

TSEA44: Computer hardware – a system on a chip

Lecture 2: A short introduction to SystemVerilog

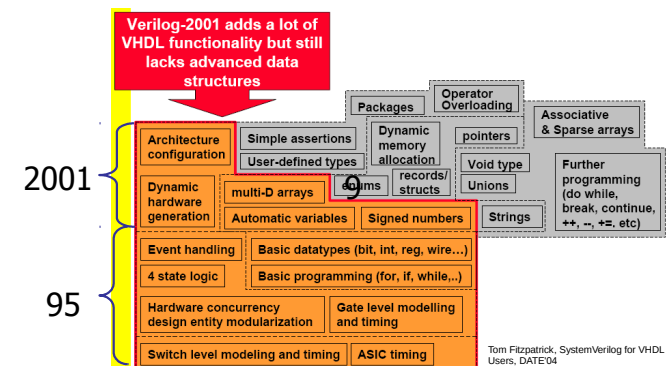
(System)Verilog

- Assume background knowledge of VHDL and logic design
- Focus on coding for synthesis
 - Testbenches will be mentioned
- Verilog was initially used for **modelling** of hardware, but later where software created that allowed the model to be synthesized into hardware
 - Adds restrictions on how the code should be written

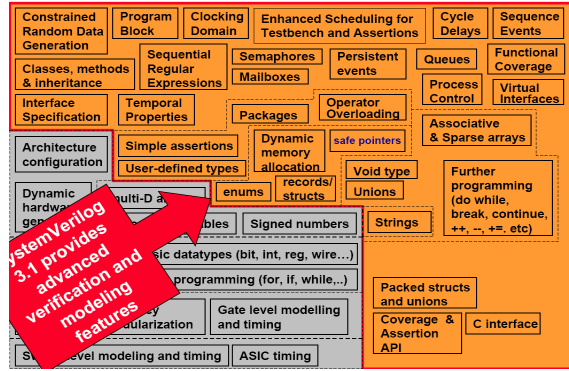
History of Verilog and SystemVerilog

- 1985 Verilog invented, C-like syntax, initial use: modelling of hardware
 - <- VHDL defined in 1987
- 1995 First standard of Verilog (Verilog-95, IEEE 1364)
- 2001 Extra features added (Verilog-2001, IEEE 1364-2001)
- 2005 Minor extension (Verilog-2005)
- 2005 SystemVerilog standardized (SystemVerilog-2005, IEEE 1800-2005) as an extension to Verilog-2005
- 2009 Merge of SystemVerilog and Verilog (IEEE 1800-2009)

Verilog vs VHDL



SystemVerilog



SystemVerilog 3.1 provides advanced verification and modeling features

Tom Fitzpatrick, SystemVerilog for VHDL Users, DATE04

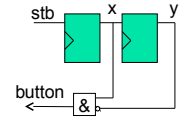
Synchronization and single pulse

- Asynchronous inputs must be synchronized to the input clock
- Useful: indicate input signal edge detection

```
reg x,y; // variable type (0,1,Z,X)
wire button; // net type (0,1,Z,X)
```

```
// SSP
always @(posedge clk) // procedural block
begin
x <= stb;
y <= x;
end
```

```
assign button = x & ~y; //continuous assignment
```



A variation of the same thing

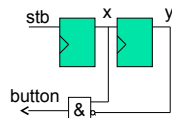
- Multiple always block

```
reg x,y; // variable type (0,1,Z,X)
wire button; // net type (0,1,Z,X)
```

```
// SSP
always @(posedge clk) // procedural block
begin
x <= stb;
end
```

```
always @(posedge clk) // procedural block
begin
y <= x;
end
```

```
assign button = x & ~y;
```



Resettable D-flipflop

```
// This is OK
always @(posedge clk)
begin
x <= stb;
if (rst)
x <= 0;
end
```

```
// same as
always @(posedge clk)
begin
if (rst)
x <= 0;
else
x <= stb;
end
```

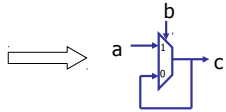
```
// This is not OK
// multiple assignment
always @(posedge clk)
begin
x <= stb;
end
```

```
always @(posedge clk)
begin
if (rst)
x <= 0;
end
```

