Работа с преобразуватели (транслатори) от един език за описание на апаратната част в друг. Verilog – основни ключови думи и тяхното използване.

Езиците за описание на апаратната част биват много и най-различни видове - VHDL (Very High Speed Integrated Circuit Description Language), Verilog, SystemC, ABEL (Advanced Boolean Expression Language), AHDL (Altera HDL), Handel-C, Lola и т.н. Най-често използваните са VHDL, Verilog and SystemC. Трудно е човек да има познания и да използва всички от тези езици. Най-често един инженер използва само един език или най-много два от тях. Както знаем от предходните лекции проектирането на дадено цифрово устройство или система става като се използват IP (Intellectual Property, интелектуална собственост) модули. Възможно е модулът, който ще се използва да бъде описан на друг (различен) език от този който използва проектанта. В този случай от полза би било да се използват пребразуватели (транслатори) от един език за описание на апаратната част в друг. По-долу ще бъде дадена кратка информация за съществуващите преобразуватели и за техните предимства.

1. Работа с преобразуватели (транслатори) от един език за описание на апаратната част в друг.

Като предимства на преобразувателите (транслаторите) от един език за описание на апаратната част в друг могат да се посочат следните неща:

- Удвояване на пазарните позиции на фирмата чрез използване на двата езика VHDL и Verilog. Вместо да се хабят седмици време на инжинерите да разработва устройства и системи на втори език (този който те не използват често) може да се използват v2v транслатори, които ще свършат същата работа за много по-малко време;

- Спестява се време за поддръжка на описаните устройства и системи чрез използването на един език за описание на апаратната част като се преобразуват само направените промени в проекта и се използва v2v конвертор.

- v2v транслаторите са добре разработени и позволяват много качествено преобразуване от един HDL (Hardware Description Language, език за описание на апаратна част) в друг. Конвертор, който преобразува всички възможни езикови конструкции би бил голям по обем и с висока цена. Част от получения HDL код (обиквено много малка или липсваща такава) ще се наложи да се преобразува от самия специалист, което от своя страна изисква поне частично познаване на втория език за описание на апаратната част;

- Когато се използва голяма и сложна HDL програма използваните коментари са от важно значение за разбирането и поддръжката на модула. V2V транслаторите обикновено запазват коментарите;

