INSTYTUT INFORMATYKI
Wydział IEiT AGH

Ćwiczenie laboratoryjne lub pokaz zdalny

Prosty mikroprocesor i system komputerowy w logice programowalnej

Nazwa kodowa: **`cpu`**. Wersja **`211002.0`**

dr inż. Ada Brzoza
ada.brzoza@agh.edu.pl

1. Wstęp

1.1 . Cel ćwiczenia

- Poznanie ogólnej zasady funkcjonowania mikroprocesorów,
- Poznanie przykładowego sposobu implementacji mikroprocesora o minimalnej funkcjonalności,
- Zapoznanie się z przykładem implementacji prostego systemu komputerowego (*Simple Computer System* - SCS)
- Sprawdzenie działania zaimplementowanego mikroprocesora i systemu komputerowego na rzeczywistej platformie sprzętowej.

1.2 . Wymagane wiadomości i umiejętności

- Znajomość idei działania mikroprocesora,
- Umiejętność obsługi płytki testowej Digilent ZYBO, dokumentacja: https://reference.digilentinc.com/reference/programmable-logic/zybo/reference-manual
- Umiejętność posługiwania się programem terminalowym.

1.3 . Dodatkowe umiejętności

Wykonanie ćwiczenia pozwoli Studentom na zdobycie także dodatkowych umiejętności:

- Poznanie elementów języka Verilog,
- Podstawy tworzenia konfiguracji dla układów FPGA w środowisku Xilinx Vivado 2016.2, zestawienie projektu, kompilacja, generowanie plików wynikowych, tworzenie i dostosowywanie modułu przy pomocy narzędzia IP Integrator.

1.4 . Wykorzystywane narzędzia

Ćwiczenie wykonywane jest na płycie testowej ZYBO produkcji Digilent. Płyta jest wyposażona w zintegrowany programator z interfejsem USB. Do tworzenia konfiguracji FPGA używane jest środowisko Xilinx Vivado w wersji 2016.2. Do wykonania ćwiczenia można posłużyć się wersją darmową Webpack tego środowiska (nie jest potrzebna wersja komercyjna).

Dodatkowo będziemy korzystać z programu terminalowego.

1.5 . Dostępne materiały

Do wykonania ćwiczenia niezbędne są następujące materiały:

- dokumentacja techniczna rdzenia mikroprocesora z pliku ***scs_trm.pdf***,
- projekt startowy dla środowiska Vivado.

2 . System testowy

Modułem nadrzędnym systemu testowego jest ***toplevel.v***. Po jego otwarciu warto pobieżnie zapoznać się z jego zawartością. Pewnym ułatwieniem może być opis w dalszej części niniejszego

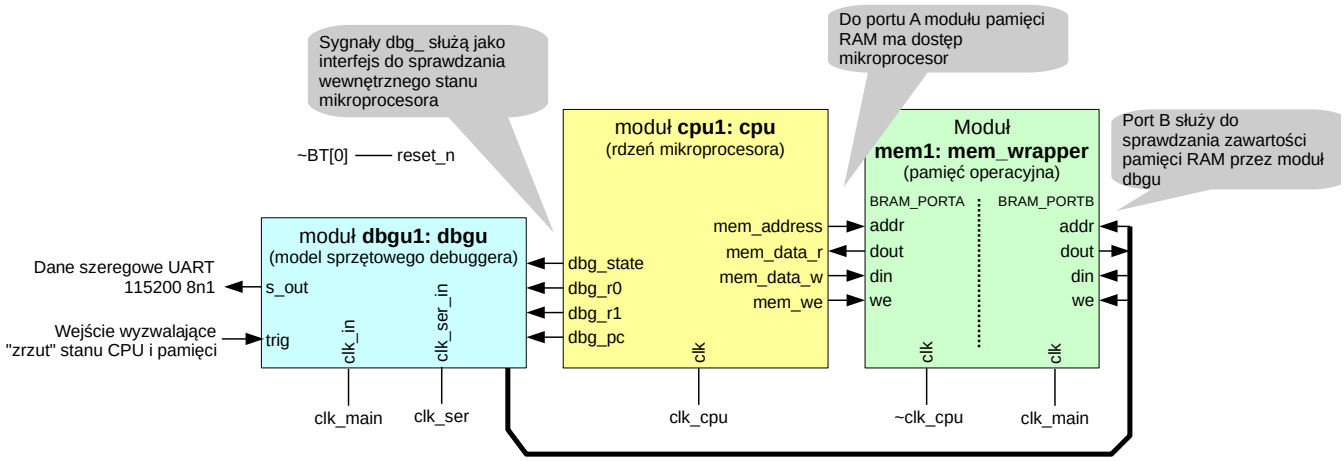
punktu.

2.1 . Ogólny schemat blokowy

W dostępnym projekcie startowym znajduje się rdzeń mikroprocesora oraz podstawowe układy umożliwiające jego działanie:

- blok pamięci operacyjnej,
- układy kondycjonujące sygnały wejściowe z przycisków i przełączników,
- generatory sygnałów zegarowych.

Uproszczony schemat blokowy systemu testowego z projektu bazowego został przedstawiony na poniższym rysunku.



Podstawowymi blokami systemu są mikroprocesor (moduł ***cpu1*** typu ***cpu***, plik ***cpu.v***) oraz jego pamięć operacyjna (moduł ***mem1***, ***mem_wrapper***).

Mikroprocesor posiada trzy typy interfejsów:

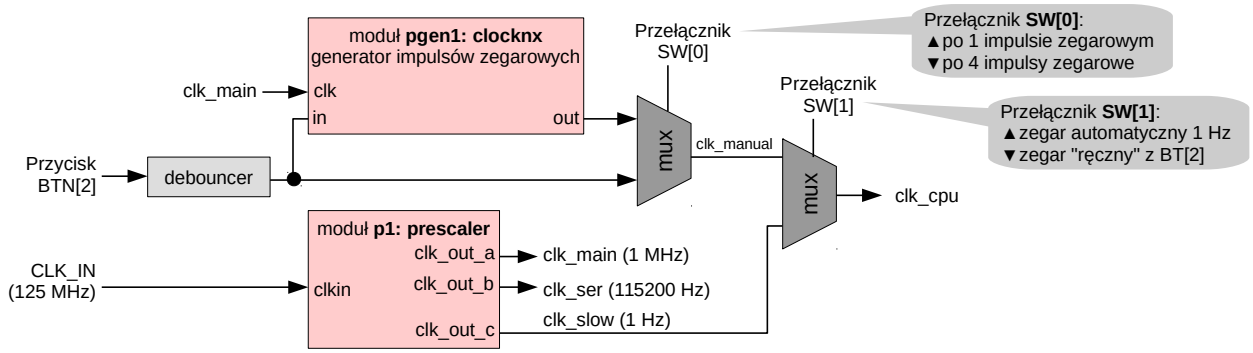
- do sterowania pracą: sygnał ***reset_n*** aktywowany stanem logicznym niskim,
- dostępu do pamięci:
  - ***mem_address*** - do adresowania aktywnej komórki pamięci RAM,
  - ***mem_data_r*** - do odczytu danych z pamięci,
  - ***mem_data_w*** - do zapisu danych do pamięci,
  - ***mem_we*** - wyjście zezwolenia na zapis do pamięci.
- do debugowania:
  - ***dbg_state*** - informujący o aktywnym etapie wykonywania pojedynczej instrukcji (każda instrukcja wykonywana jest w 4 cyklach sygnału zegarowego clk),
  - ***dbg_r0, dbg_r1, dbg_pc*** - informujące o aktualnej zawartości rejestrów R0, R1 i PC (licznika programu).

