Procesor s jednoduchou instrukční sadou

Datum zadání: 15.10.2013
Datum a forma odevzdání: do 15.12.2013 23:59, POUZE přes IS FIT, 4 soubory
Počet bodů: max. 20 bodů
Poznámka: součástí zadání je archiv project2.zip
Dotazy: v případě problémů souvisejících se zadáním se obracejte na vasicek@fit.vutbr.cz

1 Úvod

Cílem tohoto projektu je implementovat pomocí VHDL procesor, který bude schopen vykonávat program napsaný v jazyce BrainF*ck [3]. I když tento jazyk používá pouze osm jednoduchých příkazů (instrukcí), jedná se o výpočetně úplnou sadu, pomocí které je možné realizovat libovolný algoritmus.

Činnost procesoru

Jazyk BrainF*ck definuje osm příkazů zakódovaných pomocí tisknutelných 8-bitových znaků. Implementovaný procesor bude zpracovávat přímo tyto znaky (tzn. operační kód procesoru bude sestávat vždy z osmi bitů). Program v tomto jazyce sestává ze sekvence těchto příkazů (neznámé příkazy jsou ignorovány, což umožňuje vkládat komentáře přímo do programu). Vykonávání programu začíná první instrukcí a končí jakmile je detekován konec sekvence (znak s ASCII hodnotou 0). Program je uložen v nemodifikovatelné paměti ROM a je vykonáván nelineárně (tzn. obsahuje skoky). Data jsou uložena v paměti RAM, jejíž obsah je inicializován na hodnotu nula. Pro přístup do paměti se používá ukazatel (ptr), který je možné přesouvat o pozici doleva či doprava. Paměť je chápána jako kruhový buffer uchovávající 8-bitová čísla bez znaménka.

Procesor podporuje příkazy definované v následující tabulce. Operační kódy, které se v tabulce nenacházejí jsou procesorem ignorovány.

příkaz	operační kód	význam	ekvivalent v C
`>`	0x3E	inkrementace hodnoty ukazatele	ptr += 1;
`<`	0x3C	dekrementace hodnoty ukazatele	ptr -= 1;
`+`	0x2B	inkrementace hodnoty aktuální buňky	*ptr += 1;
`-`	0x2D	dekrementace hodnoty aktuální buňky	*ptr -= 1;
`[`	0x5B	je-li hodnota aktuální buňky nulová, skoč za odpovídající příkaz] jinak pokračuj následujícím znakem	while (*ptr) {
`]`	0x5D	je-li hodnota aktuální buňky nenulová, skoč za odpovídající příkaz [jinak pokračuj následujícím znakem	}
`.`	0x2E	vytiskni hodnotu aktuální buňky	putchar(*ptr);
`,`	0x2C	načti hodnotu a ulož ji do aktuální buňky	*ptr = getchar();
null	0x00	zastav vykonávání programu	return;

V případě příkazů `[` a `]` manipulujících s ukazatelem do programového kódu (instrukčním čítačem PC) je zapotřebí detekovat odpovídající pravou, respektive levou, závorku. Možností je několik, nejjednodušší je postupně inkrementovat, respektive dekrementovat, ukazatel a počítat počet závorek (viz dále).

Mikrokontroler

Aby bylo možné vykonávat smysluplný program, je procesor doplněn o paměti a vstupně-výstupní rozhraní. Výsledný mikrokontroler je ve formě projektu pro FITkit. Procesor je připojen ke dvěma odděleným pamětem, paměti programu (pouze pro čtení, kapacita 4kB) a paměti dat (dovoluje čtení i zápis, kapacita 1kB).

Vstup dat je řešen pomocí maticové klávesnice. Jakmile procesor narazí na instrukci načtení hodnoty (operační kód 0x2C), vykonávání se pozastaví do té doby, než je stisknuto některé z tlačítek klávesnice. Tlačítka 0-9 jsou interpretována jako znaky '0' až '9' s ASCII hodnotami 48 až 57. Tlačítko * a # je interpretováno jako konec řádku s ASCII hodnotu 10.

