Specyfikacja projektu, wersja 2022/1.04

Ostatnia modyfikacja: 17.05.2022

Spis treści

Uwaga: Wszystkie działania oznaczone słowem kluczowym TODO w instrukcjach i skryptach należy udokumentować w raporcie projektu.

1 Cel projektu

Celem projektu jest opracowanie prostego układu typu System-On-Chip z procesorem RISC-V, w tym:

- napisanie programu dla procesora i sprawdzenie poprawności jego działania przez symulacje funkcjonalne,
- opracowanie topografii (layout'u) procesora, weryfikacja jego parametrów

2 Oczekiwane wyniki

W wyniku projektu ma powstać (na ocenę 3.0):

A) raport z projektu (udokumentowane działania TODO),

B) program na procesor RISC-V działający zgodnie z wymaganiami,

C) plik GDS zawierający layout procesora,

D) netlista bramkowa (wynik syntezy i Place&Route),

E) plik SDF zawierający opóźnienia dla netlisty,

F) report z generacji drzewa zegara,

G) finalne wyniki analizy czasowej (sign-off, streszczenie, bez błędów)

H) raport DRC z programu INNOVUS (bez błędów),

I) raport LVS z programu INNOVUS (bez błędów),

Dodatkowo na ocenę 4.0 należy:

J) zautomatyzować przebieg projektu (synteza i P&R mają być wykonywane po uruchomieniu pojedynczej komendy),

K) wykonać symulację zsyntezowanej netlisty z opóźnieniami z pliku SDF, z poprawnym wynikiem.

Dodatkowo na ocenę 5.0 należy:

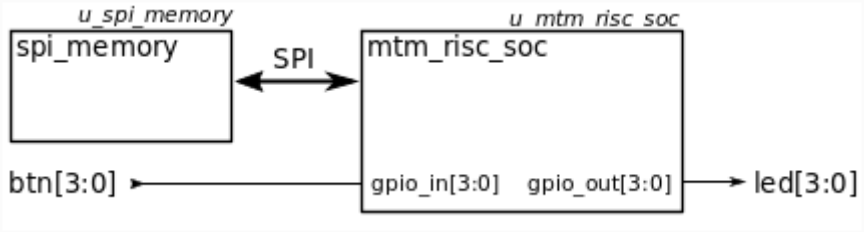
L) wykonać analizę poboru mocy układu dla opracowanego programu i przedstawić wynik.

Wystawiona ocena zależy od poprawnego wykonania wymaganych zadań.

Uwaga: BONUS - w przypadku wykonania projektu w terminie do 30 czerwca 2022 oraz uzyskania oceny co najmniej 4.0, ocena zostanie podniesiona o 0.5 (4.0 → 4.5, 4.5→5.0). Finalny termin: 15 września 2022.

3 Informacje wstępne.

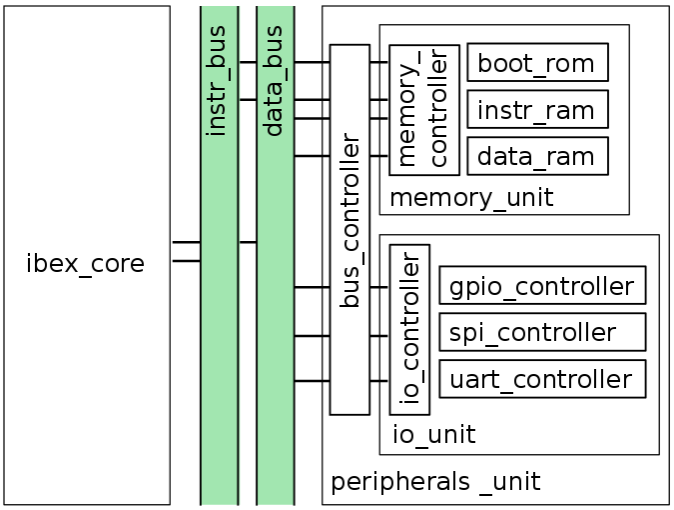
Na potrzeby projektu został opracowany przez p. Pawła Skrzypca model prostego układu SoC (będziemy nazywać go tu mikrokontrolerem) zawierającego rdzeń procesora RISC-V (Ibex) oraz peryferia GPIO. Schemat symulowanego środowiska przedstawia rysunek poniżej.



