Unsigned Integers

- Non-negative numbers (including 0)
  - Represent real-world data
    - e.g., temperature, position, time, …
  - Also used in controlling operation of a digital system
    - e.g., counting iterations, table indices
- Coded using unsigned binary (base 2) representation
  - analogous to decimal representation

Binary Representation

- Decimal: base 10
  - $124_{10} = 1\times10^2 + 2\times10^1 + 4\times10^0$
- Binary: base 2
  - 124_{10}
    $= 1\times2^6+1\times2^5+1\times2^4+1\times2^3+1\times2^2+0\times2^1+0\times2^0$
    $= 1111100_2$
- In general, a number x is represented using n bits as $x_{n-1}, x_{n-2}, \ldots, x_0$, where

$$x = x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_0 2^0$$

Binary Representation

- Unsigned binary is a code for numbers
  - n bits: represent numbers from 0 to $2^n - 1$
    - 0: 0000…00; $2^n - 1$: 1111…11
  - To represent x: $0 \le x \le N - 1$, need $\lceil \log_2 N \rceil$ bits
- Computers use
  - 8-bit bytes: 0, …, 255
  - 32-bit words: 0, …, ~4 billion
- Digital circuits can use what ever size is appropriate

Unsigned Integers in Verilog

- Use vectors as the representation
  - Can apply arithmetic operations

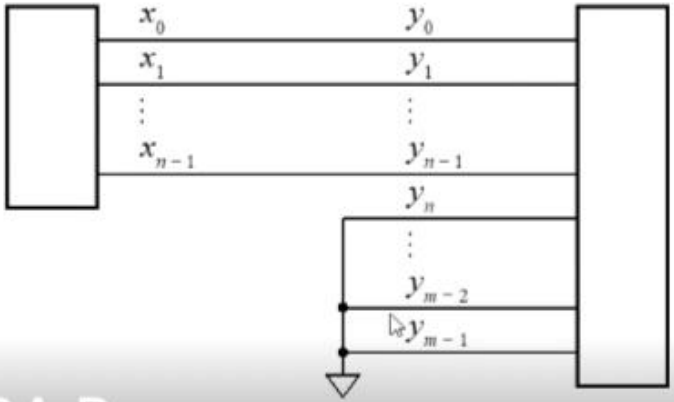
```
module multiplexer_6bit_4_to_1
  ( output reg [5:0] z,
    input      [5:0] a0, a1, a2, a3,
    input      [1:0] sel );
  always @*
    case (sel)
      2'b00: z = a0;
      2'b01: z = a1;
      2'b10: z = a2;
      2'b11: z = a3;
    endcase
endmodule
```

Octal and Hexadecimal

- Short-hand notations for vectors of bits
- Octal (base 8)
  - Each group of 3 bits represented by a digit
  - 0: 000, 1:001, 2: 010, …, 7: 111
  - 253_8 = 010 101 011_2
  - $11001011_2 \Rightarrow$ 11 001 011_2 = 313_8
- Hex (base 16)
  - Each group of 4 bits represented by a digit
  - 0: 0000, …, 9: 1001, A: 1010, …, F: 1111
  - $3CE_{16}$ = 0011 1100 1110_2
  - $11001011_2 \Rightarrow$ 1100 1011_2 = CB_{16}

Extending Unsigned Numbers

- To extend an *n*-bit number to *m* bits
  - Add leading 0 bits
  - e.g., 72_{10} = 1001000 = 000001001000



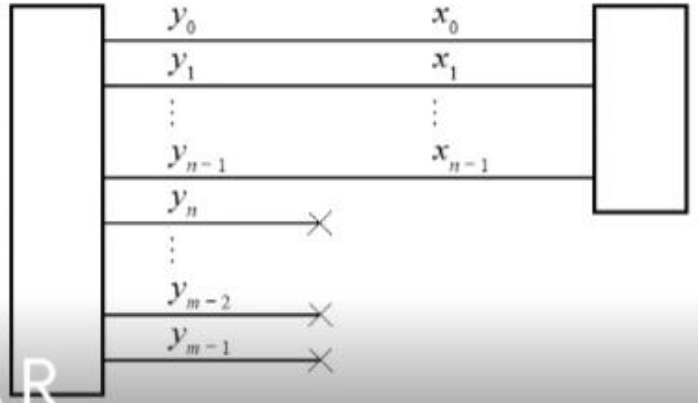

```
wire [3:0] x;
wire [7:0] y;

assign y = {4'b0000, x};
```

```
assign y = {4'b0, x};
```

```
assign y = x;
```

