UNIVERSAL CODE CONVERTER

In verilog

By Dhruv Khodke

Table of Content

1. **Introduction**
2. **Block Diagram**
3. **Verilog Modules**

3.1 Verilog Module

3.2 Verilog Test Bench

4. **Simulation Result**

4.1 Simulation Result

1) INTRODUCTION

The Universal Code Converter (UCC) is a digital system designed to convert numerical data between different coding formats used in digital electronics. These formats include Binary, Gray code, BCD (Binary Coded Decimal), and Excess-3 code. Such conversions are fundamental in digital systems where different devices or circuits represent numbers using various encoding schemes. The UCC module provides a flexible way to interconvert these codes, ensuring compatibility and data integrity between different parts of a digital system.

The converter is implemented using Verilog HDL (Hardware Description Language), which allows hardware modeling and simulation of digital systems. The module can be synthesized on FPGAs or used for simulation in tools like ModelSim, Xilinx Vivado, or Quartus.

Objective

The main objective of this project is to design and implement a Verilog-based module that can perform code conversion between multiple number systems. The system should be universal — capable of performing any combination of input-to-output code conversion depending on the control signals provided by the user.

Supported Code Conversions

The module supports the following conversions:

- ✓ Binary to Gray code
- ✓ Binary to BCD (Binary Coded Decimal)
- ✓ Binary to Excess-3
- ✓ Gray to Binary
- ✓ Gray to BCD
- ✓ Gray to Excess-3
- ✓ BCD to Binary
- ✓ BCD to Gray
- ✓ BCD to Excess-3
- ✓ Excess-3 to Binary
- ✓ Excess-3 to Gray
- ✓ Excess-3 to BCD

Module Description

The UCC module is designed as a combinational logic system. It takes an 8-bit input in, two 2-bit selection inputs selin and selout, and produces an 8-bit output out. Based on the values of selin and selout, the system determines which conversion to perform.

Input Signals:

in[7:0] : 8-bit input value (can represent binary, BCD, Gray, or Excess-3 code)
selin[1:0] : Selects the input code type
selout[1:0] : Selects the desired output code type

Output Signals:

out[7:0] : Converted output value
errors : Indicates invalid inputs or conversion errors
output_ : 5-bit representation of the converted output
input_ : 5-bit representation of the original input
Each conversion sub-module (like btog, btoex, bcdtog, etc.) handles one specific conversion logic.

Working Principle

The UCC module uses case selection controlled by the concatenated signal **{selin, selout}**. This 4-bit signal uniquely identifies one of 16 possible conversion cases. For example:

4'b0000 → Binary to Binary (no conversion)
4'b0001 → Binary to Gray
4'b0010 → Binary to BCD
4'b0011 → Binary to Excess-3
4'b0100 → Gray to Binary
4'b0101 → Gray to Gray (no conversion)
4'b0110 → Gray to BCD
4'b0111 → Gray to Excess-3
and so on...

Depending on the selected case, the output is assigned from one of the internal sub-module outputs (out_btog, out_gtobcd, etc.).

Whenever a conversion involving invalid BCD values or out-of-range numbers occurs, the module sets the errors flag to 1, indicating the user must correct the input.

Error Detection

Some conversions require checking the validity of input data. For example:
In BCD representation, valid digits are from 0000 to 1001 (0–9).
Inputs above 9 in BCD are invalid and should raise an error signal.
Similarly, invalid binary-to-Excess-3 or BCD-to-Excess-3 inputs are handled by setting the error output high.
This helps prevent undefined outputs during simulation or hardware implementation.

Internal Modules

The UCC module instantiates multiple smaller converter modules, each performing a specific function.
Examples include:
btog: Binary to Gray code converter
btoex: Binary to Excess-3 converter
bcdtog: BCD to Gray code converter
extog: Excess-3 to Gray code converter
bcdtob: BCD to Binary converter

Each sub-module has its own logic to perform the bit-wise conversion. For example:
Binary to Gray: Gray[i] = Binary[i+1] XOR Binary[i]
Gray to Binary: Cumulative XOR decoding from MSB to LSB.
Binary to Excess-3: Excess-3 = Binary + 3
BCD to Excess-3: Each decimal digit adds 3 to its binary equivalent.

Design Flow

Input Selection: User provides 8-bit input and selects source and destination code using selin and selout.
Module Activation: Based on selection, only the relevant conversion sub-module output is used.
Error Checking: Any invalid input is detected and reported.

