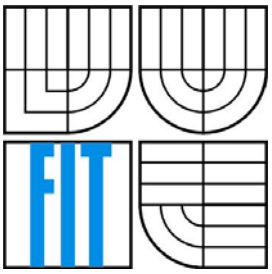


MOŽNOSTI PROPOJENÍ JAZYKA ISAC A HANDEL-C S OHLEDEM NA GENEROVÁNÍ HDL POPISU

SEMINÁRNÍ PRÁCE (VPD)

Ing. Tomáš Novotný

31. srpen 2008

Obsah

1 Úvod

V rámci jazyka ISAC (Instruction Set Architecture C) [1], jenž vzniká v rámci projektu Lissom na FIT VUT v Brně, existuje sekce s názvem BEHAVIOR. Tato sekce může obsahovat popis instrukčního chování, které je možno popsat pomocí příkazů z podmnožiny jazyka ANSI-C.

Pro potřeby simulace je popis v jazyce ANSI-C velice vhodný, pro potřeby hardwarové syntézy již tak vhodný není. Je třeba tento popis transformovat do některého z jazyků HDL (Hardware Description Language), kterým je prozatím jazyk VHDL.

Nicméně přímá transformace popisu z jazyka ANSI-C do jazyka VHDL se zachováním sémantiky je problémem velice komplexním a rozsáhlým. Hledání přímé cesty transformace by mohlo být příliš náročné a tak se nabízí využití jazyka Handel-C.

Jazyk Handel-C [2] patří do skupiny jazyků, kterou lze zařadit někam mezi jazyky HDL a klasické vyšší programovací jazyky jako je ANSI-C. Tento jazyk je svoji syntaxí velice blízký jazyku ANSI-C. Rozšířen je ale o nové konstrukce, které umožňují popisovat např. časování, komunikaci či paralelní zpracování. Díky těmto konstrukcím má tedy blíže k jazykům jako VHDL a lze provádět generování popisu z jazyka Handel-C do jazyka VHDL.

Jazyk Handel-C je navíc navržen tak, aby umožnil návrhářům integraci Handel-C kódu s kódem nejpoužívanějších jazyků Verilog a VHDL.

V této práci jsou představeny možnosti propojení jazyka Handel-C s jazykem VHDL, případně s jazykem Verilog, který je hodně používán zejména v zámoří jako ekvivalent jazyka VHDL, s využitím překladače *handelc,* který je součástí produktu DK Suite [3] od firmy Celoxica.

2 Jazyk Handel-C

Jazyk Handel-C není „čistým“ jazykem HDL jako populární jazyky VDHL a Verilog. Hlavním cílem, pro který byl jazyk vyvinut, bylo zpřístupnit návrh hardwaru komukoliv, kdo nemá mnoho zkušeností s klasickým návrhem s využitím jazyků HDL, ale má zkušenost se standardními vyššími programovacími jazyky jako je ANSI-C. Jazyk Handel-C umožňuje popisovat chování hardware. Jazyk byl vyvinut s přispěním Oxford Hardware Compilation Group. Nyní jej lze nejefektivněji využít v prostředí DK Suite od firmy Celoxica.

Tento jazyk je založen na jazyku ANSI-C, avšak obsahuje spoustu konstrukcí, které ANSI-C jazyk nemá. Rovněž však ale postrádá některé konstrukce ANSI-C jazyka.

Zde jsou ve zkratce uvedeny hlavní rozdíly mezi jazykem Handel-C a jazykem ANSI-C (převzato z [4]).

Co Handel-C přidává do jazyka ANSI-C?

- Paralelismus
- Časování
- Rozhraní
- Hodiny
- Komunikaci
- Bitové operace

Handel-C nepodporuje

- Rekurze
- Standardní knihovny
- Malloc ()
- Standard floating point
- Ukazatele

Z hlediska této práce jsem nejvíce prozkoumal možnosti rozhraní, které umožňují integraci s HDL kódem.

3 Handel-C a jazyk ISAC

Jak již bylo řečeno v předchozí kapitole, jazyk Handel-C má několik rozšíření oproti jazyku ANSI-C, ale zároveň některé konstrukce postrádá. To vše za účelem primárního použití pro návrh hardware. Podmnožina jazyka ANSI-C je využita v rámci jazyka ISAC pro popis chování operací. Z jazyka ISAC lze automaticky generovat syntetizovatelný popis architektury v jazyce VHDL.

V současné době jsou generovány s využitím šablon zdrojové prvky, s využitím párových automatů je generován centrální řadič s dekodérem. Avšak současný generátor VHDL kódu postrádá generování funkčního popisu jednotlivých operací, který je obsažen v BEHAVIOR sekci jazyka ISAC.
Generátor VHDL kódu umožňuje generovat rozhraní jednotlivých funkčních jednotek, nebo-li entit jazyka VHDL. Avšak neumožňuje generovat vlastní popis funkčnosti. Zde se nabízí využití jazyka Handel-C a jeho možnosti propojení s jazyky HDL. Generátor VHDL kódu z jazyka ISAC by bylo možné propojit s překladačem jazyka Handel-C, který by umožnil vygenerovat odpovídající popis funkčnosti jednotlivých funkčních jednotek. Je však zapotřebí připravit transformaci BEHAVIOR sekce, která je v jazyce ANSI-C, do jazyka Handel-C.