Moduł pamięci operacyjnej zbudowany jest w oparciu o blok pamięci RAM o dostępie

dwuportowym. Oznacza to, że do jednego obszaru pamięci mamy praktycznie niezależny dostęp z dwóch interfejsów oznaczonych na rysunku jako porty BRAM_PORTA i BRAM_PORTB (BRAM od *Block Random Access Memory*). Port A w przykładowym systemie zarezerwowany jest dla mikroprocesora, natomiast port B służy do obserwowania zawartości wybranej komórki pamięci RAM.

2.2 . Zarządzanie sygnałami zegarowymi

Schemat podsystemu dystrybucji sygnałów zegarowych w projekcie testowym został przedstawiony i pokrótce opisany na poniższym rysunku.



Do wyboru aktywnego źródła sygnału zegarowego służą przełączniki SW[0] i SW[1].

- Jeśli przełącznik SW[1] ustawiony jest w pozycji „w dół”, wówczas sygnał zegarowy podajemy ręcznie, wciskając przycisk BTN[2].
- Ustawienie przełącznika SW[1] na pozycję „do góry” spowoduje automatyczne generowanie powolnego sygnału zegarowego o częstotliwości 1 Hz.
- Jeśli ustawimy przełącznik SW[0] w pozycji „w dół”, wówczas ręczny sygnał zegarowy będzie podawany po 4 impulsy za każdym wciśnięciem BTN[2], co ułatwia obserwację pracy mikroprocesora wykonującego pełne **cykle instrukcji**.
- Jeśli ustawimy przełącznik SW[0] w pozycji „w górę”, wówczas ręczny sygnał zegarowy będzie podawany po jednym impulsie za każdym wciśnięciem BTN[2]. Wtedy można obserwować, co dzieje się w mikroprocesorze podczas wykonywania poszczególnych **cykli zegarowych**.

2.3 . Program testowy

W pamięci RAM systemu znajduje się prosty program testowy. Został on przedstawiony w poniższej tabeli.

Adres	Instrukcja	Kod bin.	Kod hex.
00	mov r0, #0	1000 0000	80
01	mov r1, #0	1001 0000	90
02	mov r0, #1	1000 0001	81
03	mov r1, #2	1001 0010	92
04	mov r0, r1	1010 0000	A0
05	jmp #0	0000 0000	00

Kody operacji (kolumny „Kod bin.” i „Kod hex.”) znajdują się w kolejnych komórkach pamięci RAM o adresach wyszczególnionych w lewej kolumnie. Wyjaśnienie mnemoników i kodowania poszczególnych instrukcji można znaleźć w dokumentacji mikroprocesora (***cpu_dok.pdf***).

Program testowy do wykonania musi zostać umieszczony w pliku o rozszerzeniu ***.coe*** (z odpowiednią składnią) i wskazany w edytorze schematów blokowych środowiska Vivado. Bardziej szczegółowy opis procedury dodawania i uaktualniania kodu programu znajduje się w dalszej części instrukcji.

3 . Wykonanie ćwiczenia

3.1 . Sprawdzenie podłączenia sprzętu

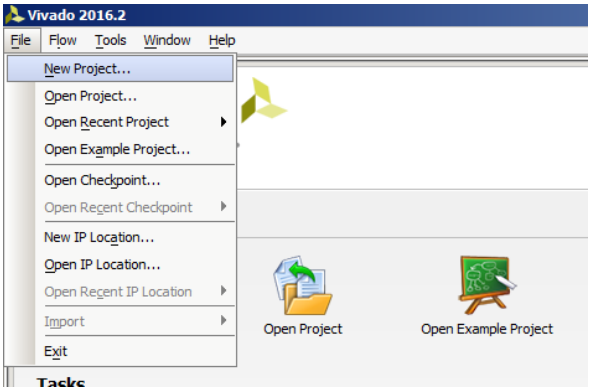
Płytka testowa Zybo powinna być podłączona do komputera PC dwoma przewodami:

1. Przewodem USB-micro ze złącza PROG/UART do gniazda USB w komputerze PC
2. Przewodem asynchronicznego łącza szeregowego pomiędzy złączem JE a przejściówką USB-RS232(TTL) włożoną do gniazda USB komputera PC. Wtyczka i gniazdo JE powinny mieć naniesiony znacznik informujący o kierunku włożenia wtyczki. Jeśli znaczniki nie zgadzają się, nie należy włączać płytki ZYBO, tylko zgłosić sytuację osobie prowadzącej zajęcia.

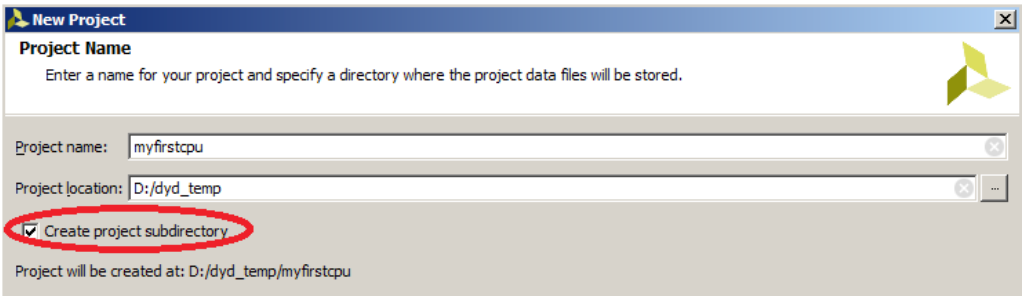
3.2 . Utworzenie systemu mikroprocesorowego w FPGA

Zaczynamy od pobrania kodów źródłowych modułów opisanych w języku opisu sprzętu Verilog. Zostały one dołączone do materiałów do ćwiczenia na stronie przedmiotu Technika Mikroprocesorowa. Następnie otwieramy środowisko **Vivado 2016.2**.

1. W ekranie powitalnym z menu głównego Vivado wybieramy ***File → New Project...***

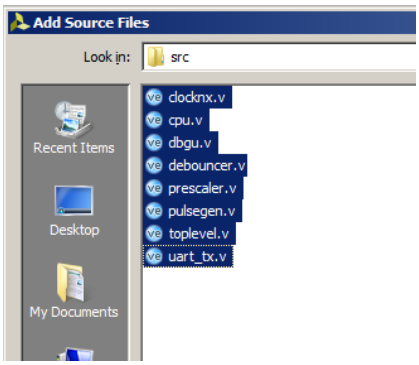


2. Pojawi się kreator *New Project*. W pierwszym ekranie przechodzimy od razu dalej przyciskiem ***Next***.
3. W drugim oknie kreatora wybieramy ścieżkę i nazwę projektu. Warto zaznaczyć pole ***Create project subdirectory***.