Truncating Unsigned Numbers

- To truncate from *m* bits to *n* bits
  - Discard leftmost bits
  - Value is preserved if discarded bits are 0
  - Result is $x \bmod 2^n$




```
assign x = y[3:0];
```

Unsigned Addition

- Performed in the same way as decimal

```
0 0 1 1 1 1 0 0 0 0
  1 0 1 0 1 1 1 1 0 0
  0 0 1 1 0 1 0 0 1 0
-------------------
  1 1 1 0 0 0 1 1 1 0
```

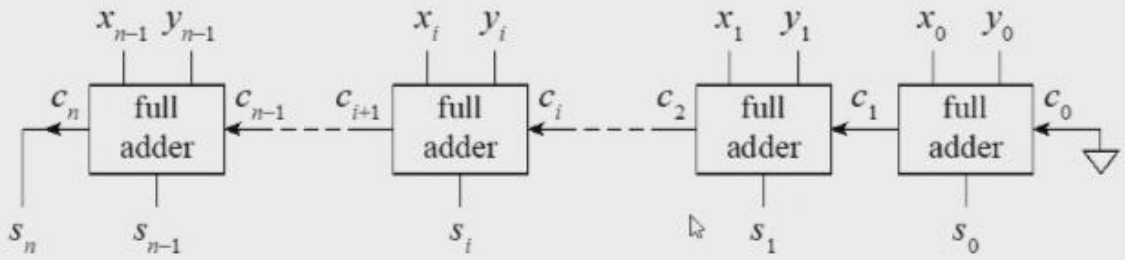
carry bits

```
1 1 0 0 1
  0 1 0 0 1
  1 1 1 0 1
-----------
1 0 0 1 1 0
```

overflow

Ripple-Carry Adder

- Full adder for each bit, $c_0 = 0$



overflow

- Worst-case delay
  - from x_0, y_0 to s_n
  - carry must ripple through intervening stages, affecting sum bits

Adders in Verilog

- Use arithmetic "+" operator

```
wire [7:0] a, b, s;
…

assign s = a + b;
```

```
wire [8:0] tmp_result;
wire       c;
…

assign tmp_result = {1'b0, a} + {1'b0, b};
assign c          = tmp_result[8];
assign s          = tmp_result[7:0];
```

```
assign {c, s} = {1'b0, a} + {1'b0, b};
```

```
assign {c, s} = a + b;
```

Improving Adder Performance

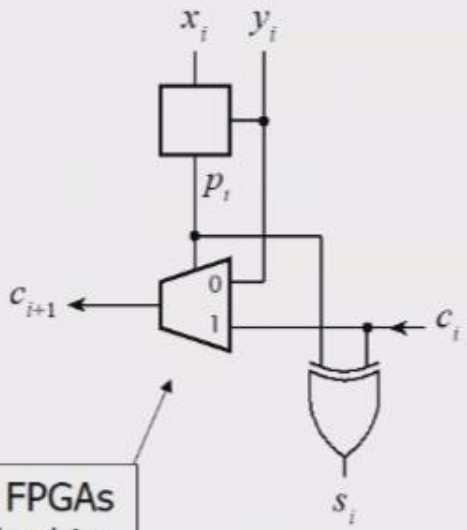
- Carry kill: $k_i = \overline{x_i} \cdot \overline{y_i}$
- Carry propagate: $p_i = x_i \oplus y_i$
- Carry generate: $g_i = x_i \cdot y_i$
- Adder equations

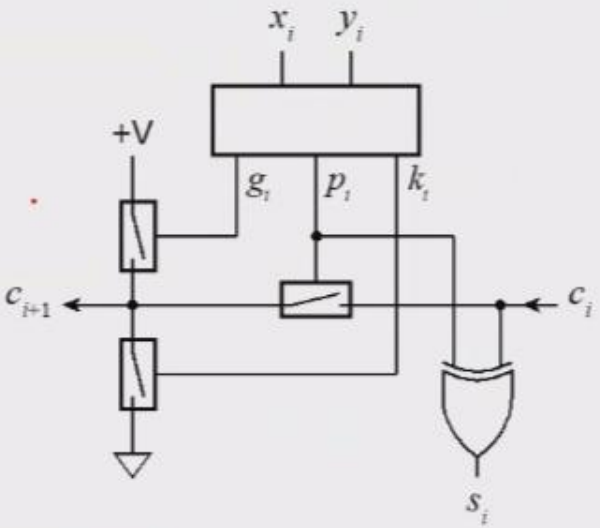
x_i	y_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	1	1
1	1	1	1	1

$$s_i = p_i \oplus c_i \qquad c_{i+1} = g_i + p_i \cdot c_i$$

Fast-Carry-Chain Adder

- Also called Manchester adder





Xilinx FPGAs include this structure