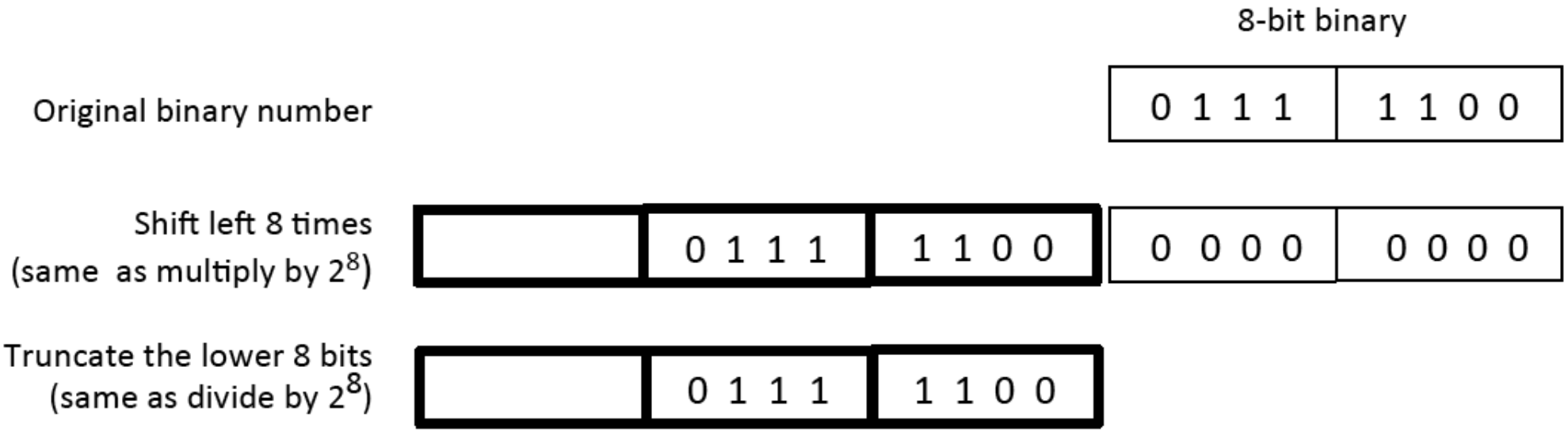
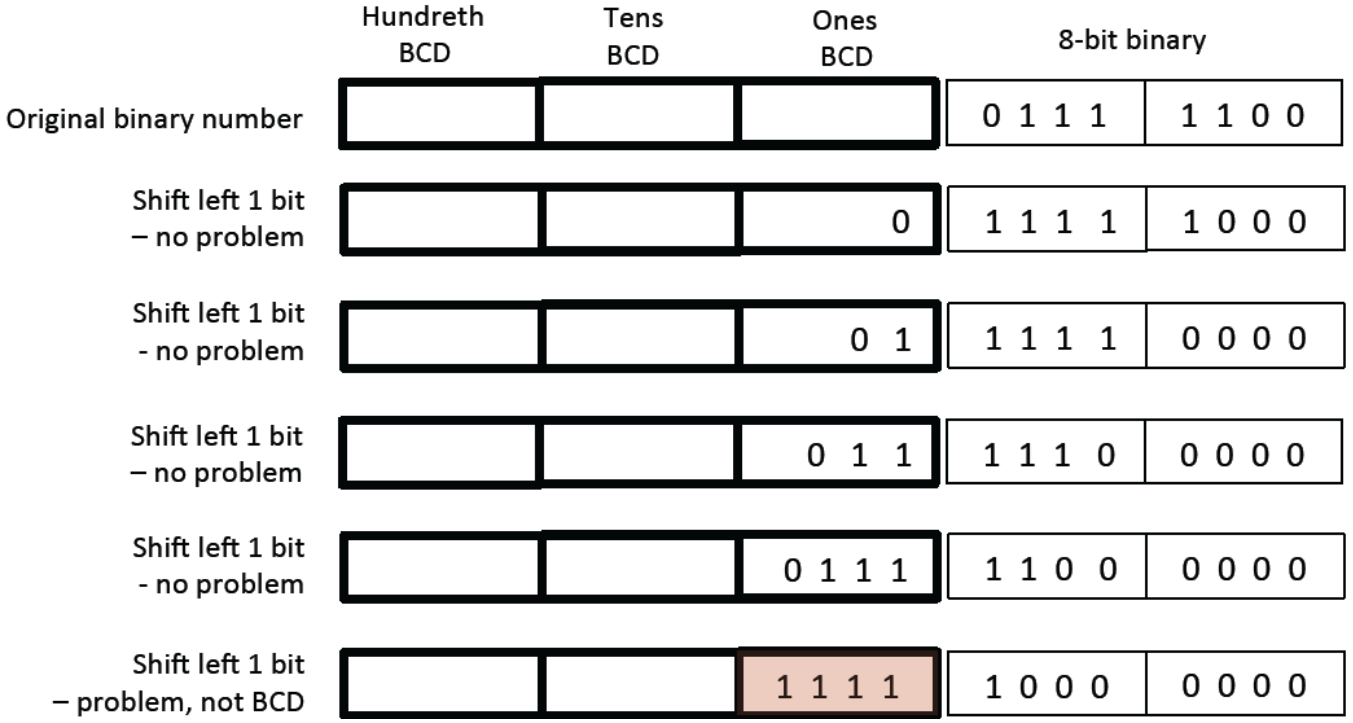
Shift and Add 3 algorithm [1] – shifting operation