3.1 Rozšíření s ohledem na transformaci do Handel-C

V této kapitole jsou uvedeny klíčové aspekty uvažované transformace z jazyka ANSI-C do jazyka Handel-C.

Zde se nabízí dvě možnosti řešení. První možností je nahradit stávající podmnožinu jazyka ANSI-C, která slouží pro popis chování, podmnožinou jazyka Handel-C. Toto řešení v zásadě obsahuje dva kroky. V prvním kroku ze stávající podmnožiny ANSI-C odebrat konstrukce, které jazyk Handel-C nepodporuje, v druhém kroku naopak přidat do této podmnožiny konstrukce, které jazyk Handel-C obsahuje navíc oproti ANSI-C.

Druhou možností je, zabývat se možnostmi transformace podmnožiny jazyka ANSI-C do jisté minimální podmnožiny jazyka Handel-C. Minimální s pohledu možností jazyka ISAC a generátoru HDL kódu z jazyka ISAC. Ne všechny konstrukce jazyka Handel-C je nutné podporovat v rámci transformace.

3.1.1 Jazyk Handel-C nepodporuje

Již v kapitole č. 2 bylo uvedeno, které prvky jazyka ANSI-C nejsou obsaženy v jazyce Handel-C. Pokud tedy chceme provést transformaci zmíněné sekce BEHAVIOR do kódu v jazyce Handel-C, nezbývá nám, než tyto konstrukce v rámci syntaxe BEHAVIOR sekce nepodporovat. Z hlediska jazyka ISAC je tedy tato část realizace velice snadná. Jedná se pouze o úpravu syntaxe jazyka ISAC, potažmo vnořeného jazyka ANSI-C.

3.1.2 Jazyk Handel-C poskytuje navíc

Tato část transformace již není tak triviální jako předchozí krok. Konstrukce, které jazyk Handel-C poskytuje navíc by bylo možné všechny zahrnout do jazyka ISAC, tím pádem bychom však mohli

přestat v souvislosti s BEHAVIOR sekcí mluvit o jazyku ANSI-C. Popis BEHAVIOR sekce by pak byl čistě podmnožinou jazyka Handel-C.

Jelikož je cílem zachovat v rámci jazyka ISAC podmnožinu jazyka ANSI-C pro popis BEHAVIOR sekce, tomuto kroku bychom se rádi vyhnuli a raději ponechali tuto sekci spíše jako podmnožinu jazyka ANSI-C. Z tohoto důvodu je třeba specifikovat možnosti transformace z jazyka ANSI-C do jazyka Handel-C.

3.1.3 Možnosti transformace z jazyka ANSI-C do jazyka Handel-C

V této kapitole jsou popsány prvky jazyka Handel-C, které nejsou součástí jazyka ANSI-C. Zároveň je vždy zmíněno, zda je třeba podporovat generování těchto prvků v případné transformaci z jazyka ANSI-C do jazyka Handle-C.

3.1.3.1 Hodiny

Hodiny se v jazyce Handel-C deklarují pomocí příkazu *set clock.*

```
set clock = Umisteni
```

kde, *Umístěni* může nabývat různých hodnot. Jelikož jsou při generování HDL popisu z jazyka ISAC hodiny určeny operací *main* jazyka ISAC, má smysl uvažovat *Umistini* jediné:

```
external = [pin]
```

Toto umístění určuje externí zdroj hodin a konkrétní pin. Každá operace *main* jazyka Handel-C musí mít nějaké hodiny deklarované. Použití hodin lze vidět na příkladě v kapitole 3.2.
Hodiny je tedy třeba generovat.

3.1.3.2 Časování

Zde je nutné vzít v úvahu, že příkaz přiřazení je v jazyce Handel-C vyhodnocen vždy za cenu jednoho hodinového taktu. Proto není možné používat přiřazení typu

```
a = b++;,
```

kde není časování jasně určeno. Příkaz musí být rozdělen na dva příkazy sekvenční, u nichž je již časování zřejmé:

```
b++;
a=b;
```

Dalším důležitým aspektem je, aby každá větev programu byla provedena s minimálně jedním hodinovým taktem. Pokud může dojít např. k nesplnění podmínky, je třeba patřičnou větev doplnit o příkaz *delay* takto.

```
if (a < b)
{
      a=b;
}
```

```
else
{
      delay;
}
```

V souvislosti s příkazem *delay* by nemusela být žádna transformace prováděna. Stejně tak jako návrhář v jazyce VHDL si musí být vědom vzniku nežádoucích latch registrů v rámci syntézy, pokud neuvede default větve při větvení, musí si být návrhář vědom podobného účinku i v rámci transformace do jazyka Handel-C. Avšak je možné provádět doplnění sekce příkazu *delay* analýzou řídících konstrukcí.