Výstup dat je řešen pomocí LCD displeje, posun kurzoru na displeji je řešen automaticky. Při zápisu většího počtu znaků, než-li dovoluje kapacita aktivní části displeje, dojde k návratu na první znak a dříve zapsané znaky se postupně přepisují,

2 Úkoly

1. Obsah souboru login.b obsahujícího program v jazyce BrainF*ck, který tiskne řetězec xlogin01, zkopírujte do debuggeru na adrese [5]. Tlačítkem "Start debugger" spusťte krokování a sledujte co způsobuje která instrukce, jak se pohybuje programový čítač a ukazatel do paměti dat. Vytvořte program, který vytiskne na displej Váš login (na velikosti písmen nezáleží). Snažte se v programu využít všechny dostupné příkazy s výjimkou příkazu načtení. Pokuste se vytvořit co nejkratší program.

2. Seznamte se s kódem v souborech *ram.vhd* (paměť dat), *rom.vhd* (paměť programu), *cpu.vhd* (rozhraní procesoru) a zejména *top.vhd* (strukturní popis mikrokontroleru). Povšimněte si, kde je definován program v jazyce BrainF*ck a jak jej lze modifikovat.

3. Do souboru *cpu.vhd* doplňte vlastní VHDL kód, který bude realizovat procesor vykonávající program zapsaný v jazyce BrainF*ck. Pro nalezení odpovídající levé závorky použijte zásobník návratových adres (uvažujte max. kapacitu 16 adres). Zásobník realizujte např. pomocí posuvného registru.

   Rozhraní procesoru je pevně dané a skládá se z pěti skupin signálů: synchronizace, rozhraní pro paměť programu, rozhraní pro paměť dat, vstupní rozhraní a výstupní rozhraní.

   Synchronizační rozhraní tvoří tři signály. **CLK** - hodinový synchronizační signál. Procesor pracuje vždy při vzestupné hraně hodinového signálu. **RESET** - asynchronní nulovací signál. Je-li RESET=1, procesor zinicializuje svůj stav (PTR=0,PC=0). **EN** - povolení činnosti procesoru. Procesor vykonává jednotlivé příkazy programu pouze je-li EN=1.

   Rozhraní *synchronní* paměti ROM uchovávající program je tvořeno třemi signály. Signál **CODE_ADDR** udává adresu buňky, signál **CODE_DATA** obsahuje 8-bitové instrukční slovo nacházející se na adrese CODE_ADDR. Hodnota signálu CODE_ADDR a CODE_EN je vzorkována a hodnota signálu CODE_DATA aktualizována vždy při vzestupné hraně hodinového signálu. K aktualizaci hodnoty signálu CODE_DATA dochází pouze pokud **CODE_EN** = 1 (tj. aktivním signálu povolení činnosti).

   Rozhraní *synchronní* paměti RAM uchovávající data je tvořeno pěti signály. Signál **DATA_ADDR** slouží k adresaci konkrétní buňky paměti. Signál **DATA_RDATA** (načtená data) obsahuje 8-bitovou hodnotu buňky na adrese DATA_ADDR. Signál **DATA_WDATA** (zapisovaná data) nechť obsahuje 8-bitovou hodnotu, kterou se má přepsat buňka na adrese DATA_ADDR. Podobně jako v předchozím případě jsou signály DATA_ADDR, DATA_EN a DATA_WDATA čteny a signál DATA_RDATA aktualizován při vzestupné hraně hodinového signálu. Rozhraní pracuje následovně. Platí-li **DATA_EN**=1 (povolení činnosti paměti) a **DATA_RDWR**=1 (volba režimu čtení / zápis), signál DATA_RDATA je aktualizován hodnotou buňky na adrese DATA_ADDR. Je-li DATA_EN=1 a DATA_RDWR=0, hodnota buňky na adrese DATA_ADDR je přepsána hodnotou signálu DATA_WDATA a signál DATA_RDATA je aktualizován hodnotou DATA_WDATA.