- Let us consider converting hexadecimal number 8'h7c (which is decimal 8'd124)
- Shift the 8-bit binary number left by 1 bit = multiply number by 2
- Shifting the number left 8 times = multiply number by 2^8
- Now truncate the number by dropping the bottom 8 bits = divide number by 2^8
- So far we have done nothing to the number – it has the same value
- The idea is that, as we shift the number left into the BCD digit "bins", we make the necessary conversion to the hex number so that it conforms to the BCD rule (i.e. falls within 0 to 9, instead of 0 to 15)



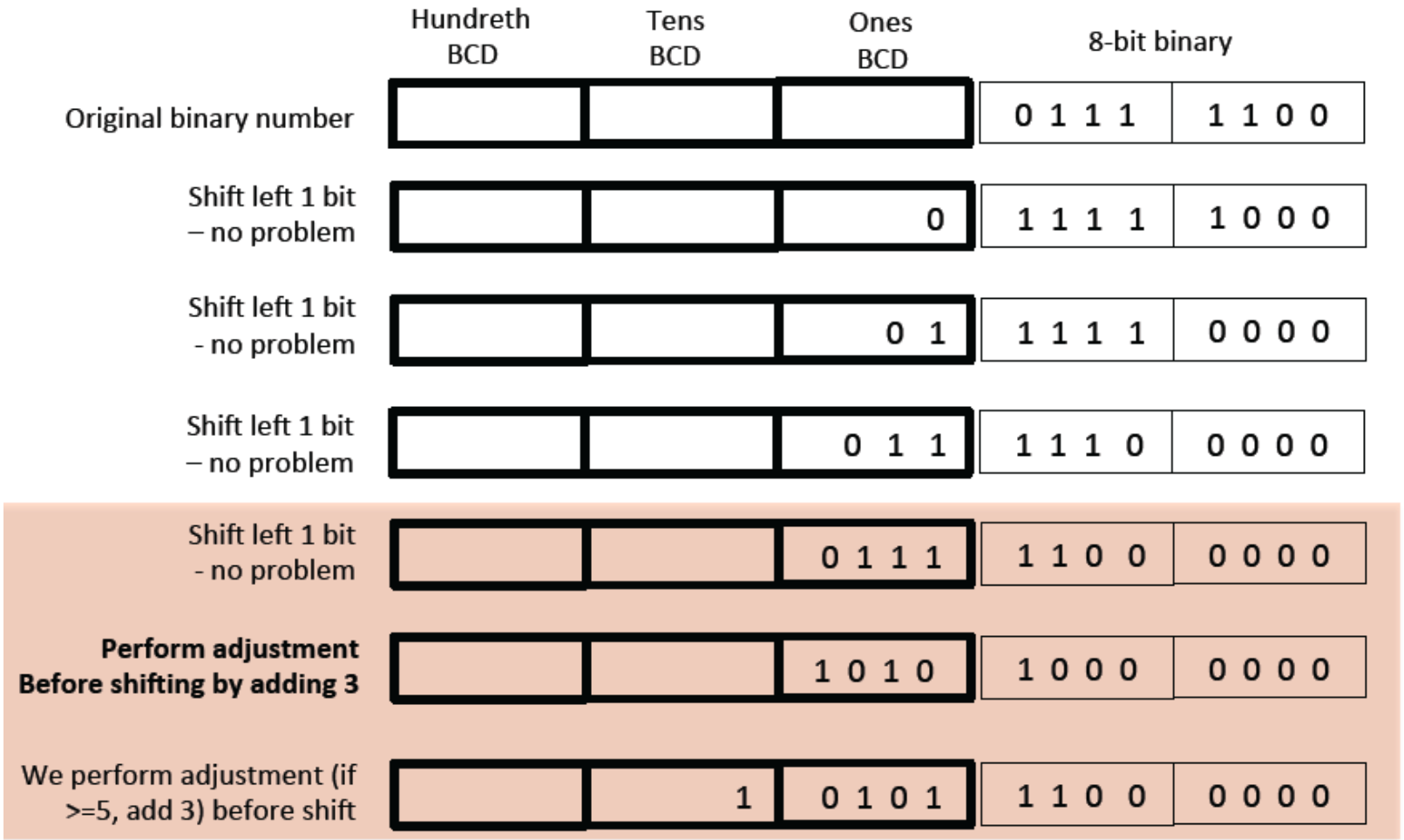
Shift and Add 3 algorithm [2] – shift left with problem

- If we take the original 8-bit binary number and shift this three times into the BCD digit positions. After 3 shifts we are still OK, because the **ones digit** has a value of 3 (which is OK as a BCD digit).
- If we shift again (4th time), the digit now has a value of 7. This is still OK. However, no matter what the next bit it, another shift will make this digit illegal (either as hexadecimal “e” or “f”, both not BCD).
- In our case, this will be a “f”!



Shift and Add 3 algorithm [3] – shift and adjust

- So on the fourth shift, we detect that the value is > or = 5, then we adjust this number by adding 3 before the next shift.
- In that way, after the shift, we move a 1 into the tens BCD digit as shown here.