SystemVerilog: always_{ff, comb, latch}

- always blocks does not guarantee capture of intent
- If not edge-sensitive then only a warning if latch inferred

```
// forgot else branch
// a synthesis warning
always @(a or b)
  if (b) c = a;
```



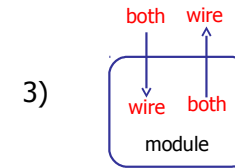
- always_ff, always_comb, always_lath are explicit
- Compiler now knows user intent and can flag errors accordingly

```
// compilation error
always_comb
  if (b)
    c = a;
else
  c = a;
```

```
// yes
always_comb
  if (b)
    c = a;
else
  c = d;
```

Reg or wire in Verilog

- 1) `always ...`
`a <= b & c;`
`reg both`
- 2) `wire both`
`assign a = b & c;`



SystemVerilog relaxes variable use

- A variable can receive a value from one of these:
 - Any number of always/initial blocks
 - One always_ff/always_comb block
 - One continuous assignment
 - One module instance
- We can skip wire/reg, use logic instead

Signed/unsigned

- Numbers in Verilog (95) are unsigned. If you write

```
// s and d2 4 bits long, d3 5 bit long
assign d3 = s + d2;
```

s and d3 gets zero-extended

```
wire signed [4:0] d3;
reg signed [3:0] s;
wire signed [3:0] d2;
```

```
assign d3 = s + d2;
```

s and d3 get sign-extended

TSEA44: Computer hardware – a system on a chip 2017-10-31 13

Blocking vs non-blocking, sequential


- Blocking assignment (=)
 - Assignments are blocked when executing
 - The statements will be executed in sequence, one after the other
 - Similar to variables in VHDL
- Non-blocking assignment (<=)
 - Assignments are not blocked
 - The statements will be executed concurrently
 - Similar to signals in VHDL

```

always_ff @(posedge c1k) begin
    B = A;
    C = B;
end

always_ff @(posedge c1k) begin
    B <= A;
    C <= B;
end
    
```

Use <= for sequential logic



TSEA44: Computer hardware – a system on a chip 2017-10-31 14

Blocking vs non-block, combinatorial


```

always_comb begin
    C = A & B;
    E = C | D;
end

always_comb begin
    C <= A & B;
    E <= C | D;
end
    
```

Same result


Use = for combinatorial logic



TSEA44: Computer hardware – a system on a chip 2017-10-31 15

Verilog constructs for synthesis

Construct type	Keyword	Notes
ports	input, inout, output	
parameters	parameter	
module definition	module	
signals and variables	wire, reg	Vectors are allowed
instantiation	module instances	E.g., mymux m(out, iO, il, s);
Functions and tasks	function, task	Timing constructs ignored
procedural	always, if, then, else, case	initial is not supported
data flow	assign	Delay information is ignored
loops	for, while, forever	
procedural blocks	begin, end, named blocks, disable	Disabling of named blocks allowed




TSEA44: Computer hardware – a system on a chip 2017-10-31 16

Operators

- Reduction: return scalar value from vector by pairwise calculation
- Replication: {3{a}} same as {a,a,a}

Operator Type	Operator Symbol	Operation Performed
Arithmetic	*	Multiply
	/	Division
	+	Add
	-	Subtract
	%	Modulus
Logical	!	Logical negation
	&&	Logical and
Relational	>	Greater than
	<	Less than
	>=	Greater than or equal
Equality	==	Equality
	!=	Inequality
Reduction	~	Bitwise negation
	~&	nand
Shift		or
	~	nor
	^	xor
	^~	xnor
	~^	xnor
Concatenation	>>	Right shift
	<<	Left shift
Conditional	{}	Concatenation
	?	conditional



TSEA44: Computer hardware – a system on a chip 2017-10-31 17

Parameters

```

module w(x,y);
    input x;
    output y;
    parameter z=16;
    localparam s=3'h1;
    ...

endmodule
    
```

```

w w0(a,b);
w #(8) w1(a,b);
w #(.z(32)) w2(.x(a),.y(b));
    
```

li.u LINKÖPING UNIVERSITY

TSEA44: Computer hardware – a system on a chip 2017-10-31 18

Constants (really substitution macros)

```

`include "myconstants.v"
`define PKMC
`define S0 1'b0
`define S1 1'b1
    
```

