Ryan Writz

Priyam Shah

ECE 4/581

Project 3

8/8/19

1:

Note as the conversion of gray code preserves the MSB and XORs everything else the Binary to Gray Code and vice versa processes are identical.

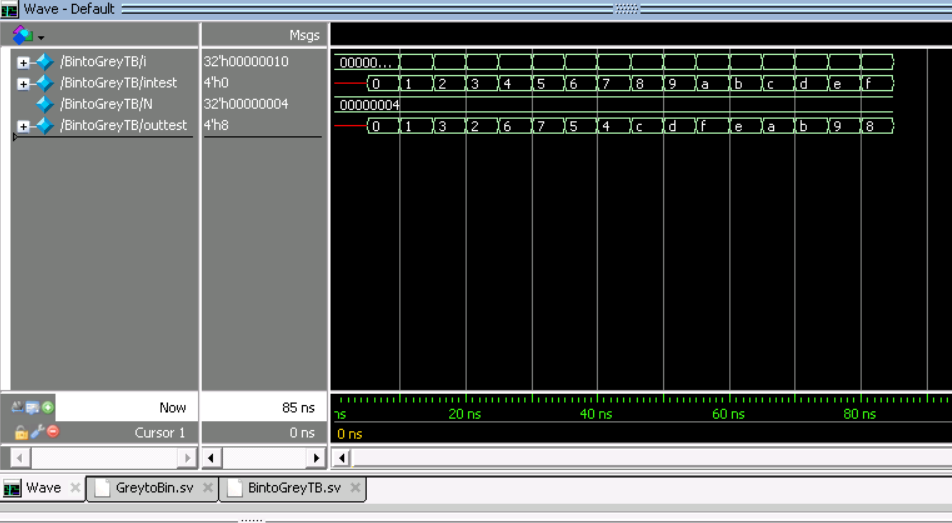
Layout of 5 Bit Gray Code Adder



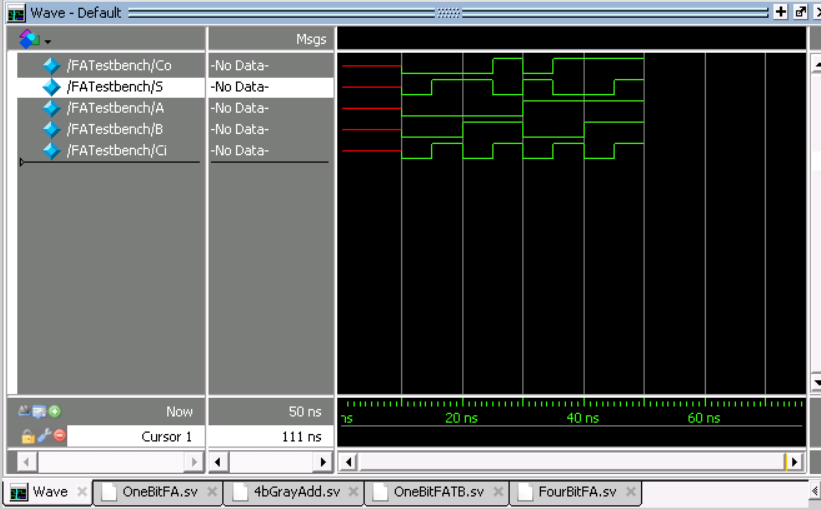
Gray Code Sequence Layout Input to Output

| Original Decimal | Original BCD | Result BCD | Result Decimal |
| --- | --- | --- | --- |
| 0 | 0000 | 0000 | 0 |
| 1 | 0001 | 0001 | 1 |
| 3 | 0011 | 0010 | 2 |
| 2 | 0010 | 0011 | 3 |
| 7 | 0111 | 0100 | 4 |
| 6 | 0110 | 0101 | 5 |
| 4 | 0100 | 0110 | 6 |
| 5 | 0101 | 0111 | 7 |
| 15 | 1111 | 1000 | 8 |
| 14 | 1110 | 1001 | 9 |
| 12 | 1100 | 1010 | 10 |
| 13 | 1101 | 1011 | 11 |
| 8 | 1000 | 1100 | 12 |
| 9 | 1001 | 1101 | 13 |
| 11 | 1011 | 1110 | 14 |
| 10 | 1010 | 1111 | 15 |

