DIGITAL VLSI PROJECTS

HRITAM MITRA- LINKDIN

HRITAM MITRA GitHub

PROJECT TITLE :-

STRUCTURAL / GATE LEVEL MODELLING OF FULL ADDER USING HALF ADDER.

BASIC CONCEPTS :-

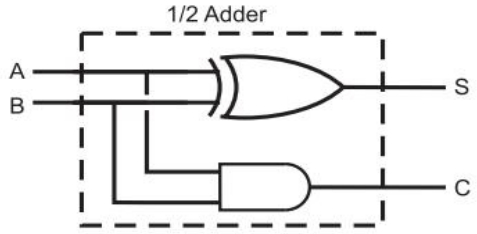
HALF ADDER:-

1) A combinational circuit that performs **addition** of **two** bits is known as ***Half Adder***.
2) The input variables designated to ***Augend*** and ***Addend bits***.
3) The output variables designated to ***Sum*** and ***Carry***.

FULL ADDER:-

1) A combinational circuit that performs **addition** of **three** bits is known as ***Full Adder***.
2) The input variables designated to ***Augend, Addend*** *and* ***carry from the lower significant position bits***.
3) The output variables designated to ***Sum*** and ***Carry.***

CIRCUIT DIAGRAM :-

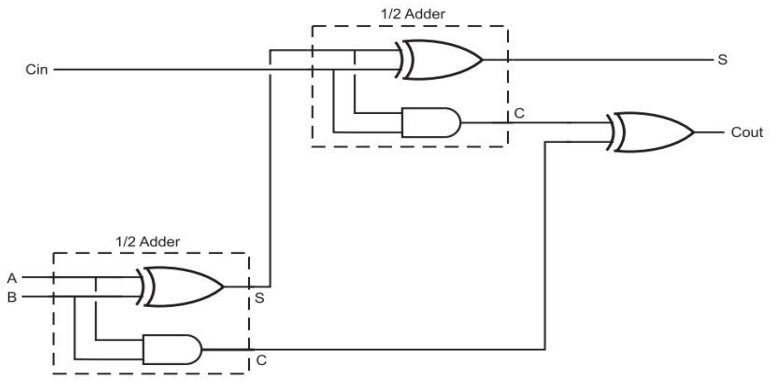


HALF ADDER CIRCUIT

TRUTH TABLES :-

"A" input	"B" input	"Sum" bit	"Carry" bit
0	0	0	0
0	1	1	0
1	1	0	1
1	0	1	0

Truth table of HALF ADDER



FULL ADDER CIRCUIT

Inputs			Outputs	
A	*B*	*Cin*	Σ	*Co*
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1
$A + B + C_{in}$			Sum	Carry out

Truth table of FULL ADDER

HRITAM MITRA- **LINKDIN**

HRITAM MITRA- **GitHub**

USED TOOLS :-

1. **GEANY (** open source IDE**)**
2. **GVIM** or **GEDIT (**Optional for code editing**)**
3. **Icarus Verilog (** Veriog code simulation & Parsing**)**
4. **GTKwave (** Waveform viewing tool **)**
5. **Yosys (** Gate level synthesis using technology library**)**
6. **Skywater 130 PDK (**Technology library has used in this project**)**