   Vstupní rozhraní, které je připojeno na řadič klávesnice pracuje následovně. Při požadavku na data procesor nastaví signál **IN_REQ** na 1 a čeká tak dlouho, dokud signál **IN_VLD** (input valid) není roven 1. Jakmile se tak stane, může procesor přečíst signál **IN_DATA**, který obsahuje ASCII hodnotu stisknuté klávesy.

   Výstupní rozhraní napojené na LCD displej pracuje následovně. Při požadavku na zápis dat procesor nejdříve musí otestovat stav signálu **OUT_BUSY**. Tento signál indikuje, že je LCD displej

zaneprázdněn vyřizování předchozího požadavku. Jakmile je OUT_BUSY=0, procesor inicializuje signál **OUT_DATA** zapisovanou ASCII hodnotou a současně na jeden hodinový takt nastaví signál **OUT_WE** (povolení zápisu) na 1.

Činnost procesoru důkladně ověřte pomocí simulace a následně na přípravku FITkit. K ověření použijte dodané testovací programy (viz soubor top.vhd) případně použijte vlastní.

4. Do souboru *top.vhd* vložte program vytvořený v prvním bodě zadání vypisující Váš login. Projekt vysyntetizujte (tj. přeložte) a odsimulujte.

   Poznámka: Hlášek během syntézy (info, warning) vzniklých mimo entitu CPU si nevšímejte, nelze je bohužel selektivně potlačit.

3 Odevzdává se

Do IS FIT se odevzdávají následující **4 soubory** (nikoliv ZIP či jiný archiv). Soubor **login.b** obsahující program v jazyce BrainF*ck vypisující Váš login (jedná se o výsledek bodu 1 zadání). Soubor **cpu.vhd** obsahující implementaci procesoru (jedná se o výsledek bodu 3 zadání). Výsledky 4. bodu zadání, tj. výsledek syntézy nacházející se v souboru build/fpga/inp.srp (soubor **inp.srp**), printscreen ze simulace zachycující konec vykonávání programu s čitelnými signály procesoru (minimálně stav automatu a signály entity cpu) a obsahem LCD displeje (soubor **inp.png**).

4 Hodnocení

Za kompletní implementaci procesoru (tj. splnění bodu 3) lze získat až 14 bodů. Za implementaci procesoru podporujícího pouze jednoduchý while cyklus (tj. nepodporující vnořené while cykly) lze získat až 7 bodů. Za implementaci procesoru podporujícího vnořené cykly while avšak nevyužívající zásobník pro nalezení odpovídající levé závorky lze získat až 10 bodů.

Za první a poslední bod zadání lze získat až 6 bodů.

5 Upozornění

Pracujte samostatně, nikomu nedávejte svoji práci k opsání. Plagiátorství se hodnotí 0 body případně dalším adekvátním postihem dle platného disciplinárního řádu VUT v Brně. Přejmenování proměnných či změna pořadí jednotlivých bloků není považována za autorské dílo. Tato změna nemá vliv na výsledný hardware (výsledek syntézy).

Odkazy

[1] http://merlin.fit.vutbr.cz/FITkit/download.html - image s vývojovými nástroji pro VirtualBox

[2] http://merlin.fit.vutbr.cz/FITkit/docs/navody/makefile2.html - překladový systém

[3] http://en.wikipedia.org/wiki/BrainFuck - popis instrukční sady, další odkazy

[4] http://www.hevanet.com/cristofd/brainfuck/ - několik programů napsaných v jazyce BrainFuck

[5] http://www.iamcal.com/misc/bf_debug/ - online debugger

Návod

Následující řádky jsou určeny těm, kteří doposud netuší jak procesor naimplementovat. Obecně platí, že procesor se skládá z datové cesty obsahující registry, ALU, apod. a řídicí cesty obsahující automat. Stejně tak je tomu i v tomto případě. Blokové schema možné implementace je uvedeno na následujícím obrázku.

