University of Tehran
College of Engineering
School of Electrical and Computer Engineering

Digital Systems 1

Dr.Navabi

Computer Assignment 3

Soroush Mesforush Mashhad

SN:810198472

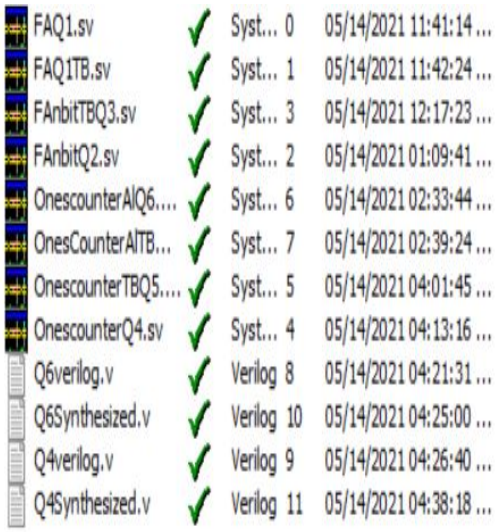
Ordibehesht 00

Abstract

In this assignment we start by creating a 1-bit full adder then we extend our definition to a n-bit full adder, then we combine these adders in six layers to create a 127-bit ones adder then we implement the adder with the always statement and lastly compare the optimization of our design to the designs given by yosys.

1 Project generated files

A full view of the generated and used SystemVerilog files in my project can be seen below:



FAQ1.sv	Syst...	0	05/14/2021 11:41:14 ...
FAQ1TB.sv	Syst...	1	05/14/2021 11:42:24 ...
FAnbitTBQ3.sv	Syst...	3	05/14/2021 12:17:23 ...
FAnbitQ2.sv	Syst...	2	05/14/2021 01:09:41 ...
OnescounterAlQ6....	Syst...	6	05/14/2021 02:33:44 ...
OnesCounterAlTB...	Syst...	7	05/14/2021 02:39:24 ...
OnescounterTBQ5....	Syst...	5	05/14/2021 04:01:45 ...
OnescounterQ4.sv	Syst...	4	05/14/2021 04:13:16 ...
Q6verilog.v	Verilog	8	05/14/2021 04:21:31 ...
Q6Synthesized.v	Verilog	10	05/14/2021 04:25:00 ...
Q4verilog.v	Verilog	9	05/14/2021 04:26:40 ...
Q4Synthesized.v	Verilog	11	05/14/2021 04:38:18 ...

Figure 1: Generated and Compiled .sv & .v files

2 Questions

2.1 Q1

2.1.1 Hand-Simulation

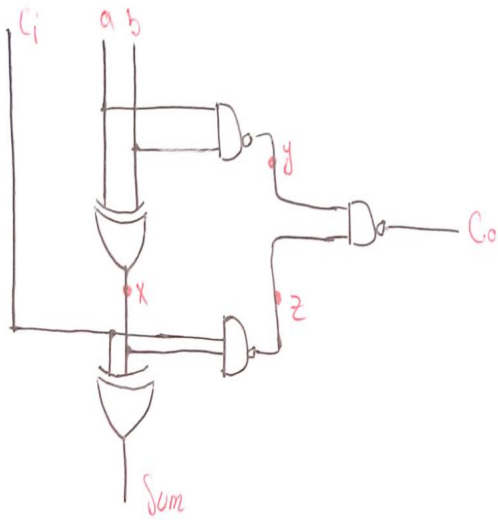


Figure 2: Hand Simulation

Based on the delays of computer assignment 1 we expect to get a 34ns delay to 1 and a 38 ns delay to 0(these are for the S output of course) for brevity we consider that the Co output comes out during these times which is not a wrongful assumption.

2.1.2 Verilog Code & Testbench

```verilog
`timescale 1ns/1ns
module FAQ1(input a,b,CIN,output SUM,COUT);
    wire x,y,z;
    nand #(10,8) (y,a,b);
    xor #(17,19) (x,a,b);
    nand #(10,8) (z,x,CIN);
    nand #(10,8) (COUT,y,z);
    xor #(17,19) (SUM,x,CIN);
endmodule
```

```verilog
`timescale 1ns/1ns
module FAQ1TB();
  logic aa=0;
  logic bb=0;
  logic ci=0;
  wire ss,ccout;
  FAQ1   UUT(aa,bb,ci,ss,ccout);
  initial begin
  #80 ci=1;
  #80 ci=0;
  #80 bb=1;
  #80 ci=1;
  #80 ci=0;
  #80 bb=0;
  #80 aa=1;
  #80 ci=1;
  #80 ci=0;
  #80 bb=1;
  #80 ci=1;
  #80 bb=0;
  #80 ci=0;
  #80 bb=1;
  #80 ci=1;
  #80 $stop;
  end
endmodule
```

