Konštrukcia demonštračného CPU v FPGA čipe

Popis a ciele ročníkového projektu:

Tento ročníkový projekt je zameraný na prácu s FPGA čipom, konkrétne nami používaný iCEstick Evaluation Kit. Jedná sa o zariadenie, ktoré je možné s použitím programovacieho jazyku HDL (hardware description language) nakonfigurovať tak, aby sa správalo ako opísaný kus hardvéru. Naša úloha je rozdelená na štyri body:

1. Naučiť sa práci s FPGA: naučíme sa používať HDL menom Verilog, a toolchain Apio, ktorý nám umožní na Icestick naše Verilog kódy uploadnuť.
2. Pozrieť si existujúce implementácie demonštračných CPU na FPGA.
3. Navrhnúť vlastnú implementáciu nášeho CPU.
4. Dané CPU naprogramovať a nahrať na Icestick.

1. Naučenie sa práce s FPGA:

Prvý krok projektu je zoznámiť sa s verilogom, naučiť sa práci s ním a vyskúšať programovať jednoduché projekty. Ako hlavný zdroj som používal Youtube tutoriál od kanálu DigiKey menom Introduction to FPGA. Výsledky mojej práce sú dostupné na github repozitári ročníkového projektu. Pokrývajú základnú syntax verilogu a kombinačné priradenia, sekvenčnú logiku a využitie registrov cez always loops, využitie vnorených modulov a parametrov pre uľahčenie práce s navrhovaním, prácu s objemnejšou pamäťou, simuláciu testbenchov modulov pre účely debuggingu, a zároveň aj teóriu na tému metastability a ako sa jej vyhýbať. Toto všetko by pre naše účely malo byť postačujúce.

Inštalácia a použitie Apio

Používal som toolchain Apio, ktorý obsahuje nástroje na kompiláciu, testovanie a uploadovanie verilogových programov na FPGA. Pred používaním ho treba nainštalovať (prerekvizitou je mať nainštalovaný Python):

1. Apio nainštalujeme cez pip príkazom *pip install -U apio*
2. Následne treba nainštalovať dodatočné súbory príkazom *apio install –all*
3. Niektoré FPGA čipy potrebujú špeciálne nastavenia driverov: *serial* alebo *ftdi*. Icestick požaduje *ftdi* driver, preto ho musíme nastaviť príkazom *apio driver --ftdi-enable* ako administrátor. V okne, ktoré sa objaví, zvolíme pripojenie na Icestick, ktoré sa objaví pri jeho zapojení, a preinštalujeme driver podľa inštrukcií v termináli.

Apio nám odteraz umožní v priečinkoch našich projektov používať zopár užitočných príkazov:

- Každý projekt potrebuje súbor *apio.ini*. Apio ho dokáže automaticky zgenerovať príkazom *apio init -b icestick*, respektíve pokiaľ používame iné FPGA, musíme dosadiť jeho názov (ak je podporované, samozrejme).

- Príkazom *apio verify* overíme, či sa v kóde nenachádzajú chyby, a je užitočné ho pred kompiláciou spustiť.
- Príkazom *apio build* skompilujeme zdrojové kódy do súborov pripravených na upload.
- Príkazom *apio upload* skompilovaný projekt nahráme na pripojené FPGA. FPGA okamžite začne pracovať, keďže syntetizovaný obvod je aktívny stále, a netreba ho (dokonca to ani nie je možné) explicitne spúšťať.

Prehľad štruktúry verilog projektu

Základom projektu je verilogový program s koncovkou *.v* a tzv. physical constraints file s koncovkou *.pcf*. V *.pcf* súbore špecifikujeme porty, ktoré využívame, či už sú to predvolené hodnoty ako hodiny či zabudované LEDky, alebo porty, do ktorých môžeme zapojiť vlastné pripojenia. Tieto vstupy a výstupy potom už využíva samotný verilog.