Problém časování u přiřazení může být řešen v průběhu transformace, ale lze také ponechat na vývojáři.

3.1.3.3 Paralelní zpracování

Klíčové slovo *par* specifikuje blok, který by měl být vykonán paralelně. Tedy každý příkaz v bloku je vykonán v jednom cyklu. Protikladem je klíčové slovo *seq***,** které uvozuje sekvenčně vykonávaný blok. Pokud není definován blok jako paralelní je implicitně sekvenční. Tedy blok bez uvedení klíčového slova je sekvenční.

Toto rozšíření lze s těží generovat bez nutnosti rozšíření podmnožiny jazyka ANSI-C o konstrukce, které takovéto paralelní zpracování podporují. Zde by bylo možné doplnit podmnožinu jazyka ANSI-C o zmíněnou sekci *par*, avšak jen na syntaktické úrovni. Sémantická kontrola by v tomto případě nebyla nutná a nejednalo by se tedy o příliš rozsáhlé rozšíření. Navíc pokud bychom si tuto konstrukci odmysleli, stále máme implicitní sekvenční chování popsané v jazyce ANSI-C, aniž bychom původní podmnožinu nějak omezili.

3.1.3.4 Bitové operátory

Většina bitových operátorů je v jazyce Handel-C shodná s jazykem ANSI-C. Navíc jsou pouze operátory pro bitovou manipulaci.

`Operátor \\` zahodí zvolený počet bitů od nejnižšího bitu a do určené proměnné uloží zbylé bity po zahození.

`operátor <-` vybírá zvolený počet bitů od nejnižšího bitu.

`Operátor @` slouží ke zřetězení dvou výrazů. Bitová šířka výsledného zřetězení je rovna součtu zřetězených výrazů.

`Operátor []` slouží pro výběr jednotlivého bitu nebo zadaného rozsahu bitů.

Zde lze říci, že bez těchto rozšíření se může návrhář obejít a stačí tedy ponechat původní podmnožinu jazyka ANSI-C.

Je nutné však transformovat operátor [], který je v BEHAVIOR sekci použit primárně pro adresaci zdrojových prvků a nemůže být tedy transformován v nezměněné podobě do jazyka Handel-C.

Příklad:

```
a = register[src];
```

Tento příkaz uvedený v sekci BEHAVIOR říká, že ze zdrojového prvku *register* se má na adrese *src* načíst hodnota. Z hlediska generování HDL popisu a tedy transformace do Handel-C, není možno ponechat tuto konstrukci. Je třeba pouze specifikovat adresový signál, který má být napojen na entitu zdrojového prvku. Určení adresy by měl zajistit dekodér a do funkční jednotky by měl být pouze přiveden datový signál ze zdrojového prvku *register*.

3.1.3.5 Datové typy

Jazyk Handel-C nepodporuje operace v plovoucí řádové čárce a tedy i příslušné datové typy. Datové typy mohou být ve variantě *signed* a *unsigned.* Již dnes podmnožina jazyka ANSI-C použitá v rámci jazyka ISAC rovněž nepodporuje operace v plovoucí řádové čárce. Z tohoto pohledu tedy není transformace nijak limitující.

Datové typy v jazyce Handel-C mohou být deklarovány s určitou bitovou šířkou. Např. *int 5 a;,* kde 5 udává právě bitovou šířku. Tato šířka by mohla být přidána v průběhu transformace.

3.1.3.6 Signál

Signál je speciální objekt, který udržuje hodnotu jemu přiřazenou pouze po dobu jednoho hodinového taktu. Poté je jeho hodnota opět nastavena na inicializační hodnotu. Signál tedy reprezentuje konkrétní vodiče v hardwaru.

Syntaxe je následující:

```
Ssignal <Type DataWidth> Nazev;
```

Příklad:

```
int 15 a, b;
signal <int> sig;
a = 7;
par
{
      sig = a;
      b = sig;
}
```

Prvku *b* je přiřazena hodnota z *a*, tedy signál umožňuje propojení konkrétních hardwarových zdrojů, realizuje vodič.

Tato konstrukce by mohla být využita místo generování signálů z P-grafu. Jelikož již generování signálů je zahrnuto v P-grafu, lze tuto konstrukci z transformace vyloučit.

3.1.3.7 Paměti

Tuto část nemá smysl při transformaci z ANSI-C na Handel-C uvažovat, neboť se jedná o specifikaci konkrétní hardwarové platformy (FPGA), což v jazyce ISAC není obsaženo.

3.1.3.8 Kanály

Slouží pro komunikaci mezi paralelními bloky. Jeden paralelní blok může do kanálu zapisovat a druhý z něj může číst.