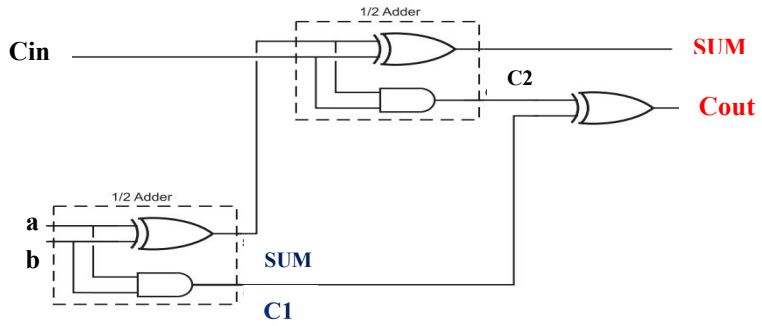
MAIN PROGRAM CODE

Symbols | fa_by_ha.v | tb_fa.v

Modules
Variables
C1 [15]
C2 [15]
Cin [13]
Cout [14]
S1 [15]
a [6]
a [13]
b [6]
b [13]
sum [14]
wire [7]

```
1
2   // STRUCTURAL OR GATE LEVEL MODELLING OF FULL ADDER USING THE HALF ADDER
3   // SOURCE CODE
4
5   module half_adder(a,b,sum,Carry);  // Half adder instance
6       input a,b;
7       output  wire sum,Carry;
8       assign sum = a^b;
9       assign Carry = a&b;
10  endmodule
11
12  module full_adder (a,b,Cin,sum,Cout); // Designing full adder module using half adder instance
13      input a,b,Cin;
14      output sum,Cout;
15      wire C1,S1,C2;
16      half_adder HA1(a,b,S1,C1); // For the first half adder instance
17      half_adder HA2(Cin,S1,sum,C2); // For the first half adder instance
18      or (Cout,C1,C2); // The output & input of OR gate
19  endmodule
```

TESTBENCH CODE



Symbols | fa_by_ha.v | tb_fa.v

Modules
tb_fa [3]
Variables
Cin [4]
Cout [5]
a [4]
b [4]
sum [5]

```
1   // TESTBENCH CODE
2
3   module tb_fa();
4       reg a,b,Cin;  // Inputs of the full adder
5       wire sum,Cout; // Outputs of the full adder
6   initial begin
7       $dumpfile("full_adder.vcd"); // all the binaries will be dumped into that file
8       $dumpvars; // all the variables will be dumped into that file
9       $display("CHECKING THE OUTPUTS OF THE FULL ADDER");
10      $display ("Time \t A \t B \t Cin \t S \t Cout");
11      $monitor("%g \t %b \t %b \t %b \t %b \t %b", $time,a,b,Cin,sum,Cout); // monitoring the change of output in accordance with change of input
12
13      // applying the inputs to test the characteristics of the full adder
14
15      #0   {a,b,Cin}=3'b000;
16      #5   {a,b,Cin}=3'b001;
17      #10 {a,b,Cin}=3'b010;
18      #15 {a,b,Cin}=3'b011;
19      #20 {a,b,Cin}=3'b100;
20      #25 {a,b,Cin}=3'b101;
21      #30 {a,b,Cin}=3'b110;
22      #35 {a,b,Cin}=3'b111;
23      #40 $finish ;
24  end
25  full_adder FA(a,b,Cin,sum,Cout);
26  endmodule
```

HRITAM MITRA- **LINKDIN**

HRITAM MITRA- **GitHub**

SIMULATION IN CLI MODE

STEP 1 :-

```
rono@DESKTOP-6FVSKOG:/mnt/f/digital_vlsi_projects$ ls
fa_by_ha.v  tb_fa.v
```

Here **fa_by_ha.v** is ***source code***
And **tb_fa.v** is ***testbench code.***

STEP 2 :-

```
/mnt/f/digital_vlsi_projects$ iverilog fa_by_ha.v tb_fa.v -o test_fa
```

Here ***test_fa*** is the ***object dump file.***
Though it's optional.
By default a file will create automatically after linting by iverilog named as ***a.out***.

STEP 3 :-

```
rono@DESKTOP-6FVSKOG:/mnt/f/digital_vlsi_projects$ ls
fa_by_ha.v  tb_fa.v  test_fa
```

After successful simulation,the ***object dump file*** (***test_fa)*** has been created.

OUTPUT / RESULT

```
/mnt/f/digital_vlsi_projects$ ./test_fa
```

Executing the abovementioned ***object dump file*** by using this command line.

```
VCD info: dumpfile full_adder.vcd opened for output.
CHECKING THE OUTPUTS OF THE FULL ADDER
Time        A        B        Cin      S         Cout
0           0        0        0        0         0
5           0        0        1        1         0
15          0        1        0        1         0
30          0        1        1        0         1
50          1        0        0        1         0
75          1        0        1        0         1
105         1        1        0        0         1
140         1        1        1        1         1
```

This is the truth table of **full adder**.