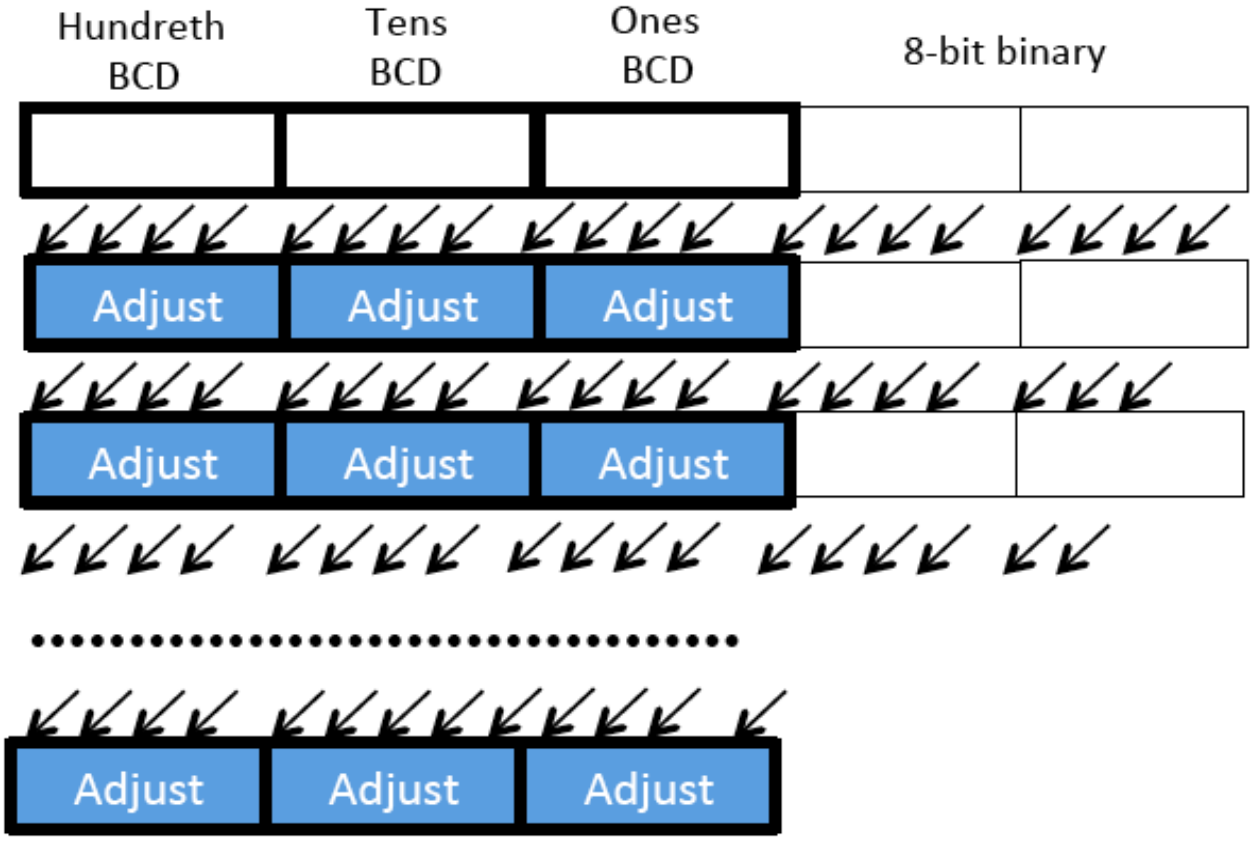
Shift and Add 3 algorithm [4] – full conversion

- In summary, the basic idea is to shift the binary number left, one bit at a time, into locations reserved for the BCD results.
- Let us take the example of the binary number 8'h7C. This is being shifted into a 12-bit/3 digital BCD result of 12'd124 as shown below.

	Hundreth BCD	Tens BCD	Ones BCD	8-bit binary	
Original binary number				0 1 1 1	1 1 0 0
Shift left three times no adjust			0 1 1	1 1 1 0	0
Shift left Ones = 7, ≥5			0 1 1 1	1 1 0 0	
Add 3			1 0 1 0	1 1 0 0	
Shift left Ones = 5		1	0 1 0 1	1 0 0	
Add 3		1	1 0 0 0	1 0 0	
Shift left 2 times Tens = 6, ≥5		1 1 0	0 0 1 0	0	
Add 3		1 0 0 1	0 0 1 0	0	
Shift left BCD value is correct	1	0 0 1 0	0 1 0 0		