Základný verilogový súbor zvyčajne obsahuje popis vstupov a výstupov daného modulu v hlavičkovom bloku. Následne je zadefinované vnútorné fungovanie: môžeme definovať registre, konštanty, priradzovať výstupom kombinačnú hodnotu a tvoriť tzv. always loops, teda bloky, ktorých obsah sa vykoná zakaždým, keď vo vstupnom kanáli nastane pozitívna resp. negatívna hrana. Takto môžeme registrom priradzovať hodnoty a tvoriť sekvenčnú logiku. Možné je aj vnoriť moduly do seba – pri naprogramovaní modulu v jednom verilogovom súbore môžeme tento modul využiť ako súčasť iného modulu tak, že ho inštancujeme v rámci iného kódu a zadefinujeme pripojenie jeho vstupov a výstupov na kanály vo vnútri vonkajšieho modulu. Tu môžeme využiť aj parametre – pri programovaní modulu môžeme parametre nechať nedefinované, a pre každú inštanciu ich špecifikovať nanovo, teda môžeme ľahko vytvoriť veľa rôznych podobných variánt jedného modulu.

Verilog podporuje aj jednoduché testovanie spravením testbench modulu. Jedná sa o modul, ktorý nebude syntetizovaný na FPGA, ale obsahuje v rámci seba inštanciu modulu, ktorý chceme testovať. V takomto module môžeme použiť aj štruktúry ako *while* a *for*, ale aj *wait*. Tieto príkazy nie sú syntetizovateľné, ale umožnia nám simulovať vstup testovaného modulu menením jeho vstupu v pevne stanovených časových intervaloch. Potom testbench simulujeme a pozorujeme signály v jeho kanáloch, a či sa v čase menia tak, ako je zamýšľané.

2. Existujúce implementácie

SAP-1

Keďže bola pomerne priamočiaro použiteľná, rozhodol som sa vyskúšať na Icesticku zbehnúť implementáciu SAP-1 (Smallest As Possible computer) opísanú v tomto článku. Problémom bolo, že bola navrhnutá pre iné FPGA. Pokúsil som sa preto kód adaptovať a znížiť množstvo výstupových portov, respektíve nejako výstup skondenzovať, keďže využíva viac portov ako má Icestick. Výsledný dizajn je na githube projektu. Rozhodol som sa sedem-segmentový displej s multiplexerom jednoducho nahradiť tak, že iba jeden zo siedmych segmentov každej číslice sa bude zobrazovať na jednej z troch LEDiek, ktoré má v sebe Icestick zabudovaný. Prekvapivo som ale musel aj zmeniť zopár detailov, ktoré vyzerali ako chyby v kóde, konkrétne priame priradenia výstupov vnorených modulov na registre (registrom by sa hodnota mala priradzovať v always cykloch, a nie kontinuálne). Neviem, či je to spôsobené iným vývojovým prostredím, ktoré možno používam, pretože pôvodný dizajn by mal byť funkčný… Každopádne som po pár zmenách dospel k funkčnému dizajnu, na ktorom aj pri

obmedzenom výstupe vidno, že naozaj funguje a čísla sa striedajú (pokúsil som sa zo siedmych segmentov zvoliť ten, ktorý sa mení najčastejšie, aby to bolo čo najzjavnejšie).

Tu sú štatistiky z kompilácie:

Number of wires:	**245**
Number of wire bits:	**968**
Number of public wires:	**245**
Number of public wire bits:	**968**
Number of memories:	**0**
Number of memory bits:	**0**
Number of processes:	**0**
Number of cells:	**514**
SB_CARRY	**45**
SB_DFF	**16**
SB_DFFE	**128**
SB_DFFER	**42**
SB_DFFN	**8**
SB_DFFNSS	**6**
SB_DFFR	**3**
SB_LUT4	**266**

Čo je zvláštne je, že vypísané množstvo použitej pamäti je nulové, aj napriek tomu, že dizajn pamäť využíva na uloženie svojho programu. Pravdepodobne sa nejdená o to, že by terminál vypisoval iba štatistiky vrchného modulu, pretože množstvo použitých wires je omnoho vyššie. Iné, pravdepodobnejšie vysvetlenie je, že pri kompilácií sa kód optimalizuje, a preto v konečnom dôsledku využíva iba registre a žiadne pamäťové bunky...