Hence, the modelling of the **full adder** using half adder has been successfully done.

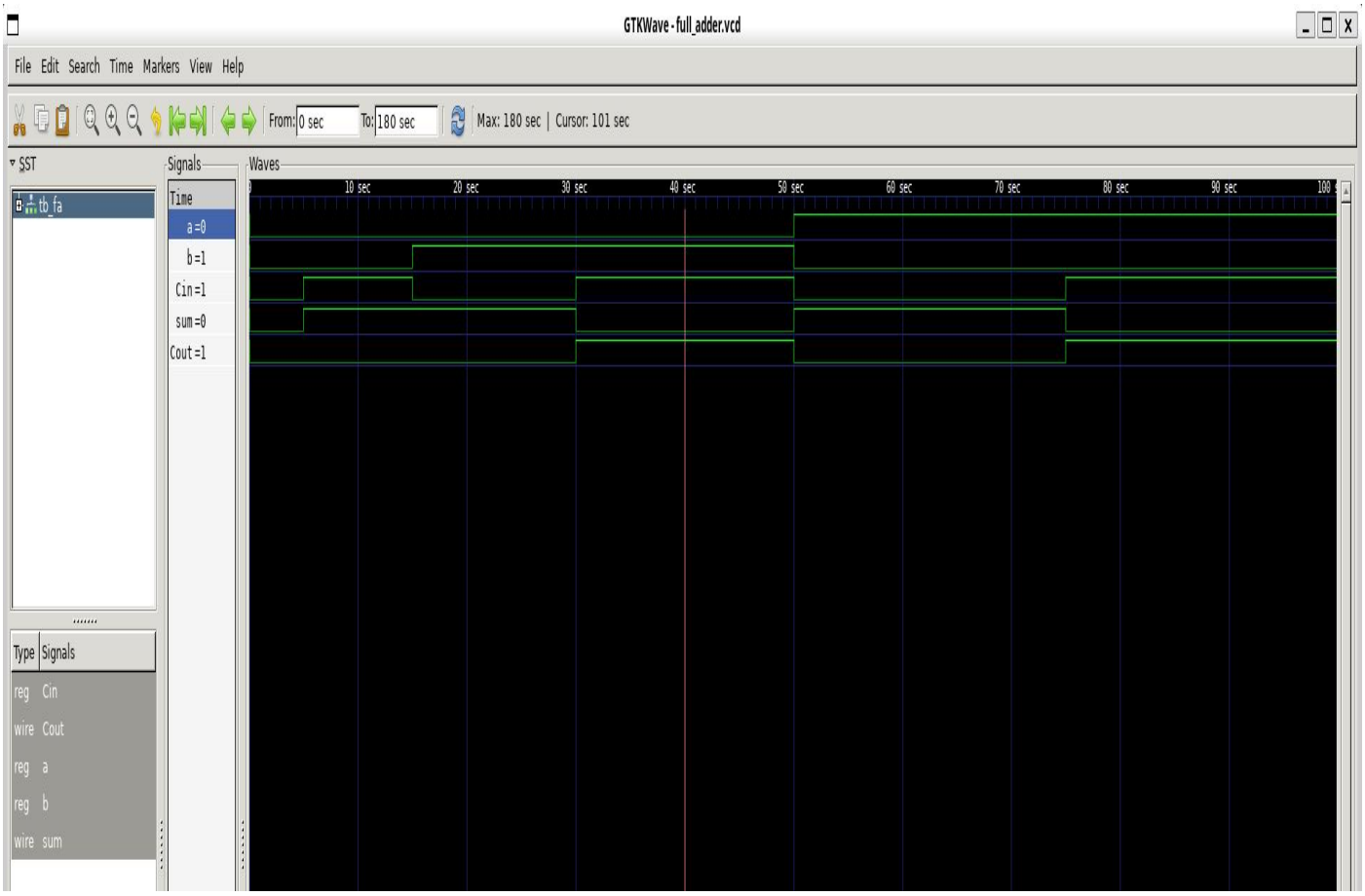
HRITAM MITRA- **LINKDIN**

HRITAM MITRA- **GitHub**

ANALYSIS OF WAVEFORM

USE THE COMMAND :-

gtkwave full_adder.vcd



BEHAVIORAL TO RTL LEVEL SYNTHESIS

STEP 1 :- Here we will use ***yosys*** for the **Behavioral level --> RTL --> Logic gate level --> Physical gate level synthesis**.

```
rono@DESKTOP-6FVSKOG:/mnt/f/digital_vlsi_projects/Synthesis/synthesis_2$ yosys

 /----------------------------------------------------------------------------\
 |                                                                            |
 |  yosys -- Yosys Open SYnthesis Suite                                       |
 |                                                                            |
 |  Copyright (C) 2012 - 2019  Clifford Wolf <clifford@clifford.at>           |
 |                                                                            |
 |  Permission to use, copy, modify, and/or distribute this software for any  |
 |  purpose with or without fee is hereby granted, provided that the above    |
 |  copyright notice and this permission notice appear in all copies.         |
 |                                                                            |
 |  THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES  |