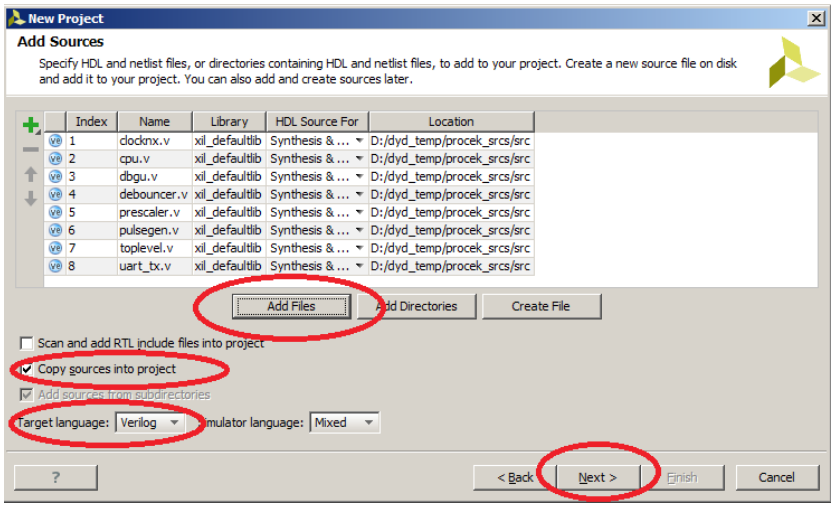


4. W kolejnym oknie wybieramy typ projektu: tutaj wybierzemy podstawowy typ, czyli ***RTL Project***. Dodatkowo pilnujemy, aby nie była zaznaczona opcja ***Do not specify sources at this time*** – pozwoli to nam dodać gotowe pliki źródłowe na etapie tworzenia projektu. Klikamy ***Next***.

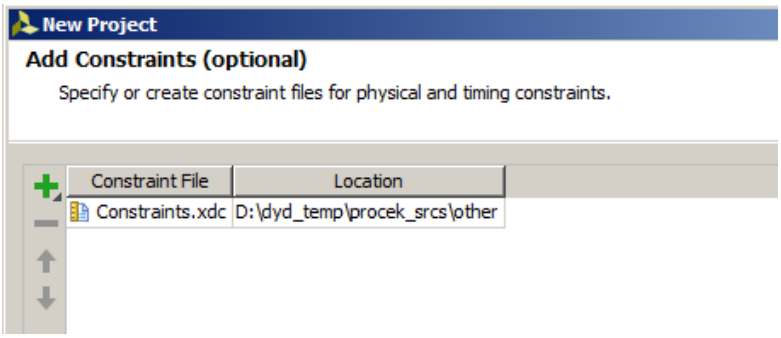
5. W kolejnej części kreatora możemy dodać gotowe pliki źródłowe. Klikamy przycisk ***Add Files*** i wskazujemy wszystkie pliki z podkatalogu ***src***. Zatwierdzamy ***OK***.



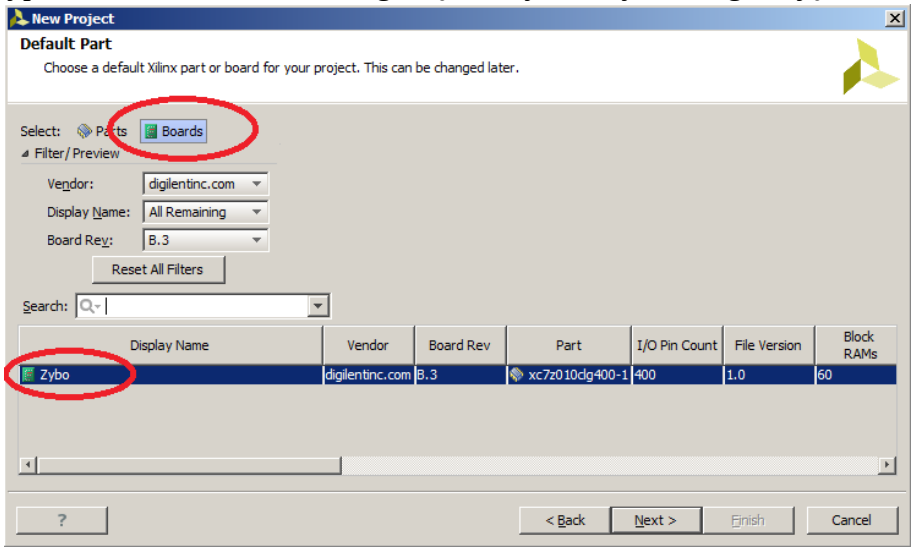
6. Sprawdzamy, aby zaznaczona była opcja ***Copy sources into project***. Oprócz tego wybieramy domyślny język (***Target language***): ***Verilog***, ***Simulator language*** pozostawiamy na ***Mixed***. Przechodzimy dalej przyciskiem ***Next***.



7. Ekran kreatora *Add Existing IP* na razie pomijamy przechodząc dalej przyciskiem ***Next***.
8. Następnie pojawi się ekran kreatora *Add Constraints (optional)*. Również wciskamy przycisk ***Add Files*** i wskazujemy plik ***Constraints.xdc*** z podkatalogu ***other***. Przechodzimy dalej przyciskiem ***Next***.

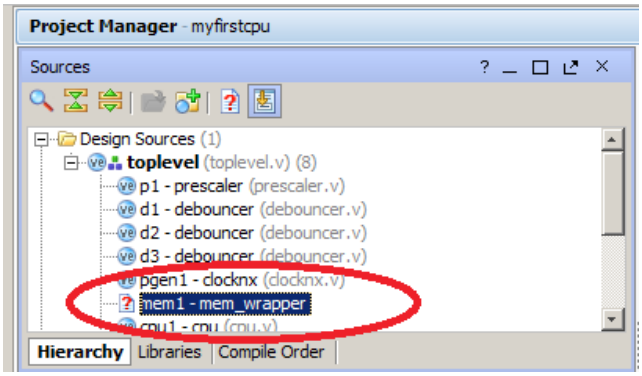


9. W ekranie *Default Part* klikamy na zakładkę ***Boards***, a następnie wybieramy ***Board Rev: B.3*** i wskazujemy ***Zybo***. Dzięki temu etapowi środowisko Vivado zostanie poinformowane m.in. o typie układu FPGA, dla którego będziemy tworzyć konfigurację.

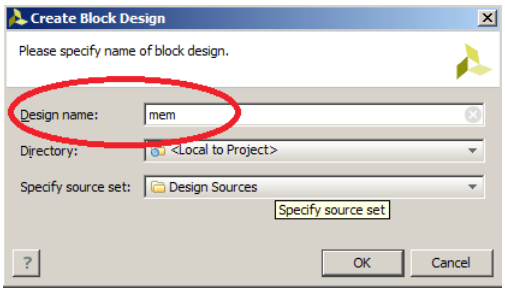


10. Ostatni krok kreatora nowego projektu to podsumowanie. Zatwierdzamy je przyciskiem ***Finish***.

11. Pojawi się główne okno środowiska Vivado. W części *Project Manager* w zakładce *Hierarchy* możemy dostrzec, że przy elemencie *mem1* typu *mem_wrapper* pojawił się znak zapytania – tego modułu brakuje w projekcie i jest to pamięć, którą wygenerujemy przy pomocy podsystemu *IP Integrator* środowiska Vivado.