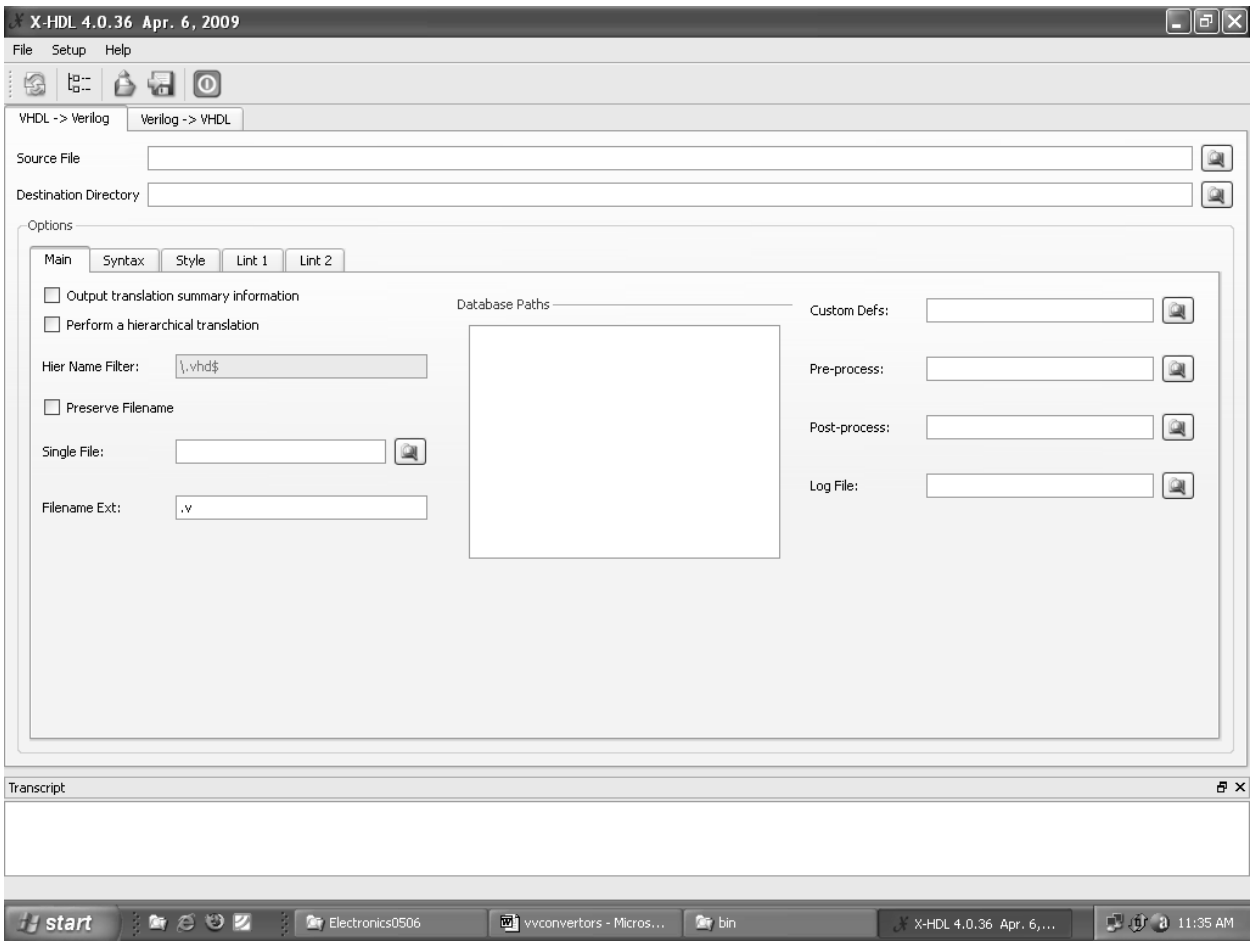
- Не без значение при използването на гореспоменатите преобразуватели е лесния начин на тяхното използване;

След споменатите предимства на преобразуватели (транслатори) от един език за описание на апаратната част в друг, на следните редове са дадени примери с такива транслатори разработени от различни фирми работещи в областта на проектирането на СГИС.

- http://www.ocean-logic.com/ – **Ocean Logic Pty Ltd., Australia.**
- http://www.syncad.com/ – **SynaptiCAD Inc., USA.**
- http://www002.upp.so-net.ne.jp/morioka/v2v.html – **Japan.**
- http://www.x-tekcorp.com/ – **X-Tek Corporation, USA.**
- http://www.veritools-usa.com/ – **VeriTools Inc.,USA.**

Като пример за използването на такъв конвертор съм избрал X-HDL продукт на фирмата X-Tek Corporation.

По-долу на фигурата е даден как изглежда прозорец на транслатора.



С него се работи по следния начин. Първо е необходимо да се избере какъв тип преобразуване ще се извърши от VHDL ->Verilog или Verilog -> VHDL. След това трябва да се избере файла, който ще се преобразува и се посочи пътят състоящ се от папки и подпапки до него (полето Source file). Необходимо и да се посочи къде да се постави преобразувания файл (Destination Directory). След като е направено всичко това се щраква с левия бутон на мишката върху бутона Преобразувай (Translate), който представлява две стрелки в зелен цвят и се намира под менюто Fail. По време на процеса на преобразуване на кода в прозореца Transcript се извежда информация за това как е протекъл той – успешно или с грешки. Ако има грешки в процеса на траслация те трябва да се коригират, след което отново да се стартира процеса по преобразуване.

Когато продукта X-HDL не е лицензиран той може да се използва в демонстрационен режим. За този режим е характерно, че могат да се преобразувт файлове с големина до 1кВ. Ако е необходимо да се транслират по-големи файлове може да се поиска 10-дневен безплатен лиценз за работа от фирмата производител.