Important: It is `, not '
(back tick instead of apostrophe)

```

`ifdef PKMC
...
`else
...
`endif
    
```

li.u LINKÖPING UNIVERSITY

TSEA44: Computer hardware – a system on a chip 2017-10-31 19

Example of an FSM (nextstate + out)

```

//NEXT
always_ff @(posedge clk) begin
    if (xst)
        s <= `S0;
    else
        case (s)
            `S0:
                if (x)
                    s <= `S1;
                default:
                    if (x)
                        s <= `S0;
        end
    
```

```

//OUT
always_comb begin
    case (s)
        `S0: if (x)
            u = 1'b1;
            else
            u = 1'b0;
        default: if (x)
            u = 1'b0;
            else
            u = 1'b1;
    end
    
```

li.u LINKÖPING UNIVERSITY

TSEA44: Computer hardware – a system on a chip 2017-10-31 20

Alternative FSM (separate register)

```

// COMB
always_comb begin
    ns = `S0; // defaults
    u = 1'b0;
    case (s)
        `S0: if (x) begin
            ns = `S1;
            u = 1'b1;
        end
        default:
            if (~x) begin
                u = 1'b1;
                ns = `S1;
            end
    end
    
```

```

// state register
always_ff @(posedge clk) begin
    if (rst)
        s <= `S0;
    else
        s <= ns;
    end
    
```

This description stops us from adding unintentional extra states and flipflops

li.u LINKÖPING UNIVERSITY

Adding more datatypes, including struct

```
typedef logic [3:0] nibble;
nibble nibbleA, nibbleB;

typedef enum {WAIT, LOAD, STORE} state_t;
state_t state, next_state;

typedef struct {
    logic [4:0] alu_ctrl;
    logic stb,ack;
    state_t state } control_t;
control_t control;

assign control.ack = 1'b0;
```

System tasks

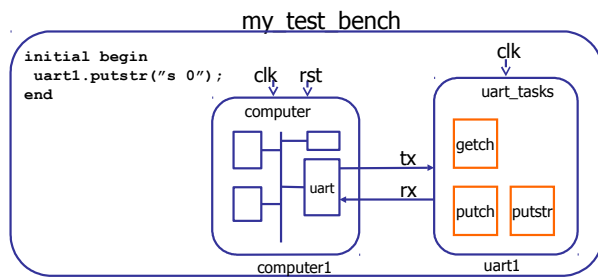
- Initialize memory from file

```
module test;
reg [31:0] mem[0:511]; // 512x32 memory
integer i;

initial begin
    $readmemh("init.dat", mem);
    for (i=0; i<512; i=i+1)
        $display("mem %0d: %h", i, mem[i]); // with CR
end
...
endmodule
```

Expand testbench functions using tasks

- Tasks are subroutines (like procedures in VHDL)
- Initial statement only in testbenches



Tasks example

```
module uart_tasks(input clk, uart_tx,
    output logic uart_rx);

initial begin
    uart_rx = 1'b1;
end

task getch();
    reg [7:0] char;
begin
    @(negedge uart_tx);
    #4340;
    #8680;
    for (int i=0; i<8; i++) begin
        char[i] = uart_tx;
    end
    #8680;
    $fwrite(32'h1,"%c", char);
end
endtask // getch

task putch(input byte char);
begin
    uart_rx = 1'b0;
    for (int i=0; i<8; i++)
        #8680 uart_rx = char[i];
    #8680 uart_rx = 1'b1;
end
endtask // putch

task putstr(input string str);
    byte ch;
begin
    for (int i=0; i<str.len; i++)
        begin
            ch = str[i];
            if (ch)
                putch(ch);
        end
    putch(8'h0d);
end
endtask // putstr
endmodule // uart_tb
```