Kanál je specifikován pomocí klíčového slova *chan*. Pro četní i zápis do kanálu se využívá:

`Kanal ? Promenna` – čtení z kanálu

`Kanal ! Vyraz`– zápis do kanálu

Syntaxe je následující:

`chan [` ***logickyTyp*** `] ` ***Jmeno*** `[with` ***specifikace***`];`

Příklad

```
void main(void)
{
     unsigned 8 Res;
     chan unsigned 8 Bill;

     par
     {
         Bill ! 23;
         Bill ? Res;
     }
 }
```

Kanály mohou vytvářet FIFO komunikaci mezi paralelně probíhajícími procesy. Z popisu v dokumentaci se lze domnívat, že by se daly kanály uplatnit v mezi procesové komunikaci na úrovni popisu v jazyce VHDL. Ovšem nepodařilo se mi vygenerovat žádný VHDL kód , který by něco takového potvrzoval. Z tohoto důvodu se domnívám, že signály zřejmě rovněž není třeba zahrnovat do transformace z jazyka ANSI-C do jazyka Handel-C.

3.2 Integrace s HDL jazyky

Klíčovým prvkem pro komunikaci s externím kódem (v jazyce VHDL, Verilog) jsou v rámci jazyka Handel-C rozhraní.

Rozhraní lze definovat buď deklarací nebo definicí. Pokud chceme používat vícenásobnou instanci rozhraní, je třeba použít deklaraci. Je nutné si uvědomit, že rozhraní pro předdefinované druhy rozhraní není třeba definovat.

Rozhraní je definováno pomocí klíčového slova *interface.* Definice či deklarace má následující syntaxi:

Deklarace:

```
interface Typ `(` {Vstupni_porty_do_HandelC} `)` `(`
     {Vystupni_porty_z_HandelC} ')' ;
```

Definice:

```
interface Typ '(' {Vstupni_porty_do_HandelC} ')' NazevInstance '('
     {Vystupni_porty_z_HandelC} ')' with {Specifikace}
```

`Typ` poskytuje několik obecně předefinovaných typů rozhraní. Jedná se o jméno „černé skříňky“, ke které má být rozhraní připojeno.

`Vstupni_porty_do_HandelC` jde o volitelnou část definice. Lze definovat jeden či více prototypů portů, které přináší data do Handel-C kódu.

`Výstupni_porty_z_HandelC` jde o volitelnou část definice. Lze definovat jeden či více prototypů portů, které odesílají data z Handel-C kódu.

`NazevInstance` uživatelsky definovaný identifikátor pro instanci rozhraní

`Specifikace` zde lze specifikovat hardwarové detaily rozhraní, jako je počet pinů na čipu nebo definice externího simulátoru.

Jak již bylo uvedeno, poskytuje jazyk Handel-C obecně několik druhů předefinovaných rozhraní. Z nich jsou z hlediska integrace s HDL jazyky nejzajímavější předdefinované typy:

- `port_in/port_out` - představuje vodič napojený na port
- `bus_in/bus_out` - představuje vodič připojený přímo na externí pin

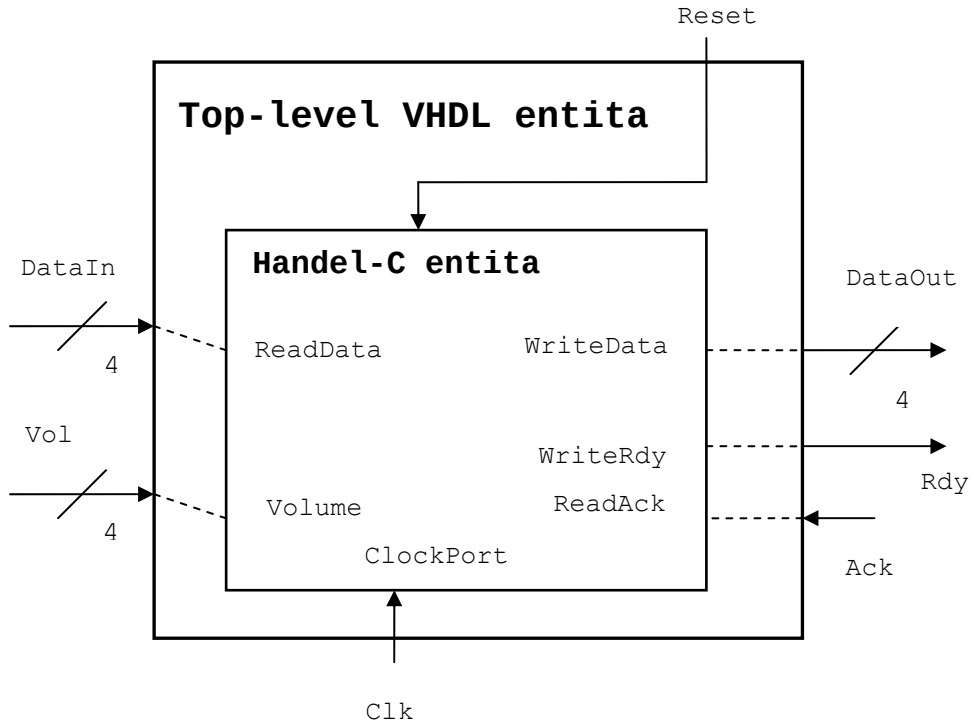
Obě rozhraní jsou použita tehdy, když chceme do existujícího VHDL popisu integrovat popis v jazyce Handel-C.

