# Introduction to Verilog and SystemVerilog

Saul Rodriguez

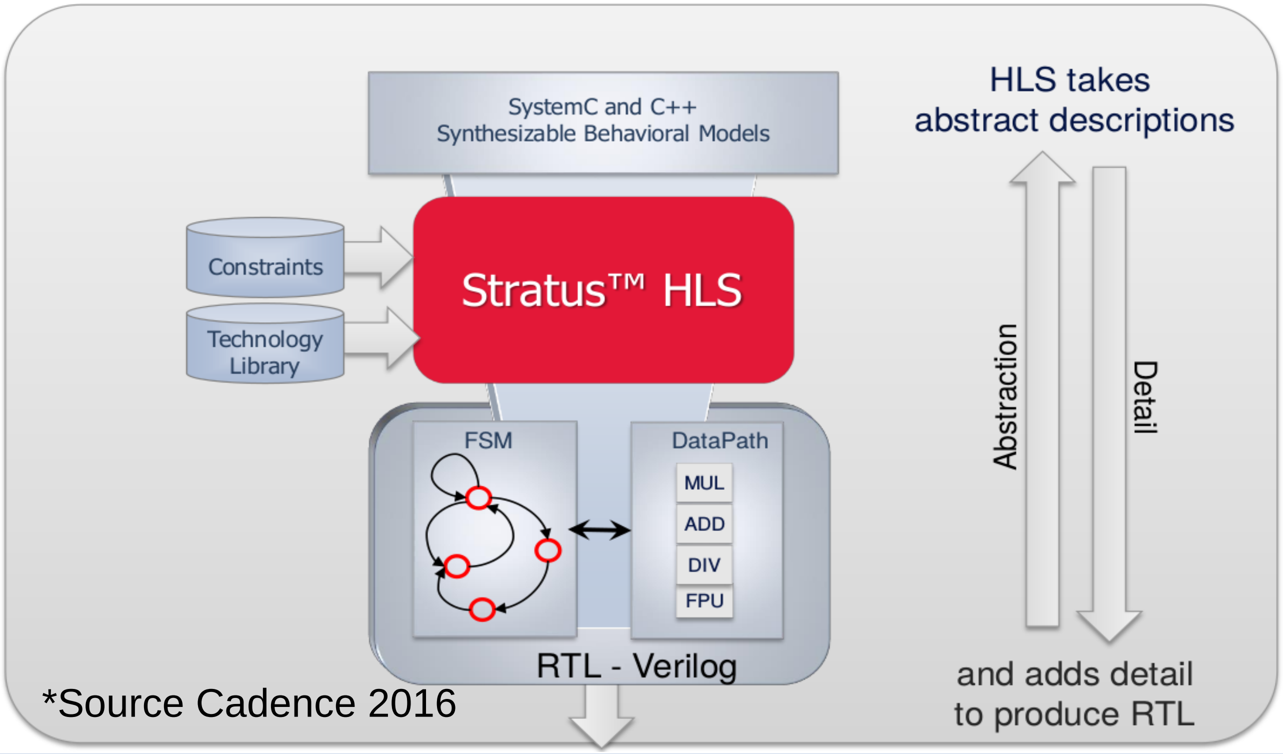
## OUTLINE

## MOTIVATION

- Digital design flows evolve towards high level languages
  - Higher productivity
  - Shorter design and verification time
  - Easy to learn
- VHDL is based on ADA
  - ADA is Strong typed, code safe
  - ADA is not very widely used
- Verilog is based on C
  - C is weak typed
  - C is the most widely spread language
  - C++, Java, C#, Objective C, Pearl, etc.

# MOTIVATION

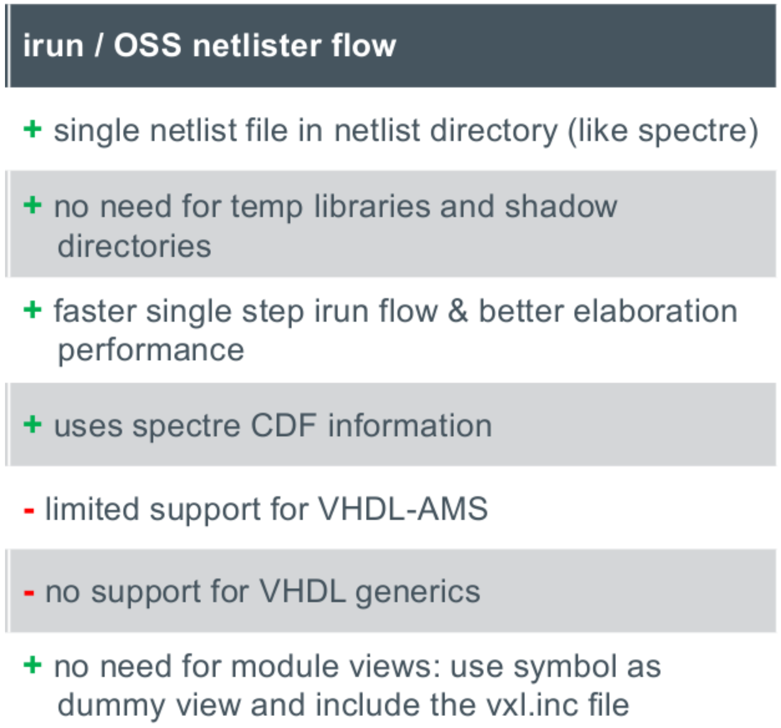
- ASIC EDA tools are influenced by a limited number of players that decide what tool you will use.



