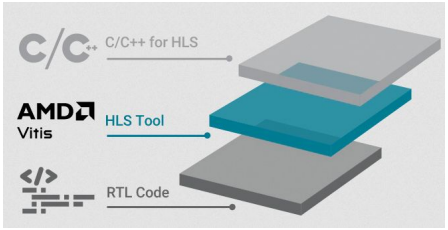
CPE 470 - Synthesis

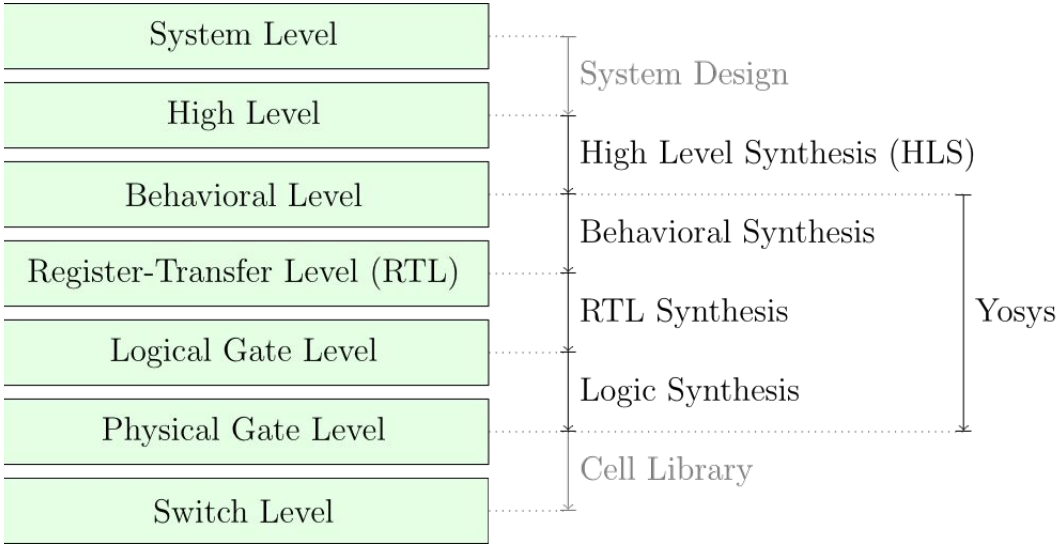
Synthesis

Glossary
HLS: High Level Synthesis

- Transformation from higher level design to lower level functional representation
  - **HLS** : higher level language like C++, gets translated to RTL or logic level



- RTL Synthesis:
  1. Start with **RTL**
  2. Synthesize to generic **logic** (NAND, AND)
  3. Map generic logic to implementation-specific **physical** gates



Yosys

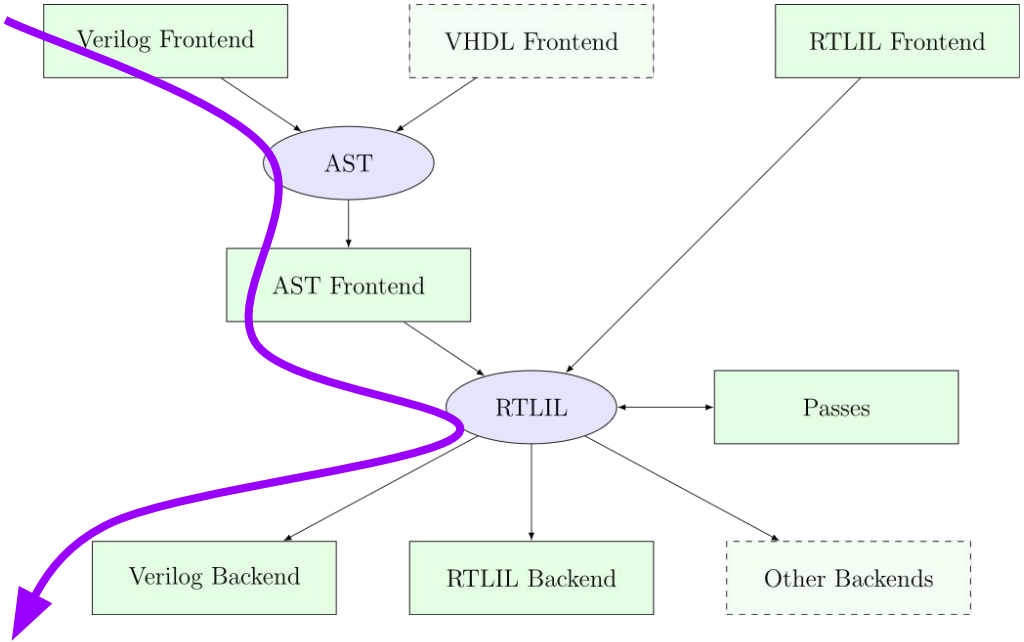
Glossary

Timing-Driven Synthesis: iteratively runs timing analysis during synthesis to optimize for speed