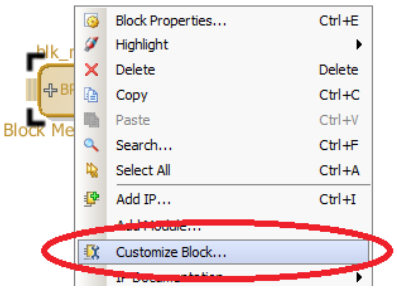


12. W polu ***Flow Navigator*** po lewej stronie okna głównego pod *IP Integrator* klikamy ***Create Block Design***. W polu *Design name* wpisujemy ***mem*** i zatwierdzamy przyciskiem *OK*.

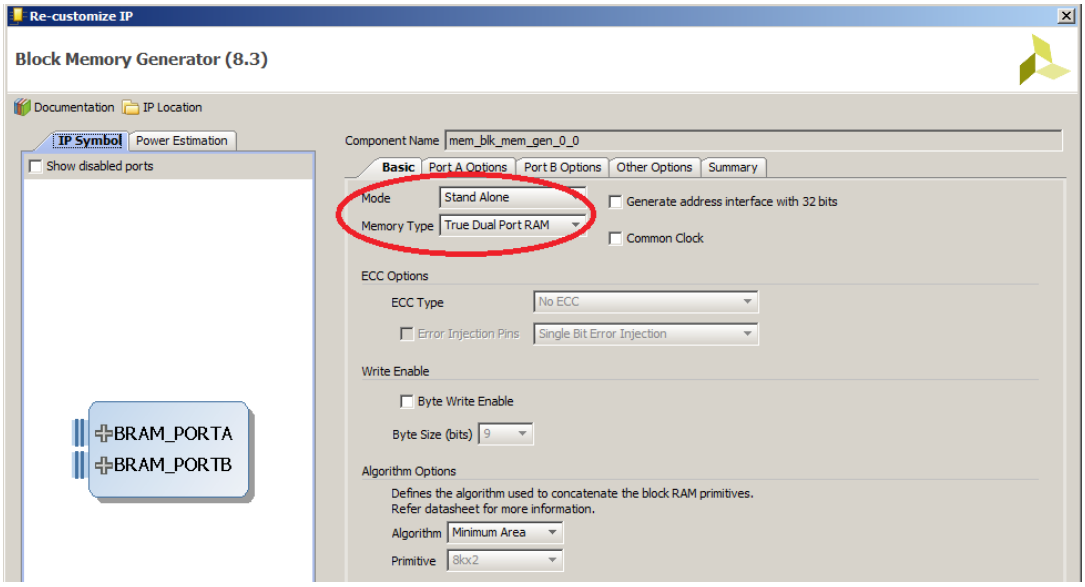


13. Otworzy się podsystem tworzenia schematu blokowego. Na razie obszar rysowania jest pusty. W oknie edycji schematu *Diagram* wskazujemy ikonkę ***Add IP***.

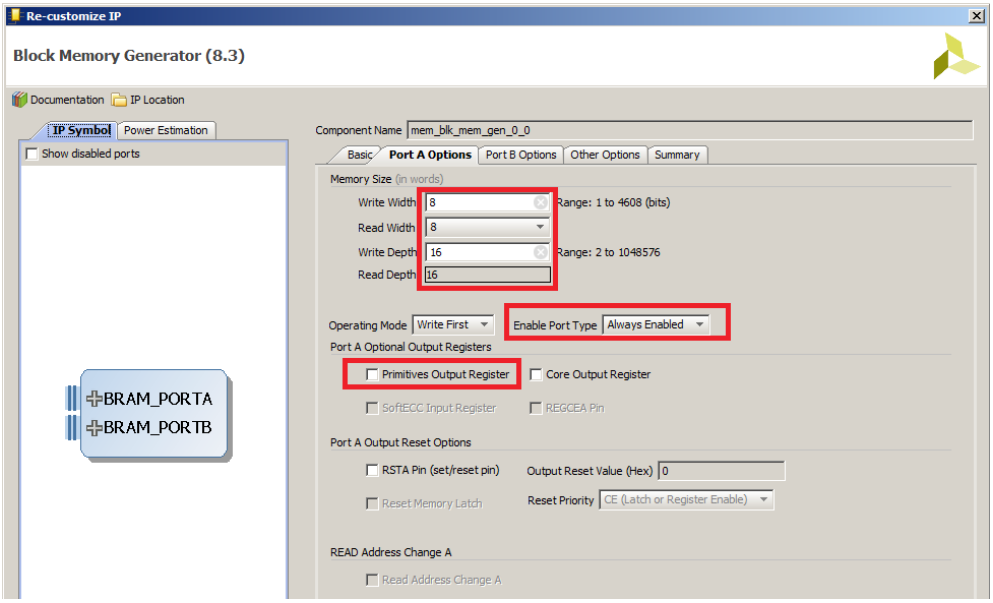
14. Na liście dostępnych modułów odnajdujemy element ***Block Memory Generator***. Klikamy go dwa razy i czekamy na dodanie do schematu blokowego.
15. Zaznaczamy wygenerowany blok lewym przyciskiem myszy, a następnie prawym przyciskiem wywołujemy jego menu kontekstowe, z którego wybieramy ***Customize Block...***



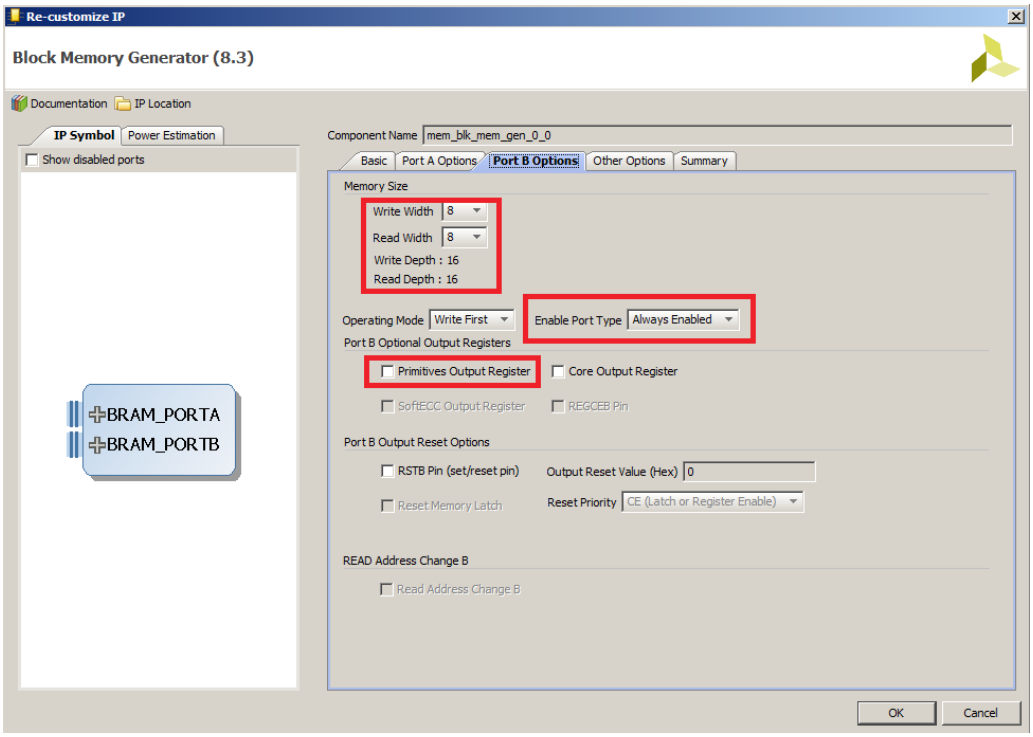
16. W zakładce *Basic* wybieramy tryb ***Stand Alone*** oraz typ pamięci: ***True Dual Port RAM***. Widzimy, że symbol bloku IP po lewej stronie okna zmienia się adekwatnie do wprowadzanych ustawień. Przechodzimy do zakładki ***Port A Options***.