Z hlediska integrace jazyka Handel-C s HDL jazyky jsou umožněny dva druhy integrace. Záleží na tom, který z jazyků používáme jako stěžejní pro popis navrhovaného hardware.

3.2.1 Popis v jazyce HDL na nejvyšší úrovní

Jestliže pro popis hardware používáme primárně jazyk HDL a do něj chceme zakomponovat některé části návrhu popsané v jazyce Handel-C, pak mluvíme o popisu HDL na nejvyšší úrovni.

Na orbázku 1. je znázorněno propojení, jak lze pomocí rozhraní zakomponovat do popisu v jazyce VHDL části systému popsané v jazyce Handel-C. Popisu v jazyce VHDL odpovídá *Top-level entita.*.



Obrázek 1. – Top-Level VHDL entita s vnořeným Handel-C kódem

Propojení mezi popisem v jazyce VHDL a vnořeným popisem v jazyce Handel-C je tedy provedeno pomocí definovaných rozhraní *port_in* nebo *port_out* jak je vidět z následujícího příkladu.

Příklad:

```
unsigned 4 Bytes_out;
unsigned 1 DataReady;
// nastavení externího reset signálu
set reset = external "RESET"  with {active_low=1};

//definice rozhraní
// vstupní hodinový signál
interface port_in
     // clk – název portu použitého ve VHDL
     (unsigned 1 clk with {clockport = 1})
     // název instance portu, pod kterým lze přistupovat k portu
v rámci Handel-C
     ClockPort
     ();

interface port_in
     // DataIn – název portu použitého ve VHDL
     (unsigned 4 DataIn)
     // název instance portu
     ReadData
     // nejsou definované výstupní porty
     ()
     // standardní logický typ v jazyce VHDL
     with {vhdl_type = "std_logic_vector"};
```

```
interface port_out
      // nejsou definované vstupní porty
      ()
      // jméno instance
      WriteData
      // bitový výstupní port, jméno použité ve VHDL
      (unsigned 4 DataOut = Bytes_out)
      // standardní logický typ v jazyce VHDL
      with {vhdl_type = "std_logic_vector"};

interface port_out
      ()
      WriteRdy
      (unsigned 1 Rdy = DataReady);

interface port_in
      (unsigned 1 Ack)
      ReadAck
      ();

interface port_in
      (unsigned 4 Vol)
      Volume
      ()
      with {vhdl_type = "std_logic_vector"};


// přivedení hodinového signálu z Top-level entity do Handel-C
entity
set clock = internal ClockPort.clk;

// main funkce musí být obsažena
void main(void)
{
      unsigned int 4 Res1;
      unsigned int 4 Res2;
      // čtení ze vstupního portu
       par
       {
            // čtení ze vstupních portů z Top-level entity
            Res1 = ReadData.DataIn * Volume.Vol;
            Res2 = ReadData.DataIn + Volume.Vol;
       }
      if (ReadAck.Ack)
      {
            // Zápis do Top-level entity
            Bytes_out = Res1 & Res2;
            DataReady = ReadAck.Ack;
      }
}
```

3.2.1.1 Generovaný HDL popis

Každý Handel-C kód je mapován do VHDL souboru, každá Handel-C funkce je mapována do VHDL entity a architektury. Současně musí existovat top-level VHDL soubor, který mapuje jednotlivé entity dohromady a obsahuje globální definici portů.

Pokud je použit kompilátor z příkazového řádku:

handelc –vhdl handelEntity.hcc –o Top-level.vhd

jsou vytvořeny dva soubory s příponou .vhd.

První, který je definován parametrem *–o,* je top-level entitou v jazyce VHDL. Tedy entitou, která bude obsahovat komponentu vytvořenou z Handel-C popisu. Takto generovaná entita obsahuje jednak definici samotné entity, rovněž však definici komponenty a popis propojení mezi entitou a komponentou.

Druhý soubor, jehož název je vytvořen ze souboru .hcc tím, že tečka před příponou je nahrazena podtržítkem a je přidána koncovka .vhd, obsahuje popis komponenty generované z Handel-C popisu. Název komponenty je stejný s názvem generovaného .vhd souboru, ovšem neobsahuje již příponu .vhd. Pro každou funkci z jazyka Handel-C je pak v generovaném popisu vytvořena další entita. Pro funkci s názvem *MojeFunkce*, která je definována v souboru *Zdroj.hcc,* je vytvořena entita s názvem *Zdroj_hcc_MojeFunkce*.

Pokud je třeba specifikovat výstupní styl generovaného kódu v závislosti na použitém syntézním nástroji, lze použít parametr `hdlstyle`*SynthesisTool,* ten může nabývat např. hodnot `QuartusII`, `Precision` nebo `Synthesi` (obecný).