Figure 3: Verilog Code & Testbench

2.1.3 Waveform

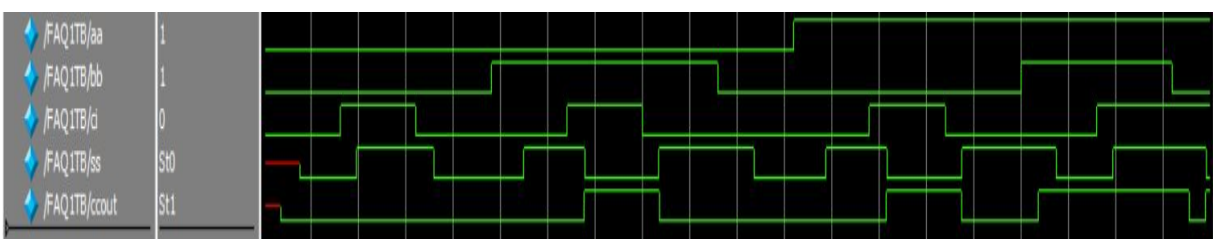


Figure 4: 1-bit FA Waveform

2.2 Q2 & Q3

2.2.1 Hand-Simulation

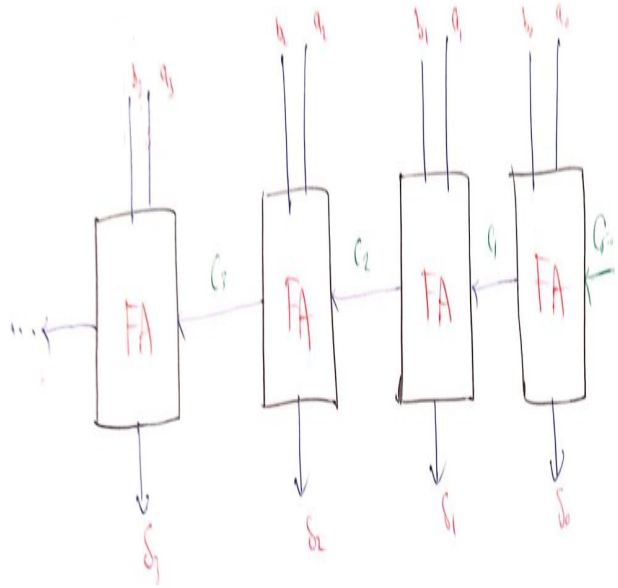


Figure 5: Hand Simulation

As we are using n full adders to make a n-bit full adder we shall multiply the delay of the previous adders by n accordingly to meet our demands.

2.2.2 Verilog Code & Testbench

```verilog
`timescale 1ns/1ns
module FAnbitQ2 #(parameter n = 4)( a,b,CIN,SUM,COUT);
  input [n-1:0]a,b;
  input CIN;
  output[n-1:0] SUM;
  output COUT;
  assign #(n*38) {COUT,SUM}=a+b+CIN;
endmodule
```

```verilog
`timescale 1ns/1ns
module FAnbitTBQ3();
  logic [3:0] aa=4'b1100;
  logic [3:0] bb=4'b1001;
  logic ci=0;
  wire[3:0] ss;
  wire ccout;
  FAnbitQ2 #4 UUT(aa,bb,ci,ss,ccout);
  initial begin
  repeat(12)#250
        {aa,bb}=$random();
  #250 $stop;
  end
endmodule
```