Output Generation: The converted code is displayed on the out port, and input-output mappings are shown via input_ and output_.

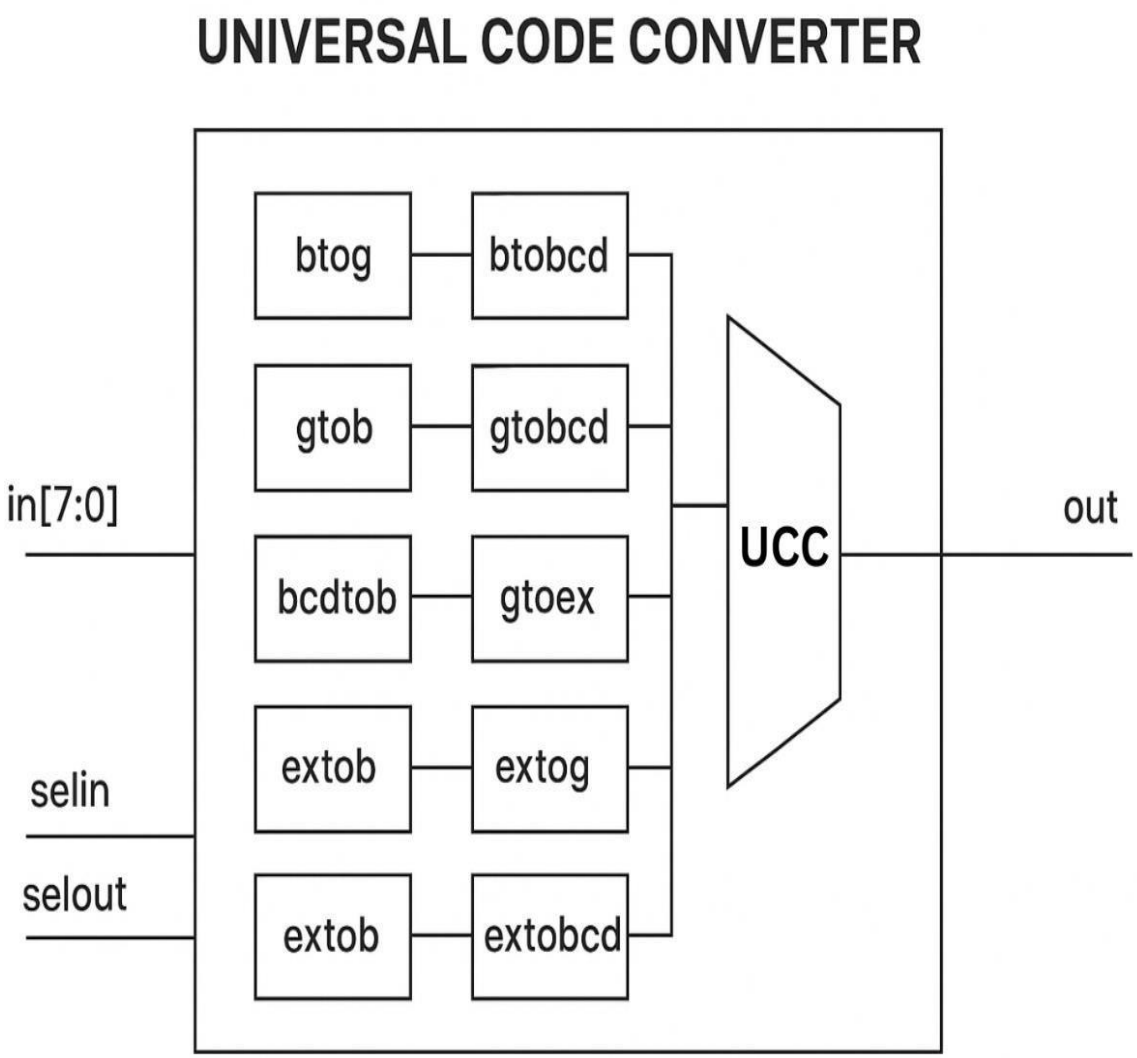
Applications

Digital communication systems (encoding and decoding).
Microcontroller or microprocessor interfacing where code compatibility is required.
Data transmission and error detection circuits.
Digital signal processing applications involving multiple number systems.