Následující příklad demonstruje, jakým způsobem vypadá generovaný HDL kód.

Příklad

Pokud uvažujeme výše zmíněné pojmenování (*handelEntity.hcc* a *Top-level.vhd),* bude soubor *Top-level.vhd* obsahovat jednu entitu, komponentu a *port_map* konstrukci.

```
ENTITY Top_level IS
  -- interface description
  PORT (
    Ack : IN std_logic;
    clk : IN std_logic;
    DataIn : IN std_logic_vector(3 DOWNTO 0);
    DataOut : OUT std_logic_vector(3 DOWNTO 0);
    PIN_RESET : IN std_logic;
    Rdy : OUT std_logic;
    Vol : IN std_logic_vector(3 DOWNTO 0)
  );
END;

  -- Component declaration
  COMPONENT handelEntity_hcc
  PORT (
    Ack : IN std_logic;
```

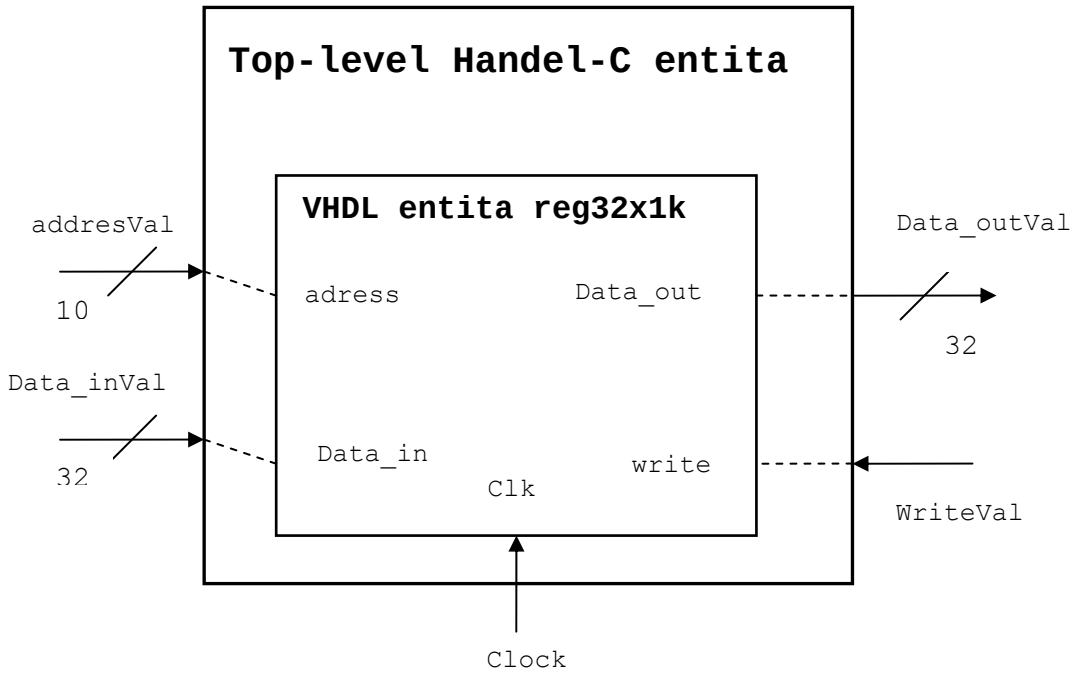
```
    DataIn : IN std_logic_vector(3 DOWNTO 0);
    G_Bytes_out_out : OUT unsigned(3 DOWNTO 0);
    G_DataReady_out : OUT unsigned(0 DOWNTO 0);
    PIN_RESET : IN std_logic;
    Vol : IN std_logic_vector(3 DOWNTO 0);
    clk : IN std_logic
  );
  END COMPONENT;

  -- Instantiation
  I0_file : handelEntity_hcc
    PORT MAP (
      Ack => Ack,
      DataIn => DataIn,
      G_Bytes_out_out => G_Bytes_out,
      G_DataReady_out => G_DataReady,
      PIN_RESET => PIN_RESET,
      Vol => Vol,
      clk => clk
    );
```

Soubor *handelEntity_hcc.vhd* pak bude obsahovat definici entity *handelEntity_hcc* a popis jejího chování. Rovněž pak definici a popis chování entity generované pro funkci *main.*

3.2.2 Popis v jazyce Handel-C na nejvyšší úrovní

Jestliže pro popis hardware používáme primárně jazyk Handel-C a do něj chceme zakomponovat některé části návrhu popsané v jazyce HDL, pak mluvíme o popisu Handel-C na nejvyšší úrovni.

Na orbázku 2. je znázorněno propojení, jak lze pomocí rozhraní zakomponovat do popisu v jazyce Handel-C části systému popsané v jazyce VHDL. Popisu v jazyce VHDL odpovídá entita s názvem *reg32x1k.*



Obrázek 2. – Top-Level VHDL entita s vnořeným Handel-C kódem

Propojení je provedeno na základě shody jména VHDL entity a rozhraní definovaného v rámci popisu v jazyce Handel-C.

