

Latvijas Universitāte
Datorikas fakultāte

Aparatūras apraksta valodas. Verilog.

Kurss "ievads digitālajā projektēšanā"
Lekcija 11.11.2011

Autors: Rinalds Ruskuls

Lekcijas saturs

- HDL valodu pielietojums
- Verilog valodas pamati
 - Verilog programmas sastāvdaļas
 - Objekti
 - Datu tipi
- Ciparu shēmu veidošana izmantojot Verilog

Shēmu aprakstīšanas veidi

- Līdz šim iepazināties ar ciparu shēmtehnikas pamatiem!
- Ir arī citi veidi kā izveidot ciparu principiālo elektrisko shēmu:
 - Pielietojot programmēšanas valodas (**H**ardware **D**escription **L**anguages – ***HDL***)
 - Kombinējot kopā – HDL un shēmtehniku

Ciparu shēmas projektēšana izmantojot HDL valodu

```
module signed_adder
#(parameter WIDTH=16)
(
    input signed [WIDTH-1:0] dataa,
    input signed [WIDTH-1:0] datab,
    input cin,
    output [WIDTH:0] result
);

    assign result = dataa + datab + cin;

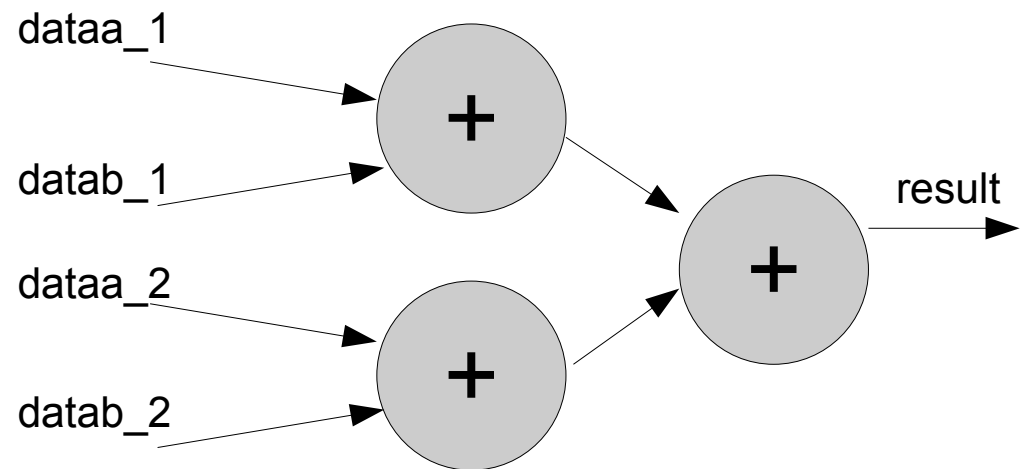
endmodule
```

Ciparu shēmas projektēšana kombinējot HDL valodas un shēmtehniku

a

Ir iespējams izmantot dažāda veida **HDL** valodas un izveidotos moduļus izmantot kā shēmas blokus!

```
module signed_adder
#(parameter WIDTH=16)
(
    input signed [WIDTH-1:0] dataa,
    input signed [WIDTH-1:0] datab,
    input cin,
    output [WIDTH:0] result
);
    assign result = dataa + datab + cin;
endmodule
```



Verilog

Kāpēc radās nepieciešamība pēc
programmēšanas valodām, kas ir paredzētas
ciparu elektronikai?

HDL kopīgais un atšķirīgais ar citām valodām

- Kopīgais ar klasiskajām programmēšanas valodām
 - Ir iespējams aprakstīt secīgas darbības
 - Ļauj strādāt dažādos abstrakciju līmeņos
- Atšķirīgais no klasiskajām programmēšanas valodām:
 - Ir iespējams aprakstīt procesus, kuri izpildās paralēli

HDL – hardware description language

- ***HDL*** valodas izmanto, lai formāli aprakstītu ciparu elektroniskas:
 - shēmu uzbūvi
 - shēmas darbību
 - testus darbības simulēšanai
- ***HDL*** iedalās:
 - VHDL
 - ***Verilog***
 - System Verilog

Verilog HDL

- Verilog valoda ir līdzīga **C/C++**, **Python** programmēšanas valodai
- Verilog valoda ir "case sensitive", visi rezervētie vārdi ir ar maziem burtiem

