Icarus Verilog: egy éves késéssel itt a nyílt forrású Verilog

Stephen és Michael a Verilog-tervezéssel kapcsolatos részleteket ismerteti.

Körülbelül 16 hónappal ezelőtt, 2001 februárjában egy ***Stephen Williams***-szel készült beszélgetés keretében egyszer már bemutattuk a Linux Journal oldalain (lásd a ➲ http://www.linuxjournal.com/article/4428-on található cikket) a nyílt forráskódú elektronikus tervezési automatizmusok (Electronic Design Automation, azaz EDA) világát.

Az Icarus Veriloghoz hasonló projektek számára a nyílt forrású fejlesztési módszer számtalan előnnyel jár. Ez a cikk a Verilog segítségével történő tervezésről egy kicsit több módszertani részletet tartalmaz, illetve felfedi az Icarus Verilog fordító alapjainak néhány részletét. Ezenkívül a cikk végén felsorolunk néhány igen kiváló Veriloggal foglalkozó művet, illetve olyan webcímet, ahol jó néhány egyéb nyílt forrású EDA-projektről olvashatunk.

Az Icarus Verilog parancssoros eszköz, amely a Verilogban íródott tervezési forrást fordítja le a célformátumra. Ez a célformátum általában a `vvp` szimulációs motor, a parancssorban azonban más célformátumot is kiválaszthatunk.

Első példánk az ismerős „Szia, Világ!” program lesz:

```
module proba;
initial $display("Szia, vilÆg!");
endmodule
```

Ezt értelemszerűen a következő parancssorozat fordítja le és hajtja végre:

```
% iverilog -oa.vvp hello.v
% vvp a.vvp
Szia, vilÆg!
```

Természetesen a mérnök gyorsan valamilyen érdekesebb példára szeretne továbblépni, ami valamilyen tervezési feladatot old meg. Erre jellemző, egyszerű példa a számlálómodell próbája (lásd az 1. listát: 40. CD Magazin/Verilog könyvtár). A ***szia.v*** és a ***szamlalo1.v*** példákban a fordító egyetlen forrásfájlt kapott meg, amelyet `vvp` formátumú kimeneti fájllá alakított, majd az így elkészített fájlt a `vvp` program hajtotta végre.

A Verilog programokban a modulok azok az objektumtípusok, amelyeket a tervező az eszközök modellezésére hoz létre.

A modulok más modulokat is hívhatnak, példányosíthatnak, illetve saját kódot is tartalmazhatnak, amellyel a modellezendő eszközt írhatják le. A tervező ezután a fő modult hívja meg a teljes modellezett eszköz példányosításához.

A Verilog-fordítók általában úgy állapítják meg, hogy az adott tervben melyek a fő modulok, hogy megvizsgálják, mely modulok nem voltak máshol példányosítva a programozó által adott forrásban. A ***szia.v*** példában mindössze egyetlen `proba` nevű modul volt, ez képezte tehát a fő modult. A ***szamlalo1.v*** programban a `szamlalo` modult a `proba` példányosította, és kizárólag a `proba` volt az egyetlen, amelyet máshol nem példányosítottunk, így a `proba` lett a főmodul.

Az Icarus Verilog használata során a programozók ezt a főmodul-azonosító heurisztikus módszert engedélyezhetik, de a kívánt főmodulokat a `-s` kapcsoló segítségével közvetlen módon is felsorolhatják.

Ahogy növekszik a program, a programozónak egyre inkább több fájlból álló forrás- és programkönyvtárak készítésére lesz szüksége. A programkönyvtárak igen hasznosak a más fejlesztők kiadta piaci eszközöket modellező modulok terjesztésében is. Az Icarus Verilog az önműködő programkönyvtárakat hordozható módon támogatja.

Az önműködő programkönyvtár-formátum ipari szabvány. Ez tulajdonképpen egy olyan Verilog-forrásfájlokat tartalmazó könyvtár, ahol a fájlok egy-egy modult tartalmaznak, és a fájl neve a benne található modulnak felel meg. Például a ***szamlalo.v*** a `szamlalo` modult tartalmazza. A könyvtár helyét az Icarus Verilog-fordítónak parancssorból a `-y` kapcsolóval, vagy parancsfájl használatával adhatjuk meg.

Önműködő programkönyvtárakat használva ***szamlalo1.v*** példánk két részre törik: a könyvtármodulra és a főprogramra. A mostani példában az a logikus, ha a `counter` modult mozgatjuk a ***lib/szamlalo.v***-be és a `proba` modult tartjuk meg a ***szamlalo2.v*** programban. A 2. *listában* megfigyelhetjük, miképpen kell a ***szamlalo2.v***-t két részből álló módon használni.

Az Icarus Verilog a ***szamlalo2.v*** programot először megpróbálja lefordítani. Amikor a `szamlalo` modul példányosításához ér, észreveszi, hogy nincsen `szamlalo` modulmeghatározás, ezért végignézi az általa ismert (`-y` kapcsolóval megadott) könyvtárakat, és megtalálja a ***lib/szamlalo.v*** fájlt. Értelmezi az új Verilog-forrást, menti a megtalált modulmeghatározásokat, és folytatja az eredeti forrás fordítását. A programkönyvtárban való keresés rekurzív, így a könyvtármodulok más könyvtármodulokat is példányosíthatnak, miközben a fordító a modulmeghatározásokat folyamatosan gyűjti, míg a program el nem készül.

