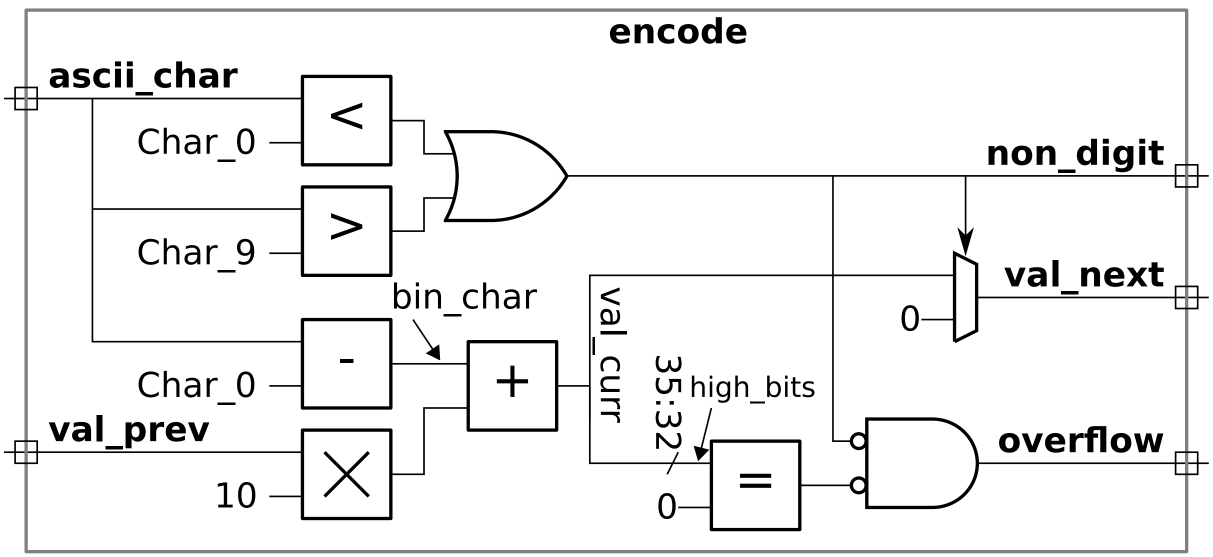
Synthesis of Combinational Logic from Behavioral Code

Example: ASCII Decimal to Binary Converter

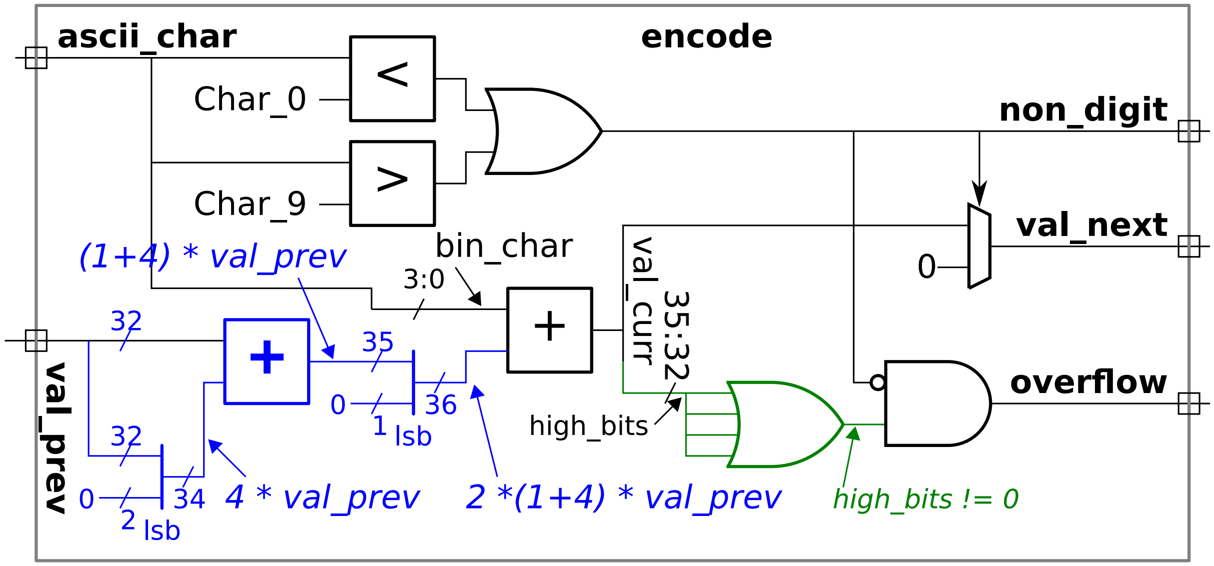
```
module encode #( int width = 32 )
   ( output logic [width-1:0] val_next,
     output logic overflow,                     output non_digit,
     input [7:0] ascii_char,                    input [width-1:0] val_prev );

   logic [width+3:0] val_curr;                  logic [3:0] high_bits, bin_char;

   assign non_digit = ascii_char < Char_0 || ascii_char > Char_9;

   always @* begin

      bin_char = ascii_char - Char_0;
      val_curr = 10 * val_prev + bin_char;
      high_bits = val_curr >> width;

      if ( non_digit ) begin
         overflow = 0;   val_next = 0;
      end else begin
         overflow = high_bits != 0;
         val_next = val_curr;
   end end
endmodule
```