In the testbench

```

wire tx,rx;
...
// send a command
initial begin
    #100000 // wait 100 us
    uart1.putstr("s 0");
end

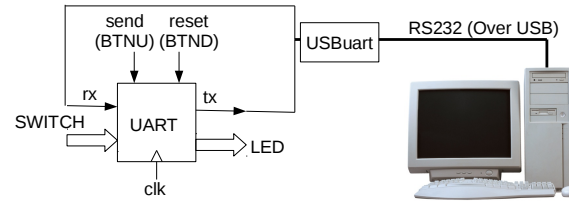
// instantiate the test UART
uart_tasks uart1(.*) ;

// instantiate the computer
computer computer1(.*) ;
    
```



Lab 0: Build an UART in Verilog, Zedboard

- Clk = 100 Mhz
- Baud rate = 115200
- Full duplex (support concurrent send and receive)



UCF = User Constraint File, Zedboard

```

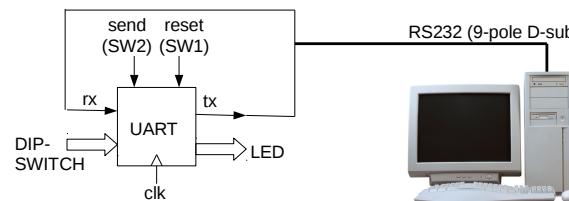
NET "clk_i" LOC = "Y9" | IOSTANDARD=LVCMOS33; // 100 Mhz on Zedboard
NET "rst_i" LOC = "R16" | IOSTANDARD=LVCMOS18; // BTND (downward) on green flexo
NET "send_i" LOC = "T18" | IOSTANDARD=LVCMOS18; // BTNU (up) on green flexo
// switches
NET "switch_i<0>" LOC = "F22" | IOSTANDARD=LVCMOS18; // SWITCH 0
NET "switch_i<1>" LOC = "G22" | IOSTANDARD=LVCMOS18; // SWITCH 1
NET "switch_i<2>" LOC = "H22" | IOSTANDARD=LVCMOS18; // SWITCH 2
NET "switch_i<3>" LOC = "F21" | IOSTANDARD=LVCMOS18; // SWITCH 3
NET "switch_i<4>" LOC = "H19" | IOSTANDARD=LVCMOS18; // SWITCH 4
NET "switch_i<5>" LOC = "H18" | IOSTANDARD=LVCMOS18; // SWITCH 5
NET "switch_i<6>" LOC = "H17" | IOSTANDARD=LVCMOS18; // SWITCH 6
NET "switch_i<7>" LOC = "M15" | IOSTANDARD=LVCMOS18; // SWITCH 7
// row of LEDs
NET "led_o<0>" LOC = "T22" | IOSTANDARD=LVCMOS33; // LED LD0
NET "led_o<1>" LOC = "T21" | IOSTANDARD=LVCMOS33; // LED LD1
NET "led_o<2>" LOC = "U22" | IOSTANDARD=LVCMOS33; // LED LD2
NET "led_o<3>" LOC = "U21" | IOSTANDARD=LVCMOS33; // LED LD3
NET "led_o<4>" LOC = "V22" | IOSTANDARD=LVCMOS33; // LED LD4
NET "led_o<5>" LOC = "W22" | IOSTANDARD=LVCMOS33; // LED LD5
NET "led_o<6>" LOC = "U19" | IOSTANDARD=LVCMOS33; // LED LD6
NET "led_o<7>" LOC = "U14" | IOSTANDARD=LVCMOS33; // LED LD7
// USBUART Pmod on top row of JB
NET "rx_1" LOC = "V10" | IOSTANDARD=LVCMOS33; // PMOD B, JB1
NET "tx_o" LOC = "W11" | IOSTANDARD=LVCMOS33; // PMOD B, JB2
    
```

Net names must match names used in the module description



Lab 0: Build an UART in Verilog, VirtexII

- Clk = 40 Mhz
- Baud rate = 115200
- Full duplex (support concurrent send and receive)



(Voltage level shifter etc not shown)



UCF = User Constraint File, VirtexII

```
CONFIG PART = XC2V4000-FF1152-4 ;
```

```
NET "clk" LOC = "AK19";
```