- Primary Open-Source Synthesis Tool
  - Developed by Claire Xenia Wolf
- Implementation Independent
  - Can be used for FPGAs, mapping to LUTs
  - Can be used for ASICs, mapping to Standard Cells
- Primarily optimizes for area
  - Follows the philosophy that smaller is usually faster
  - Generally does **not** do **Timing-Driven Synthesis**

How Yosys Synthesis Works

- Behavioral / RTL Verilog goes in
- Verilog **Netlist** comes out



Glossary

Netlist: description of an electric circuit, including devices (gates) and the nets (wires) that connect them

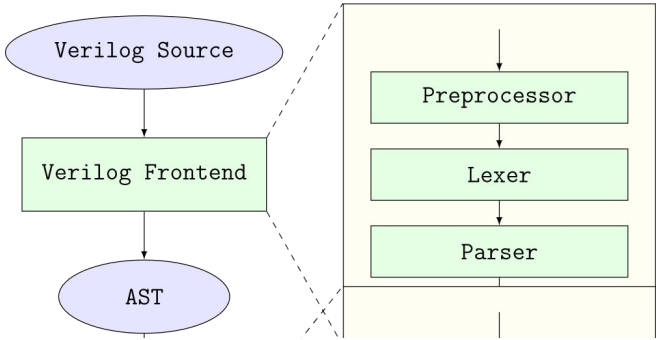
Glossary

AST: Abstract Syntax Tree

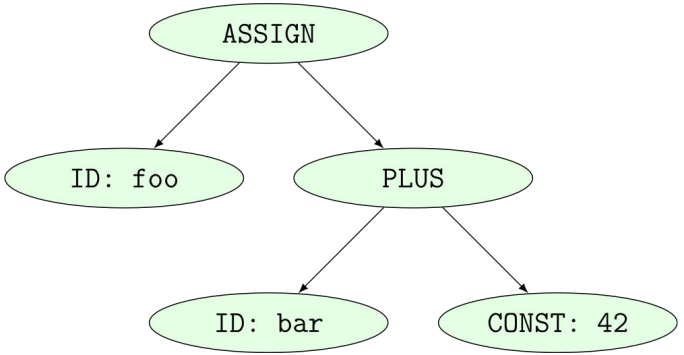
Verilog to AST

First, we have to process our verilog code into a meaningful graph

1. **Preprocessor** evaluates parameters, macros, etc.
   - Macros like `` `define, `ifdef, `` or `` `include ``
2. **Lexer** processes character by character into tokens
3. **Parser** builds tokens into **AST**
   - Abstract Syntax Tree is language agnostic




```
assign foo = bar + 42;
```