 |  WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF          |
 |  MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR   |
 |  ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES    |
 |  WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN     |
 |  ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF   |
 |  OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.            |
 |                                                                            |
 \----------------------------------------------------------------------------/

 Yosys 0.9 (git sha1 1979e0b)

yosys>
```

STEP 2 :- Now, reading the verilog file (***fa_by_ha.v***).

```
yosys> read_verilog fa_by_ha.v
1. Executing Verilog-2005 frontend: fa_by_ha.v
Parsing Verilog input from `fa_by_ha.v' to AST representation.
Generating RTLIL representation for module `\half_adder'.
Generating RTLIL representation for module `\full_adder'.
Successfully finished Verilog frontend.
```

STEP 3 :- Now analyse of the ***hierarchy*** of our design.

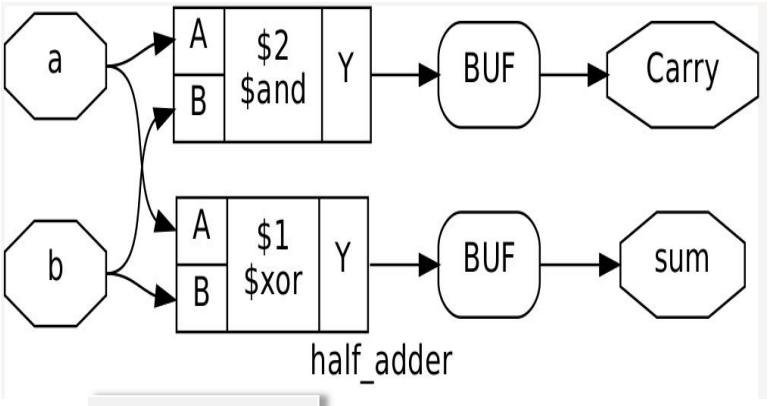
```
yosys> hierarchy -check -top full_adder

7. Executing HIERARCHY pass (managing design hierarchy).

7.1. Analyzing design hierarchy..
Top module:  \full_adder
Used module:     \half_adder

7.2. Analyzing design hierarchy..
Top module:  \full_adder
Used module:     \half_adder
Removed 0 unused modules.
```