Hardware implementation (1) – binary to BCD

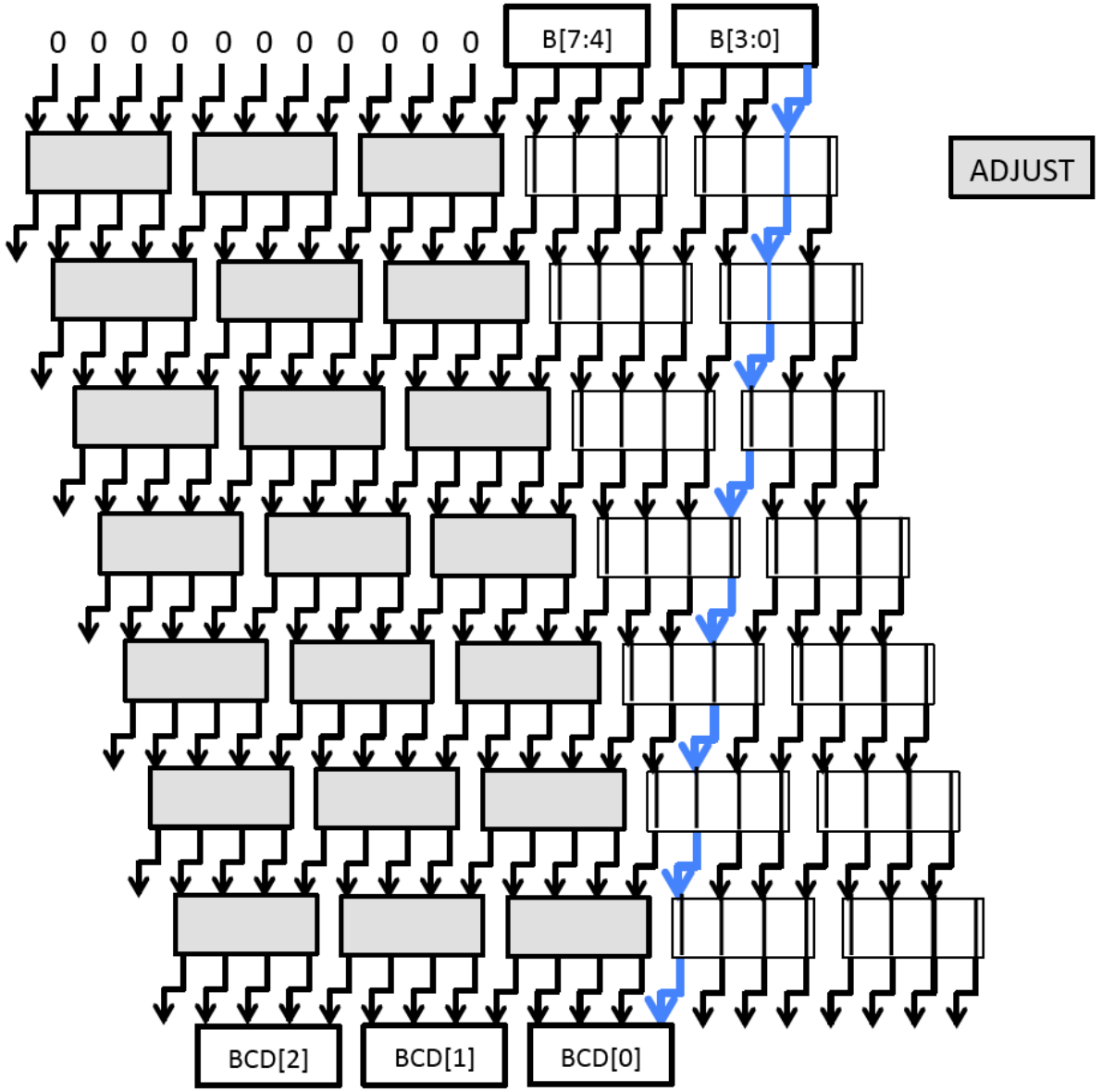
- The hardware to perform binary to BCD conversion is shown below.
- Shifting is easy – just wiring all signals one position to the left.
- For each of the BCD locations, we need an “adjust” module which perform the follow operation: if the value is ≥5, then add 3.