Program użytkownika jest umieszczany w zewnętrznej pamięci, komunikującej się z mikrokontrolerem przez interfejs SPI. Po uruchomieniu procesora program jest kopiowany do pamięci wewnętrznej mikrokontrolera i wykonywany.

Mikrokontroler posiada 4 uniwersalne wejścia i 4 uniwersalne wyjścia, które mogą być obsługiwane przez użytkownika.

Strukturę wewnętrzną samego mikrokontrolera przedstawia rysunek poniżej.



Rdzeń procesora (*ibex_core*) jest podłączony do dwóch magistral: instrukcji (*instr_bus*) i danych *(data_bus*). Procesor czyta kolejne instrukcje z pamięci przez magistralę instrukcji. Instrukcje mogą być dostarczane z dwóch pamięci:

- modułu *boot_rom*, jest to pamięć stała (ROM), zawierająca program wykonywany po starcie systemu. Program ten przepisuje zawartość zewnętrznej pamięci SPI do pamięci wewnętrznej *instr_ram*,
- modułu *instr_ram*; jest to zwykła pamięć instrukcji procesora, używana w trakcie wykonywania programu użytkownika.

W projekcie, na etapie symulacji, używamy prostych modeli pamięci, nie związanych z żadną technologią.

Poprzez magistralę danych (*data_bus*) procesor może się komunikować z zewnętrznymi peryferiami. Kontroler SPI jest używany tylko przy boot'owaniu. W tym projekcie nie będziemy używać UART'a. Jedynym używanym peryferium będzie kontroler GPIO.

4 Repozytorium

Pliki źródłowe do projektu znajdują się w repozytorium git.

Repozytorium należy sklonować hierarchicznie (zawiera odniesienie do repozytorium IBEX'a).
Repozytorium można przeglądną online:
https://github.com/agh-riscv/mtm_ppcu_vlsi_riscv

TODO:

Sklonuj repozytorium projektu. Będziesz potrzebować do tego konta na github.com.

Na serwerze studenckim wygeneruj klucze ssh (terminal bash), bez hasła, podając email, którego używasz na githubie:

```
ssh-keygen -t ecdsa -b 521 -C youremailaddress@xyz.com
```

Zawartość pliku `~/.ssh/id_rsa.pub` skopiuj w odpowiednie miejsce w ustawieniach na koncie github (Settings -> SSH and GPG keys -> New SSH key).

W wybranym katalogu na serwerze studenckim sklonuj repozytorium:

```
git clone --recursive git@github.com:agh-riscv/mtm_ppcu_vlsi_riscv.git
```

Szczegółowe informacje nt struktury repozytorium znajdują się w pliku README w głównym katalogu.

5 Etapy projektu.

5.1 Program

5.1.1 Kompilacja

Struktura programistyczna dla procesora umieszczona jest w katalogu `sw`. W każdym używanym katalogu umieszczony jest plik konfiguracyjny `CmakeLists.txt` dla programu `cmake`. Program `cmake` stanowi nakładkę na `make`, tworzący pliki `Makefile` i wykonujący odpowiednie komendy `make` w całym drzewie. Więcej informacji można znaleźć na stronach:

https://www.gnu.org/software/make/manual/html_node/Introduction.html

https://cmake.org/cmake/help/latest/guide/tutorial/index.html

Sama kompilacja będzie przeprowadzana w osobnym drzewie katalogów o nazwie `build`.