HRITAM MITRA- **LINKDIN**

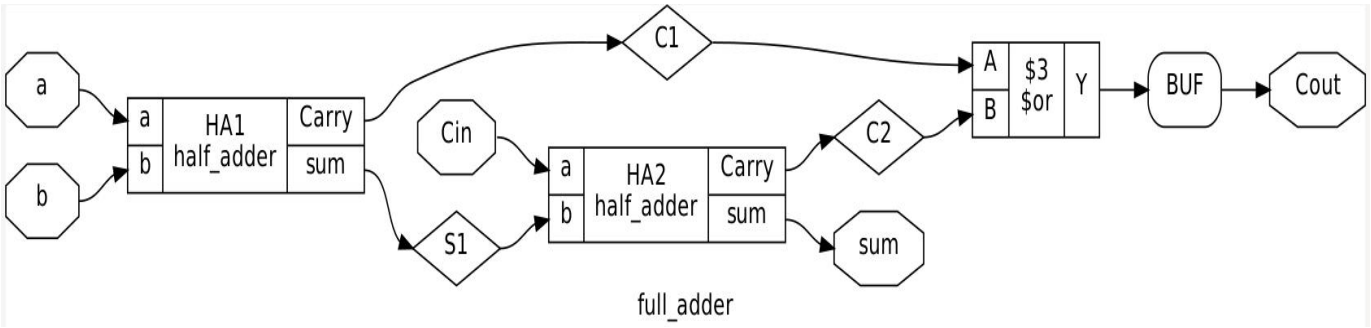
HRITAM MITRA- **GitHub**

RTL to logic gate level synthesis
(For HA and FA modules)

HALF ADDER



FULL ADDER



Instructing **Yosys** to write a verilog file after the logic gate level synthesis
(**FOR DETAILS PLEASE VISIT MY GITHUB REPO**)

Open | synth_1.v | Save

```
/* Generated by Yosys 0.9 (git sha1 1979e0b) */

module full_adder(a, b, Cin, sum, Cout);
  wire _0_;
  wire C1;
  wire C2;
  input Cin;
  output Cout;
  wire S1;
  input a;
  input b;
  output sum;
  assign _0_ = C1 | C2;
  half_adder HA1 (
    .Carry(C1),
    .a(a),
    .b(b),
    .sum(S1)
  );
  half_adder HA2 (
    .Carry(C2),
    .a(Cin),
    .b(S1),
    .sum(sum)
  );
  assign Cout = _0_;
endmodule

module half_adder(a, b, sum, Carry);
  wire _0_;
  wire _1_;
  output Carry;
  input a;
  input b;
  output sum;
  assign _0_ = a & b;
  assign _1_ = a ^ b;
  assign sum = _1_;
  assign Carry = _0_;
endmodule
```

Logic Gate Level to Physical Gate Level Synthesis

STEP 1 :-

Performing synthesis steps by using commands.

Printing the statistics after synthesis. There will be all the informations regarding the *number of gates*, *memory bits*, *cells*,*wires* etc.

```
yosys> synth -top full_adder

4. Executing SYNTH pass.

4.1. Executing HIERARCHY pass (managing design hierarchy).

4.1.1. Analyzing design hierarchy..
Top module:  \full_adder
Used module:     \half_adder

4.1.2. Analyzing design hierarchy..
Top module:  \full_adder
Used module:     \half_adder
Removed 0 unused modules.

4.2. Executing PROC pass (convert processes to netlists).

4.2.1. Executing PROC_CLEAN pass (remove empty switches from decision trees).
Cleaned up 0 empty switches.

4.2.2. Executing PROC_RMDEAD pass (remove dead branches from decision trees).
Removed a total of 0 dead cases.

4.2.3. Executing PROC_INIT pass (extract init attributes).

4.2.4. Executing PROC_ARST pass (detect async resets in processes).

4.2.5. Executing PROC_MUX pass (convert decision trees to multiplexers).

4.2.6. Executing PROC_DLATCH pass (convert process syncs to latches).

4.2.7. Executing PROC_DFF pass (convert process syncs to FFs).

4.2.8. Executing PROC_CLEAN pass (remove empty switches from decision trees).
```

```
17.26. Printing statistics.

=== full_adder ===

   Number of wires:                 8
   Number of wire bits:             8
   Number of public wires:          8
   Number of public wire bits:      8
   Number of memories:              0
   Number of memory bits:           0
   Number of processes:             0
   Number of cells:                 3
     $_OR_                          1
     half_adder                     2

=== half_adder ===

   Number of wires:                 4
   Number of wire bits:             4
   Number of public wires:          4
   Number of public wire bits:      4
   Number of memories:              0
   Number of memory bits:           0
   Number of processes:             0
   Number of cells:                 2
     $_AND_                         1
     $_XOR_                         1

=== design hierarchy ===

   full_adder                       1
     half_adder                     2

   Number of wires:                16
   Number of wire bits:            16
   Number of public wires:         16
   Number of public wire bits:     16
   Number of memories:              0
   Number of memory bits:           0
   Number of processes:             0
   Number of cells:                 5
     $_AND_                         2
     $_OR_                          1
     $_XOR_                         2
```