Abychom mohli vykonávat program, obsahuje datová cesta tři registry (čítače) s možností inkrementace a dekrementace a zásobník návratových adres (RAS). Registr PC slouží jako programový čítač (tj. ukazatel do paměti programu), registr PTR jako ukazatel do paměti dat a registr CNT slouží ke korektnímu určení odpovídajícího začátku/konce příkazu while (počítání otevíracích / uzavíracích závorek, viz. popis instrukční sady). Mimo to datová cesta obsahuje multiplexor, pomocí kterého je možné určit zapisovanou hodnotu do paměti dat. Zapsat je možné buď hodnotu načtenou ze vstupu, hodnotu v aktuální buňce sníženou o jedničku nebo hodnotu aktuální buňky zvýšenou o jedničku. V případě, že se rozhodnete NEimplementovat podporu vnořených while cyklů, registr CNT ani zásobník nepotřebujete. Stačí vyhledat odpovídající levou případně pravou závorku.

Všechny řídicí signály jsou ovládány automatem, tak jak je uvedeno ve schematu. Při tvorbě VHDL kódu se inspirujte procesorem probíraným na cvičeních, jedná se de facto pouze o jednodušší variaci. V prvním kroku implementujte registry a poté postupujte od jednodušších instrukcí ke složitějším. Implementaci smyček si ponechte až úplně na závěr. Zásobník lze implementovat několika způsoby, nejjednodušší z pohledu zápisu je využít posuvný registr o kapacitě 192 bitů (šestnáct 12-bitových položek). Prvních 12 bitů poté stačí propojit s registrem PC a korektně obsluhovat posuv doleva (operace push) a doprava (operace pop). Korektní činnost ověřte pomocí simulace a vlastních či přiložených programů (pozor, v simulaci nelze provádět vstup dat).

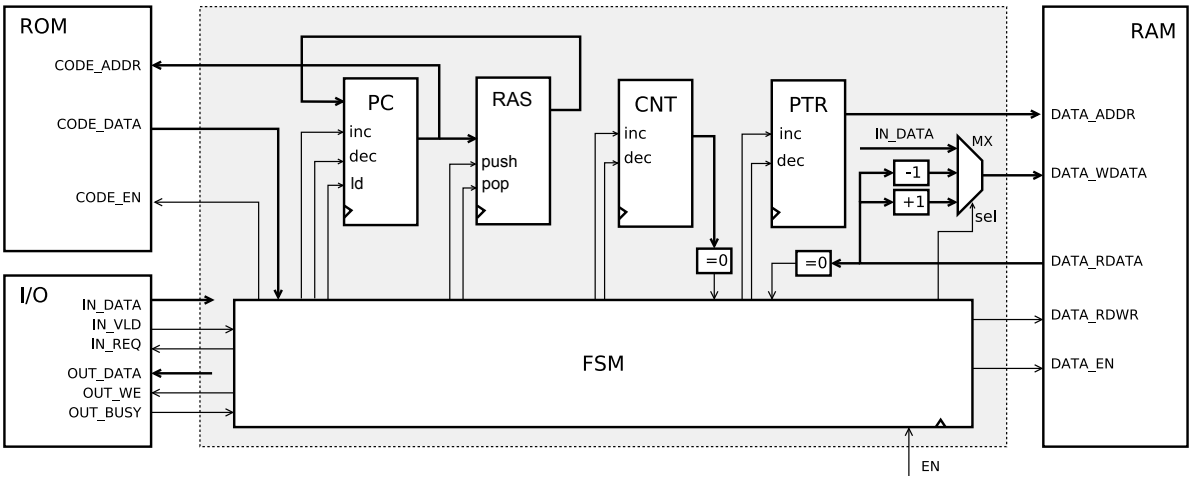

Figure 1: Blokové schema mikrokontroleru

Nevíte-li jak implementovat automat, podívejte se do materiálů ke cvičením INP. Jako návod pro implementaci automatu by měl posloužit pseudokód popisující chování jednotlivých instrukcí uvedený v tabulce 1. Máte-li s implementací problémy, vytvořte jednodušší verzi automatu, který nepodporuje vnořené smyčky. Pseudokód je uveden v tabulce 2.

Projekt pro FITkit

Zdrojové soubory využívají komponenty a překladové skripty, které se používají v rámci projektu FITkit. K usnadnění práce doporučujeme využít image pro VirtualBox, který obsahuje všechny potřebné nástroje. Po stažení image z adresy [1] a spuštění v aplikaci VirtualBox nezapomeňte aktivovat licenci pro Xilinx ISE (spuštěním průvodce `xlcm`).