Net names must match names used in the module description

```
# SWx buttons
```

```
NET "stb" LOC = "B3" ;
```

```
NET "rst" LOC = "C2" ;
```

```
# LEDs
```

```
NET "u<0>" LOC = "N9"; //leftmost
```

```
NET "u<1>" LOC = "P8";
```

```
NET "u<2>" LOC = "N8";
```

```
NET "u<3>" LOC = "N7";
```

```
...
```

```
# DIP switches
```

```
NET "kb<0>" LOC = "AL3"; //leftmost
```

```
NET "kb<1>" LOC = "AK3";
```

```
NET "kb<2>" LOC = "AJ5";
```

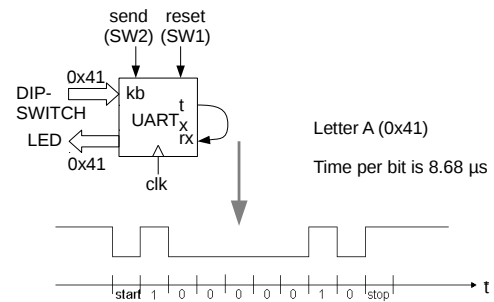
```
NET "kb<3>" LOC = "AH6";
```

```
...
```

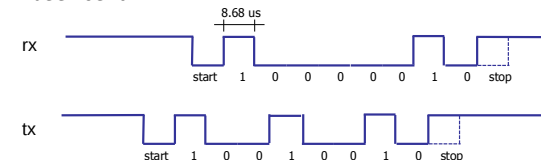
Additional requirement 2017!

- New requirement this year on lab 0 result
 - Only transmit a character on rising edge of pushbutton
- Requirements added 2016
 - The reset state on the LED should indicate the value of your student id:s last two digits.
 - Example: Linus123 should have the value 23 on the LED when reset is applied (and kept there until a value is received on rx).
 - $23 = 16+4+2+1 = 00010111$
- Remember lab 0 is individual

Lab 0: Testbench



Things to look out for

- Testbench have automatically synchronization of rx and tx
 - In real life, a rx bit may start when half o a tx bit have been sent
- 
- Do not forget to wait for the stop bit in rx!
 - May otherwise detect last data bit as new start bit!

Bigger example: Personal number check

- Swedish social security number (personnummer) consists of 10 digits, where the last is a checksum digit.

$$d_1 d_2 d_3 d_4 d_5 d_6 d_7 d_8 d_9 d_{10}$$

$$d_{10} = (10 - ([2d_1]+d_2+[2d_3]+d_4+\dots+[2d_9])) \bmod 10$$

where $[2d_x]$ is digit sum of $2*d_x$ (example $d_x=6 \Rightarrow 2*6=12 \Rightarrow [2d_x]=3$)

- Iterative solution

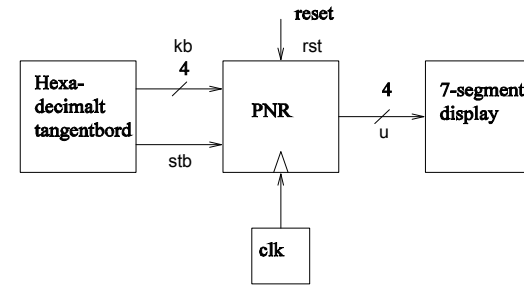
$$S_0=0$$

$$S_k = S_{k-1} \bmod_{10} X 2(d_k) \quad k=1..9$$

$$d_{10} = 10 - S_9$$

Overall system

- PNR is calculating the checksum, presenting it on the display



Top module (pnr module)

```

`include "timescale.v"
`timescale 1ns / 1ps

module pnr(input clk,rst,skb,
input [3:0] kb,
output [3:0] u);

```

time unit precision

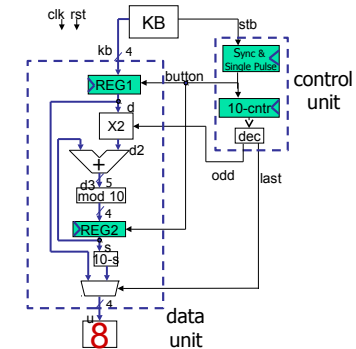
// our design

endmodule

* No entity/architecture distinction => just module

Block schematic (thinking hardware!)