Advantages

Universal – handles multiple conversions in one module.
Modular design – easy to extend with new converters.
Error detection capability for invalid codes.
Suitable for FPGA/ASIC design and educational experiments.

2) BLOCK DIAGRAM



3) VERILOG MODULE

1) Main UCC (Universal code converter) module

```
module UCC(
    input [7:0] in,
    output reg [7:0] out,
    output reg errors,
    output reg [4:0]output_,
    output reg [4:0]input_,
    input [1:0] selin,
    input [1:0] selout
    );
    wire [7:0]error;
wire [7:0]out_btobcd, out_gtobcd, out_extobcd;
wire [3:0]out_btog, out_btoex, out_bcdtob,out_bcdtog, out_bcdtoex, out_extob, out_extog, out_gtob, out_gtoex;
wire [3:0]sel={selin, selout};

btog m1(.b(in[3:0]),.g(out_btog));
btobcd m2(.b(in[3:0]),.bcd(out_btobcd));
btoex m3(.b(in[3:0]),.ex(out_btoex), .error(error[0]));
gtob m4(.g(in[3:0]),.b(out_gtob));
gtobcd m5(.g(in[3:0]),.bcd(out_gtobcd));
gtoex m6(.g(in[3:0]),.ex(out_gtoex),.error(error[1]));
bcdtob m7(.bcd(in),.b(out_bcdtob), .error(error[2]));
bcdtog m8(.bcd(in),.g(out_bcdtog), .error(error[3]));
bcdtoex m9(.bcd(in),.ex(out_bcdtoex), .error(error[4]));
extob m10(.ex(in[3:0]),.b(out_extob), .error(error[5]));
extog m11(.ex(in[3:0]),.g(out_extog), .error(error[6]));
extobcd m12(.ex(in[3:0]),.bcd(out_extobcd), .error(error[7]));

always @(*)
begin
    case (sel)
    4'b0000:begin out=in[3:0];
    output_=out[4:0];
    input_=in[4:0];
    errors=0;
    end
    4'b0001:begin out=out_btog[3:0];
    output_=out[4:0];
    input_=in[4:0];
    errors=0;
    end
    4'b0010:begin out=out_btobcd;
    output_=out[4:0];
    input_=in[4:0];
    errors=0;
    end
    4'b0011:begin out=out_btoex[3:0];
    output_=out[4:0];
    input_=in[4:0];
    errors=error[0];
    end
    4'b0100:begin out=out_gtob[3:0];
    output_=out[4:0];
    input_=in[4:0];
    errors=0;
    end
    4'b0101:begin out=in[3:0];
    output_=out[4:0];
    input_=in[4:0];
    errors=0;
    end
    4'b0110:begin out=out_gtobcd;
    output_=out[4:0];
    input_=in[4:0];
    errors=0;
```

```
85          input_=in[4:0];
86          errors=0;
87          end
88          4'b0111:begin out=out_gtoex[3:0];
89          output_=out[4:0];
90          input_=in[4:0];
91          errors=error[1];
92          end
93          4'b1000:begin out=out_bcdtob[3:0];
94          output_=out[4:0];
95          input_=in[4:0];
96          errors=error[2];
97          end
98          4'b1001:begin out=out_bcdtog[3:0];
99          output_=out[4:0];
100         input_=in[4:0];
101         errors=error[3];
102         end
103         4'b1010:begin out=in;
104         if(in>4'b1001 && in<5'b10000)
105         begin
106         errors=1;
107         output_=5'bxxxxx;
108         input_=in[4:0];
109         end
110         else
111         errors=0;
112         output_=out[4:0];
113         input_=in[4:0];
114         end
115         4'b1011:begin out=out_bcdtoex[3:0];
116         output_=out[4:0];
117         input_=in[4:0];
118         errors=error[4];
119         end
120         4'b1100:begin out=out_extob[3:0];
121         output_=out[4:0];
122         input_=in[4:0];
123         errors=error[5];
124         end
125         4'b1101:begin out=out_extog[3:0];
126         output_=out[4:0];
127         input_=in[4:0];
128         errors=error[6];
129         end
130         4'b1110:begin out=out_extobcd;
131         output_=out[4:0];
132         input_=in[4:0];
133         errors=error[7];
134         end
135         4'b1111:begin out=in[3:0];
136         if(in>4'b1001)
137         begin
138         errors=1;
139         output_=4'bx;
140         input_=in[4:0];
141         end
142         else
143         begin
144         errors=0;
145         output_=out[4:0];
146         input_=in[4:0];
147         end
148         end
149         default:
150         begin
151         output_=8'bxxxxxxxx;
152         end
153         endcase
154         end
155 endmodule
```