STEP 2 :-

Technology mapping by the tool , **abc**.
Here we are using the liberty file (**.lib**) of ***skywater 130 technology***.
(**FOR DETAILS PLEASE VISIT MY GITHUB REPO**)

```
yosys> abc -liberty sky130_fd_sc_ls__ff_085C_1v95.lib

5. Executing ABC pass (technology mapping using ABC).

5.1. Extracting gate netlist of module `\full_adder' to `<abc-temp-dir>/input.blif'.
Extracted 1 gates and 3 wires to a netlist network with 2 inputs and 1 outputs.
```

OUTPUT OF abc tool:-

```
5.2.2. Re-integrating ABC results.
ABC RESULTS:   sky130_fd_sc_ls__and2_1 cells:        1
ABC RESULTS:   sky130_fd_sc_ls__xor2_1 cells:        1
ABC RESULTS:        internal signals:        0
ABC RESULTS:           input signals:        2
ABC RESULTS:          output signals:        2
Removing temp directory.
```

TECHNOLOGY MAPPING

```
yosys> techmap

6. Executing TECHMAP pass (map to technology primitives).

6.1. Executing Verilog-2005 frontend: <techmap.v>
Parsing Verilog input from `<techmap.v>' to AST representation.
Generating RTLIL representation for module `\_90_simplemap_bool_ops'.
Generating RTLIL representation for module `\_90_simplemap_reduce_ops'.
Generating RTLIL representation for module `\_90_simplemap_logic_ops'.
Generating RTLIL representation for module `\_90_simplemap_compare_ops'.
Generating RTLIL representation for module `\_90_simplemap_various'.
Generating RTLIL representation for module `\_90_simplemap_registers'.
Generating RTLIL representation for module `\_90_shift_ops_shr_shl_sshl_sshr'.
Generating RTLIL representation for module `\_90_shift_shiftx'.
Generating RTLIL representation for module `\_90_fa'.
Generating RTLIL representation for module `\_90_lcu'.
Generating RTLIL representation for module `\_90_alu'.
Generating RTLIL representation for module `\_90_macc'.
Generating RTLIL representation for module `\_90_alumacc'.
Generating RTLIL representation for module `\$__div_mod_u'.
Generating RTLIL representation for module `\$__div_mod'.
Generating RTLIL representation for module `\_90_div'.
Generating RTLIL representation for module `\_90_mod'.
Generating RTLIL representation for module `\_90_pow'.
Generating RTLIL representation for module `\_90_pmux'.
Generating RTLIL representation for module `\_90_lut'.
Successfully finished Verilog frontend.

6.2. Continuing TECHMAP pass.
No more expansions possible.
```

OPTIMIZATION OF CELLS

```
yosys> opt

13. Executing OPT pass (performing simple optimizations).

13.1. Executing OPT_EXPR pass (perform const folding).
Optimizing module full_adder.
Optimizing module half_adder.

13.2. Executing OPT_MERGE pass (detect identical cells).
Finding identical cells in module `\full_adder'.
Finding identical cells in module `\half_adder'.
Removed a total of 0 cells.

13.3. Executing OPT_MUXTREE pass (detect dead branches in mux trees).
Running muxtree optimizer on module \full_adder..
  Creating internal representation of mux trees.
  No muxes found in this module.
Running muxtree optimizer on module \half_adder..
  Creating internal representation of mux trees.
  No muxes found in this module.
Removed 0 multiplexer ports.

13.4. Executing OPT_REDUCE pass (consolidate $*mux and $reduce_* inputs).
  Optimizing cells in module \full_adder.
  Optimizing cells in module \half_adder.
Performed a total of 0 changes.

13.5. Executing OPT_MERGE pass (detect identical cells).
Finding identical cells in module `\full_adder'.
Finding identical cells in module `\half_adder'.
Removed a total of 0 cells.

13.6. Executing OPT_RMDFF pass (remove dff with constant values).

13.7. Executing OPT_CLEAN pass (remove unused cells and wires).
Finding unused cells or wires in module \full_adder..
Finding unused cells or wires in module \half_adder..
Removed 0 unused cells and 3 unused wires.
<suppressed ~2 debug messages>

13.8. Executing OPT_EXPR pass (perform const folding).
Optimizing module full_adder.
Optimizing module half_adder.

13.9. Finished OPT passes. (There is nothing left to do.)
```

