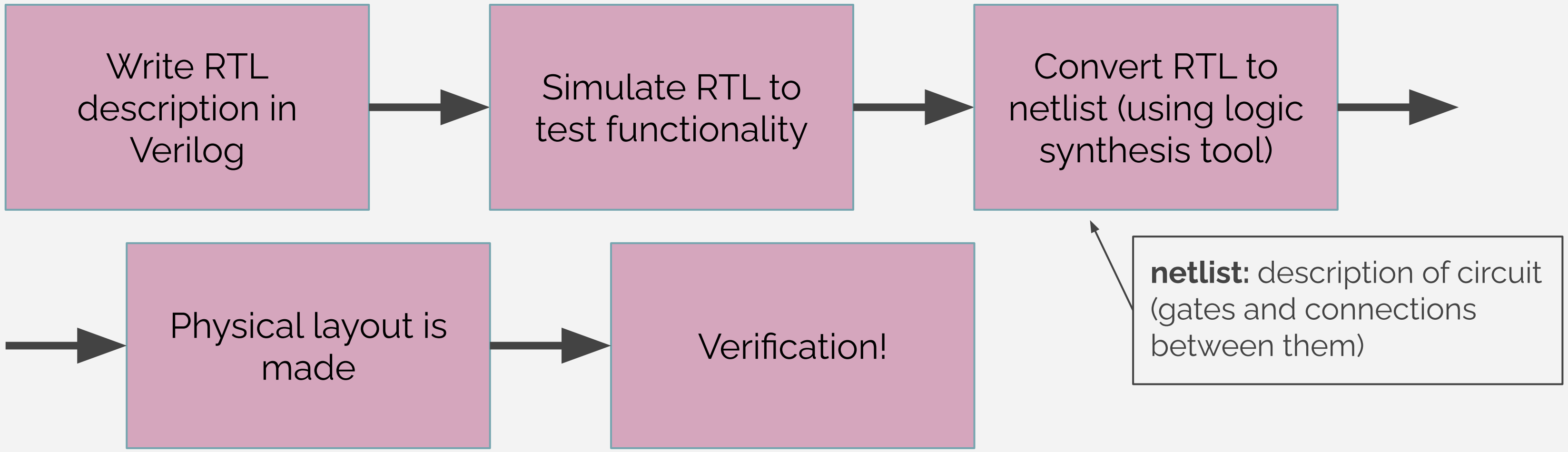
Verilog Introduction Part 1

7/25/24

Ashika Palacharla

VLSI Design Flow & HDLs

- **Hardware Description Language**
  - Used to describe any digital hardware at any level
- **RTL description is done with HDLs**


Write RTL description in Verilog
Simulate RTL to test functionality
Convert RTL to netlist (using logic synthesis tool)
Physical layout is made
Verification!
netlist: description of circuit (gates and connections between them)

HDLs and Abstraction

- **Describing designs in HDL are independent of technology**
  - Easy for designing and debugging
- **Levels of Abstraction**
  - **Behavioral level**
    - Describing system by sequential algorithms
    - Functions, tasks, blocks
  - **Register-transfer level**
    - Specify characteristics of circuit
    - Operations and transfer of data between registers
  - **Gate level**
    - Characteristics described by logical links and timing properties
    - Discrete signals with definite logic values (ex. 0 or 1)
    - (Not super important for our uses, usually generated using synthesis tools and used for gate-level simulation)

General Syntax

- **Numbers**
  - Can use binary, octal, decimal, or hexadecimal
- **Operators (open link for more)**
  - Typical arithmetic, relational
  - Bit-wise (does bit-by-bit comparison between 2 operands)
- **Identifiers**
  - Used to define the object (function, module, register)
  - Start with alphabetical characters or underscore
  - `A_Z, a_z, _`
- **Keywords**
  - Do not use as identifiers, reserved in Verilog
  - `assign, case, while, wire, reg, and, or, nand, module`
- **Comments**
  - `// this is single line comment`
  - `/* this is multiline comment */`