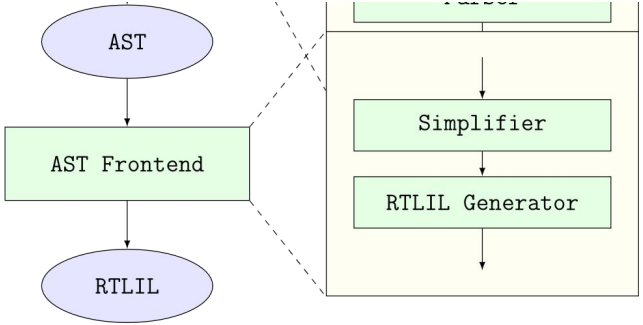
AST to RTLIL

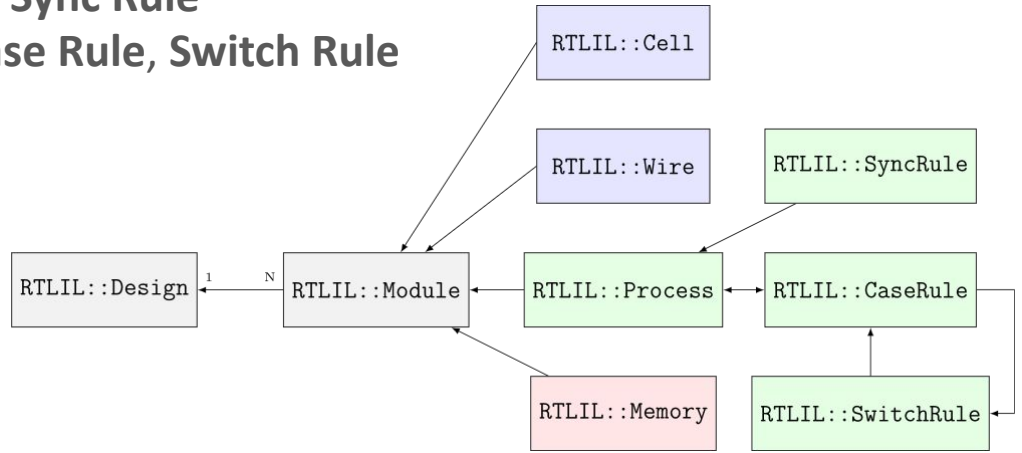
Glossary

IR: Intermediate Representation, code internal to a compiler

RTLIL: Register Transfer Layer Intermediate Language

- Graphical Representation of code gets translated to an Intermediate Representation (**IR**), specifically **RTLIL**
  - Logic Gates -> **Cells**
  - Connections between gates -> **Wires**
  - 2D Arrays -> **Memory**
  - Control Logic -> **Process**
    - Clocks and Sensitivity -> **Sync Rule**
    - Switch Statements -> **Case Rule**, **Switch Rule**





RTLIL to Verilog Netlist

- **RTLIL** contains all necessary functionality of circuit
  - Use it to rebuild verilog netlist of only wires, flip flops, and minimal logic functions

Set of minimal logic functions include:

- $_AND_
- $_BUF_
- $_MUX_
- $_NAND_
- $_NOR_
- $_NOT_
- $_OR_
- $_XNOR_
- $_XOR_

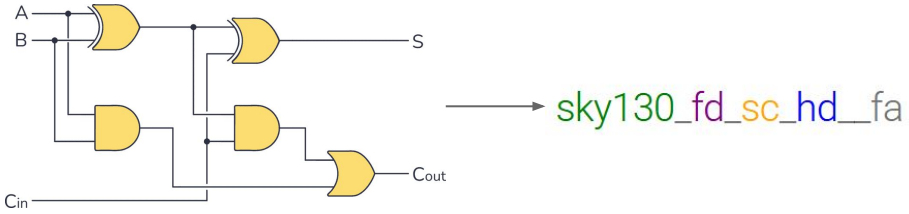
```
module adder (
    input [2:0] a,
    input [2:0] b,
    output logic [2:0] s
);

assign s = a + b;

endmodule
```

```
module adder(a, b, s);
  wire _00_;
  wire _01_;
  wire _02_;
  wire _03_;
  wire _04_;
  wire _05_;
  input [2:0] a;
  wire [2:0] a;
  input [2:0] b;
  wire [2:0] b;
  output [2:0] s;
  wire [2:0] s;
  assign _00_ = b[1] ^ a[1];
  assign _01_ = ~(b[0] & a[0]);
  assign s[1] = ~(_01_ ^ _00_);
  assign _02_ = ~(b[2] ^ a[2]);
  assign _03_ = ~(b[1] & a[1]);
  assign _04_ = _00_ & ~(_01_);
  assign _05_ = _03_ & ~(_04_);
  assign s[2] = _05_ ^ _02_;
  assign s[0] = b[0] ^ a[0];
endmodule
```

Technology Mapping

- Uses **ABC,** a tool developed by Berkeley, to match generic logic gates to specific physical standard cells
  - Standard Cell functionalities defined in **.lib** Liberty file
- Matching of individual cells
  - $_NAND_ -> sky130_fd_sc_hd__nand2
- Matching of subcircuits
  - 

- ABC is able to do some timing optimization, in progress towards timing-driven synthesis