Následující příklad se snaží demonstrovat, jakým způsobem je realizované propojení na základě shody jména implementované.

Příklad

Definice rozhraní v rámci popisu v jazyce Handel-C.

```
// definice signálu použitých v Handel-C popise
unsigned 32 counter;
signal unsigned 32 data_outVal=0;
signal unsigned 10 addressVal=0;
signal unsigned 1 writeVal=0;
signal unsigned 32 data_inVal;
set clock = external "CLK";

// Definice rozhraní
interface
  // jméno entity, která má být z VHDL popisu připojena
  reg32x1k
     // jméno výstupního portu připojované VHDL entity
    (unsigned 32 data_out)
     // název instance v Handel-C
  registers
     // jména vstupních portů z připojované VHDL entity
    (unsigned 1 clk = __clock,
     unsigned 10 address = addressVal,
     unsigned 32 data_in = data_inVal,
     unsigned 1 write = writeVal) with {busformat = "B_I"};

// hlavní program
void main(void)
{
    counter ++;
    addressVal = counter[9:0];
    data_outVal = registers.data_out;
}
```

3.2.2.1 Generovaný HDL popis

Po provedení kompilace stejným způsobem jako při popisu s VHDL entitou na nejvyšší úrovni, tedy pokud je použit kompilátor z příkazového řádku:

handelc –vhdl handelEntity.hcc –o Top-level.vhd

je do .vhd souboru *handelCEntity_hcc.vhd* vygenerována komponenta s názvem *reg32x1k* a propojení s touto komponentou. Pokud již tedy máme připraven popis samotné entity *reg32x1k* i s popsaným chováním, můžeme tento popis snadno připojit ke generovanému VHDL popisu bez nutnosti provádění změn.

Následující příklad ukazuje, jak vypadá generovaná komponenta v jazyce VHDL.

Příklad

```
-- Deklarace externí entity
COMPONENT reg32x1k
     PORT (
          address : IN unsigned(9 DOWNTO 0);
          clk : IN std_logic;
          data_in : IN unsigned(31 DOWNTO 0);
          data_out : OUT unsigned(31 DOWNTO 0);
          write : IN std_logic
     );
END COMPONENT;
```

3.2.3 Vytvoření top-level VHDL entity s reset a clock signálem

Prakticky vždy potřebujeme získat top-level VHDL entitu, která obsahuje generované signály pro hodiny a reset. Tato entita je automaticky tvořena při překladu Handel-C kódu. Je však třeba deklarovat tyto signály na úrovni Handel-C.

```
interface port_in(unsigned 1 USER_CLOCK with {clockport = 1}) clock_in();
set clock = internal clock_in.USER_CLOCK;
```

Nebo

```
interface port_in(unsigned 1 USER_RESET) reset_in();
set reset = internal reset_in.USER_RESET;
```

Výsledná top-level entita poté obsahuje tyto porty:

```
ENTITY test_top IS
   -- popis rozhraní
   PORT (
     USER_CLOCK : IN std_logic;
     USER_RESET : IN std_logic
   );
END;
```

4 Možnosti využití

S ohledem na výše zmíněnou specifikaci jazyka Handel-C (specifikace není úplná, vybral jsem z mého pohledu to nejdůležitější, více lze najít v [1] a [2]) lze uvažovat o začlenění překladače a samotného jazyka Handel-C do procesu generování HDL popisu z jazyka ISAC.
V souvislosti s jazykem ISAC přichází využití jazyka Handel-C v úvahu při generování HDL popisu z BEHAVIOR sekce. V následující podkapitole se snažím nastínit možný způsob využití jazyka Handel-C při generování HDL popisu z jazyka ISAC.

4.1 Generátor HDL popisu a jazyk Handel-C

Zde se nabízí několik otázek, jejichž odpověděním současně přinesu návrh řešení.

- **Bylo by možné provést generování úplného popisu architektury z jazyka ISAC do jazyka Handel-C?**

  Generátor HDL popisu je připraven generovat HDL popis v jazycích VHDL a Verilog. Jazyk Handel-C se od těchto jazyků značně liší, zejména v úrovni abstrakce popisu. Za současného stavu si myslím, že popis v jazyce Handel-C není možné generovat pro celou popisovanou architekturu v jazyce ISAC. Generování popisu v jazyce Handel-C by ale mohlo být uplatněno na BEHAVIOR sekci jazyka ISAC.

- **Lze nahradit současný jazyk ANSI-C v sekci BEHAVIOR jazykem Handel-C?**

  Jak bylo zmíněno v kapitolo 3.1, lze tuto změnu provést. Pro sekci BEHAVIOR však bude jazyk ANSI-C ponechán, neboť je jazykem obecnějším, než jazyk Handel-C. Z kapitoly 3.1 vyplývá, že by bylo možné provést jistou transformaci z jazyka ANSI-C do jazyka Handel-C s několika málo změnami provedenými ve stávající podmnožina jazyka ANSI-C, která je v BEHAVIOR sekci použita.