2) All converter module

a) Binary to Gray module

```
module btog(
    input [3:0]b,
    output [3:0]g
    );
    assign g[3]=b[3];
    assign g[2]=b[3]^b[2];
    assign g[1]=b[2]^b[1];
    assign g[0]=b[1]^b[0];
endmodule
```

b) Binary to BCD module

```
module btobcd(
    input [3:0]b,
    output reg [7:0]bcd
    );
    always@(*)
    begin
    if(b>4'd9)
    bcd={4'b0001,b+4'd6};
    else
    bcd = {4'b000,b};
    end
endmodule
```

c) Gray to Binary module

```
module gtob(input[3:0]g, output[3:0]b);
assign b[3]=g[3];
assign b[2]=g[3]^g[2];
assign b[1]=g[3]^g[2]^g[1];
assign b[0]=g[3]^g[2]^g[1]^g[0];
endmodule
```

d) BCD to Excess-3 module

```
module bcdtoex(
    input [7:0]bcd,
    output [3:0]ex, output error
    );
    wire [3:0]b;
    wire [1:0]errort;
    bcdtob m1(.b(b), .bcd(bcd), .error(errort[0]));
    btoex m2 (.b(b), .ex(ex),.error(errort[1]));
    assign error = |errort;
endmodule
```

e) BCD to Gray module

```
module bcdtog(
    input [7:0]bcd,
    output [3:0]g, output error
    );
    wire [3:0]b;
    wire errort;
    bcdtob m1(.b(b), .bcd(bcd), .error(errort));
    btog m2 (.b(b), .g(g));
    assign error = errort;
endmodule
```

f) Gray to BCD module

```
module gtobcd(
    input [3:0]g,
    output [7:0]bcd
    );
    wire [3:0]b;
    gtob m1 (.b(b), .g(g));
    btobcd m2(.b(b), .bcd(bcd));
endmodule
```

g) Gray to Excess-3 module

```
module gtoex(
    input [3:0]g,
    output [3:0]ex, output error
    );
    wire [3:0]b;
    gtob m1 (.g(g), .b(b));
    btoex m2 (.b(b), .ex(ex), .error(error));

endmodule
```

h) Excess-3 to Gray

```
module extog(
    input [3:0] ex,
    output [3:0] g,
    output error
    );
    wire [3:0]b;
    extob m1 (.ex(ex), .b(b), .error(error));
    btog m2 (.b(b), .g(g));

endmodule
```

i) Binary to Excess-3 module

```
module btoex(
    input [3:0]b,
    output reg [3:0]ex
    ,output reg error
    );
    always @(*)
    begin
    if(b>5'd9)
    begin
    error=1;
    ex=4'bx;
    end
    else
    begin
    ex=b+4'd3;
    error=0;
    end
    end
endmodule
```

j) BCD to Binary module

```
module bcdtob(
    input [7:0]bcd,
    output reg [3:0]b, output reg error
    );
    reg [4:0]temp;
    always @(*)
    begin
    if(bcd[7:4]==4'b0000 && bcd[3:0]<=4'd9)
    begin
    b=bcd[3:0];
    error=0;
    end
     else
     begin
     if(bcd[7:4]==4'b0001 && bcd[3:0]<=4'd5)
     begin
          temp=bcd[3:0] + 5'd10;
          b=temp[3:0];
          error=0;
     end
    else
    begin
    error=1;
    b=4'bx;
    end
    end
    end
endmodule
```

k) Excess-3 to Binary module

```
module extob(
    input [3:0]ex,
    output reg [3:0]b,
    output reg error
    );

    always @(*)
    if(4'd3<=ex && ex<=4'd12)
    begin
    error=0;
   b=ex-4'd3;
    end
    else
    begin
     error=1;
    b=4'bx;
    end
endmodule
```