- Split into datapath and control
- Green boxes synch FSM
- White boxes comb
- Button is singlepulsed



TSEA44: Computer hardware – a system on a chip 2017-10-31 37

Timing diagram

- When button is pressed:

$$d2 = \text{digitsum}(2 \cdot 9) = \text{digitsum}(18) = 9$$

New s =

$$(d2 + s) \bmod 10 = (9 + 5) \bmod 10 = 14 \bmod 10 = 4$$

h.u LINKÖPING UNIVERSITY

TSEA44: Computer hardware – a system on a chip 2017-10-31 38

Synch and single pulse shown before

- Always synchronize external inputs!!

```

reg x,y; // variable type (0,1,Z,X)
wire button; // net type (0,1,Z,X)

// SSP
always @(posedge clk) // procedural block
begin
x <= stb;
end

always @(posedge clk) // procedural block
begin
y <= x;
end

assign button = x & ~y;
    
```

h.u LINKÖPING UNIVERSITY

TSEA44: Computer hardware – a system on a chip 2017-10-31 39

Decade counter

```

reg [3:0] p;
wire odd,last;

// 10 counter
always_ff @(posedge clk) begin
if (rst)
p <= 4'd0;
else begin
if (button)
if (p<9)
p <= p+1;
else
p <= 4'd0;
end
end

assign odd = ~p[0];
assign last = (p==4'h9) ? 1'b1 : 1'b0;
    
```

h.u LINKÖPING UNIVERSITY

TSEA44: Computer hardware – a system on a chip 2017-10-31 40

X2 (multiply and add digits when odd=0)

```

always_comb begin
if (~odd)
case (d)
4'h1: d2 = 4'h2;
4'h2: d2 = 4'h4;
4'h3: d2 = 4'h6;
4'h4: d2 = 4'h8;
4'h5: d2 = 4'h1;
4'h6: d2 = 4'h3;
4'h7: d2 = 4'h5;
4'h8: d2 = 4'h7;
4'h9: d2 = 4'h9;
default: d2 = 4'h0;
endcase
else
d2 = d;
end
    
```

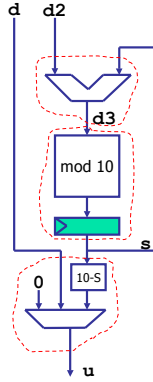
h.u LINKÖPING UNIVERSITY

ADD Reg2, Mod10 K

```
// ADD
assign d3 = {1'b0,s} + {1'b0,d2};

// REG2 and MOD10
always_ff @(posedge clk) begin
  if (rst)
    s <= 4'h0;
  else if (button)
    if (d3 < 10)
      s <= d3[3:0];
    else
      s <= d3[3:0] + 4'd6;
end

// K
assign u = (last == 1'b0) ? d :
  (s == 4'd0) ? 4'd0 :
  4'd10 - s;
```



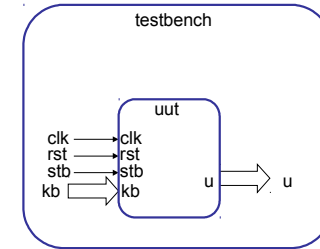
Testbench for PNR (1 of 2)

```
`include "timescale.v"

module testbench();
  // Inputs
  reg clk;
  reg rst;
  reg [3:0] kb;
  reg stb;

  // Outputs
  wire [3:0] u;

  // Instantiate the UUT
  pnr uut (
    .clk(clk),
    .rst(rst),
    .kb(kb),
    .stb(stb),
    .u(u));
```



```
// SystemVerilog instantiation
// automatic mapping port-wire
pnr uut(.*);
```

Testbench for PNR (2 of 2)

```
// Initialize Inputs
initial begin
  clk = 1'b0;
  rst = 1'b1;
  kb = 4'd0;
  stb = 1'b0;
  #70 rst = 1'b0;
  //
  #30 kb = 4'd8;
  #40 stb = 1'b1;
  #30 stb = 1'b0;
  //
  #30 kb = 4'd0;
  #40 stb = 1'b1;
  #30 stb = 1'b0;
end

always #12.5 clk = ~clk; // 40 MHz
endmodule
```

← Add more digits and strobe pulses
to test a complete number