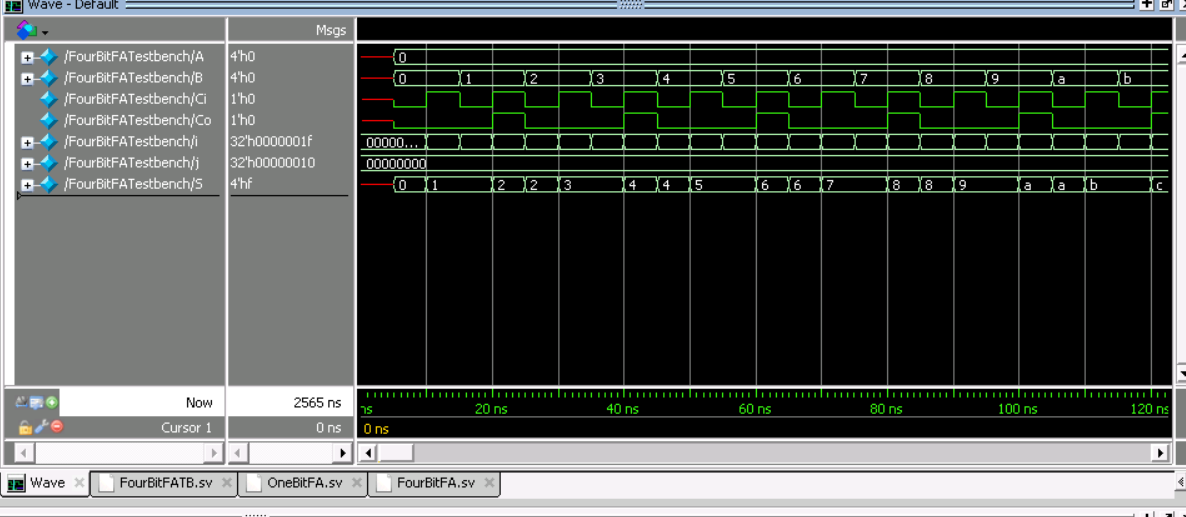
4 Bit Binary to Gray Code/ Gray Code to Binary Code Results