*Source Cadence 2016

## MOTIVATION

- ASIC Foundries decide which tools are primarily supported in their Process Design Kits (PDKs)

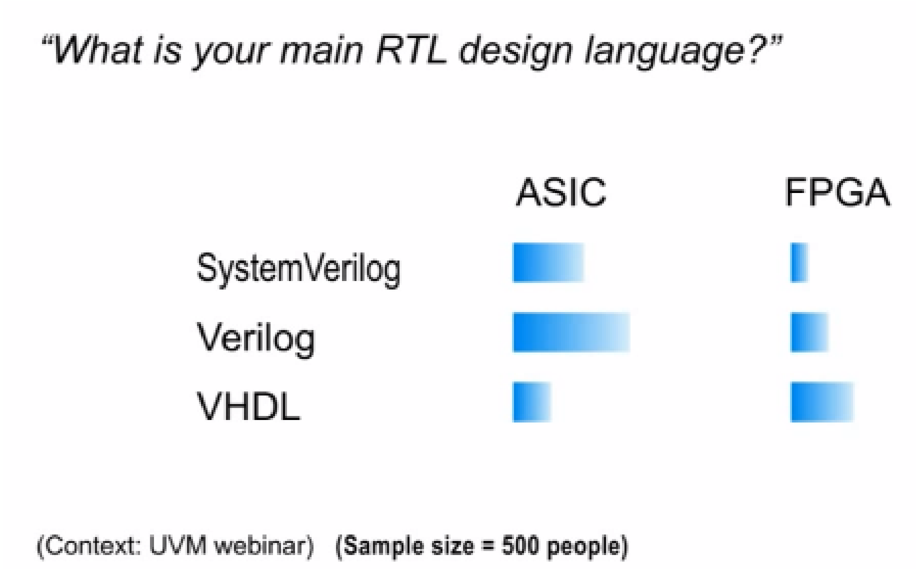


| irun / OSS netlister flow |
|---|
| + single netlist file in netlist directory (like spectre) |
| + no need for temp libraries and shadow directories |
| + faster single step irun flow & better elaboration performance |
| + uses spectre CDF information |
| - limited support for VHDL-AMS |
| - no support for VHDL generics |
| + no need for module views: use symbol as dummy view and include the vxl.inc file |

*Source: AMS 0.18um CMOS PDK

## MOTIVATION

- ASIC design is dominated by Verilog



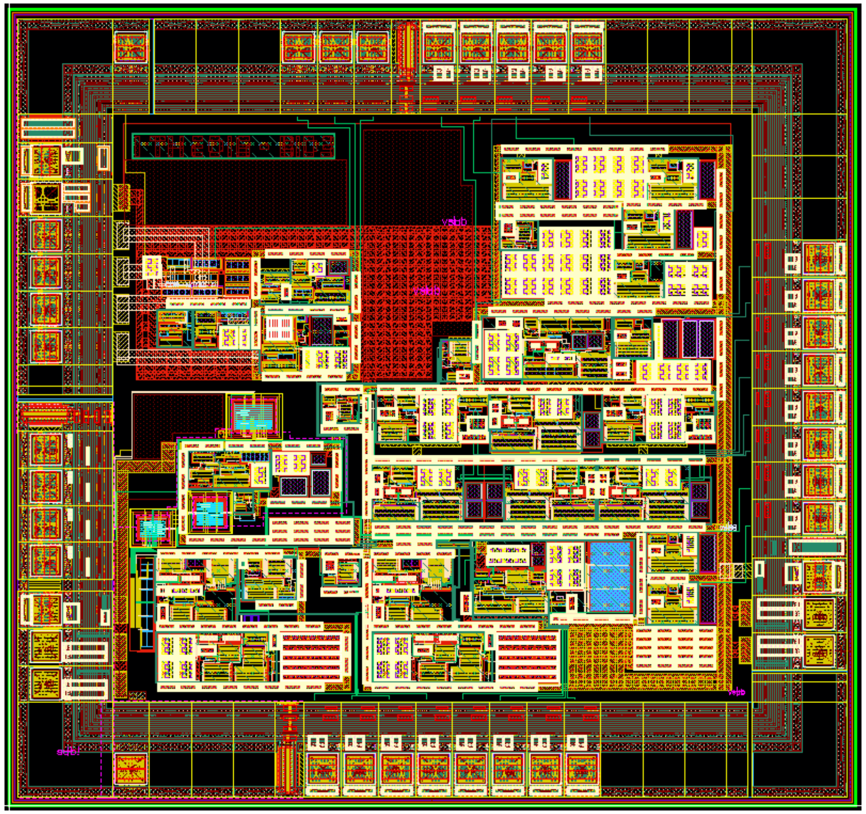
*Source: www.doulos.com

## VERILOG FAMILY

- Verilog
  - RTL level, verification
- VerilogA
  - Modeling of analog circuits
  - Used directly with an analog simulator!
- VerilogAMS
  - Analog and mixed signal circuits
  - Used with an mixed signal simulator!
- SystemVerilog
  - Extensions to Verilog (safer code)
  - Object-oriented paradigm (verification)

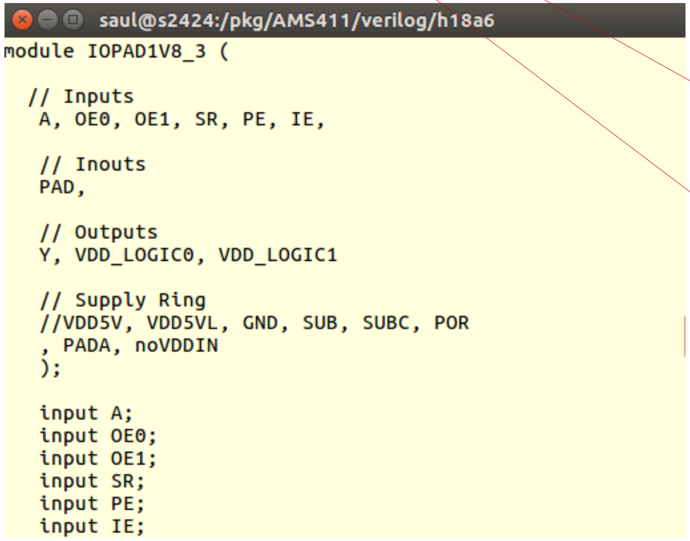
## VERILOG FAMILY

- ASIC Example
  - KTH2016-BIO3
  - Mixed-signal ASIC
  - Top-Bottom approach
  - Verilog, VerilogA, VerilogAMS