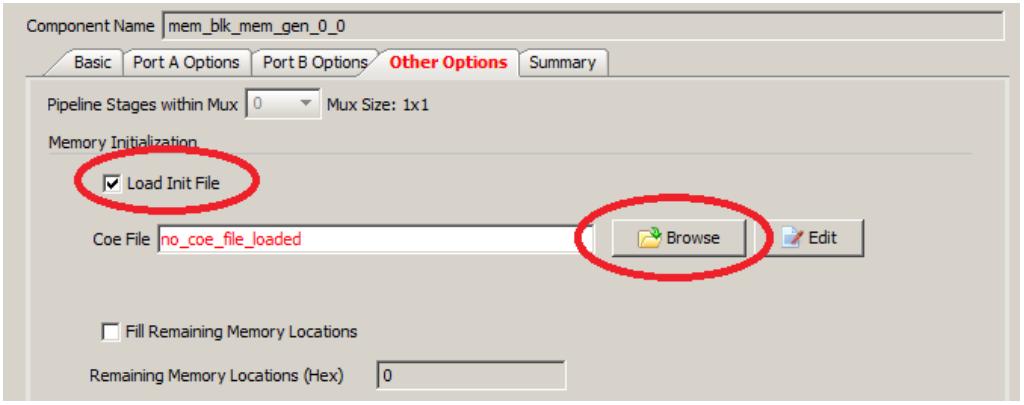
17. W zakładce *Port A Options* zmieniamy opisane dalej parametry.
    1. *Write Width* (szerokość zapisywanego słowa) ustawiamy na **8 bitów**,
    2. *Read Width* powinien zmienić się automatycznie również na **8 bitów**.
    3. *Write Depth* (ilość słów pamięci) zmieniamy na **16**.
    4. *Read Depth* również zmieni się adekwatnie.
    5. *Operating mode* pozostawiamy na ***Write First***
    6. *Enable Port Type* ustawiamy na ***Always Enabled***
    7. **Odznaczamy** opcję ***Primitives Output Register***
    8. Przechodzimy do zakładki ***Port B Options***



18. W zakładce Port B Options pozostało:
    1. Ustawić *Enable Port Type* na ***Always Enabled***
    2. **Odznaczyć** opcję ***Primitives Output Register***
    3. Przejść do zakładki ***Other Options***

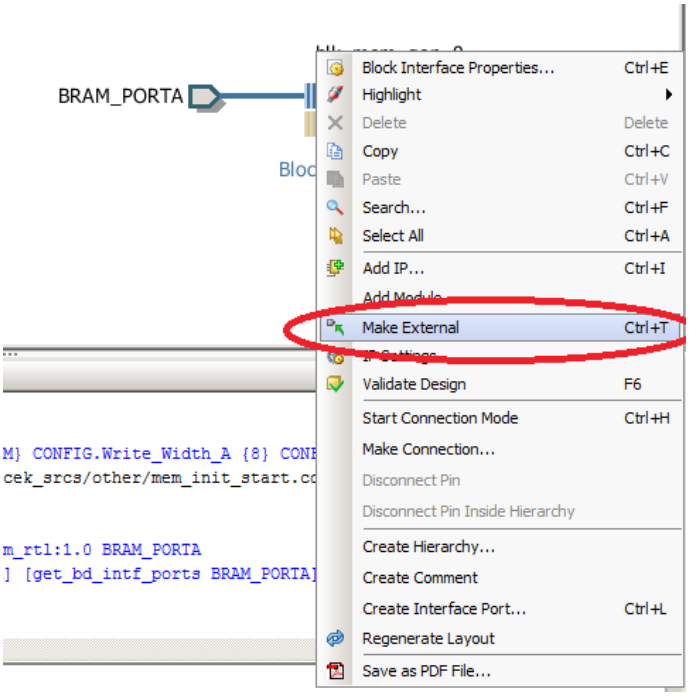


19. W zakładce *Other Options* wybieramy opcję ***Load Init File*** i po wciśnięciu przycisku ***Browse*** wskazujemy plik z testową zawartością pamięci naszego systemu mikroprocesorowego, w naszym przypadku ***mem_init_start.coe*** z podkatalogu ***other***.