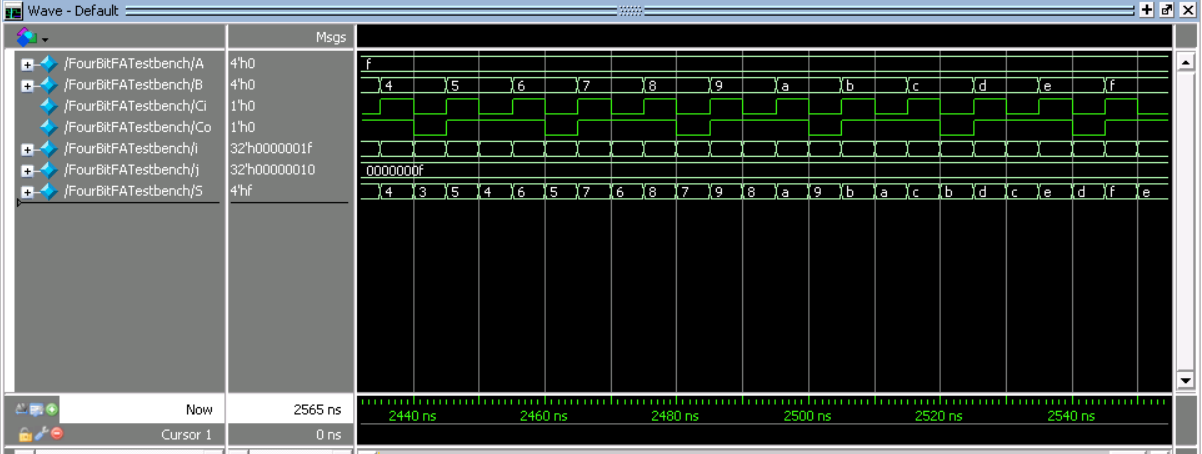


OneBit FA Results

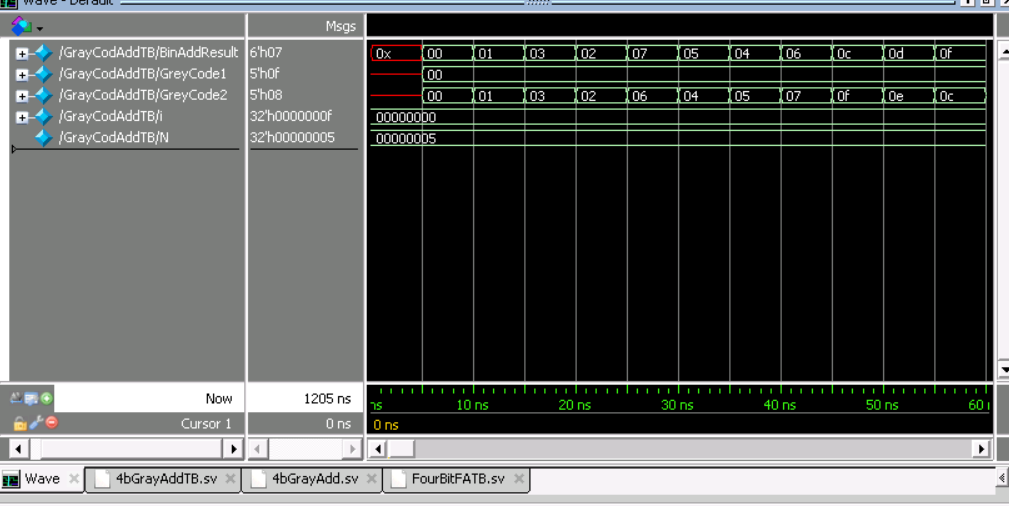


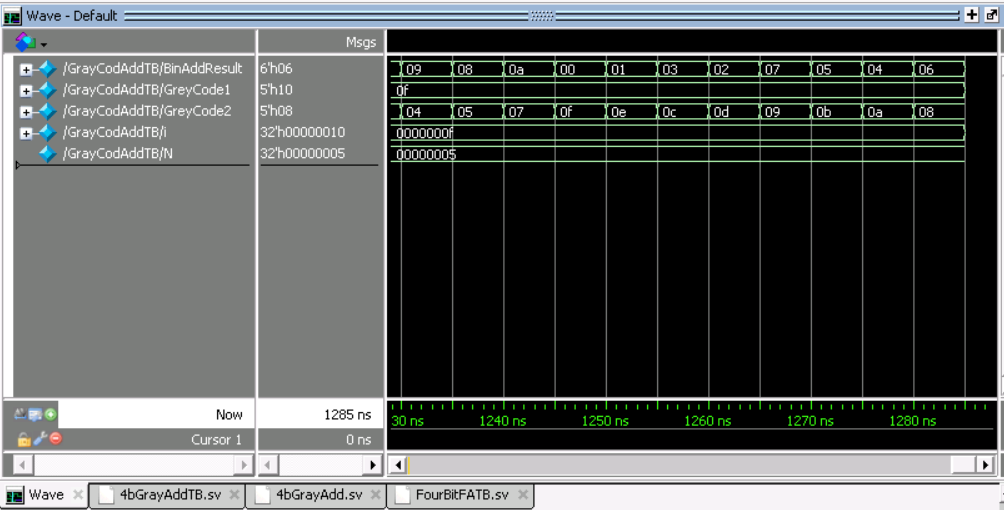
FourBit FA Results





Gray Code 5 bit Add Results





GreytoBin.sv / BintoGrey.sv (Code is identical except for module name as the same principles apply for transition either way)

module GreyToBin #(parameter Bits = 4)

(output logic [Bits - 1:0]out,

input logic [Bits -1:0]in);

int i;

always\_comb

begin

for(i = 0; i < Bits; i++)

begin

if(i == Bits- 1) //If MSB

begin //MSB of output

out[i] = in[Bits-1]; //equal to MSB of input

end

else //Else Output bit

begin //Equal to current bit XOR

out[i] = in[i] ^ in[i+1]; //next highest bit

end

end

end

endmodule

GreytoBinTB.sv

module BintoGreyTB();

parameter N = 4;

logic [N-1:0] intest;

logic [N-1:0] outtest;

GreyToBin #(N) F1(outtest,intest);

int i;

initial

begin

#5 intest = 0;

for(i = 0; i < (2\*\*(N));i++)

begin

#5 intest = intest + 1;

end

end

endmodule

OneBitFA.sv

module OneBitFA

(output logic Co, S,

input logic A, B, Ci);

logic X,Y,Z;

always\_comb

begin

X = A ^ B;