Figure 6: Verilog Code & Testbench

2.2.3 Waveform

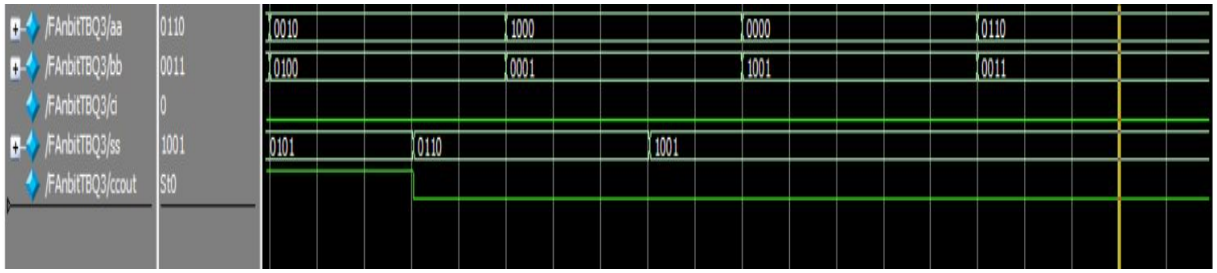


Figure 7: 4-bit FA Waveform

2.3 Q4 & Q5

2.3.1 Hand-Simulation

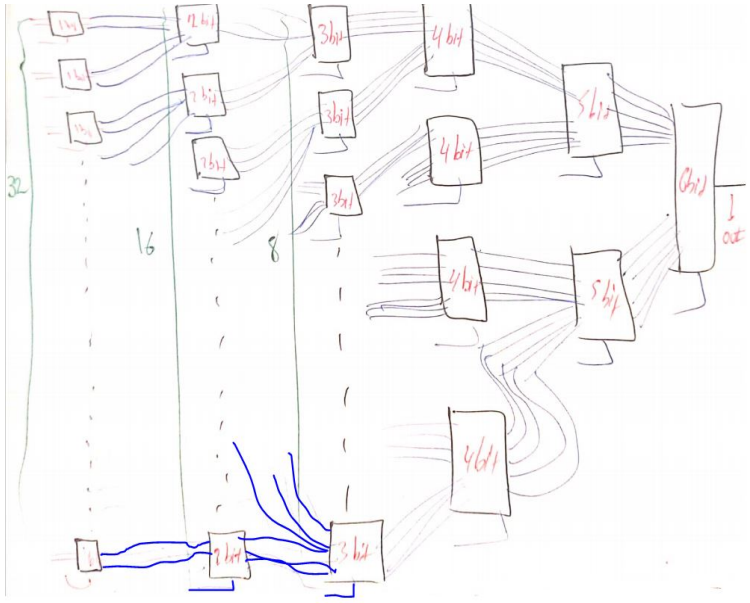


Figure 8: Hand Simulation

The concept of this part is to have a tree implementation consisted of 6 layers,to do this we withdraw some wires to give to the carry inputs of the adders,because we are adding it is not important which wire goes to which carry,hence we complete our design.

2.3.2 Verilog Code & Testbench

```verilog
`timescale 1ns/1ns
module Onescounter04 #(parameter n = 6) (input [2 ** (n + 1) - 2:0]a,output [n:0]OUT);
  wire [(2**n)-1:0]V1;
  wire [3*(2**(n-2))-1:0]V2;
  wire[2**(n-1)-1:0]V3;
  wire [19:0]V4;
  wire[11:0]V5;
  wire[6:0]V6;
  genvar i;
  generate
    for(i=0;i<=2**(n-1)-1;i=i+1)begin:Veneer1
      FAnbitQ2 #(1) FA(.a(a[3*i]),.b(a[3*i+1]),.CIN(a[3*i+2]),.SUM(V1[2*i]),.COUT(V1[2*i+1]));
    end
    for(i=0;i<=2**(n-2)-1;i=i+1)begin:Veneer2
      FAnbitQ2 #(2) FA(.a(V1[4*i+1:4*i]),.b(V1[4*i+3:4*i+2]),.CIN(a[i+96]),.SUM(V2[3*i+1:3*i]),.COUT(V2[3*i+2]));
    end
    for(i=0;i<=2**(n-3)-1;i=i+1)begin:Veneer3
      FAnbitQ2 #(3) FA(.a(V2[6*i+2:6*i]),.b(V2[6*i+5:6*i+3]),.CIN(a[i+112]),.SUM(V3[4*i+2:4*i]),.COUT(V3[4*i+3]));
    end
    for(i=0;i<=2**(n-4)-1;i=i+1)begin:Veneer4
      FAnbitQ2 #(4) FA(.a(V3[8*i+3:8*i]),.b(V3[8*i+7:8*i+4]),.CIN(a[i+120]),.SUM(V4[5*i+3:5*i]),.COUT(V4[5*i+4]));
    end
    for(i=0;i<=2**(n-5)-1;i=i+1)begin:Veneer5
      FAnbitQ2 #(5) FA(.a(V4[10*i+4:10*i]),.b(V4[10*i+9:10*i+5]),.CIN(a[i+124]),.SUM(V5[6*i+4:6*i]),.COUT(V5[6*i+5]));
    end
    FAnbitQ2 #(6) FA(.a(V5[5:0]),.b(V5[11:6]),.CIN(a[126]),.SUM(V6[5:0]),.COUT(V6[6]));
  endgenerate
  assign OUT=V6;
endmodule
```

```verilog
`timescale 1ns/1ns
module OnescounterTBQ5;
        logic [126 : 0] aa;
        wire [6 : 0] ww;
        Onescounter04 #6 UUT(aa, ww);
        initial begin
                repeat (12) #1000 aa=$random();
                #1000;
                #1000 aa = 127'b1;
                repeat(254) #1000 aa={~aa[0],aa[126:1]};
                #1000 $stop;
        end
endmodule
```