2. Verilog – основни ключови думи и тяхното използване

Тъй като по време на лекции и лабораторни упражнения по дисциплината “Системи с програмируема логика” е представен VHDL то ако използваме v2v конвертор ще получим Verilog файл. В тази връзка е добре да познаваме някои основни ключови думи от този език за да можем да се ориентираме в проекта. Разбира се, за да сме сигурни, че транслацията е успешна транслирания модул или система може да да се подложи на проверка.

За Verilog е характерно, че при описанието се има в предвид дали са използвани малки или големи букви (case sensitive). Основните елементи на езика са:

-Модули (Modules)

-Поведенческо описание (Behavioral modeling)

-Непрекъснато присвояване (Continous assignments)
-Поддържа се йерархия (Hierarchy)
-Задаване на отделни компоненти (Component instantiation)
-…

Verilog описва цифровите системи като съвкупност от модули. Всеки модул се състои от сигнали за връзка (интерфейс), а освен това съдържа и описание на функцията, която реализира. Пример за описание на модул използвайки езика Verilog, е показан по-долу:

```
module Nand (q, a, b);
 output q;
 input a, b;

 nand (q, a, b);
endmodule
```

При описанието на логическите елементи Verilog използва предварително зададени (примитивни) функции:

-and, nand, nor, or, xor, xnor (стандартни логически функции)
-buf (буфер)
-not (инвертор)
-bufif0, bufif1, notif0, notif1 (буфери и инвертори съдържащи вход за привеждането им във високоимпедансно състояние)
-nmos, pmos, cmos, rnmos, rpmos, rcmos, tran, tranif0, tranif1, rtran, rtraninf0, rtranif1, pullup, pulldown (модели на транзисторни ключове)

За стандартните логически функции е характерно, че те имат един изход или двупосочен порт и различен брой входове. Изхода е първия по ред от описаните входове и изходи на логическия елемент. Що се отнася до буферите и инверторите те могат да имат произволен брой изходи. Синтактично при описанието на логическите елементи се използва следната конструкция

<GATETYPE> <drive_strength> <delay> <gate_instance>, <gate_instance> ;
където <GATETYPE> е една от примитивните функции описваща логическия елемент, <gate_instance> се състои от <name_of_gate_instance> (<terminal>, <terminal>)
като <name_of_gate_instance> се състои от идентификатор. Пример е даден на следващите редове

```
nand (strong0, strong1) #3
        Name1 (Out1, Ain, Bin),
        OtherName (Cout, x12, x1);
```

където чрез символа # е зададено закъснение.

Когато е зададен параметъра drive strenght (коефициент за разклонение по изход, товароспособност) и закъснение, то те важат за всички зададени логически елементи в списъка, отделени един от друг със запетати. Ако е небходимо да се въведат различни параметри за всеки един от логическите елементи трябва задаване на логическия елемент да завършва със символа ;. При описанието на цифровите блокове се използват сигнали наречени nets. За тях е характерно, че те не запомнят логическите стойности, които се предават по тях. Примерът е

wire #3 x2, където е зададен свързващ сигнал от тип wire със закъснение 3 единици и товароспособност 2 входа.

При описанието на логическите блокове при Verilog може да се използва и ключовата дума reg, с което се задава сигнал от тип регистър. Примери са дадени на следващите редове

reg TempBit; , където е зададен едно битов регистър
reg [15:0] TempNumber; , където е заден 16 битов регистър

При задаването на логическия модул и по-точно при формирането на неговия интерфейс се използват входни, изходни и комбинирани (inout) типове портове. Те могат да бъдат еднобитови сигнали или магистрали (сигнали с различна разрядност).

Беше споменато по-горе, че Verilog има особеност за последователно присвояване. Пример е даден на следващите редове

```
module GateLavel12 (Cout, Sum, A, B, Cin);
 output Cout, Sum;
 input A, B, Cin;

 assign Sum = A ^ B ^Cin,
        Cout = (A&B) | (B & Cin) | (A & Cin);
Endmodule
```

Ако някоя от стойностите на входа се промени, то ще бъде изведена новополучената изходна стойност. Логическите оператори, които се използват от Verilog са показани в следващата таблица.