Hardware implementation (2) – array of gates

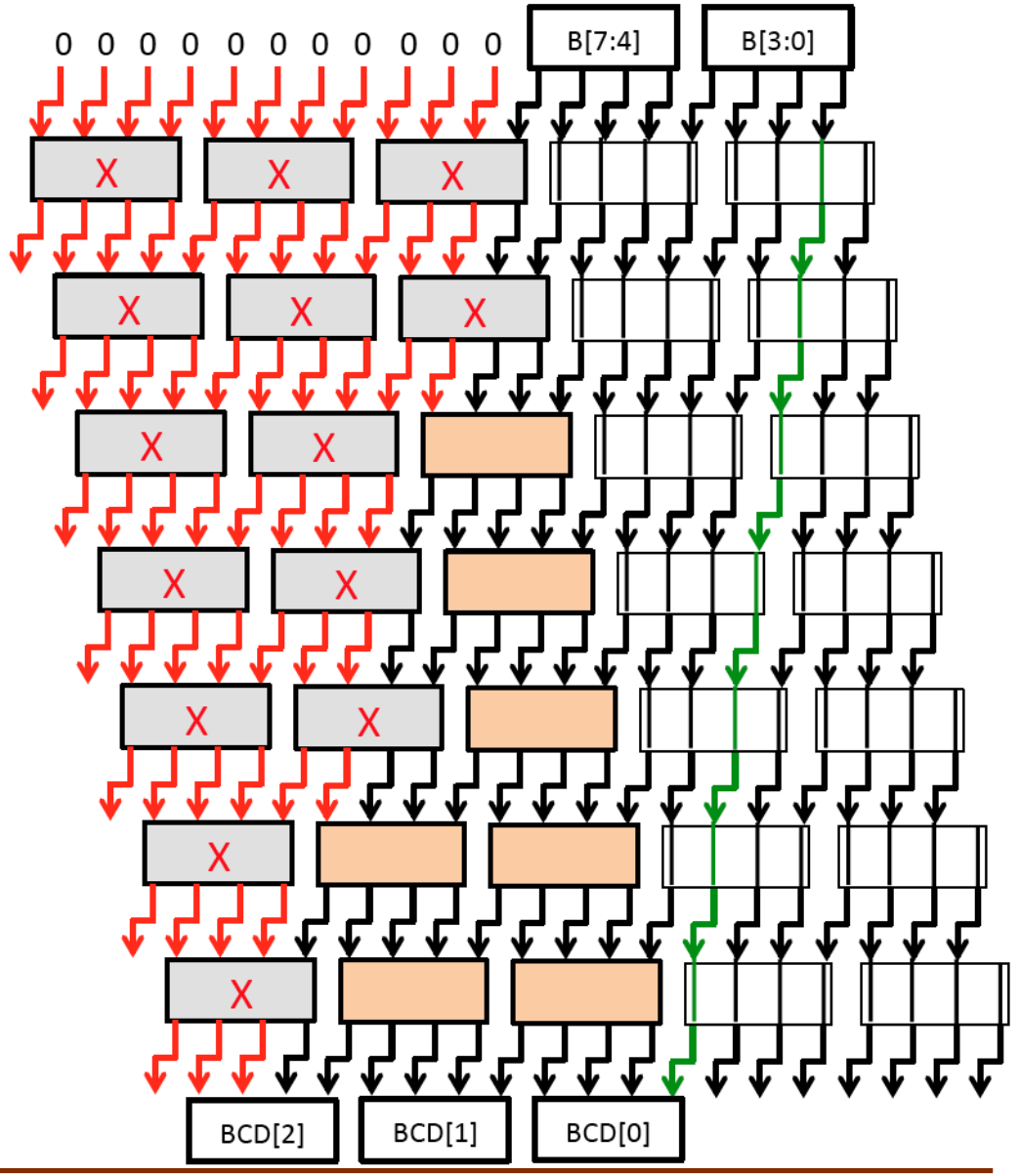
- Here is the full array of logic gates to do the conversion.
- After 8 shift and adjustment on the way, the result should be three BCD digits.
- Each ADJUST block perform the following operation:

```
if (input >= 5)
        output = input + 3
else
        output = input
```