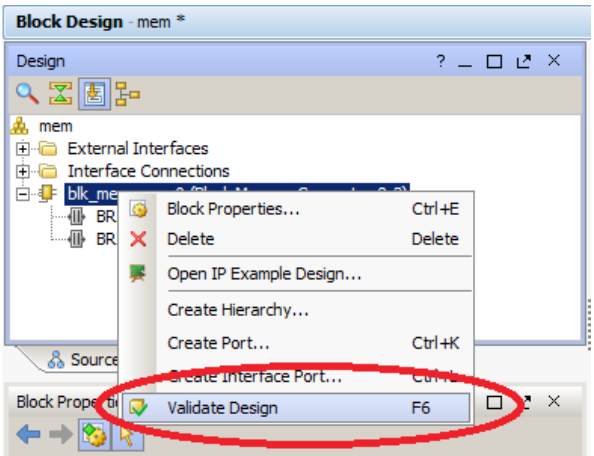


20. Po wybraniu pliku .coe zatwierdzamy ustawienia bloku IP klikając ***OK*** w oknie *Re-customize IP*.

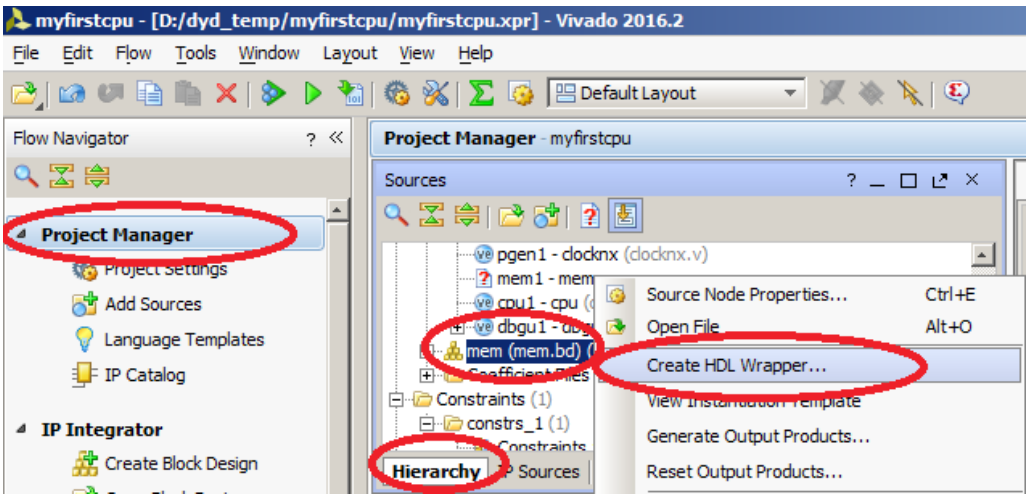
21. Powróciliśmy do edytora schematu blokowego. Klikamy w nim prawym klawiszem na jeden z portów bloku IP stanowiącego pamięć operacyjną naszego systemu mikroprocesorowego i z menu kontekstowego wybieramy ***Make External***. To samo robimy dla drugiego portu pamięci. Zapisujemy schemat blokowy skrótem klawiszowym Ctrl+S.



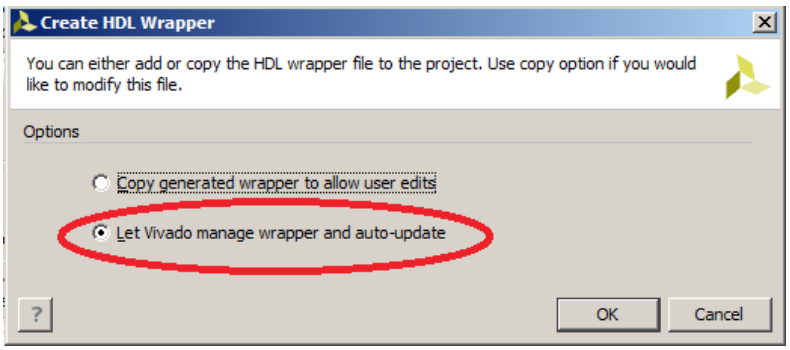
22. W obszarze *Design* edytora *Block Design* klikamy prawym przyciskiem myszy na ***blk_mem_gen_0*** i, z menu kontekstowego, wybieramy ***Validate Design***. Jeśli sprawdzenie nie zakończyło się błędami przechodzimy dalej.



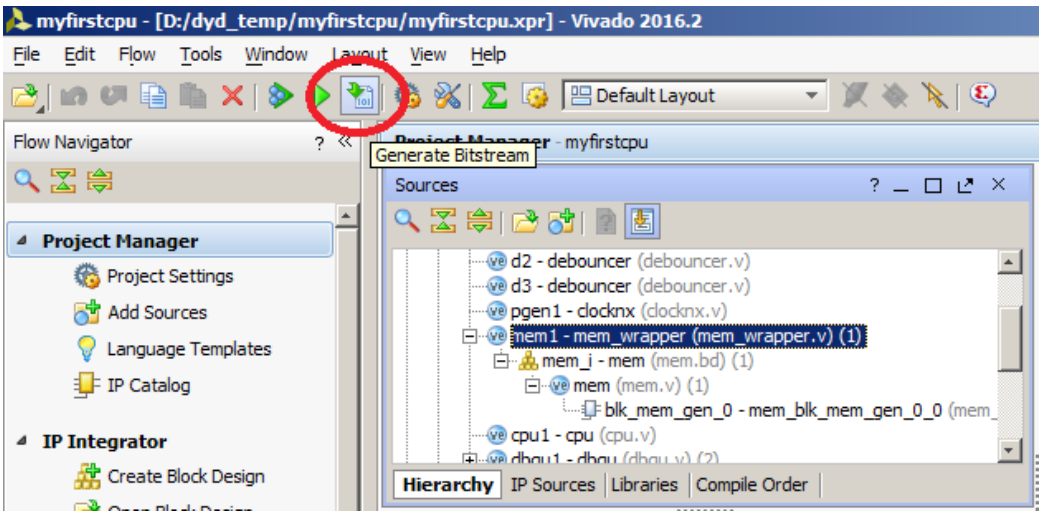
23. Przechodzimy do *Project Manager* i w polu *Sources* klikamy prawym klawiszem na element *mem*. Z menu kontekstowego wybieramy ***Create HDL Wrapper...***



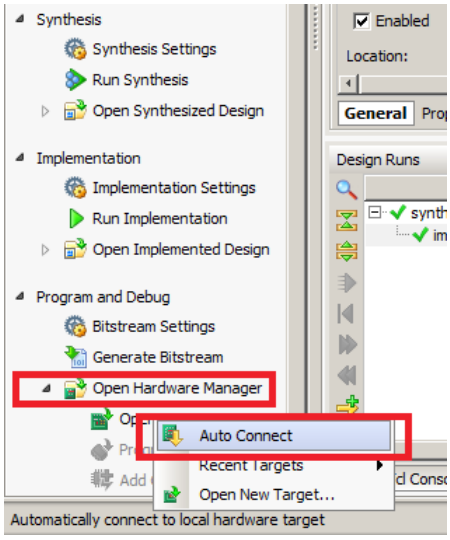
24. W okienku dialogowym *Create HDL Wrapper* zalecane jest wybranie opcji *Let Vivado manage wrapper and auto-update*.



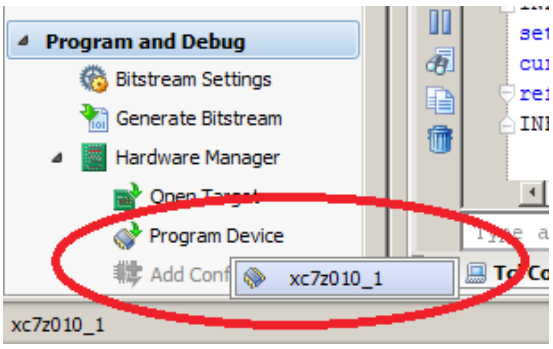
25. W obszarze *Sources* po rozwinięciu hierarchii *mem1* powinniśmy zobaczyć podobny widok jak na poniższym rysunku. Oznacza to, że właśnie wygenerowany moduł pamięci został prwidłowo rozpoznany przez środowisko Vivado. Jeśli nie widzimy błędów – klikamy ikonkę ***Generate Bitstream***.