Означение	**Логическа функция**	**Описание на действието на лог.функция**
~	Побитово лог. отрицание	Извършва лог. отрицание побитово между операндите
&	Побитово лог. умножение	Извършва лог. умножение побитово между операндите
~&	Побитова лог. ф-я И-НЕ	Извършва лог. фукция И-НЕ побитово между операндите
|	Побитова лог. ф-я ИЛИ	Извършва лог. събиране побитово между операндите
~|	Побитова лог. ф-я ИЛИ-НЕ	Извършва лог. фукция ИЛИ-НЕ побитово между операндите
^	Побитова лог. ф-я сума по модул 2	Извършва лог. фукция сума по модул 2 побитово между операндите
^~ или ~^	Побитова лог. ф-я еквивалентност	Извършва лог. фукция еквивалентност побитово между операндите
&	Редуцираща лог. функция И	Получава се 1 бит, който е в резултат на лог.функция И на всички битове на операнда
|	Редуцираща лог. функция ИЛИ	Получава се 1 бит, който е в резултат на лог.функция ИЛИ на всички битове на операнда
~|	Редуцираща лог. функция ИЛИ-НЕ	Получава се 1 бит, който е в резултат на лог.функция ИЛИ-НЕ на всички битове на операнда

^	Редуцираща лог. функция сума по модул 2	Получава се 1 бит, който е в резултат на лог.функция сума по модул 2 на всички битове на операнда
^~ или ~^	Редуцираща лог. функция еквивалентност	Получава се 1 бит, който е в резултат на лог.функция еквивалентност на всички битове на операнда

Представените логически операции имат приоритет т.е. при написани няколко логически оператора някои се изпълняват по-напред от другите. С най-висок приритет са логическите операции за редукция (намаляване) на разрядността!, &, ~&, |, ~|, ^, ~^, +, -, с по-нисък приоритет са *, /, % , а след тях са +, -, а още по-нисък приоритет имат << >>, като списъка продължава с < <= > >=, следващата група е== != === !==, по-следващата & ~& ^ ~^. На последните три места по приоритет са | ~|, следващата група е &&, предпоследната група е ||, а на последно място по приоритет е ?.

Езикът Verilog позволява да се задават числови константи използващи десетичната, шестнадесетичната, осмичната или двоичната бройни системи. Числата могат да се представят като десетично число без да е зададена неговата големина. Вторият начин задава размера на константата и изглежда по следния начин:

ss…s ‘f nn…n,

където ss…s е размера в брой битове на константата. Размера е зададен като са използвани десетични числа.
‘f задава използваната бройна система, като използваните символи са – d – за десетична бройна система, h – шестнадесетична, o-осмична и b за двоична бройна система. Използваните спецификатори могат да бъдат и главни букви.
nn…n е стойността на константата. За шестнадесетична бройна система, използваните букви от A до F ногат да бъдат изписани с главни букви. Позволено е между цифрите да се вмъква и символа за долно тире с оглед по-лесно да се представи съответното число., като символът долно тире не може да бъде първи символ от числовата константа.На следващите редове са представени примерни числа използващи първия или втория начин на представянето им.

С незададен размер в битове
792 // Десетично число (след символа // се въвежда коментар)
‘h 7d9 // Шестнадесетично число
‘o 7746 // Осмично число

С точно определен размер в битове
12 ‘h x // 12 битово число
10 ‘d 17 // десет битова константа със стойност 17
4 ‘b 110z //четири битово число

В езика Verilog се използват т.нар. инициализиращ израз – това е израз, който описва процес, който се изпълнява еднократно. Използва се за първоначално задаване (инициализиране) на променливи или сигнали, преди да е започнало същинското описание на цифровия блок. След като веднъж се изпълни инициализиращия израз след това остава неактивен. Пример за използването на инициализиращ израз е показан на следните няколко реда.

```
module ffNandSim;
reg preset, clear;
```

```
…
initial
        begin
        #10 preset = 0; clear = 1;
        #10 preset = 1;
        #10 clear  = 0;
        #10 clear = 1;
        end
endmodule
```

