzeżeniach związanych z syntezą IP core, które można zignorować.

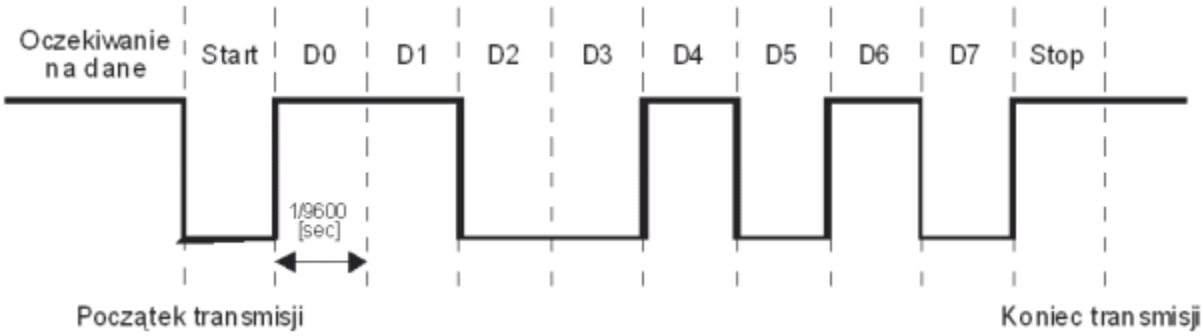
Wymagane do zaliczenia testy układu na płytce FPGA

Układ testować za pomocą programu PUTTY (program jest domyślnie skonfigurowany na 9600bps, 8 bitów danych, 1 bit stopu, bez parzystości). Echa lokalnego nie włączać. Proszę pamiętać o właściwym ustawieniu numeru portu COM (sprawdzić w „Device Manager” systemu Windows).

Nadawanie i odbiór sygnałów w standardzie RS232

Nadawanie i odbiór sygnałów w standardzie RS232 odbywa się w sposób szeregowy, oddzielnie na dwóch liniach w kierunkach do i od urządzenia. W czasie braku transmisji sygnał na danej linii jest w stanie wysokim. Rozpoczęcie transmisji inicjowane jest przez tzw. bit startu będący **"0"** logicznym, który powinien trwać przez okres równy odwrotności prędkości transmisji, w naszym przypadku wynoszący 1/9600 [sekund]. Wszystkie następne informacje są

przesyłane z identycznym okresem trwania. Następnie nadawane są szeregowo dane począwszy od najmniej znaczącego bitu aż do bitu najbardziej znaczącego (**D0**-**D7**). Później występuje bit parzystości, będący operacją logiczną **XOR** na danych **D0**-**D7**. Bit parzystości jest opcjonalny i w przypadku niniejszego zadania nie występuje. Zakończenie transmisji sygnalizowane jest bitami stopu, w ilości zazwyczaj od 1 do 2. Przykład sygnału, zgodnego z wymaganiami zadania, przenoszącego kod **01010011** przedstawiony jest na poniższym rysunku.



Rys.1 Przykładowa transmisja kodu 01010011 przez RS-232

Informacje dodatkowe o standardzie RS232:
http://www.fizyka.umk.pl/~ptarg/labview/folie/RS232.pdf
http://pl.wikipedia.org/wiki/RS-232