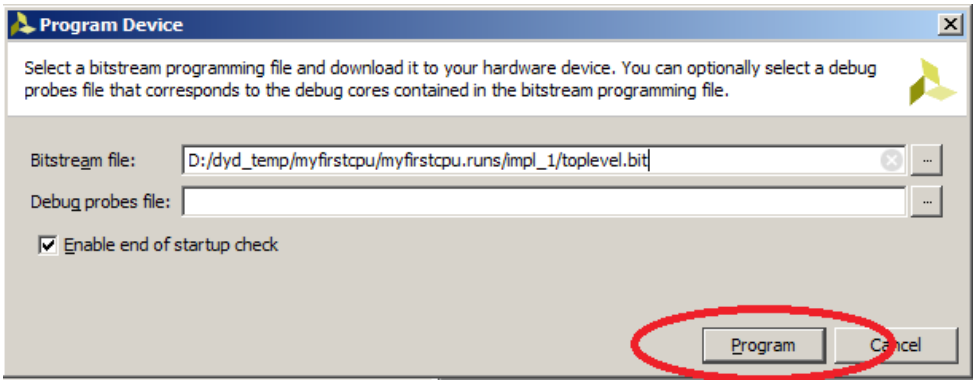
26. Jeśli środowisko Vivado zapyta, czy zapisać projekt przed kompilacją, oczywiście potwierdzamy to przyciskiem ***Save***. W oknie dialogowym z pytaniem, czy najpierw Vivado ma wykonać syntezę i implementację projektu, odpowiadamy ***Yes***.
27. Wszystkie etapy kompilacji mogą długo trwać, co jest normalne i w dużej mierze uzależnione od zasobów i mocy obliczeniowej komputera, na którym pracujemy. Można teraz upewnić się, czy płytka Zybo na pewno jest podłączona do złącza USB komputera, na którym pracujemy oraz, czy kabel dodatkowego łącza szeregowego jest poprawnie podłączony do płytki. **W razie jakichkolwiek wątpliwości należy skonsultować konfigurację sprzętową z osobą prowadzącą zajęcia**.
28. Po zakończeniu kompilacji i generowania plików wynikowych może pojawić się okno dialogowe *Bitstream Generation Completed*. Na tym etapie nie korzystamy z żadnych oferowanych w nim opcji, więc możemy wcisnąć przycisk *Cancel* lub zamknąć je klawiszem ESC.
29. Na samym dole lewej części okna środowiska Vivado rozwijamy listę *Open Hardware Manager*, a następnie klikamy ***Open Target*** i ***Auto Connect***.



30. Z tego samego obszaru wybieramy Program Device i xc7z010_1 (jedyna dostępna opcja).



31. W oknie dialogowym Program Device pozostajemy przy domyślnie zasugerowanym pliku .bit i wciskamy przycisk Program.



3.3 . Nawiązanie komunikacji przez port szeregowy

1. Upewniamy się, że kabel dodatkowego łącza szeregowego został poprawnie podłączony do gniazda JE na płytce ZYBO (zgadza się położenie znaczników o tym samym kolorze), a przejściówka z USB na port szeregowy znajduje się w złączu USB komputera na stole roboczym.
2. Sprawdzamy, który port szeregowy został wygenerowany dla przejściówki USB-RS232-TTL. Port szeregowy będzie reprezentowany w systemie jako urządzenie **`/dev/ttyUSBn`** lub **`/dev/ttyACMn`**, gdzie ***n*** jest liczbą całkowitą 0 lub większą. Listę dostępnych portów szeregowych z przejściówek USB-serial możemy uzyskać np. przy pomocy polecenia
   `ls /dev/ttyUSB*`
   lub
   `ls /dev/ttyACM*`
3. Otwieramy program terminalowy, np. ***putty*** i łączymy się z wykrytym w poprzednim punkcie portem szeregowym generowanym przez przejściówkę USB. Poprawne parametry transmisji to: **115200 baud, 8 bitów danych, 1 bit stopu, brak bitu parzystości**.
4. Po wyborze parametrów połączenia w putty wciskamy przycisk ***Connect***. Połącznie powinno teraz zostać nawiązane.
5. Z zaprogramowanym układem FPGA, wciskamy na płytce Zybo przycisk BTN[2] przy przełącznikach SW[3:0] ustawionych np. na 0101 (od lewej: „dół-góra-dół-góra”).
6. Jeśli wszystko wykonane zostało poprawnie, na terminalu powinniśmy po każdym wciśnięciu przycisku BTN[2] ujrzeć kolejną linijkę np.

   c=1 r0=1 r1=0 pc=3 A=3 R=81 W=00 M: 80 90 81 92 A0 00 00 00 00 00 00 00 00 00 00 00

3.4 . Obserwacja działania programu testowego

Po zaprogramowaniu układu FPGA przechodzimy do obserwacji prostego programu domyślnego.

1. Włączamy „ręczne” podawanie sygnału zegarowego
2. Zerujemy system przyciskiem BTN[0].

3. Obserwujemy wykonanie poszczególnych instrukcji wciskając przycisk BTN[2]. Przy ustawieniach przełączników SW[3:0] na 0100 (od lewej „dół-góra-dół-dół”), mikroprocesor będzie wykonywał po jednej pełnej instrukcji, ponieważ jedno wciśnięcie BTN[2] odpowiada podaniu do mikroprocesora 4 okresów sygnału zegarowego.
4. Obserwujemy zawartość rejestrów: PC, R0, R1 oraz danych odczytywanych z pamięci. Porównujemy dane odczytane z pamięci z zawartością pliku ***mem_init_start.coe***. Zawartość poszczególnych fragmentów każdej linijki obserwowanej na terminalu jest następująca:
   - c: wewnętrzny cykl mikroprocesora. Jeden taki cykl odpowiada jednemu cyklowi (okresowi) sygnału zegarowego podanego do mikroprocesora. Cykl instrukcji składa się z 4 cykli zegarowych,
   - r0, r1, pc: zawartość głównych rejestrów mikroprocesora w formacie szesnastkowym,
   - A: dane na magistrali adresowej pamięci,
   - W: dane na magistrali zapisu z procesora do pamięci,
   - R: dane na magistrali odczytu z pamięci do procesora,
   - M: zrzut wszystkich komórek pamięci (jest ich 16).
5. Możemy bez trudu zaobserwować, co dzieje się w poszczególnych stanach mikroprocesora w czasie wykonywania pojedynczej instrukcji i odnieść swoje obserwacje do treści pliku ***cpu.v***.

3.5 . Pisanie własnego programu

Korzystając z nabytych do tej pory umiejętności oraz z dokumentacji mikroprocesora (***cpu_dok.pdf***) należy samodzielnie napisać (w kodzie maszynowym) program, który będzie wykonywał wymienione poniżej czynności. Kod maszynowy wraz z całą zawartością pamięci umieszczamy w pliku o rozszerzeniu .coe. Najwygodniej będzie wykonać kopię pliku mem_init_start.coe i ją modyfikować przy pomocy dowolnego edytora tekstowego.

Zaczynamy od wpisania liczby ***0x05*** do komórki pamięci o adresie **13** – będzie to jedna z danych wejściowych do obliczeń w programie. Program ma wykonać następujące czynności:

1. Wyczyścić zawartość rejestrów R0 i R1 wpisując do nich wartość 0,
2. Wyczyścić zawartość komórki pamięci o adresie 14 wpisując do niej wartość 0,
3. Do rejestru R0 wpisać liczbę 0x6
4. Dodać do rejestru R0 wartość przechowywaną w pamięci RAM pod adresem 13,
5. Umieścić zawartość rejestru R0 w pamięci RAM w komórce o adresie 14,
6. Zapętlić wykonywanie programu.