Основно във Verilog се използва ключовата дума always за задаване на процес, като това се повтаря непрекъснато без спиране или без възможност за излизане от него. При поведенческо описание използвано при Verilog може да съдържат един или няколко always израза. Модул, който не съдържа always израз е модул, който описва йерархична структура. Пример за always израз е показан на следните редове.

```
module ClockGen (Clock);
        output Clock;
        reg Clock;

        initial
        #5 Clock = 1;
        always #50 Clock = ~ Clock;
        endmodule
```

Във Verilog символа за промяна (събитие) е @. Има възможност да се следи дали е настъпила промяна в логическата стойност на определен вход (определен фронт на сигнала) и ако това е така да се започне да се изпълнява определено действие. Пример за реакция на фронта на сигнала е показан във следния пример, който представлява D тригер описан с езика Verilog.

```
module DflipFlop (Q, Clk, Data);
        output Q;
        reg  Q;
        input Clk, Data;

        initial Q =0;
        always @(negedge Clk) Q = Data;
        endmodule
```

Възможно е да се следи положителния (нарастващия) фронт на сигнала

always @ (posedge Foo) Out1 =In2; или дори да се следят и двата фронта
@ (ControlSignal) Out2 = In1;

Промяната на сигнала, чийто фронт се следи, може да е изход на логически елемент, от тип wire или от тип reg. Могат да се следят и няколко сигнала, като промяната на само един ще доведе до изпълнение на определената последователност от действия.

@(posedge InputA or posedge Timer) …

Друга конструкция, която се използва при Verilog е wait конструкцията, която е чувствителна към логическото ниво на определен сигнал и се използва основно за синхронизацията на два конкурентни процеса. Пример е даден на следващите редове

```
module WaitModule (DataOut, DataIn, Ready);
        output [7:0] DataOut;
        input [7:0] DataIn;
        input Ready;
        reg Internal;

        always
                begin
                wait (Ready)
                Internal =DataIn;
                …
                end
                endmodule
```

При Verilog се използва директивата ‘define, която позволява да се задава стойност на определена текстова константа, като например

```
`define DvLen 15
`define DdLen 31
`define Qlen 15
```

При поведенческото описание, много често се използва if-then-else конструкцията. Може да използва или да не се използва else израза.Примерите са

```
        if (NegDivisor)
                Divisor = -Divisor

        if  (!Divident [`DdLen])
        Quotient = Quotient +1;
        Else …
```

Когато имаме няколко последователни if израза, else твърдението се отнася за най-близкия до него if оператор. Използват се и оператори за сравнение като > , >=, <, <=, == , !=, === (побитово равенство, изход TRUE или FALSE) !== (побитово неравенство, изход TRUE или FALSE) като операндите от тип reg или net се взимат без знак, а реалните и целите числа могат да съдържат и знак.

Оператор за условие може да се използва при програмите на Verilog , като начина на записването му е

:: = <expression> ? <expression> : <expression>,

като ако резултата от първия израз е TRUE, тогава стойността на оператора е второто твърдение, в противен случай стойността на изриза се определя от третия израз.

Verilog съдържа и четири оператора за цикъл – repeat, for, while , forever.
Ако се използва repeat цикъл, брояч за изпълнение на цикъла трябва да бъде посочен в скоби. Правилният формат на изписването му е
repeat (<expression>) <statement>, а пример за неговото използване е показан на следващите редове

```
repeat (`DvLen + 1)
begin
…
end
```

Операторът за цикъл for записва в следния вид

::=for (<assignment>;<expression>;<assignment>) <statement>, а примера е

```
for (i=16; i; i=i-1)
        begin
        /* statement(s)*/
end
```

Операторът за цикъл while се записва в следния вид

::=while (<expression>) <statement>

```
i=16;
while (i)
begin
//statement
i= i –1;
end
```

Операторът за цикъл forever се записва в следния вид

::=forever <statement>

```
always
        begin
                PowerOninitializations;
                forever
                        begin
                                FetchAndExecute
                        End
        end
```