PRINTING THE SATISTICS
(NO. Of cells used & chip area for modules)

```
yosys> stat -liberty sky130_fd_sc_ls__ff_085C_1v95.lib

25. Printing statistics.

=== full_adder ===

   Number of wires:                 8
   Number of wire bits:             8
   Number of public wires:          8
   Number of public wire bits:      8
   Number of memories:              0
   Number of memory bits:           0
   Number of processes:             0
   Number of cells:                 3
     half_adder                     2
     sky130_fd_sc_ls__or2_1         1

   Area for cell type \half_adder is unknown!

   Chip area for module '\full_adder': 7.992000

=== half_adder ===

   Number of wires:                 4
   Number of wire bits:             4
   Number of public wires:          4
   Number of public wire bits:      4
   Number of memories:              0
   Number of memory bits:           0
   Number of processes:             0
   Number of cells:                 2
     sky130_fd_sc_ls__and2_1        1
     sky130_fd_sc_ls__xor2_1        1

   Chip area for module '\half_adder': 20.779200
```

```
   Chip area for module '\full_adder': 7.992000

=== half_adder ===

   Number of wires:                 4
   Number of wire bits:             4
   Number of public wires:          4
   Number of public wire bits:      4
   Number of memories:              0
   Number of memory bits:           0
   Number of processes:             0
   Number of cells:                 2
     sky130_fd_sc_ls__and2_1        1
     sky130_fd_sc_ls__xor2_1        1

   Chip area for module '\half_adder': 20.779200

=== design hierarchy ===

   full_adder                       1
     half_adder                     2

   Number of wires:                16
   Number of wire bits:            16
   Number of public wires:         16
   Number of public wire bits:     16
   Number of memories:              0
   Number of memory bits:           0
   Number of processes:             0
   Number of cells:                 5
     sky130_fd_sc_ls__and2_1        2
     sky130_fd_sc_ls__or2_1         1
     sky130_fd_sc_ls__xor2_1        2

   Chip area for top module '\full_adder': 49.550400
```

HRITAM MITRA- **LINKDIN**

HRITAM MITRA- **GitHub**

Writing the synth log file after technology mapping

```
yosys> write_verilog -noattr synth_2.v

14. Executing Verilog backend.
Dumping module `\full_adder'.
Dumping module `\half_adder'.
```

Synthesis log file after technology mapping

Open — synth_2.v — /mnt/f/digital_vlsi_projects/Synthesis

```
/* Generated by Yosys 0.9 (git sha1 1979e0b) */

module full_adder(a, b, Cin, sum, Cout);
  wire C1;
  wire C2;
  input Cin;
  output Cout;
  wire S1;
  input a;
  input b;
  output sum;
  sky130_fd_sc_ls__or2_1 _0_ (
    .A(C2),
    .B(C1),
    .X(Cout)
  );
  half_adder HA1 (
    .Carry(C1),
    .a(a),
    .b(b),
    .sum(S1)
  );
  half_adder HA2 (
    .Carry(C2),
    .a(Cin),
    .b(S1),
    .sum(sum)
  );
endmodule

module half_adder(a, b, sum, Carry);
  output Carry;
  input a;
  input b;
  output sum;
  sky130_fd_sc_ls__and2_1 _0_ (
    .A(b),
    .B(a),
    .X(Carry)
  );
  sky130_fd_sc_ls__xor2_1 _1_ (
    .A(b),
    .B(a),
    .X(sum)
  );
endmodule
```

We can use a script file for routine execution steps.
Though this step is optional. We can use this file instead of doing the baby steps.
(FOR DETAILS PLEASE VISIT MY GITHUB REPO)