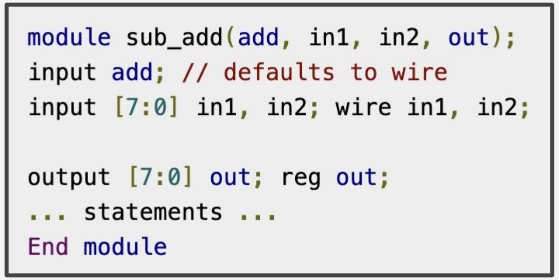
General Syntax

- **Basic values stored in data types**
    - 0 (logic zero/false condition)
    - 1 (logic one/true condition)
    - x (unknown logic value)
    - z (high impedance state)
- **Wire**
    - Represents a physical wire in circuit
    - Cannot assign a value in a procedural block
    - Can drive a value onto wire using continuous assignment
- **Register**
    - Data object, holds the value from one assignment to the next
    - Used only in functions and procedural blocks
    - `A_Z, a_z, _`
- **Input, Output, Inout**
    - Used to declare input, output, and bidirectional ports of a module

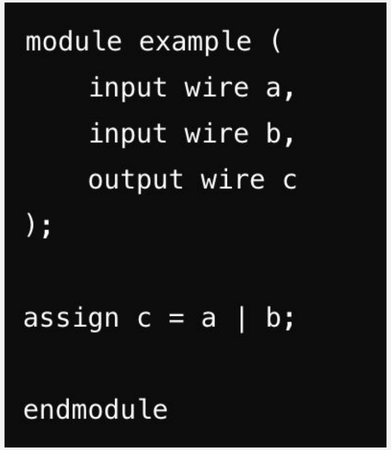
General Syntax

- **Modules**
  - Principal design entity
  - Indicates name and port list
  - Specifies input/output type and port width

- **Continuous Assignment**
  - Module for assigning a value onto a wire
  - Explicit assign statement
  - Continuously executed at time of simulation (evaluated whenever right-hand side signals change)



```
module sub_add(add, in1, in2, out);
input add; // defaults to wire
input [7:0] in1, in2; wire in1, in2;

output [7:0] out; reg out;
... statements ...
End module
```

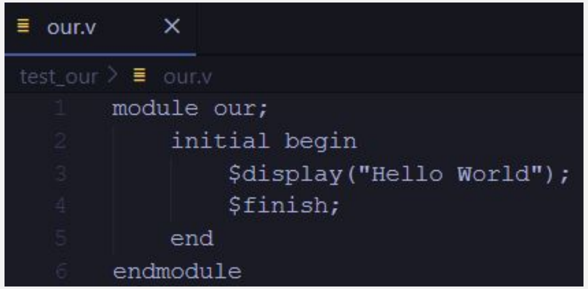


```
module example (
    input wire a,
    input wire b,
    output wire c
);

assign c = a | b;

endmodule
```

Important Commands

- **Verilator command to translate design (VHDL) into executable**
  - Generates C++ and compiles it
  - `verilator --exe --main --build --cc -Wall [VHDL file name]`
    - `--cc:` translates design into C++ code
    - `--main:` creates a C++ top wrapper to read CLI arguments
    - `--exe:` creates makefiles to generate simulation executable
    - `--build:` builds model
- **Run the executable**
  - `obj_dir/[name of executable]`

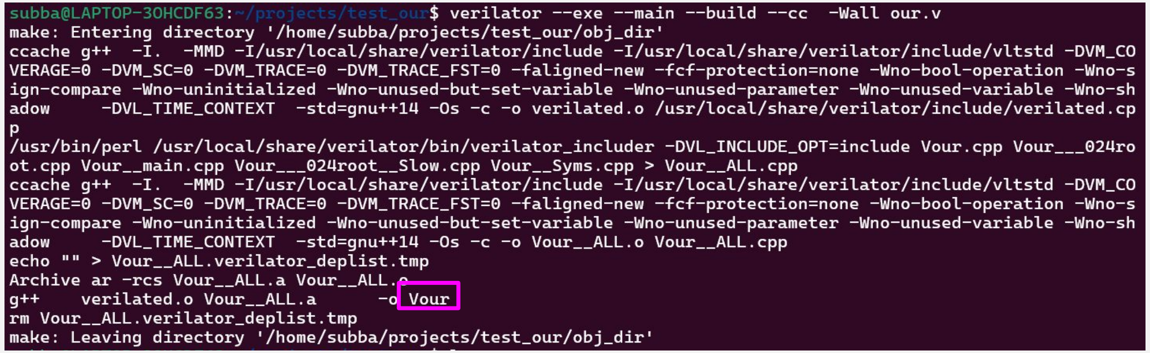
Hello World program



```
module our;
    initial begin
        $display("Hello World");
        $finish;
    end
endmodule
```