Inicjalizację drzewa należy przeprowadzić w katalogu `sw` komedą: `cmake -H. -Bbuild`

Całe drzewo kompilujemy komendą: `cmake --build build`

5.1.2 Aplikacja użytkownika

Programy użytkownika należy tworzyć w katalogu `sw/app/src`. W pliku `main.c` znajduje się przykładowy program sterujący wyjściami GPIO procesora. Skompiluj program przykładowy i uruchom symulację, żeby zobaczyć jak działa całe środowisko (informacje o symulacji poniżej).

Wszystkie potrzebne funkcje znajdziesz w pliku źródłowym sterownika GPIO: `sw/drivers/src/gpio.c`

TODO: Należy napisać program, który po uruchomieniu procesora wyśle na kolejnych czterech wyjściach GPIO następujące sekwencje:

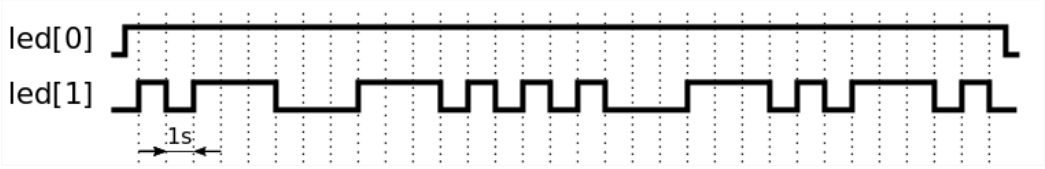
– wyjście *led[0]* = sygnał ważności sekwencji; sekwencja jest nadawana, gdy *led[0] ==1*

– wyjście *led[1] = „SOS” w kodzie Morse’a (nadawane tylko raz),*

– wyjście *led[2]* = twoje imię w kodzie Morse’a, bez polskich znaków,

– wyjście *led[3]* = twoje nazwisko w Morse’a, bez polskich znaków.

Sygnał krótki w kodzie Morse’a ma trwać 1 s ± 10%, sygnał długi: 3 s ± 10%, przerwa między sygnałami: 1 s ± 10%, przerwa między znakami: 3 s ± 10%. Nadawanie sygnału to logiczna 1 na konkretnym wyjściu. Przykładowo dla nadania sekwencji „ABC” przebiegi powinny wyglądać następująco:



Uwaga: należy założyć, że wywołanie funkcji `msleep(1000)` da w rzeczywistości 1 sekundę opóźnienia w wykonaniu programu. Na potrzeby symulacji opóźnienie jest zmniejszane do wybranej liczby taktów zegara (patrz plik `sw/misc/src/delay.c`). W symulacji używany jest sygnał zegara o częstotliwości 50 MHz. Nie należy tego zmieniać.

TODO: **Do raportu należy załączyć:**

- kod programu,
- zrzut ekranu z widocznymi przebiegami sygnałów wyjściowych *led[3:0]* mikrokontrolera,
- odpowiedzi na pytania:
  - ✓ Ile bajtów ma program bootloader'a? Odpowiedź uzasadnij wydrukiem zawartości odpowiedniego pliku `*.vmem`. Plików `*.vmem` szukaj w drzewie kompilacji.
  - ✓ Ile bajtów pamięci zajmuje opracowany przez Ciebie program? Odpowiedź odpowiednio udokumentuj.
  - ✓ Ile bajtów pamięci maksymalnie zajmuje stos procesora (podpowiedź: stosu należy szukać w górnych adresach pamięci danych). Odpowiedź udokumentuj.

5.1.3 Symulacje

Do uruchamiania symulacji służy skrypt *run.sh* w katalogu *./sim*. Skrypt na początku uruchamia kompilację części sprzętowej oraz skrypty służące do:

- translacji bootloadera do języka SystemVerilog (pamięć bootloadera jest mała i jest implementowana jako bramki),
- translacji programu użytkownika do modelu pamięci SPI.