Open — script.v — /mnt/f/digital_vlsi_projects/Synthesis — Save

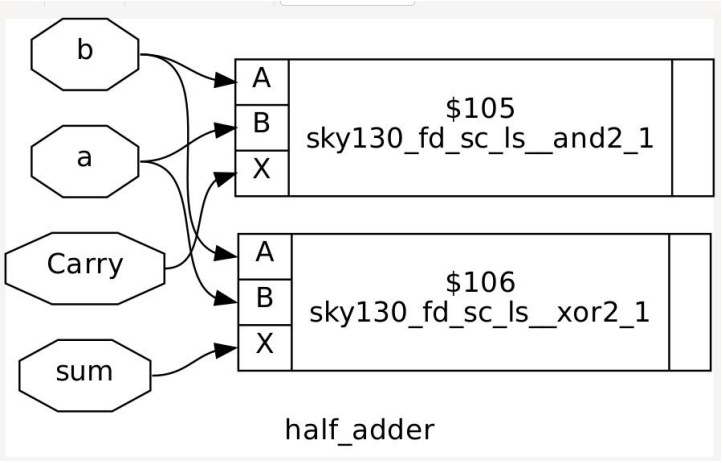
```
read_verilog fa_by_ha.v
hierarchy -check -top full_adder
write_verilog -noattr synth_1.v
synth -top full_adder
abc -liberty sky130_fd_sc_ls__ff_085C_1v95.lib
techmap
opt
write_verilog -noattr synth_2.v
stat -liberty sky130_fd_sc_ls__ff_085C_1v95.lib
```

HRITAM MITRA- **LINKDIN**

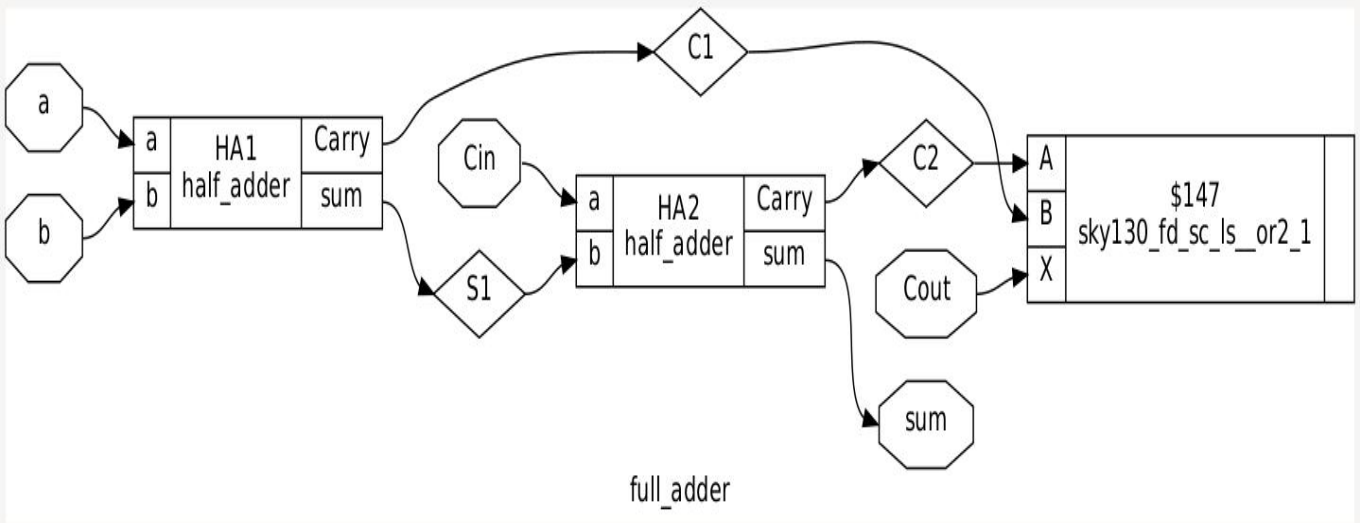
HRITAM MITRA- **GitHub**

Physical gate level synthesis
(For HA and FA modules)

HALF ADDER



FULL ADDER



THANK YOU