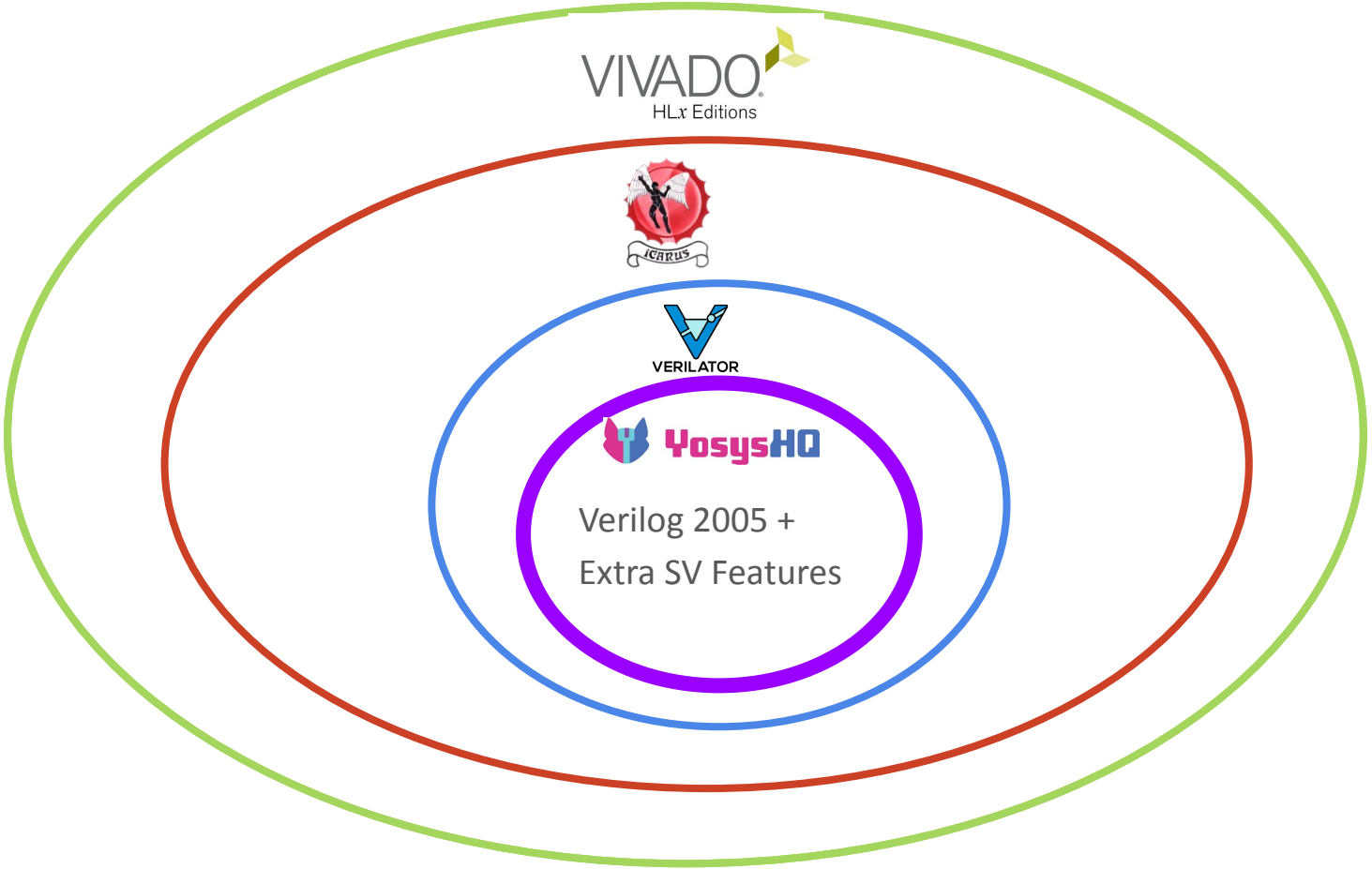
```
sky130_fd_sc_hd__xnor2_1 _25_ (
  .B(_15_),
  .Y(_19_)
);
sky130_fd_sc_hd__o21ai_0 _26_ (
  .A1(_12_),
  .A2(_14_),
  .B1(_13_),
  .Y(_16_)
);
sky130_fd_sc_hd__xnor2_1 _27_ (
  .A(_11_),
  .B(_08_),
  .Y(_17_)
);
sky130_fd_sc_hd__xnor2_1 _28_ (
  .A(_16_),
  .B(_17_),
  .Y(_20_)
);
sky130_fd_sc_hd__xor2_1 _29_ (
  .A(_09_),
  .B(_06_),
  .X(_18_)
);
```

Supported Features

- Yosys does not have complete system verilog support!
  - Technically supports Verilog 2005

- Still includes all the features we care about
  - `always_comb`, `always_ff` and `always_latch`
  - `logic` and `bit`
  - `typedef` and `enum`
  - Packed Structs
  - Multidimensional Arrays
    - Not on inputs/outputs though
  - interfaces and modports
    - Use with caution, recent additions

Scope Continued



Synthesizable or Not?