encode, inferred hardware.

```
module encode #( int width = 32 )
   ( output logic [width-1:0] val_next,
     output logic overflow,                  output non_digit,
     input [7:0] ascii_char,                 input [width-1:0] val_prev );

   logic [width+3:0] val_curr;              logic [3:0]       high_bits, bin_char;

   assign non_digit = ascii_char < Char_0 || ascii_char > Char_9;

   always @* begin

      bin_char = ascii_char - Char_0;
      val_curr = 10 * val_prev + bin_char;
      high_bits = val_curr >> width;

      if ( non_digit ) begin
         overflow = 0;
         val_next = 0;
      end else begin
         overflow
           = high_bits != 0;
         val_next = val_curr;
      end end
endmodule
```



encode with some optimizations

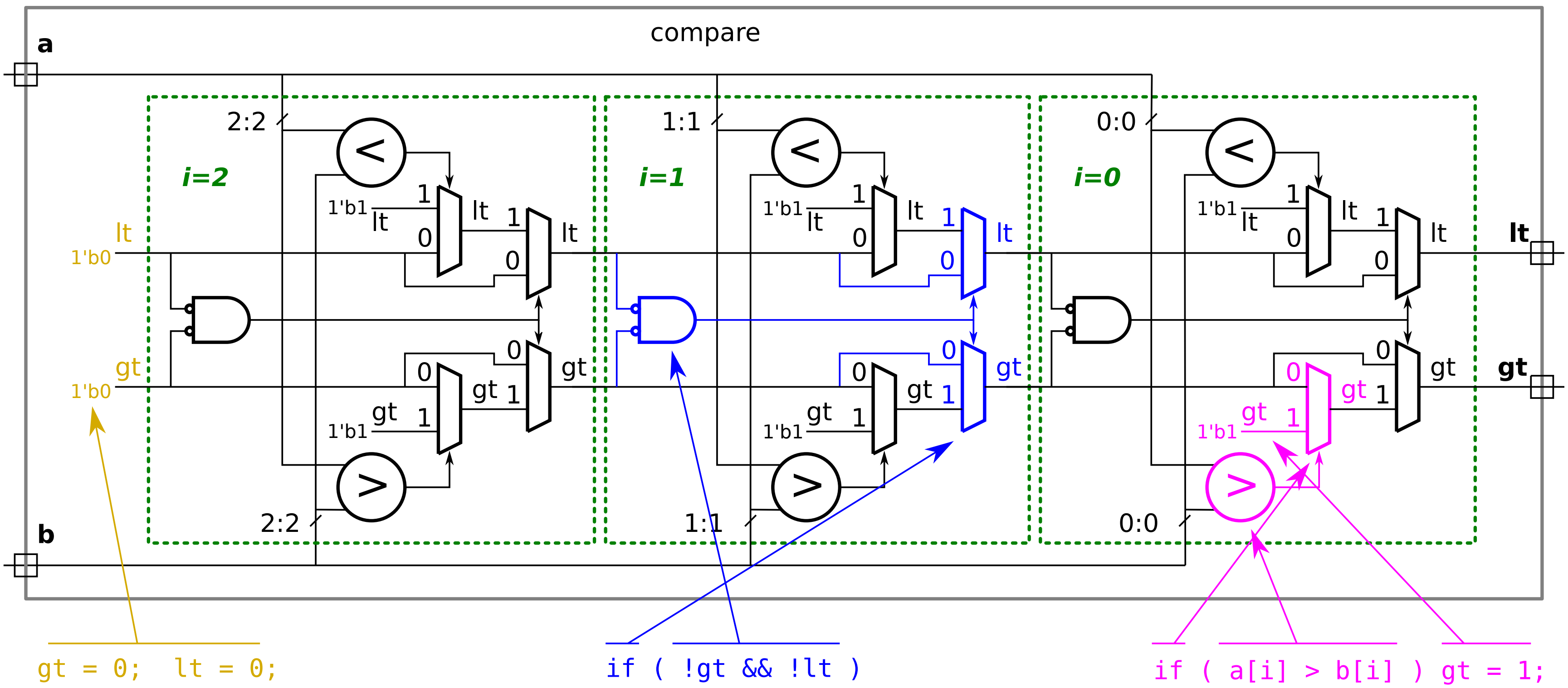


Example: compare

```
module compare( output logic gt, lt,  input wire [2:0] a, b );
   always @* begin
     gt = 0; lt = 0;
     for ( int i=2; i>=0; i-- )
       if ( !gt && !lt ) begin
           if ( a[i] < b[i] ) lt = 1;
           if ( a[i] > b[i] ) gt = 1;
        end
   end
endmodule
```

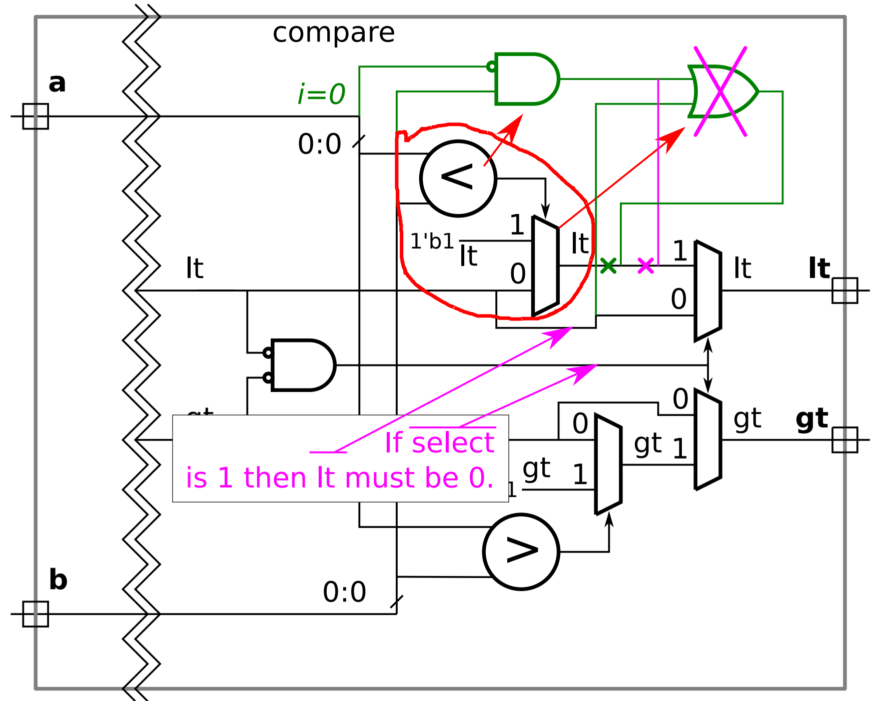
compare Inferred Hardware, No Optimization

```
module compare( output logic gt, lt,  input wire [2:0] a, b );
   always @* begin
      gt = 0; lt = 0;
      for ( int i=2; i>=0; i-- )
        if ( !gt && !lt ) begin
           if ( a[i] < b[i] ) lt = 1;
           if ( a[i] > b[i] ) gt = 1;
end end endmodule
```