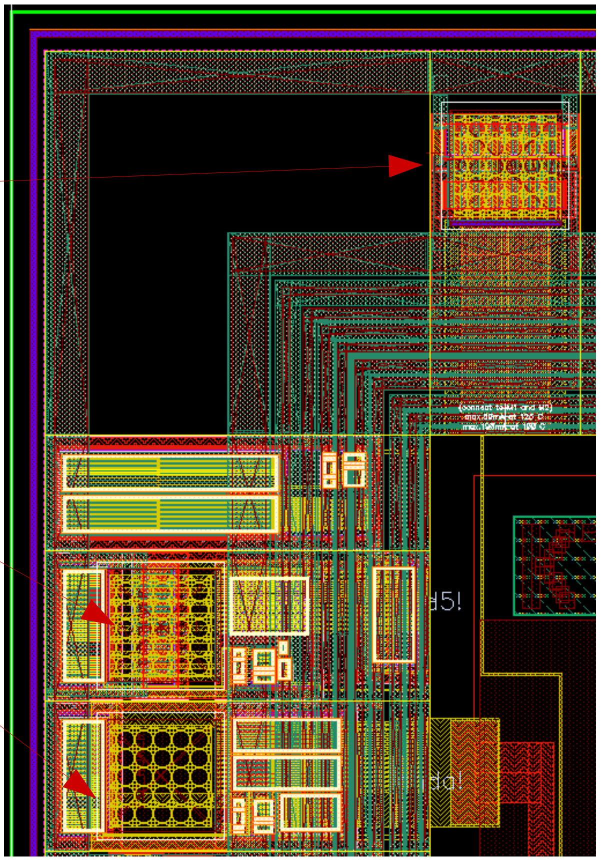
# VERILOG FAMILY

- Foundry provides Verilog models for IOs



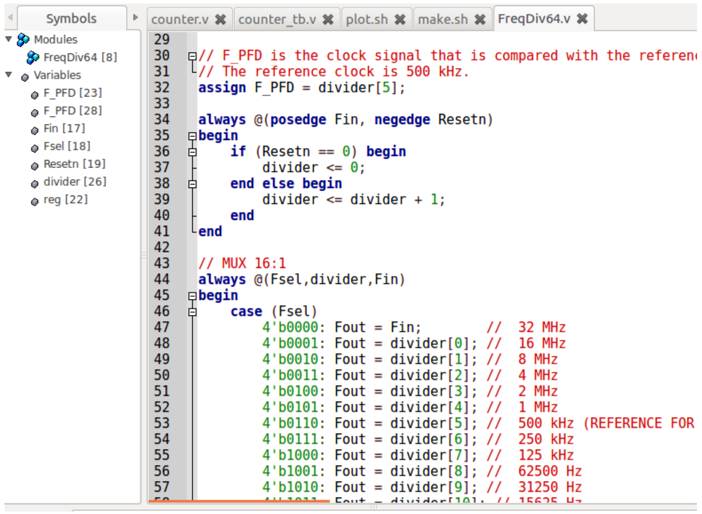
saul@s2424:/pkg/AMS411/verilog/h18a6

```
module IOPAD1V8_3 (

  // Inputs
   A, OE0, OE1, SR, PE, IE,

   // Inouts
   PAD,

   // Outputs
   Y, VDD_LOGIC0, VDD_LOGIC1

   // Supply Ring
   //VDD5V, VDD5VL, GND, SUB, SUBC, POR
   , PADA, noVDDIN
   );

   input A;
   input OE0;
   input OE1;
   input SR;
   input PE;
   input IE;
```