`verilator --exe --main --build --cc -Wall our.v`

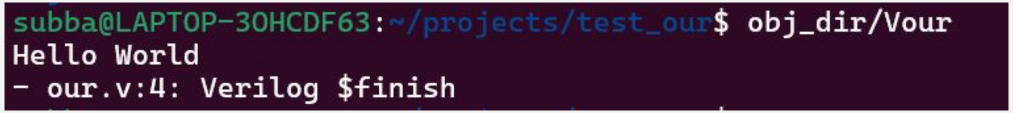
Translate our.v VHDL into executable



```
subba@LAPTOP-3OHCDF63:~/projects/test_our$ verilator --exe --main --build --cc  -Wall our.v
make: Entering directory '/home/subba/projects/test_our/obj_dir'
ccache g++  -I.  -MMD -I/usr/local/share/verilator/include -I/usr/local/share/verilator/include/vltstd -DVM_CO
VERAGE=0 -DVM_SC=0 -DVM_TRACE=0 -DVM_TRACE_FST=0 -faligned-new -fcf-protection=none -Wno-bool-operation -Wno-s
ign-compare -Wno-uninitialized -Wno-unused-but-set-variable -Wno-unused-parameter -Wno-unused-variable -Wno-sh
adow     -DVL_TIME_CONTEXT   -std=gnu++14 -Os -c -o verilated.o /usr/local/share/verilator/include/verilated.cp
p
/usr/bin/perl /usr/local/share/verilator/bin/verilator_includer -DVL_INCLUDE_OPT=include Vour.cpp Vour___024ro
ot.cpp Vour__main.cpp Vour___024root__Slow.cpp Vour__Syms.cpp > Vour__ALL.cpp
ccache g++  -I.  -MMD -I/usr/local/share/verilator/include -I/usr/local/share/verilator/include/vltstd -DVM_CO
VERAGE=0 -DVM_SC=0 -DVM_TRACE=0 -DVM_TRACE_FST=0 -faligned-new -fcf-protection=none -Wno-bool-operation -Wno-s
ign-compare -Wno-uninitialized -Wno-unused-but-set-variable -Wno-unused-parameter -Wno-unused-variable -Wno-sh
adow     -DVL_TIME_CONTEXT   -std=gnu++14 -Os -c -o Vour__ALL.o Vour__ALL.cpp
echo "" > Vour__ALL.verilator_deplist.tmp
Archive ar -rcs Vour__ALL.a Vour__ALL.o
g++    verilated.o Vour__ALL.a      -o Vour
rm Vour__ALL.verilator_deplist.tmp
make: Leaving directory '/home/subba/projects/test_our/obj_dir'
```

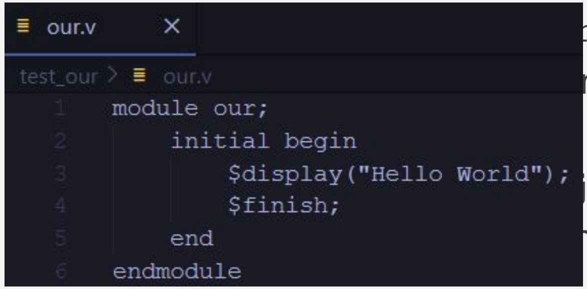
`obj_dir/Vour`

Run executable, produces output to terminal:



```
subba@LAPTOP-3OHCDF63:~/projects/test_our$ obj_dir/Vour
Hello World
- our.v:4: Verilog $finish
```

Half Bit Adder program



```
module our;
    initial begin
        $display("Hello World");
        $finish;
    end
endmodule
```

```
rilator --exe --main --build --cc -Wall our.v
```

nslate our.v VHDL into executable

```
_dir/Vour
```

n executable, produces output to terminal:

References

https://www.tutorialspoint.com/vlsi_design/vlsi_design_verilog_introduction.htm

https://verilator.org/guide/latest/verilating.html