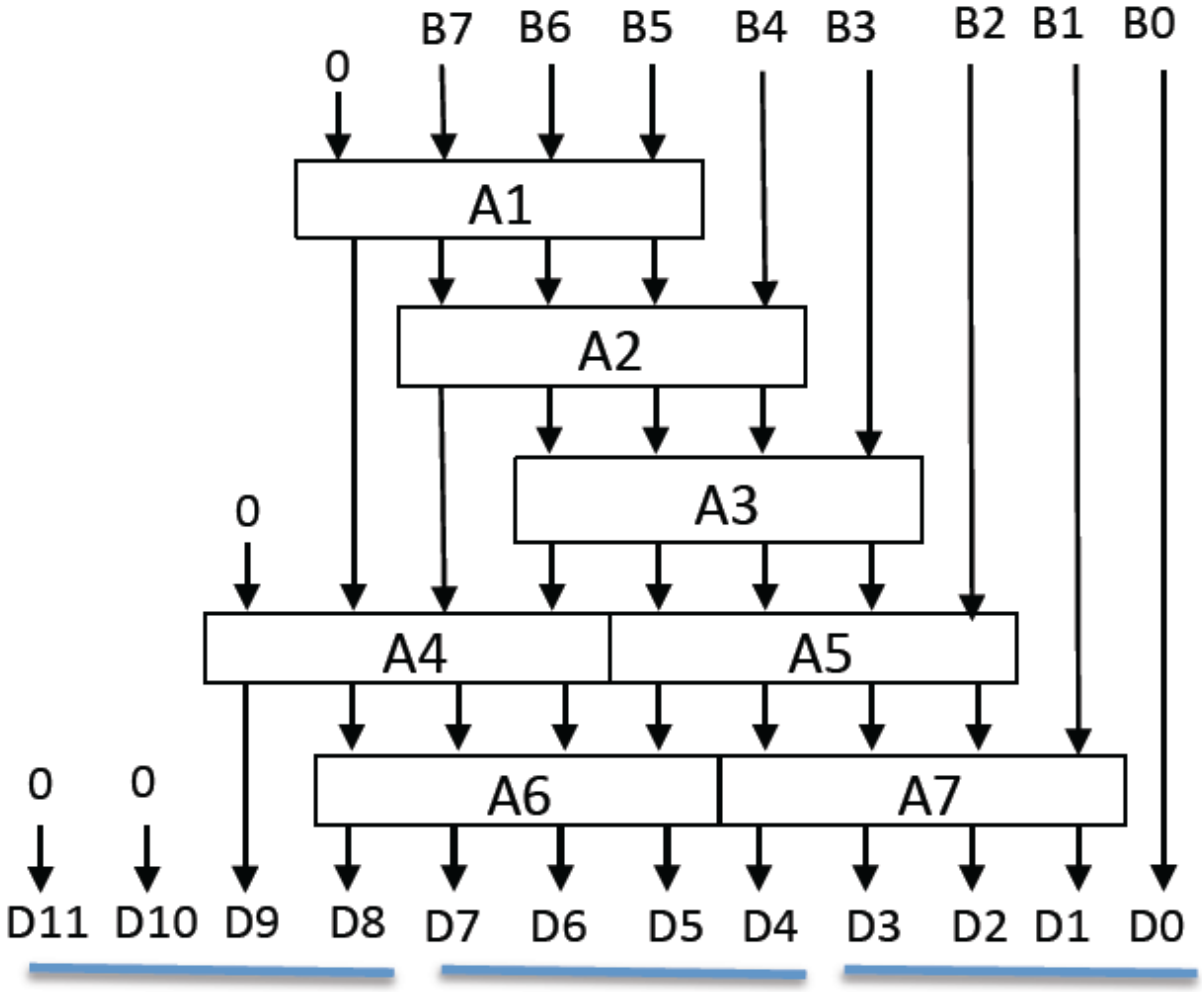
Hardware implementation (3) – propagate 0 to simplify

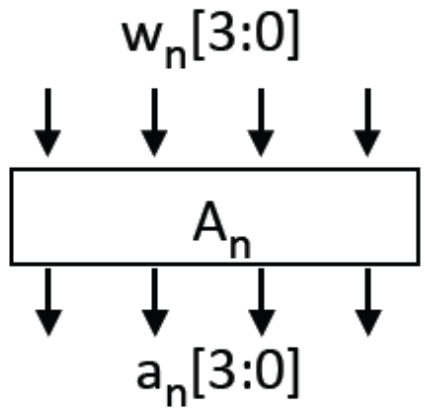
- If we now propagate forward all the 0s, we can eliminate all ADJUST modules except those in RED.
- All the others are just wires from input to output because the input values are GUARANTEED to be smaller than 5.