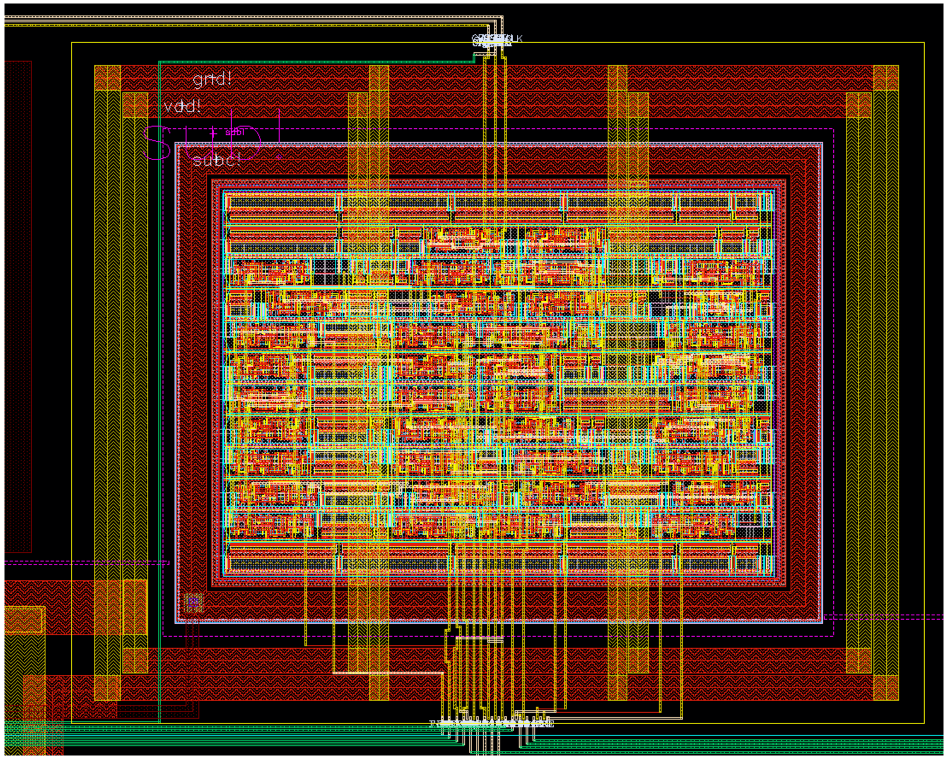


# VERILOG FAMILY

- Digital blocks coded in Verilog RTL
- Synthesized, placed and routed

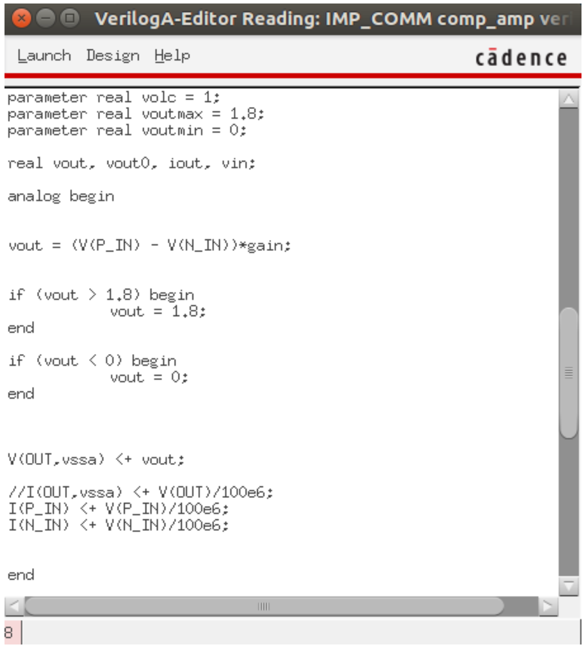


```verilog

// F_PFD is the clock signal that is compared with the referen
// The reference clock is 500 kHz.
assign F_PFD = divider[5];

always @(posedge Fin, negedge Resetn)
begin
    if (Resetn == 0) begin
        divider <= 0;
    end else begin
        divider <= divider + 1;
    end
end

// MUX 16:1
always @(Fsel,divider,Fin)
begin
    case (Fsel)
        4'b0000: Fout = Fin;         //  32 MHz
        4'b0001: Fout = divider[0]; //  16 MHz
        4'b0010: Fout = divider[1]; //  8 MHz
        4'b0011: Fout = divider[2]; //  4 MHz
        4'b0100: Fout = divider[3]; //  2 MHz
        4'b0101: Fout = divider[4]; //  1 MHz
        4'b0110: Fout = divider[5]; //  500 kHz (REFERENCE FOR
        4'b0111: Fout = divider[6]; //  250 kHz
        4'b1000: Fout = divider[7]; //  125 kHz
        4'b1001: Fout = divider[8]; //  62500 Hz
        4'b1010: Fout = divider[9]; //  31250 Hz
        4'b1011: Fout = divider[10]; // 15625 Hz
```

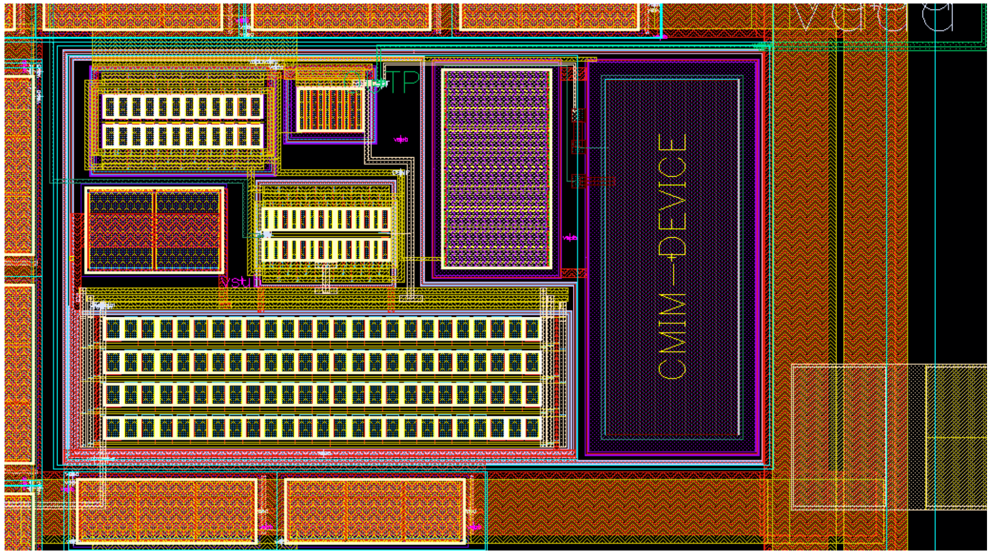


# VERILOG FAMILY

- Analog blocks modeled using VerilogA

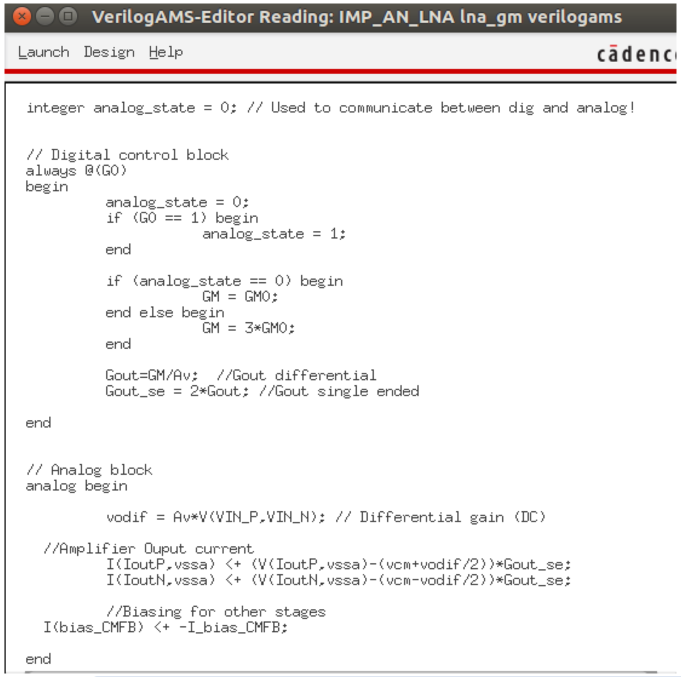


```
parameter real volc = 1;
parameter real voutmax = 1.8;
parameter real voutmin = 0;

real vout, vout0, iout, vin;

analog begin


vout = (V(P_IN) - V(N_IN))*gain;


if (vout > 1.8) begin
            vout = 1.8;
end

if (vout < 0) begin
            vout = 0;
end



V(OUT,vssa) <+ vout;

//I(OUT,vssa) <+ V(OUT)/100e6;
I(P_IN) <+ V(P_IN)/100e6;
I(N_IN) <+ V(N_IN)/100e6;


end
```