- **Jak začleněnit generování popisu v jazyce Handel-C do stávajícího generátoru HDL popisu z jazyka ISAC.**

  Zde se nabízí využití možností integrace jazyka Handel-C s jinými HDL jazyky jako je VHDL či Verilog. Nejlepším řešením se zdá být využití integrace, kde VHDL popis tvoří top-level entitu návrhu, neboť

  - BEHAVIOR sekci jazyka ISAC lze chápat jako funkci jazyka ANSI-C. Ve skutečnosti tak k této sekci přistupuje i simulátor. Její kód lze převést na funkci jazyka Handel-C. Z této funkce může být poté generována s pomocí překladače *handelc* entita jazyka VHDL, která bude představovat funkční jednotku z pohledu procesorové architektury.

- Pro generovanou funkci jazyka Handel-C je třeba specifikovat rozhraní, tedy vstupní a výstupní porty přes které bude komunikovat s okolními entitami, které již jsou generovány (zdrojové prvky, kontrolér a dekodér). Rozhraní je možné generovat z P-grafu, který obsahuje informace o propojení jednotlivých entit. V P-grafu by Handel-C funkce představovala funkční jednotku, která má vstupní a výstupní signály.
- V průběhu generování Handel-C popisu je třeba provést analýzu názvů všech zdrojových prvků použitých v popise BEHAVIOR sekce. Z názvů zdrojových prvků jsou generovány signály, jejichž názvy je nutné použit pro definici rozhraní. Následující příklad ukazuje, jak by mohla vypadat transformace ANSI-C kódu, kde je použit zdrojový prvek jazyka ISAC.

Příklad

ANSI-C kód

```
res[dst] =  reg1[src] & 10;
```

Handel-C kód

```
interface port_in
   // reg1_src_DataIn – název portu použitého ve VHDL
   (unsigned 4 reg1_src_DataIn)
   // název instance portu
   ReadData
   // nejsou definované výstupní porty
   ()
   // standardní logický typ v jazyce VHDL
   with {vhdl_type = "std_logic_vector"};

interface port_out
   // nejsou definované vstupní porty
   ()
   // jméno instance
   WriteData
   // bitový výstupní port, jméno použité ve VHDL
   (unsigned 4 res_dst_DataOut = Bytes_out_res_dst)
   // standardní logický typ v jazyce VHDL
   with {vhdl_type = "std_logic_vector"};

   Bytes_out_res_dst =  ReadData.reg1_src_DataIn & 10;
```

Samotný příkaz přiřazení tedy zůstane, jen je třeba doplnit definici rozhraní a použít jiná jména identifikátorů.

Druhý způsob integrace, kde by top-level entitu tvořil popis v jazyce Handel-C prozatím nemá smysl uvažovat, neboť nejsme nyní schopni generovat celý popis architektury v jazyce Handel-C. Museli bychom generovat časování a řízení architektury v jazyce Handel-C.

5 Závěr

Lze říci, že využití jazyka Handel-C v rámci generování HDL popisu v jazyce VHDL či Verilog z jazyka ISAC by mohlo být přínosné. Zvláště s využitím překladače *handelc* od firmy Celoxica, který lze bez problému používat bez grafické nadstavby. Překladač by mohl být začleněn jako samostatný nástroj, který by komunikoval či spolupracoval s generátorem HDL kódu jazyka ISAC. Otázkou zde však zůstává, zda je možné překladač od firmy Celoxica využít v rámci celého projektu jazyka ISAC, neboť je to nástroj třetí strany.

S využitím tohoto nástroje by bylo možné tedy do stávajícího generátoru HDL kódu doplnit generování popisu funkčních jednotek a získat tak již téměř kompletní generátor. Popis v jazyce Handel-C by nebyl výstupem samotného generátoru, zde by zůstal výstup v jazyce VHDL. Popis v Handel-C jazyce by tvořil jen dočasný mezistupeň pro generování cílového VHDL popisu a po ukončení procesu generování by mohl zaniknout.

Do budoucna zůstává otázkou, zda popis v Handel-C kódu nezachovat a nesnažit se o generování kompletního popisu architektury v jazyce Handel-C.

Literatura

[1] HRUŠKA T., *Instruction Set Architecture C*, interní dokumentace FIT VUT Brno, 2004

[2] *Handel-C Language Reference Manual*, dokumentace k vývojovému prostředí DK Suite 5.0, Celoxica, www.celoxica.com.

[3] *DK Design Suite User Guide,* dokumentace k vývojovému prostředí DK Suite 5.0, Celoxica, www.celoxica.com, 2005

[4] Šída J. *Návrh číslicových obvodů pomocí Handel C - Bakalářská práce.* Praha: ČVUT v Praze, Elektrotechnická fakulta, 2008