Putting things together – structural design

- Once you have specified the **adjust module (A_n)** in Verilog, you can wire up the entire converter as shown here:

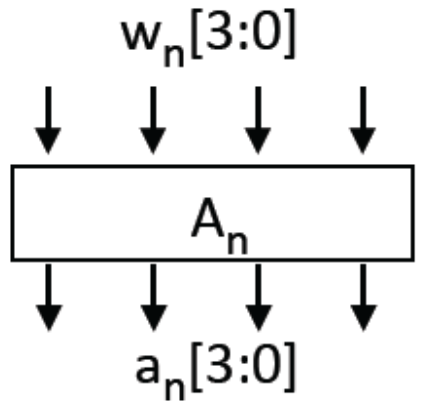




Binary to BCD conversion in Verilog

- Here is the Verilog code to perform the 8-bit binary to BCD conversion:

```
module bin2bcd8 (B, BCD_0, BCD_1, BCD_2);

    input [7:0] B;     // binary input number
    output [3:0]   BCD_0, BCD_1, BCD_2;    // BCD digit LSD to MSD

    wire [3:0]  w1,w2,w3,w4,w5,w6,w7;
    wire [3:0]  a1,a2,a3,a4,a5,a6,a7;

    // Instantiate a tree of add3-if-greater than or equal to 5 cells
    //  ... input is w_n, and output is a_n
    add3_ge5 A1 (w1,a1);
    add3_ge5 A2 (w2,a2);
    add3_ge5 A3 (w3,a3);
    add3_ge5 A4 (w4,a4);
    add3_ge5 A5 (w5,a5);
    add3_ge5 A6 (w6,a6);
    add3_ge5 A7 (w7,a7);

    // wire the tree of add3 modules together
    assign  w1 = {1'b0, B[7:5]};      // wn is the input port to module An
    assign  w2 = {a1[2:0], B[4]};
    assign  w3 = {a2[2:0], B[3]};
    assign  w4 = {1'b0, a1[3], a2[3], a3[3]};
    assign  w5 = {a3[2:0], B[2]};
    assign  w6 = {a4[2:0], a5[3]};
    assign  w7 = {a5[2:0], B[1]};

    // connect up to four BCD digit outputs
    assign BCD_0 = {a7[2:0],B[0]};
    assign BCD_1 = {a6[2:0],a7[3]};
    assign BCD_2 = {2'b0,a4[3],a6[3]};

endmodule
```



Behaviour design in Verilog – Much better way!

```
module bin2bcd (
   input[7:0] x,         // value ot be converted
   output reg[3:0] BCD0,
   output reg[3:0] BCD1,
   output reg[3:0] BCD2  // BCD digits
);

   // Concatenation of input and output
   reg [19:0] result;  // no of bits = no_of_bit of x + 4* no of digits
   integer i;

   always @(*)
   begin
      result[19:0] = 0;
      result[7:0] = x;

      for (i=0; i<8; i=i+1) begin
         // Check if unit digit >= 5
         if (result[11:8] >= 5)
            result[11:8] = result[11:8] + 4'd3;

         // Check if ten digita >= 5
         if (result[15:12] >= 5)
            result[15:12] = result[15:12] + 4'd3;

         // Shift everything left
         result = result << 1;
      end

      // Decode output from result
      BCD0 = result[11:8];
      BCD1 = result[15:12];
      BCD2 = result[19:16];
   end
endmodule
```