Figure 9: Verilog Code & Testbench

2.3.3 Waveform

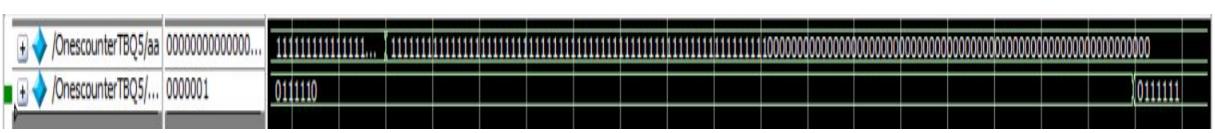


Figure 10: Ones counter Waveform

2.4 Q6

In this part similiar to C-Programming we add the bits of the number one by one and since we have annotated the delays previously we get the following result.

2.4.1 Verilog Code & Testbench

```
`timescale 1ns/1ns
module OnescounterAlQ6 #(parameter n = 6) (input [0 : 2 ** (n + 1) - 2] in,output [n : 0] out);
        integer k=0;
        integer Dummy=0;
        always@(in) begin
              Dummy=0;
              for(k=0;k<=126;k=k+1)begin
                    Dummy=Dummy+in[k];
              end
        end
        assign #798 out=Dummy;
endmodule
```

```
`timescale 1ns/1ns
module OnescounterAlTBQ6;
        logic [126 : 0] aa;
        wire [6 : 0] ww;
        OnescounterAlQ6 #6 UUT(aa, ww);
        initial begin
              repeat (12) #1000 aa=$random();
              #1000;
              #1000 aa = 127'b1;
              repeat(254) #1000 aa={~aa[0],aa[126:1]};
              #1000 $stop;
        end
endmodule
```

Figure 11: Verilog Code & Testbench

2.4.2 Waveform

Figure 12: Ones counter Waveform

2.5 Q7

Here we use yosys to synthesize the codes of part 4 and 6 the results are as follows:

```
=== OnescounterAlQ6 ===

   Number of wires:                 682
   Number of wire bits:             845
   Number of public wires:            3
   Number of public wire bits:      166
   Number of memories:                0
   Number of memory bits:             0
   Number of processes:               0
   Number of cells:                 686
     $_AND_                           9
     $_AOI3_                         43
     $_NAND_                         18
     $_NOR_                         231
     $_NOT_                         100
     $_OAI3_                          9
     $_OR_                           18
     $_XNOR_                        223
     $_XOR_                          35
```

```
4.1.2. Re-integrating ABC results.
ABC RESULTS:              NAND cells:      554
ABC RESULTS:               NOR cells:      752
ABC RESULTS:               NOT cells:      284
ABC RESULTS:        internal signals:      679
ABC RESULTS:           input signals:      127
ABC RESULTS:          output signals:        7
Removing temp directory.
```

Figure 13: Synthesis of art 4

```
   OnescounterQ4                          1
     $paramod\FAnbitQ2\n=1               32
     $paramod\FAnbitQ2\n=2               16
     $paramod\FAnbitQ2\n=3                8
     $paramod\FAnbitQ2\n=4                4
     $paramod\FAnbitQ2\n=5                2
     $paramod\FAnbitQ2\n=6                1

   Number of wires:                     692
   Number of wire bits:                1172
   Number of public wires:              323
   Number of public wire bits:          803
   Number of memories:                    0
   Number of memory bits:                 0
   Number of processes:                   0
   Number of cells:                     552
     $_AND_                              50
     $_AOI3_                             46
     $_NAND_                             85
     $_NOR_                               1
     $_NOT_                              44
     $_OAI3_                             80
     $_OR_                                6
     $_XNOR_                            128
     $_XOR_                             112
```

```
4.6.2. Re-integrating ABC results.
ABC RESULTS:            NAND cells:     18
ABC RESULTS:             NOR cells:     39
ABC RESULTS:             NOT cells:     19
ABC RESULTS:       internal signals:    25
ABC RESULTS:         input signals:     13
ABC RESULTS:        output signals:      7
Removing temp directory.
```

Figure 14: Synthesis of art 6

Due to the concept which is repeatedly mentioned in class which I quote ”There is no free lunch” because we have designed the counter of problem 4 with more difficulty and precission, hence when we synthesize it with yosys it shall use less gates to give us a fully functional circuit.