S = Ci ^ X;

Y = Ci & X;

Z = A & B;

Co = Y | Z;

end

endmodule

OneBitFATB.sv

module FATestbench;

logic Co, S;

logic A, B, Ci;

OneBitFA FATest(Co, S, A, B, Ci);

initial

begin

#5ns;

#5ns A = 0; B = 0; Ci = 0; //{A,B,Ci} = 000

#5ns Ci = 1; //{A,B,Ci} = 001

#5ns B = 1; Ci = 0; //{A,B,Ci} = 010

#5ns Ci = 1; //{A,B,Ci} = 011

#5ns A = 1; B = 0; Ci = 0; //{A,B,Ci} = 100

#5ns Ci = 1; //{A,B,Ci} = 101

#5ns B = 1; Ci = 0; //{A,B,Ci} = 110

#5ns Ci = 1; //{A,B,Ci} = 111

#5ns;

end

endmodule

FourBitFA.sv

module FourBitFA

(output logic Co, [3:0]S,

input logic [3:0]A, B,

input logic Ci);

logic Co2,Co3,Co4;

OneBitFA FA1(Co, S[0], A[0], B[0], Ci);

OneBitFA FA2(Co2, S[1], A[1], B[1], Co);

OneBitFA FA3(Co3, S[2], A[2], B[2], Co2);

OneBitFA FA4(Co4, S[3], A[3], B[3], Co3);

endmodule

FourBit FATB.sv

module FourBitFATestbench();

logic Co,Ci;

logic [3:0]S,A,B;

FourBitFA FourFATest(Co, S, A, B, Ci);

int i,j;

initial

begin

#5ns; B=0; A = 0; Ci = 0;

for(j=0;j < 16; j++)

begin

for(i = 0; i < 31; i++) //For each value of A, check

begin //each of the 16 B cases and

#5ns Ci = ~Ci; //2 Ci options resulting in 32 options

if(i%2 == 0)

B = B;

else

B = B +1;

end

#5ns A = A+1;

end

ends

endmodule

4bGrayAdd.sv

//Ryan Writz and Priyam Shah (c) July 2019

//ECE 4/581 Project 2 Question 1

//Designed a 4-bit grey code adder by creating two 4-bit grey-to-binary converters,

// a 4-bit binary adder, and a 5-bit binary-to-grey code convertor, modeling the

// design as a combinational block, ie. two 4 bit grey code inputs

//d) Write one test bench to verify the SV model.

//e) Synthesize the design and show the complexity (gate counts), area, and power of synthesized circuits using report command from DC. (Choose osu05\_stdcells.db as your target library, generate the netlist file in Verilog)

//f) Go back to your simulator, and simulate the synthesized Verilog netlist by including std05\_stdcell.v as the standard cell library.

//g) By simulation, verify the SV model and the synthesized Verilog netlist with the same testbench.

module NBitGrayCodeAdd #(parameter N = 5)

(output logic [N:0] BinAddResult,

input logic [N-1:0] GreyCode1,GreyCode2);

logic [N-1:0] BinCode1,BinCode2,Sum;

logic Ci = 0;

logic Co;

GreyToBin #(N-1) GTBC1(BinCode1,GreyCode1); //1st 4-bit grey-to-binary converter

GreyToBin #(N-1) GTBC2(BinCode2,GreyCode2); //2nd 4-bit grey-to-binary converter

FourBitFA AddBinFour(Co, Sum, BinCode1,BinCode2,Ci); //Four Bit Binary Adder

GreyToBin #(N) BTGC1(BinAddResult,{Co,Sum}); //5-bit binary-to-grey code converter

endmodule

4bGrayAddTB.sv

module GrayCodAddTB();

parameter N = 5;

logic [N:0]BinAddResult;

logic [N-1:0] GreyCode1,GreyCode2;

NBitGrayCodeAdd #(N)

GCAdd(BinAddResult,GreyCode1,GreyCode2);

int i;

initial

begin

#5 GreyCode1 = 0; GreyCode2 = 0;

for(i = 0; i <16; i++)

begin

#5 GreyCode2 = 1;

#5 GreyCode2 = 3;

#5 GreyCode2 = 2;

#5 GreyCode2 = 6;