Po spuštění QDevKitu stáhněte aktuální zdrojové kódy a do podadresáře *apps* rozbalte obsah souboru *project2.zip*. Při následném spuštení QDevKitu se ve stromu dostupných aplikací objeví položka INP a pod ní tento projekt. Pomocí kontextového menu případně dvojklikem na této položce lze připojený FITkit naprogramovat případně projekt přeložit či spustit simulaci. Celým procesem lze projít i v příkazové řádce (příkazy `gmake`, `gmake load`, `gmake isim`, `gmake term`) [2]. Před použitím příkazu gmake je zapotřebí vytvořit Makefile pomocí příkazu `fcmake`.

Součástí zadání je vysyntetizované vzorové řešení (soubor *build/inp.bin*) obsahující program, který po spuštění očekává na vstupu číslo (tj. sekvence číslic ukončená stiskem klávesy #), které je následně rozloženo na prvočísla. Pokud FITKit naprogramujete (pozor, nesmíte projekt přeložit) dodaným kódem a spustíte terminál, procesor bude čekat na stisk klávesy. Tento stav je indikován svitem LED diody D4. Vzorové řešení je určeno pro FITkit s dvouřádkovým displejem (tzn. verze 2.x). Pro ověření funkce na FITkitu s jednořádkovým displejem je zapotřebí nakopírovat obsah adresáře *build/fitkit_v1* do adresáře *build* jinak v důsledku nekompatibility nebudou vidět znaky na druhé polovině displeje.

Projekt je možné **simulovat** pomocí přiloženého test bench souboru (*fpga/sim/tb.vhd*) a simulátoru Xilinx ISIM. Simulaci lze vyvolat z aplikace QDevKit případně zadáním příkazu `gmake isim` v adresáři s projektem. Skript starající se o vložení signálů do simulace a její spuštění je umístěn v souboru *fpga/sim/isim.tcl*. Skript lze modifikovat dle potřeb. Test bench (VHDL kód v souboru tb.vhd) obsahuje FPGA entitu, generátor hodinového a resetovacího signálu a emulátor LCD displeje.

příkaz / stav	**pseudokód**
výchozí stav	PC ← 0, PTR ← 0, CNT ← 0
`>`	PTR ← PTR + 1, PC ← PC + 1
`<`	PTR ← PTR - 1, PC ← PC + 1
`+`	DATA_RDATA ← ram[PTR] ram[PTR] ← DATA_RDATA + 1, PC ← PC + 1
`-`	DATA_RDATA ← ram[PTR] ram[PTR] ← DATA_RDATA - 1, PC ← PC + 1
`.`	while (OUT_BUSY) {} OUT_DATA ← ram[PTR], PC ← PC + 1
`,`	IN_REQ ← 1 while (!IN_VLD) {} ram[PTR] ← IN_DATA, PC ← PC + 1

`[`	PC ← PC + 1
	if (ram[PTR] == 0)
	  CNT ← 1
	  while (CNT != 0)
	  c ← rom[PC]
	  if (c == '[') CNT ← CNT + 1 elsif (c == ']') CNT ← CNT - 1
	  PC ← PC + 1
	else
	  stack.push(PC)

`]`	if (ram[PTR] == 0)
	  PC ← PC + 1
	  stack.pop()
	else
	  PC ← stack.top()

null	PC ← PC
ostatní	PC ← PC + 1

Table 1: Popis chování (pseudokód) jednotlivých intrukcí procesoru

příkaz / stav	**pseudokód**

`[`	PC ← PC + 1
	if (ram[PTR] == 0)
	  do
	  c ← rom[PC]
	  PC ← PC + 1
	  until (c == ']')

`]`	if (ram[PTR] != 0)
	  do
	  PC ← PC - 1
	  c ← rom[PC]
	  until (c == '[')
	PC ← PC + 1

Table 2: Zjednodušená implementace instrukcí procesoru nepodporující vnořené smyčky a tímpádem nevyžadující zásobník