l) Excess-3 to BCD module

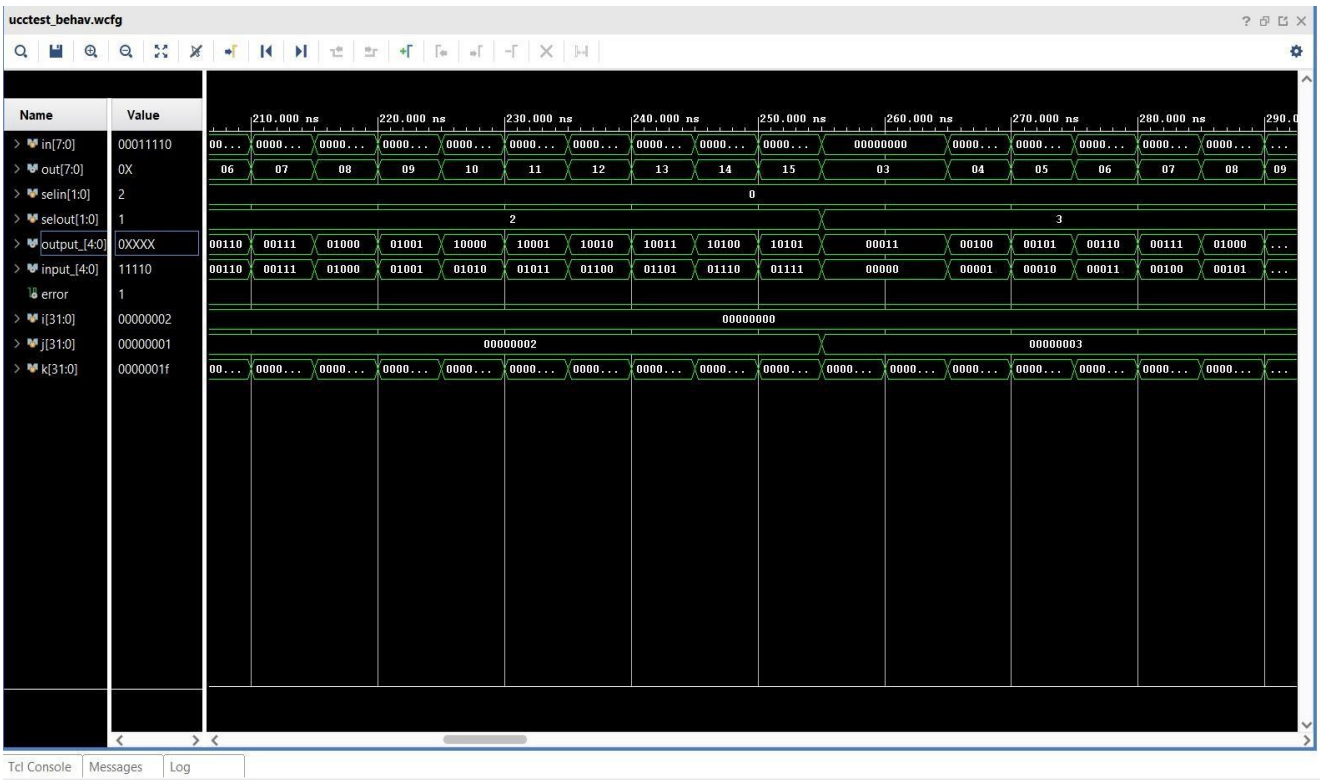
```
module extobcd(
    input [3:0] ex,
    output [7:0] bcd,
    output error
    );
    wire errort;
    wire [3:0]b;
    extob m1 (.ex(ex), .b(b), .error(errort));
    btobcd m2 (.b(b), .bcd(bcd));
    assign error=errort;

endmodule
```

VERILOG TEST BENCH

```
module ucctest();
    reg [7:0] in;
    reg [1:0] selin;
    reg [1:0] selout;
    wire [7:0] out;
    wire [4:0] output_;
    wire [4:0] input_;
    wire error;
    UCC uut (.in(in),.out(out),.errors(error),.output_(output_),.input_(input_),.selin(selin),.selout(selout));
    integer i, j, k;
    initial begin
       for(i=0; i<4; i=i+1)
       begin
       selin=i[1:0];
       for(j=0; j<4; j=j+1)
       begin
       selout=j[1:0];
       in=8'b00000000;
       if(selin==2'b10)
       begin
       for(k=0;k<=32;k=k+1)
       begin
       #5 in=k;
       end
       end
       else
       begin
       for(k=0;k<=16;k=k+1)
       begin
       #5 in=k;
       end
       end
       end
       end
```

4) SIMULATION RESULT



Thank You!