`compare` Optimizations

```
module compare
 ( output logic gt, lt,
   input wire [2:0] a, b );
   always @* begin
      gt = 0; lt = 0;
      for ( int i=2; i>=0; i-- )
        if ( !gt && !lt ) begin
           if ( a[i] < b[i] ) lt = 1;
           if ( a[i] > b[i] ) gt = 1;
     end end
endmodule
```



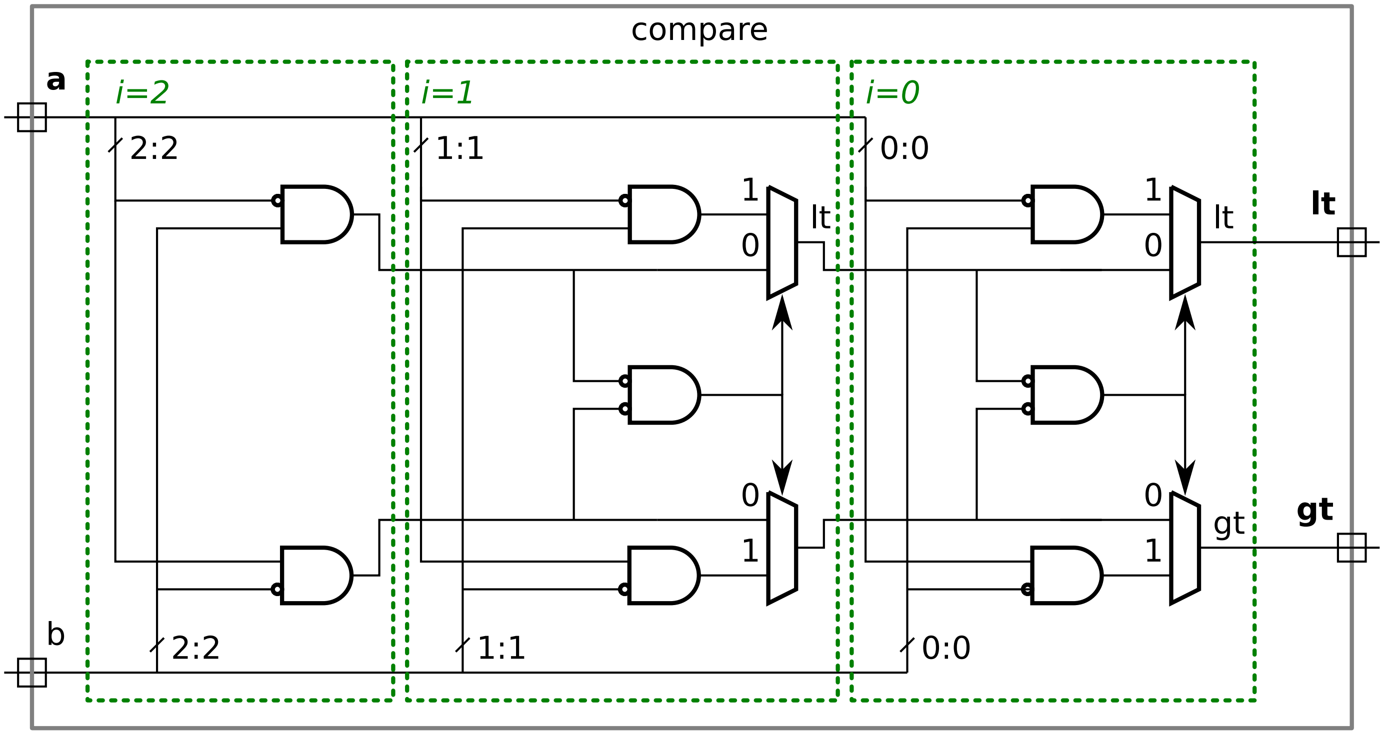
Notice that operations are on one-bit quantities . . .

. . . and so `a[i] < b[i]` equivalent to `!a[i] && b[i]` or $\overline{a_i} b_i$.

Notice that `if ( a[i] < b[i] ) lt = 1;`. . .

. . . is equivalent to `lt = lt || !a[i] && b[i];` . . .

. . . or $lt_{\text{new}} = lt_{\text{old}} + \overline{a_i} b_i$.


compare
a
i=2
i=1
i=0
2:2
1:1
0:0
1
0
lt
1
0
lt
lt
0
1
0
1
gt
gt
b
2:2
1:1
0:0