Procesor s jednoduchou instrukční sadou

Datum zadání: 13.11.2011
Datum a forma odevzdání: do 23.12.2011 23:59, POUZE přes IS FIT
Počet bodů: max. 16 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 [4]. Ačkoliv tento jazyk definuje pouze osm jednoduchých instrukcí, jedná se o výpočetně úplnou sadu, pomocí které je možné popsat libovolný algoritmus.

Aby bylo možné vykonávat smysluplný program, je procesor doplněn o paměti a vstupně-výstupní rozhraní. 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. 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,

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 pomocí přesouvat o pozici doleva či doprava. Paměť RAM je organizována jako 1024 buněk uchovávajících 8-bitové hodnoty bez znaménka.

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

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).

Zprovoznění projektu

Zdrojové soubory využívají komponenty a překladové skripty, které se používají v rámci projektu FITkit. Archiv je nutné rozbalit do adresáře *apps* a pomocí standardních nástrojů (fcmake + gmake) kód přeložit, vysyntetizovat a nahrát do FITkitu.

K usnadnění práce je možné využít prostředí QDevKit dostupné ze stránek projektu [1]. Předpokladem je úspěšné zprovoznění FITkitu (viz [2]). QDevKit nainstalujte, stáhněte aktuální zdrojové kódy a do adresáře *apps* rozbalte obsah souboru *project2.zip*. Při spuštení QDevKit 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. Totéž lze realizovat i v příkazové řádce (`gmake`, `gmake load`, `gmake isim`) [3]. Před použitím příkazu gmake je zapotřebí vytvořit Makefile pomocí příkazu `fcmake`.

Součástí zadání je vysyntetizovaný kód (soubor *build/inp.bin*) obsahující kód, který po spuštění očekává na vstupu číslo (tj. sekvence číslic ukončená stiskem klávesy #), které je následně rozloženo na prvočísla. Pro vyzkoušení tedy postačí naprogramovat dodaný kód do FITkitu (tj. bez překladu). Pokud procesor čeká na stisk klávesy, svítí LED dioda D4. Součástí zadání je též vzorové řešení (soubor cpu.ngc). V případě, že budete chtít použít tento kód, řiďte se pokyny v souboru *fpga/ngc/info.txt*.

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.

2 Úkoly

1. Seznamte se s kódem v souborech *ram.vhd* (paměť dat), *rom.vhd* (paměť programu) a *top.vhd* (mikrokontroler). Povšimněte si, kde je zapsaný kód pro mikrokontroler.

2. Do souboru *cpu.vhd* doplňte VHDL kód, který bude realizovat procesor vykonávající program zapsaný v jazyce BrainF*ck.

   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. **RESET** - asynchronní nulovací signál. Je-li RESET=1, procesor zinicializuje svůj stav (PTR=0,PC=0). **EN** - povolení činnosti procesoru. Pokud je signál RESET uvolněn (RESET=0) a EN=1, procesor postupně začne vykonávat program od adresy 0.

   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. Je-li signál povolení **DATA_EN** (povolení činnosti paměti) roven 1 a **DATA_RDWR** (volba režimu čtení / zápis) je roven 0, signál DATA_RDATA je aktualizován hodnotou buňky na adrese DATA_ADDR. Je-li signál povolení

DATA_EN roven 1 a DATA_RDWR je roven 1, 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 je signalizována validita, 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 zapíše ASCII data na **OUT_DATA** a současně nastaví signál **OUT_WE** (povolení zápisu) na 1.

3. Ověřte činnost procesoru pomocí simulace a následně na přípravku FITkit. K ověření použijte dodané testovací programy (viz soubor top.vhd) případně použijte vlastní.

4. V souboru *top.vhd* si zvolte program označený jako "[2]" (nebo vytvořte vlastní obsahující alespoň jeden neprázdný cyklus) vypisující data na displej, který nepotřebuje vstup z klávesnice. 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 **cpu.vhd** obsahující implementaci procesoru (jedná se o výsledek bodu 2 zadání). Výsledky a vstupy 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**). Nezapomeňte též přiložit použitý program v jazyce BrainF*ck (soubor **inp.b**).

4 Hodnocení

Za kompletní implementaci procesoru (tj. splnění bodu 2) lze získat až 12 bodů. Za implementaci procesoru podporujícího pouze jednoduchý while cyklus (tj. nepodporující vnořené while cykly) lze získat až 8 bodů. Za poslední bod zadání lze získat až 4 body.

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).

6 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. 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 nepotřebujete.

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. Korektní činnost ověřte pomocí simulace a vlastních či přiložených programů (pozor, v simulaci nelze provádět vstup dat).

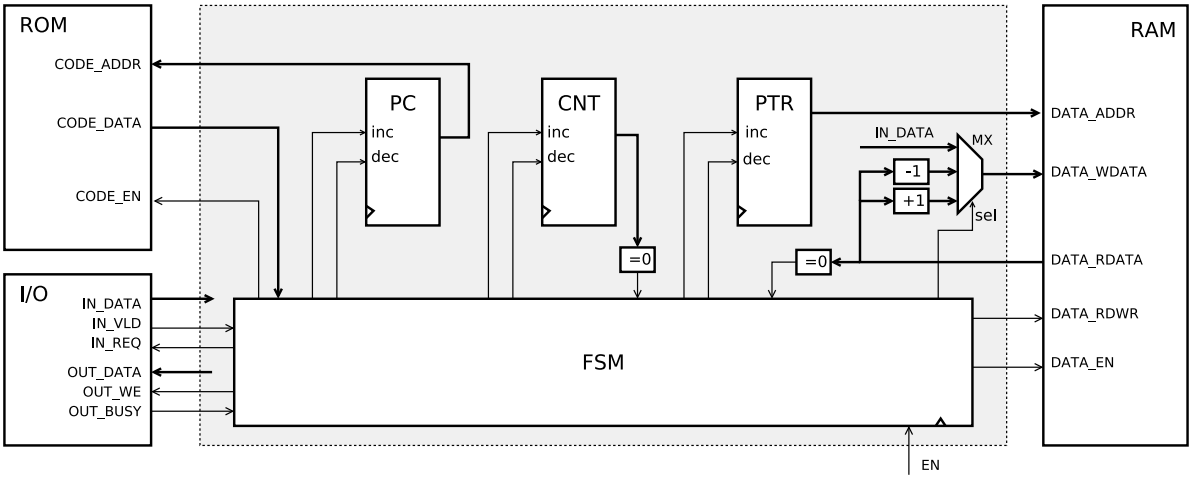

Figure 1: Blokové schema mikrokontroleru

Nevíte-li jak implementovat automat, podívejte se do cvičení, kde byly automaty probírány. Jako návod pro implementaci automatu by měl posloužit pseudokód popisující chování jednotlivých instrukcí uvedený v tabulce 1.

7 Revize

13.11.2011 - první verze dokumentu
1.12.2011 - opraveny překlepy v zadání; upraven LCD řadič, aby se při simulaci vždy uvažoval jednořádkový displej

Odkazy

[1] http://merlin.fit.vutbr.cz/FITkit - stránky projektu FITkit

[2] http://merlin.fit.vutbr.cz/FITkit/docs/navody/200810start.html - informace o zprovoznění FITkitu

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

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

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

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

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

null	PC ← PC
ostatní	PC ← PC + 1

Table 1: Popis chování (pseudokód) jednotlivých intrukcí procesoru