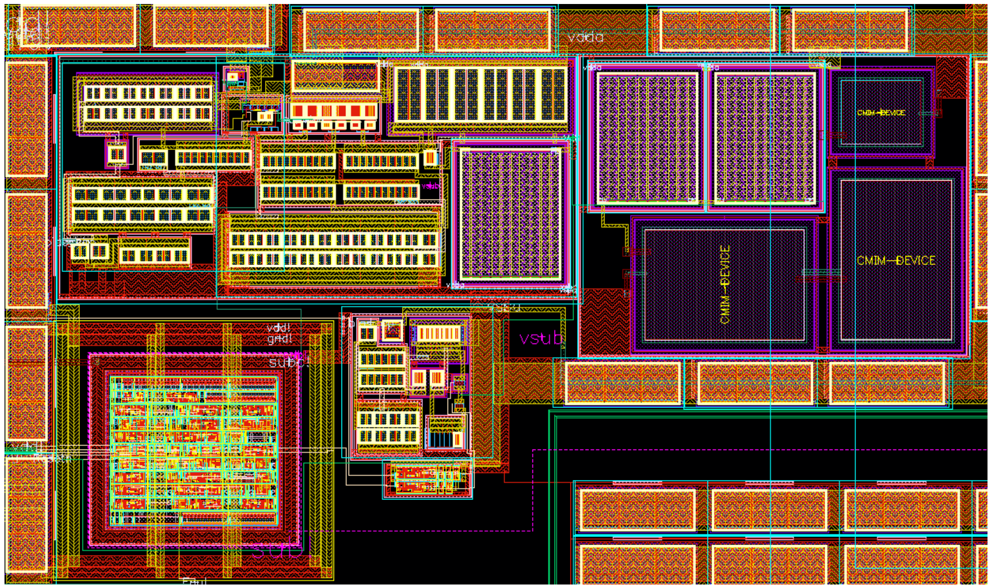


# VERILOG FAMILY

- Digitally controlled Analog blocks are modeled in VerilogAMS



```
integer analog_state = 0; // Used to communicate between dig and analog!


// Digital control block
always @(GO)
begin
        analog_state = 0;
        if (GO == 1) begin
                analog_state = 1;
        end

        if (analog_state == 0) begin
                GM = GM0;
        end else begin
                GM = 3*GM0;
        end

        Gout=GM/Av;  //Gout differential
        Gout_se = 2*Gout; //Gout single ended

end


// Analog block
analog begin

        vodif = Av*V(VIN_P,VIN_N); // Differential gain (DC)

  //Amplifier Ouput current
        I(IoutP,vssa) <+ (V(IoutP,vssa)-(vcm+vodif/2))*Gout_se;
        I(IoutN,vssa) <+ (V(IoutN,vssa)-(vcm-vodif/2))*Gout_se;

        //Biasing for other stages
  I(bias_CMFB) <+ -I_bias_CMFB;

end
```



# VERILOG RTL AND SYNTHESIS EXAMPLES

- This is a short demo of Verilog RTL+ Verilog test bench code
- You can download all these examples from:
- Recommended literature:
  - S. Brown, Z. Vranesic, "Fundamentals of Digital Logic with Verilog Design" 3rd Ed, McGRaw-Hill
  - Pong P. Chu, "FPGA Prototyping by Verilog Examples", Wiley 2008

## VERILOG RTL AND SYNTHESIS EXAMPLES

We will use these free open-source tools:

- Icarus Verilog compiler http://iverilog.icarus.com/
- GTKWave  waveform viewer
- http://gtkwave.sourceforge.net/
- Yosys Open Synthesis Suite http://www.clifford.at/yosys/download.html
- Geany IDE
- https://www.geany.org/

## VERILOG

- Documentation
  // This is a short comment
  /*This is a long Verilog comment
  that spans two lines */
- White spaces
  f = f = w0; if (s == 1) f = w1;w0; if (s == 1) f = w1;
- Signals
  **type** [range] signal_name{, signal_name};

## VERILOG

- Identifier names

  Any character may be used (no digits at the beginning, no reserved words). Verilog is case sensitive!
- Signal Values

  0 = logic value 0  
  1 = logic value 1  
  z = tri-state (high impedance)  
  x = unknown value
- Numbers

  d = decimal  
  b = binary  
  h = hexadecimal  
  o = octal

## VERILOG

- Example:

| | |
|---|---|
| 0 | number 0 |
| 10 | decimal number 10 |
| ’b10 | binary number 10 = $(2)_{10}$ |
| ’h10 | hex number 10 = $(16)_{10}$ |
| 4’b100 | binary number 0100 = $(4)_{10}$ |
| 4’bx | unknown 4-bit value xxxx |
| 8’b1000_0011 | “_” can be inserted for readability |
| 8’hfx | same as 8’b1111_xxxx |

## VERILOG

- Example:

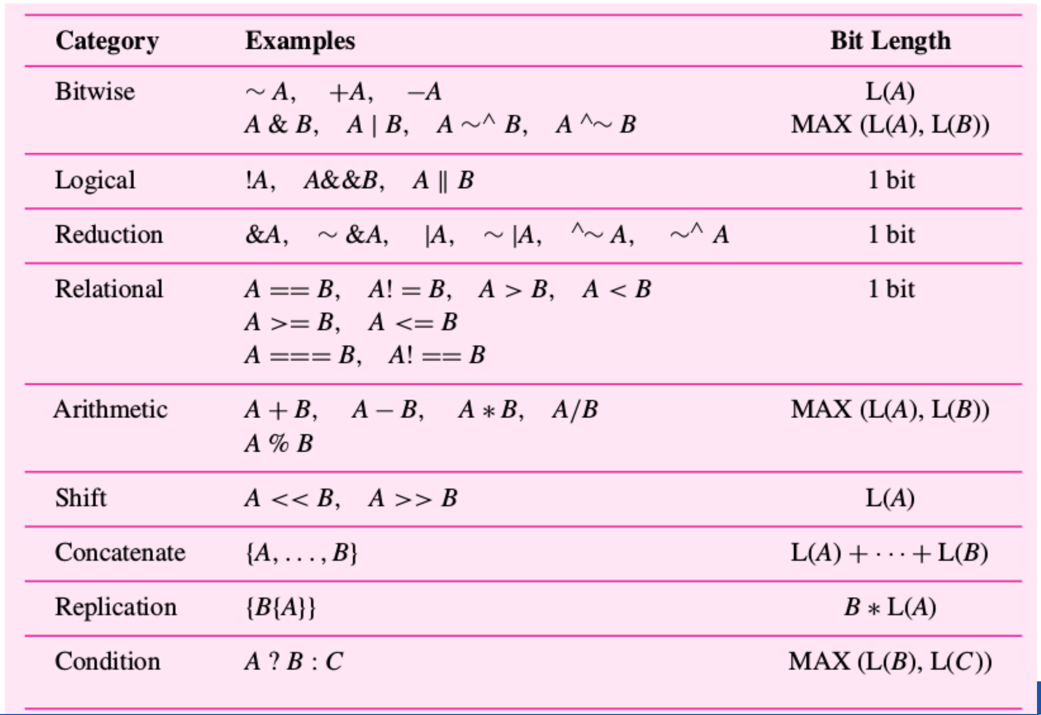
| | |
|---|---|
| 0 | number 0 |
| 10 | decimal number 10 |
| 'b10 | binary number 10 = $(2)_{10}$ |
| 'h10 | hex number 10 = $(16)_{10}$ |
| 4'b100 | binary number 0100 = $(4)_{10}$ |
| 4'bx | unknown 4-bit value xxxx |
| 8'b1000_0011 | "_" can be inserted for readability |
| 8'hfx | same as 8'b1111_xxxx |