Az Icarus Verilog további kényelmi szolgáltatása, hogy parancsfájlokat képes feldolgozni. A parancsfájlok olyan szöveges állományok, amelyek fájlneveket, útvonalmegadásokat és más fordítási irányelveket tartalmaznak. A ***szamlalo2.v*** példánkhoz ilyesféle parancsfájlt tudnánk készíteni:

```
szamlalo2.txt:
  # ez egy sajÆt k nyvtÆr
  -y lib
```

2. lista A szamlalo2.v példa

```
\begin{verbatim}
module proba;

/* Egyszer pulzÆl  reset kØsz tØse. */
   reg reset = 0;
   initial begin
      # 17 reset = 1;
      # 11 reset = 0;
      # 29 reset = 1;
      # 11 reset = 0;
      # 100 $stop;
   end

/* hagyomÆnyos pulzÆl   ra. */
   reg clk;
   always #5 clk = !clk;
   wire [7:0] value;
   szamlalo c1 (value, clk, reset);


   initial $monitor("At time %t, value
                    ➥= %h (%0d)",
                    ➥$time, value, value);
endmodule // test

\end{verbatim}
\caption{Example Verilog simulation file
  \texttt{szamlalo2.v}}\label{fig2}

    % iverilog -o a.vvp -ylib szamlalo2.v
    % vvp a.vvp
    [...]
```

```
  # az ehhez a beÆll tÆshoz tartoz  projekt-
  #  forrÆsfÆjlok
  szamlalo2.v
```

Ezután a programot a következőképpen futtathatnánk:

```
% iverilog -c szamlalo2.txt -o a.vvp
% vvp a.vvp
```

Egy ilyen kis program esetében a parancsfájl használatának nincs túl sok értelme, de ahogyan a terv egyre nagyobbá válik (forrásfájlok százai sem ritkák), a parancsfájl értéke igencsak megnő.
Az Icarus Verilog rendelkezik egy más Verilog-fordítókban fel nem lelhető különleges képességgel is: támogatja a betölthető cél-API-kat. A betölthető modulokhoz készült C API-t a fordító akkor hívhatja meg, amikor a kimenetet új formátumban kell előállítania. A vvp kódelőállító maga is ilyen betölthető célmodul, amely a(z alapértelmezett) vvp-szimuláció kérése esetén indul el. Létezik egy `null` kódelőállító és egy `fpga`-kódelőállító is.
A betölthető cél-API a C-programozók számára az Icarus Verilog-csomagokkal együtt érkező és települő ***ivl_target.h header*** fájlon keresztül érhető el. Ezáltal a C-programozók új kimenet-előállítókat írhatnak az Icarus Verilog fordítóhoz.

A vvp futtatható állomány szintén támogatja a szabványos Verilog VPI-csatolófelület egy részét. Itt egy olyan futásidejű C API-ról van szó, amely a programozóknak lehetővé teszi, hogy a Verilog-forrás által meghívható új rendszerfeladatokat illesszenek be. Ilyen rendszerfeladatokra példa a `$stop` és a `$monitor` utasítás a ***szamlalo2.v*** példa forrásfájljában. Minden szabványos magrendszerfeladat Icarus Verilogban íródott, a VPI API segítségével.

Linux Journal 2002. július, 99. szám

Stephen Williams
egy szép napon eltévedt egyetemének számítógéptermében. Bár elektromérnök-hallgató volt, informatikusként (computer science) végzett. Tíz évvel később, miközben számos programfejlesztésben részt vett – különös tekintettel a nagy és erősen osztott rendszerekre –, egyszer csak azon kapta magát, hogy nagysebességű digitális kamerák vezetékeibe gabalyodik. Azóta e tapasztalatokat ötvözve eszközmeghajtókon, beágyazott rendszereken és EDA-eszközökön dolgozik.

Michael Baxter
kilencéves kora óta dolgozik a számítógépiparban, amióta magával ragadta az 1969-es 2001-vízió, az Ürodüsszeia. Tapasztalt számítógép-, rendszer-, áramkör- és FPGA-tervező. Tíz USA-szabadalmat jegyez számítógépek és áramkörök tervezésével kapcsolatban, valamint további ötnek társfeltalálója. Szeret túrázni, érdekli az amatőr rádiózás és a Lisp nyelven való programozás.

Hasznos Verilog-források

IEEE Std. 1364-1995 (ISBN 1-55937-727-5)
➲ http://standards.ieee.org
IEEE Std. 1364-2001 (ISBN 0-7981-2806-6)
➲ http://standards.ieee.org
Verilog 2001: A guide to the New Features of the Verilog Hardware Description Language by *Stuart Sutherland*. Kluwer Academic Publishers, 2001. ISBN 0-7923-7568-8.
The Verilog PLI Handbook: A User's Guide and Comprehensive Reference on the VERILOG Programming Language Interface by *Stuart Sutherland*. Kluwer Academic Publishers, 1999. ISBN 0-7923-8489-x.
Verilog Quickstart, Second Edition by *James M. Lee*. Kluwer Academic Publishers, 1997. ISBN 0-7923-8515-2.

Néhány nyílt forrású EDA-honlap

gEDA ➲ http://www.geda.seul.org
Icarus Verilog ➲ http://icarus.com/eda/verilog
Open Collector ➲ http://www.opencollector.org
OPENCORES.ORG ➲ http://www.opencores.org