Program należy uruchomić i zaprezentować jego działanie prowadzącemu zajęcia.

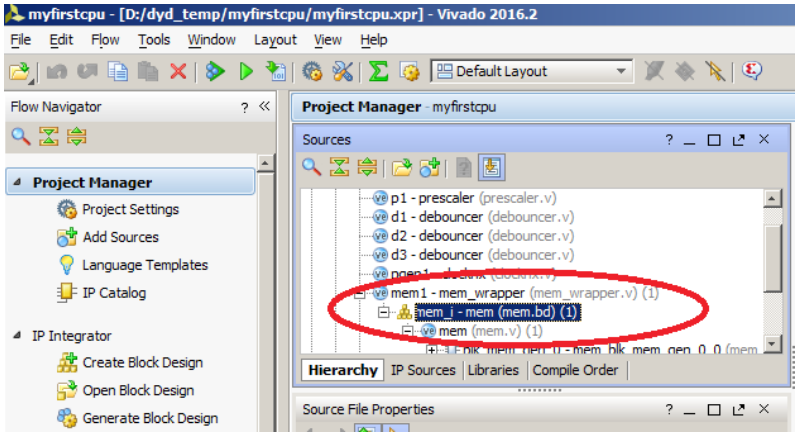
Uwaga: aby nowy program znalazł się w systemie mikroprocesorowym w układzie FPGA należy wykonać poniższe kroki. Procedura została opisana w dwóch alternatywnych opcjach – wybieramy jeden ze sposobów (najczęściej jeśli pierwszy nie zadziała, to możemy spróbować sposobu drugiego).

3.5.1 . Sposób pierwszy, „oficjalny”

1. Przede wszystkim trzeba mieć przygotowany plik o rozszerzeniu .coe z zawartością

pamięci.

2. W głównym oknie Vivado uaktywnić ***Project Manager***, a następnie, w sekcji *Sources* → *Hierarchy* kliknąć dwukrotnie na element reprezentujący schemat blokowy z pamięcią.



3. Po otwarciu edytora schematów blokowych należy przejść do trybu dostosowania (***customize***) elementu reprezentującego pamięć i tam wskazać nowy plik COE, analogicznie jak przy wstępnej konfiguracji bloku pamięci (część 3.2 niniejszej instrukcji), tym razem jednak wskazując **nowo utworzony plik .coe**.
4. Po zakończeniu wskazywania nowego pliku COE jeszcze raz wykonujemy kompilację projektu i generowanie pliku wynikowego (*bitstream*) i ponownie programujemy FPGA na płytce Zybo.

3.5.2 . Sposób alternatywny

Jeśli nie działa pierwszy sposób (widzimy poprzednią zawartość pamięci sprzed edycji), to może oznaczać, że środowisko Vivado nie wykonało konwersji nowego pliku wejściowego .coe na plik .mif służący dalej do programowania zawartości pamięci.

1. W menedżerze plików otwieramy katalog, w którym znajduje się nasz projekt dla Vivado.
2. Wyszukujemy pliki o rozszerzeniu ***.mif*** (od *memory initialization file*). Powinniśmy znaleźć jeden plik, np. o nazwie ***mem_blk_mem_gen_0_0.mif***. Otwieramy ten plik przy pomocy edytora tekstowego, np. Programmer's Notepad. Edytujemy ten plik tak, aby zawierał nasz program i dane. Nie zmieniamy sposobu zapisu danych w pliku – musi on zostać taki, jak przy otwieraniu. Wprowadzamy i zapisujemy zmiany.
3. Aby środowisko Vivado zauważyło zmianę, należy wykonać drobną modyfikację któregoś pliku źródłowego .v przy pomocy edytora wbudowanego w Vivado. Wystarczy np. dodać i usunąć znak na końcu pustej linii kodu, a następnie zapisać plik i projekt.
4. Wykonujemy polecenie generowania plików wynikowych (*Generate Bitstream*) i programujemy FPGA.

3.6 . Utworzenie nowej instrukcji mikroprocesora

Rozszerzyć możliwości instrukcji dodawania w taki sposób, by można było wybrać, do którego rejestru (R0 czy R1) wpisywany będzie wynik. Wybór rejestru powinien odbywać się przez zmianę wartości bitu 4 kodu operacji.

Następnie dodać nową instrukcję wykonującą wybrane działanie. Może to być np. mnożenie, odejmowanie lub jeszcze inna operacja uzgodniona z prowadzącym zajęcia.

Prezentację działania nowej instrukcji należy wykonać na płytce testowej modyfikując odpowiednio program napisany w punkcie 3.6 i prezentując jego działanie.

4 . Co należy umieć i/lub co należy uwzględnić w sprawozdaniu

Wymienione w niniejszej sekcji zagadnienia należy opracować samodzielnie, a następnie dla każdego punktu odnieść się zwięźle w sprawozdaniu i/lub utrwalić zdobytą wiedzę i umiejętności zależnie od ogłoszonej przez osobę prowadzącą formy weryfikacji efektów uczenia dla tego zadania.

1. Rozróżnienie cykli zegarowych i cykli instrukcji w mikroprocesorze dydaktycznym z ćwiczenia *cpu* oraz w innych mikroprocesorach.
2. Klasyfikacja mikroprocesora, na którym wykonywane jest ćwiczenie oraz umiejętność uzasadnienia takiej klasyfikacji pod kątem:
   a) organizacji ścieżek danych: Harvard / von Neumann
   b) zestawu i kodowania instrukcji: RISC / CISC
   c) wspieranej szerokości bitowej jego najważniejszych elementów
3. Proszę omówić w kontekście wsparcia dla konkretnych klas instrukcji podstawowe możliwości mikroprocesora dydaktycznego w bazowej konfiguracji:
   a) klasa: instrukcje skoków – które rodzaje instrukcji skoków może wykonywać?
   b) klasa: instrukcje przesłań – które rodzaje przesłań wspiera, tj. pomiędzy którymi elementami? (np. czy rejestr-rejestr, czy pamięć-pamięć, czy rejestr-pamięć itd.)
   c) klasa: instrukcje przetwarzania danych – które rodzaje obliczeń może wykonywać? (np. które operacje arytmetyczne wspiera?)
4. Proszę wymienić i zwięźle opisać zastosowanie rejestrów wewnętrznych, w które jest wyposażony mikroprocesor, oraz rozmiaru (szerokości bitowej) tych rejestrów.
5. Proszę zwięźle omówić parametry wydajnościowe mikroprocesora dydaktycznego:
   a) czy ma wsparcie dla przetwarzania potokowego?
   b) jaka jest liczba cykli zegarowych przypadających na wykonanie jednej instrukcji i czy liczba ta jest stała dla każdej instrukcji, czy zmienia się zależnie od wykonywanej instrukcji?
   c) jak nazywają się poszczególne etapy, z których składa się wykonanie jednej instrukcji?