# VERILOG

- Net Types:
  **wire** x;
  **wire** Cin, AddSub;
  **wire** [3:0] S;
  **wire** [1:2] Array;
  **tri** z;
  **tri** [7:0] DataOut;
- Variable Types:
  **reg** [2:0] Count; //reg does not denote a storage element!
  **integer** k;

# VERILOG

- Memories:
  **reg** [7:0] R[3:0]; // four eight-bit variables R[0]-R[3]
  **reg** [7:0] R[3:0][1:0]
- Operators:



| **Category** | **Examples** | **Bit Length** |
|---|---|---|
| Bitwise | $\sim A,\quad +A,\quad -A$<br>$A \,\&\, B,\quad A \mid B,\quad A \sim\wedge B,\quad A \wedge\sim B$ | $\mathrm{L}(A)$<br>$\mathrm{MAX}\ (\mathrm{L}(A), \mathrm{L}(B))$ |
| Logical | $!A,\quad A\&\&B,\quad A \parallel B$ | 1 bit |
| Reduction | $\&A,\quad \sim\&A,\quad \vert A,\quad \sim\vert A,\quad \wedge\sim A,\quad \sim\wedge A$ | 1 bit |
| Relational | $A == B,\quad A! = B,\quad A > B,\quad A < B$<br>$A >= B,\quad A <= B$<br>$A === B,\quad A! == B$ | 1 bit |
| Arithmetic | $A + B,\quad A - B,\quad A * B,\quad A/B$<br>$A \,\%\, B$ | $\mathrm{MAX}\ (\mathrm{L}(A), \mathrm{L}(B))$ |
| Shift | $A << B,\quad A >> B$ | $\mathrm{L}(A)$ |
| Concatenate | $\{A, \ldots, B\}$ | $\mathrm{L}(A) + \cdots + \mathrm{L}(B)$ |
| Replication | $\{B\{A\}\}$ | $B * \mathrm{L}(A)$ |
| Condition | $A\ ?\ B : C$ | $\mathrm{MAX}\ (\mathrm{L}(B), \mathrm{L}(C))$ |

## VERILOG

- Modules:

```
module module_name [(port name{, port name})];
   [parameter declarations]
   [input declarations]
   [output declarations]
   [inout declarations]
   [wire or tri declarations]
   [reg or integer declarations]
   [function or task declarations]
   [assign continuous assignments]
   [initial block
   [always blocks]
   [gate instantiations]
   [module instantiations]
endmodule
```

## VERILOG

- Example full-adder 1:

```
module fulladd (Cin, x, y, s, Cout);
   input Cin, x, y;
   output s, Cout;

   assign s = x ^ y ^ Cin;
   assign Cout = (x & y) | (Cin & x) | (Cin & y);

endmodule
```

## VERILOG

- Example full-adder 2:

```
module fulladd (Cin, x, y, s, Cout);
   input Cin, x, y;
   output s, Cout;

   assign {Cout, s} = x + y + Cin;

endmodule
```

## VERILOG

- Example full-adder 3:

```
module fulladd (Cin, x, y, s, Cout);
   input Cin, x, y;
   output s, Cout;
   wire z1, z2, z3, z4;

   and And1 (z1, x, y);
   and And2 (z2, x, Cin);
   and And3 (z3, y, Cin);
   or Or1 (Cout, z1, z2, z3);
   xor Xor1 (z4, x, y);
   xor Xor2 (s, z4, Cin);

endmodule
```

## VERILOG

- Concurrent Statements

  **assign** net_assignment{, net_assignment};

- Example:

  **assign** Cout = (x & y) | (x & Cin) | (y & Cin);
  **assign** s = x ∧ y ∧ z;
  **wire** [1:3] A, B, C;
  .
  .
  **assign** C = A & B;

  **wire** [3:0] X, Y, S;
  **wire** carryin, carryout;
  **assign** {carryout, S} = X + Y + carryin;

## VERILOG

- Using Parameters

```
module addern (X, Y, S, S2s);
  parameter n = 4;
  input [n-1:0] X, Y;
  output [2*n-1:0] S, S2s;

  assign S = X + Y,
  S2s = {{n{X[n-1]}}, X} + {{n{Y[n-1]}}, Y};
endmodule
```

## VERILOG

- Procedural Statements (Sequential statements)

  **always** @(sensitivity_list)
      **[begin]**
          [procedural assignment statements]
          [**if-else** statements]
          [**case** statements]
          [**while**, **repeat**, and **for** loops]
          [**task** and **function** calls]
      **[end]**

## VERILOG

- Procedural Assignments Statements
  - **Blocking Assignments** (evaluated in order at $t_i$)

    ```
    S = X + Y;
    p = S[0];
    ```
  - **Non-blocking Assignments** (evaluated in order but with variable values at the start of $t_i$)