#5 GreyCode2 = 4;

#5 GreyCode2 = 5;

#5 GreyCode2 = 7;

#5 GreyCode2 = 15;

#5 GreyCode2 = 14;

#5 GreyCode2 = 12;

#5 GreyCode2 = 13;

#5 GreyCode2 = 9;

#5 GreyCode2 = 11;

#5 GreyCode2 = 10;

#5 GreyCode2 = 8;

#5 GreyCode1 = GreyCode1 + 1;

end

end

endmodule

2:



module FSM(output logic out, input logic in, clk);

enum {A, B, C, D, E} state, next\_state;

always\_ff@(posedge clk)

begin

case (state)

A: next\_state = B;

B: next\_state = C;

C: next\_state = D;

D: next\_state = E;

E: next\_state = A;

endcase

end

if(in == 1)

begin

out == 1;

end

endmodule

module FSM(output logic out, input logic in, clk);

enum {A, B, C, D, E} state, next\_state;

always\_ff@(posedge clk)

begin

case (state)

A: next\_state = B;

B: next\_state = C;

C: next\_state = D;

D: next\_state = E;

E: next\_state = A;

endcase

end

if(in == 1)

begin

out == 1;

end

endmodule

module TB();

logic out, in, clk;

int i = 0;

initial

begin

clk = 0;

end

always\_comb

begin

#5 clk = ~clk;

end

FSM(out, in, clk);

//FSM\_reverse\_case(out, in, clk);

endmodule

3:

typedef enum { OFF, //power off

RED, //red state

YELLOW, //yellow state

GREEN, //green state

PRE\_GREEN //state before green

}

lights\_t;

module trafficlight

(

output lights\_t ns\_light, //North-South light status, main road

output lights\_t ew\_light, //East-West light status

input ew\_sensor, //East-West sensor for new car

input emgcy\_sensor, //emergency sensor

input reset\_n, //synchronous reset

input clk //master clock

);

timeunit 1ns;

timeprecision 100ps;

parameter FAIL =1'b0;

logic [1:0] ns\_green\_timer; //timer for NS light in GREEN

logic [1:0] ew\_green\_timer; //timer for EW light in GREEN

logic reset\_s;

int i = 0;

int j = 0;

always\_comb

begin

reset\_s = reset\_n && clk;

end

always\_ff @(posedge clk)

begin

if(reset\_s == 0)

begin

if(i!=3)

begin

for(i = 0; i < 3; i++)

begin

ns\_green\_timer <= i;

end

end

else if (i == 3)

begin

i = 0;

ns\_green\_timer <= 0;

end

end

else if(reset\_s == 1 || ns\_light yellow == 1)

begin

ns\_green\_timer <= 0;

end

end

always\_ff @(posedge clk)

begin

if (reset\_s == 0)

begin

if (j != 3)

begin

for (j = 0; j < 3; j++)

begin

ew\_green\_timer <= j;

end

end

else if (j == 3)

begin

j = 0;

ew\_green\_timer <= j;

end

end

else if(reset\_s == 1 || ew\_light yellow == 1)

begin

ew\_green\_timer <= 0;

end

end

always\_ff @(posedge clk)

begin

if(ew\_sensor == 0 && emgcy\_sensor ==0)

begin

ns\_light red <= 0;

ns\_light pre\_greem <= 0;

ns\_light yellow <= 0;

ns\_light off <= 0;

ns\_light green <= 1;

ew\_light red <= 1;

ew\_light pre\_greem <= 0;

ew\_light yellow <= 0;

ew\_light off <= 0;

ew\_light green <= 0;

end

else if(ns\_light green == 1 && emgcy\_sensor==1)

begin

ns\_light red <= 0;

ns\_light pre\_greem <= 0;

ns\_light yellow <= 1

ns\_light off <= 0;

ns\_light green <=0;

ew\_light red <= 1;

ew\_light pre\_greem <= 0;

ew\_light yellow <= 0;

ew\_light off <= 0;

ew\_light green <= 0;

end

else if(ew\_light pre\_green == 1)

begin

for(ew\_green\_timer = ew\_green\_timer + 1)

begin

ew\_light green <= 1

end

end

else if (ew\_light yellow == 1)

begin

ew\_light yellow = 0;

ew\_light red == 1;

end

end

endmodule