- Bit Flipper
  - Flips r every clock cycle
  - Active low asynchronous reset

```
module bit_flipper (
    input clk,
    input rst_n
);

logic r;

always_ff @( posedge clk ) begin
    r <= ~r;
end

always_ff @( negedge rst_n ) begin
    r <= 0;
end

endmodule
```

Not!

```
module bit_flipper (
    input clk,
    input rst_n
);

logic r;

always_ff @( posedge clk ) begin
    r <= ~r;
end

always_ff @( negedge rst_n ) begin
    r <= 0;
end

endmodule
```

- Register **r** has multiple drivers!
  - Verilator will actually warn about this
- Fix by combining into one sensitivity list

```
module bit_flipper (
    input clk,
    input rst_n,
    output logic r
);

always_ff @( posedge clk or negedge rst_n ) begin
    if (~rst_n) begin
        r <= 0;
    end else begin
        r <= ~r;
    end
end

endmodule
```

Synthesizable or Not? 2

- Integer Adder

```
module int_adder (
    input [31:0] a,
    input [31:0] b,
    output int s
);

assign s = a + b;

endmodule
```

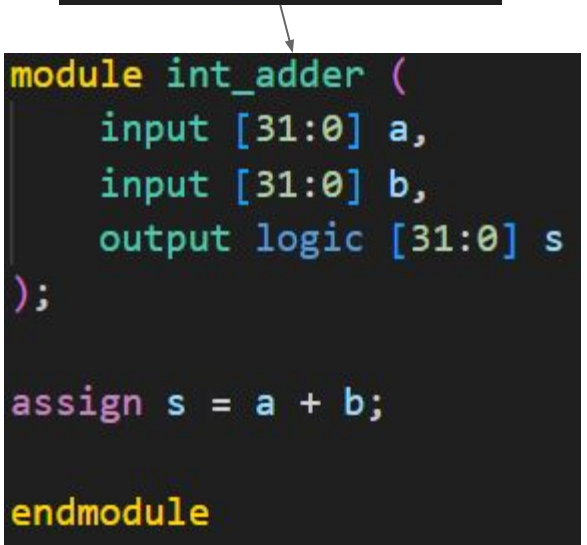
Yes, but…

- Variable types of fixed size will be translated to logics of equal size
  - `int` -> 32 bit logic
  - `byte` -> 8 bit logic
  - `bit` -> 1 bit logic
- Don’t use them in RTL!!!
  - Always use logic/reg/wire
  - variable types are simulated as 2 state instead of 4 state
- Only use them in testbenches

```
module int_adder (
    input [31:0] a,
    input [31:0] b,
    output int s
);

assign s = a + b;

endmodule
```



Synthesizable or Not? 3

```
module test (
    output logic r1,
    output logic r2
);


inverter I1 (.a(r1), .y(r2));
inverter I2 (.a(r2), .y(r1));
```

```
module inverter (
    input a,
    output logic y
);
assign y = ~a;

endmodule
```

Yes, but…

- Yosys will let you create combinational loops if you want!
  - Generally it will warn you about them
  - There are times when you want to make an oscillator, such as for random number generation

- Verilator will get more upset than Yosys about this

Synthesizable or Not? 4

- In1 drives the output high
- In2 drives the output low

```
module test (
    input in1,
    input in2,
    output logic out
);

always_comb begin
    if (in1) begin
        out = 1'b1;
    end else if (in2) begin
        out = 1'b0;
    end
end

endmodule
```

Not!

```
module test (
    input in1,
    input in2,
    output logic out
);

always_comb begin
    if (in1) begin
        out = 1'b1;
    end else if (in2) begin
        out = 1'b0;
    end
end

endmodule
```

- A latch is inferred at the output
  - This is because `out` is not defined in the case where `in1` and `in2` are 0
- Yosys does not allow inferred latches
- Always declare default output values at the beginning of an always_comb block

```
always_comb begin
    out = 1'b0;
    if (in1) begin
        out = 1'b1;
    end else if (in2) begin
        out = 1'b0;
    end
end

endmodule
```

References

- https://yosyshq.readthedocs.io/en/latest/