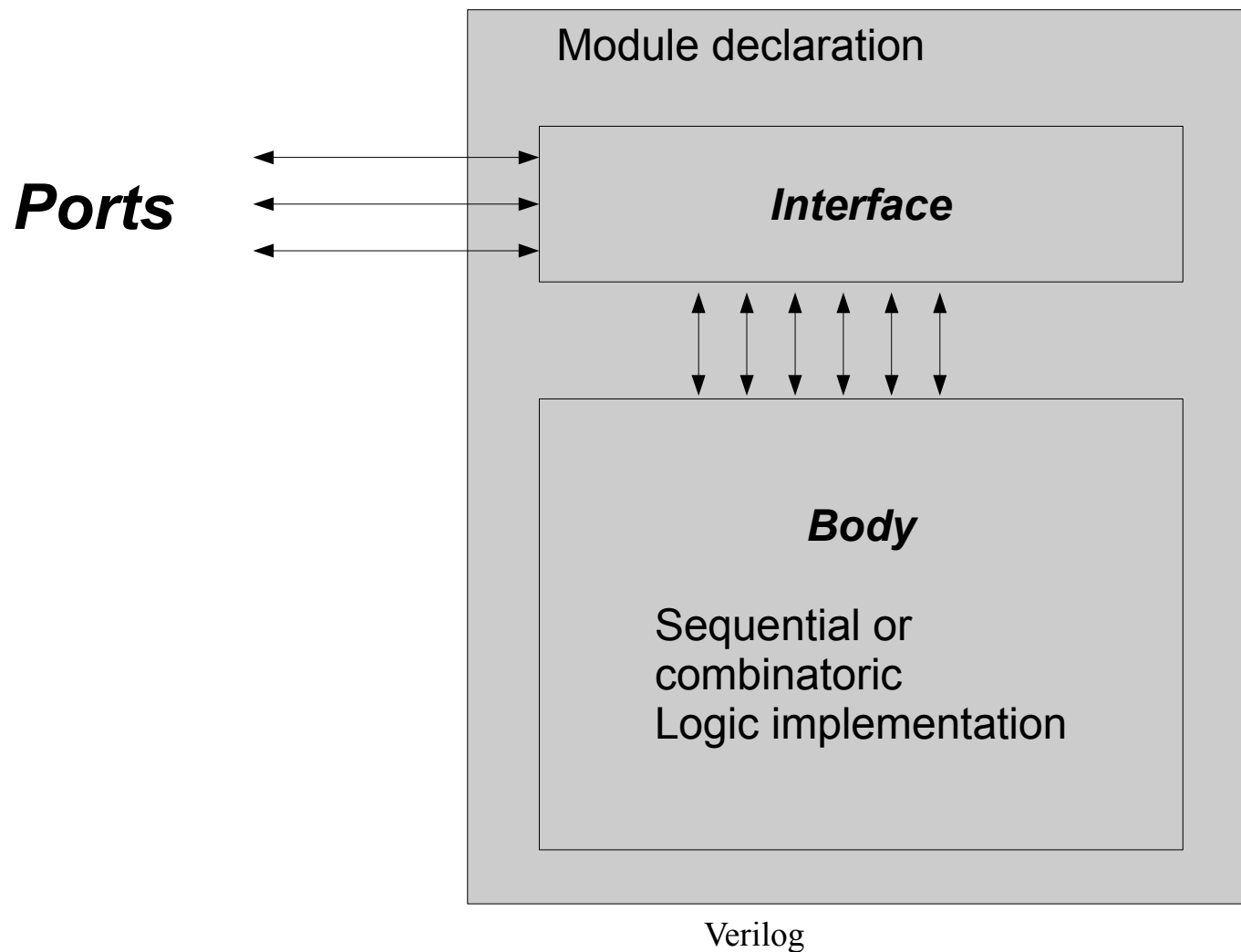
```
input // a Verilog Keyword  
wire // a Verilog Keyword  
WIRE // a unique name ( not a keyword)  
Wire // a unique name (not a keyword)
```

Verilog HDL

- Verilog koda iekomentēšana ir iespējama divos veidos:
 - Vienas rindas komentārs `//`
 - Bloka komentārs `/* */`
- Komentēšanas piemērs

```
module my_and (a, b, result)
    input a; // vienas rindas komentārs
    /* input b;   bloka komentārs
    output result; */
endmodule
```

Verilog satāvdaļas



Module declaration

- Pirmais solis ir definēt moduli (***module_name***) un izmantoto signālu nosaukumus (***sig_name_1, sig_name_2...***)
- Lietotāja definētiem nosaukumiem jāievēro šādi likumi
 - Jāsākas ar burtiem [a-z] vai [A-Z]
 - Nedrīkst sākties ar \$ vai [0-9]
 - ***Case sensitivity*** – my_sig ≠ My_sig

```
module module_name (sig_name_1, sig_name_2, ..., sig_name_n);  
.....  
endmodule
```

Module declaration

- Otrais solis ir definēt signālu virzienu un datu tipu
 - Signālu virzieni var būt – ieeja, izeja, divvirzienu
 - Datu tipi – bits, bitu vektors, utt.

```
module module_name (sig_name_1, sig_name_2,..., sig_name_n);  
    input sig_name_1;           // viens bits  
    Input [3:0] sig_name_2;    // bitu vektors – 4 biti  
    output sig_name_n;        // viens bits;  
endmodule
```