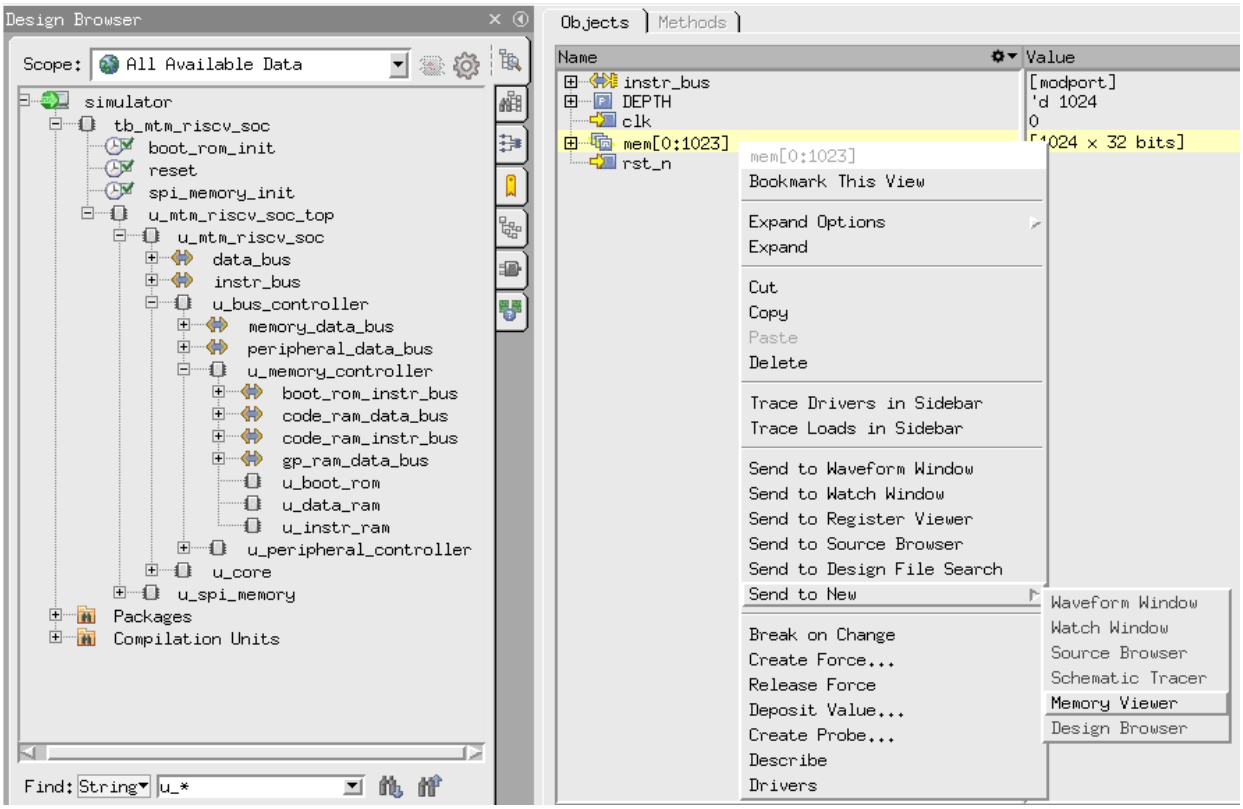
Skrypt umożliwia on włączenie trybu „debug" odpowiednią opcją (uruchomienie skryptu bez opcji wyświetli szczegółowe informacje na ten temat).

W trybie „debug" w trakcie symulacji robione są trzy zrzuty zawartości pamięci SPI, kodu i danych:

- po inicjalizacji w czasie 0,
- po zakończeniu sekwencji bootowania,
- na koniec symulacji.

Porównanie zawartości tych plików pomoże określić, jaki obszar pamięci jest używana na stos, i jaki na dane programu.

Poniżej przedstawiono sposób uruchamiania przeglądarki pamięci w środowisku SimVision, co może być przydatne podczas debugowania programu.



Zagadnienia implementacji i symulacji post-layout zostaną przedstawione w osobnych dokumentach.