    ```
    S <= X + Y;
    p <= S[0];
    ```
- Blocking Assignments should be used for combinational circuits
- Non-blocking Assignments should be used for sequential circuits

## VERILOG

- IF-ELSE STATEMENTS

```
if (expression1) begin
    statement;
end else if (expression2) begin
    statement;
end else begin
    statement;
end
```

- Careful with default assignments, otherwise latches are implied!
- Mux2to1 Live demo

## VERILOG

- CASE STATEMENT

```
case (expression)
   alternative1: begin
                     statement;
                 end
   alternative2: begin
                     statement;
                 end
   [default:     begin
                    statement;
                 end]
endcase
```

- fulladd Live demo

## VERILOG

- FOR LOOP STATEMENT

```
for (initial_index; terminal_index; increment)
    begin
        statement;
    end
```

- Example:

```
always @(X, Y, carryin)
    begin: fulladders
        integer k;
        C[0] = carryin;
        for (k = 0; k <= n−1; k = k+1) begin
            S[k] = X[k] ∧ Y[k] ∧ C[k];
            C[k+1] = (X[k] & Y[k]) | (C[k] & X[k]) | (C[k] & Y[k]);
        end
        carryout = C[n];
end
```

## VERILOG

- USING SUBCIRCUITS

  module_name [#(parameter overrides)] instance_name (, .port_name ( [expression] ) {, .port_name ( [expression] )} );
- EXAMPLE

```
module adder4 (carryin, X, Y, S, carryout);
   input carryin;
   input [3:0] X, Y;
   output [3:0] S;
   output carryout;
   wire [3:1] C;

   fulladd stage0 (carryin, X[0], Y[0], S[0], C[1]);
   fulladd stage1 (C[1], X[1], Y[1], S[1], C[2]);
   fulladd stage2 (C[2], X[2], Y[2], S[2], C[3]);
   fulladd stage3 (.Cout(carryout), .s(S[3]), .y(Y[3]), .x(X[3]), .Cin(C[3]));
endmodule
```

## VERILOG

- SEQUENTIAL CIRCUITS
  - Require procedural statements
- Gated D Latch

```
module latch (D, clk, Q);
        input D, clk;
        output reg Q;

        always @(D, clk)
            if (clk)
                Q = D;
endmodule
```

## VERILOG

- D FLIP-FLOP

```
module flipflop (D, Clock, Q);
        input D, Clock;
        output reg Q;

        always @(posedge Clock)
            Q <= D;
endmodule
```

- D FLIP-FLOP with asynchronous RESET (L)

```
module flipflop_ar (D, Clock, Resetn, Q);
        input D, Clock, Resetn;
        output reg Q;

        always @(posedge Clock, negedge Resetn)
            if (Resetn == 0)
                Q <= 0;
            else
                Q <= D;
endmodule
```

## VERILOG

- D FLIP-FLOP with synchronous RESET (L)

```
module flipflop_sr (D, Clock, Resetn, Q);
        input D, Clock, Resetn;
        output reg Q;

        always @(posedge Clock)
            if (Resetn == 0)
                Q <= 0;
            else
                Q <= D;
endmodule
```

- LIVE DEMO: synthesis of flip-flops with synchronous and asynchronous reset

## VERILOG

- Register with asynchronous clear

```
module reg4 (D, Clock, Resetn, Q);
   input [3:0] D;
   input Clock, Resetn;
   output reg [3:0] Q;

   always @(posedge Clock, negedge Resetn)
      if (Resetn == 0)
         Q <= 4'b0000;
      else
         Q <= D;
endmodule
```

## VERILOG

- n-bit register with asynchronous clear and enable

```
module regne (D, Clock, Resetn, E, Q);
        parameter n = 4;
        input [n-1:0] D;
        input Clock, Resetn, E;
        output reg [n–1:0] Q;

        always @(posedge Clock, negedge Resetn)
            if (Resetn == 0)
                    Q <= 0;
            else if (E)
                    Q <= D;
endmodule
```

- Live demo: synthesis of registers with asynch. Clear and enable

## VERILOG

- Shift registers

```
module shift3 (w, Clock, Q);
   input w, Clock;
   output reg [1:3] Q;

   always @(posedge Clock)
   begin
      Q[3] <= w;
      Q[2] <= Q[3];
      Q[1] <= Q[2];
   end
endmodule
```

- Live demo: synthesis of shift register

## VERILOG

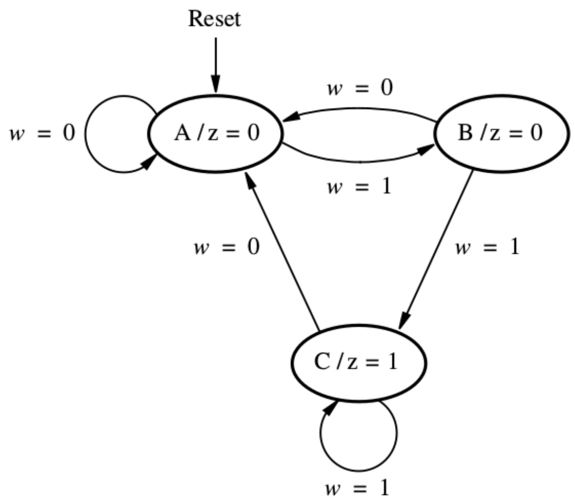
- Counters

```
module count4 (Clock, Resetn, E, Q);
    input Clock, Resetn, E;
    output reg [3:0] Q;

    always @(posedge Clock, negedge Resetn)
        if (Resetn == 0)
            Q <= 0;
        else if (E)
            Q <= Q + 1;
endmodule
```

- Live demo: simulation and synthesis of counter

## VERILOG

- Moore-Type FSM
  - 2 always blocks:
    - 1 combinational to define next state
    - 1 sequential to assign the next state during clock edge
  - Outputs assigned in combinational block or with separate assignment statements
- EXAMPLE:



## VERILOG

```
module moore (Clock, w, Resetn, z);
        input Clock, w, Resetn;
        output z;
        reg [1:0] y, Y; // y = current state, Y = next state
        parameter A = 2’b00, B = 2’b01, C = 2’b10;

        always @(w, y)
        begin
                case (y)
                        A: if (w == 0)    Y=A;
                            else           Y=B;
                        B: if (w == 0)    Y=A;
                            else           Y=C;
                        C: if (w == 0)    Y=A;
                            else           Y=C;
                        default:           Y=2’bxx;
                endcase
        end
CODE CONTINUE IN NEXT SLIDE...
```

## VERILOG

**CONTINUATION…**

```
always @(posedge Clock, negedge Resetn)
begin
        if (Resetn == 0)
                y <= A;
        else
                y <= Y;
end

assign z = (y == C); //could also be moved to top block!

endmodule
```

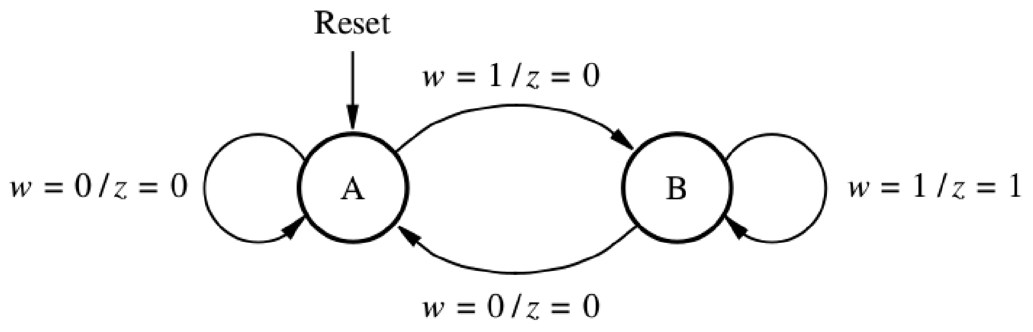
# VERILOG

## ALTERNATIVE CODE IN 1 ALWAYS BLOCK:

```
module moore (Clock, w, Resetn, z);
input Clock, w, Resetn;
output z;
reg [1:0] y;
parameter A = 2'b00, B = 2'b01, C = 2'b10;

always @(posedge Clock, negedge Resetn)
begin
        if (Resetn == 0)
                y <= A;
        else
                case (y)
                    A: if (w == 0) y <= A;
                       else        y <= B;
                    B: if (w == 0) y <= A;
                       else         y <= C;
                    C: if (w == 0) y <= A;
                       else         y <= C;
                    default:        y <= 2'bxx;
        endcase
end

assign z = (y == C);
endmodule
```

## VERILOG

- Mealy-Type FSM
  - Same as before2 allways blocks:
    - 1 combinational to define next state
    - 1 sequential to assign the next state during clock edge
  - Outputs assigned in combinational block
- EXAMPLE:



## VERILOG

```
module mealy (Clock, w, Resetn, z);
input Clock, w, Resetn;
output reg z;
reg y, Y;
parameter A = 1'b0, B = 1'b1;

always @(w, y)
        case (y)
                A: if (w == 0)  begin
                        Y = A; z = 0;
                    end else begin
                        Y = B; z = 0;
                    end
                B: if (w == 0) begin
                        Y = A; z = 0;
                    end else begin
                        Y = B; z = 1;
                    end
        endcase
```

CONTINUES NEXT SLIDE...

## VERILOG

**CONTINUATION…**

```
  always @(posedge Clock, negedge Resetn)
  begin
      if (Resetn == 0)
              y <= A;
      else
              y <= Y;
  end

endmodule
```

## SYSTEMVERILOG

- Hardware description and Verification language HDVL
  - Combines Verilog/VHDL features with C and C++ including object oriented programming (OOP).
- Natural successor to Verilog
- Advanced tools for
  - RTL Design
  - Assertion
  - Verification
- The following is just a brief overview. For a comprehensive guide and tutorials visit: https://www.doulos.com

## SYSTEMVERILOG

- New data types from C:

| **TYPE** | **Description** | **Example** |
|---|---|---|
| bit | user-defined size | bit [3:0] a_nibble; |
| byte | 8 bits, signed | byte a, b; |
| shortint | 16 bits, signed | shortint c, d; |
| int | 32 bits, signed | int i,j; |
| longint | 64 bits, signed | longint lword; |

## SYSTEMVERILOG

- New data types, same syntax as C:

```
typedef struct packed {
    logic [3:0] X;
    int Y;
    bit flag;
} my_type;
```

- Packages (VHDL, Java, C++ namespaces)
- Strings
- Classes (full OOP with single inheritance)
  - Very useful in verification

## SYSTEMVERILOG

- New data types, same syntax as C:

```
typedef struct packed {
        logic [3:0] X;
        int Y;
        bit flag;
} my_type;
```

- Packages (VHDL, Java, C++ namespaces)
- Strings
- Classes (full OOP with single inheritance)

## SYSTEMVERILOG

- Easier port connections
- Instantiation in Verilog:

  **module** Design (input Clock, Reset, input [7:0] Data, output [7:0] Q);

  Design DUT ( Clock, Reset, Data, Q );
  Design DUT ( .Clock(Clock), .Reset(Reset), .Data(Data), .Q(Q) );

- Instantiation in SystemVerilog:

  Design DUT(.*);
  Design DUT ( .Clock(SysClock), .* );

## SYSTEMVERILOG

- Synthesis idioms always_comb, always_ff, always_latch
  - Enforce checks to avoid bugs during synthesis

- Example1:

```
always_comb
  if (sel)
     f = x;
    else
    f = y;
```

- Example2:

```
always_ff @(posedge clock iff reset == 0 or posedge reset)
  if (reset)
       q <= 0;
 else if (enable)
       q++;
```

## SYSTEMVERILOG

- Immediate Assertions

```
assert (A == B) $display ("OK. A equals B");
   else $error("It's gone wrong");
```

- Concurrent Assertions

```
assert property ( !(Read && Write) );
assert property ( @(posedge Clock) Req |-> ##[1:2] Ack);
```

## FINAL REMARKS

- Verilog and SystemVerilog increase in popularity
- General programming skills (C,C++,etc) are becoming very important for digital designers
- Learning Verilog after understanding VHDL is relatively easy