Module declaration - example

Pirmais solis ir definēt moduli un izmantoto signālu nosaukumus

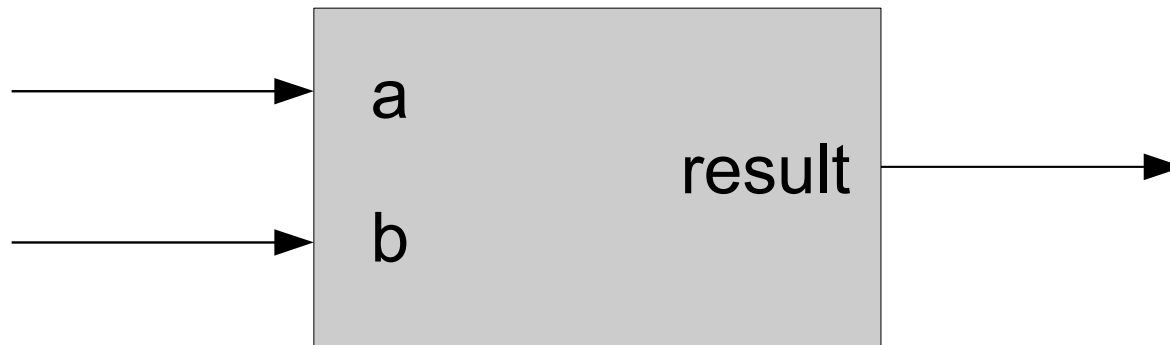
```
module my_and (a, b, result)
.....
endmodule
```

Otrais solis ir definēt moduļa ieejas un izjas.

```
module my_and (a, b, result)
    input a;
    input b;
    output result;
endmodule
```

Kas mums ir sanācis!?

- Izveidojām moduli, kuram ir:
 - Divas ieejas – a un b;
 - Viena izeja - result



Turpmākie darbi – definēt izveidotajam modulim tā veicamo funkcionalitāti

Funkcionalitātes definēšana

- Kombinatoriskas shēmas veidošana

```
module my_and (a, b, result)
    input a;
    input b;
    output result;
    assign result = a & b;
endmodule
```


Funkcionalitātes definēšana

- Kombinatoriskas shēmas veidošana

```
module my_and (a, b, result)
  input a;
  input b;
  output result;
  wire int_sig;
  assign int_sig = a | b;
  assign result = a & b ^ int_sig;
endmodule
```

Bitu līmeņa operandi

- & → AND
- | → OR
- ~ → NOT
- ^ → XOR
- ~^ vai ^~ → XNOR

Loģiskās operācijas

`&&` → AND

`||` → OR

`!` → NOT

Izejā ir viena bita vērtība – 0, 1 vai x

`A = 6;` `A && B` → `1 && 0` → 0;

`B = 0;` `A || !B` → `1 || 1` → 1;

`C = x;` `C || B` → `x || 0` → x;

Salikšanas operācija

- $\{op_1, op_2, ..\}$ - saliek signālu no vairākiem atsevišķiem signāliem

```
reg a;
```

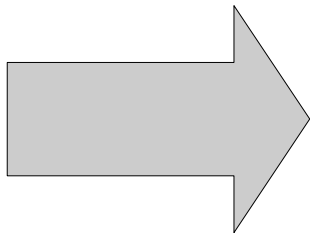
```
reg [2:0] b, c;
```

```
..
```

```
a = 1'b 1;
```

```
b = 3'b 010;
```

```
c = 3'b 101;
```



```
Catx = {a, b, c}; // 1_010_101
```

```
Caty = {b, 2'b11, a}; // 010_11_1
```

Funkcionalitātes definēšana

- Secīgas shēmas veidošana - skaitītājs

```
always @ (posedge clk or posedge rst)
  if (rst) // This causes reset of the cntr
    count <= {size{1'b0}};
  else
    if (cet && cep) // Enables both true
      begin
        if (count == length-1)
          count <= {size{1'b0}};
        else
          count <= count + 1'b1;
      end
end
```

```
// the value of tc is continuously assigned
```

```
// the value of the expression
```

```
assign tc = (cet && (count == length-1));
```

```
endmodule
```

Papildus uzdevumi mājās

- Patstāvīgi iepazīties ar Verilog valodu
 - Secīgu un paralēlu shēmu veidošanas principi
 - Izveidoto moduļu izsaukšana
 - simbola izveidošana no izveidotajiem moduļiem

Praktiskie darbi

- Strādājam pie kursa projekta
- Drīkst strādāt arī pie tiem mājasdarbiem, kuri vēl nav iesniegti

***Paldies par uzmanību!
Jautājumi?***