Reconfigurable architecture #2

osana@eee.u-ryukyu.ac.jp

This and next week

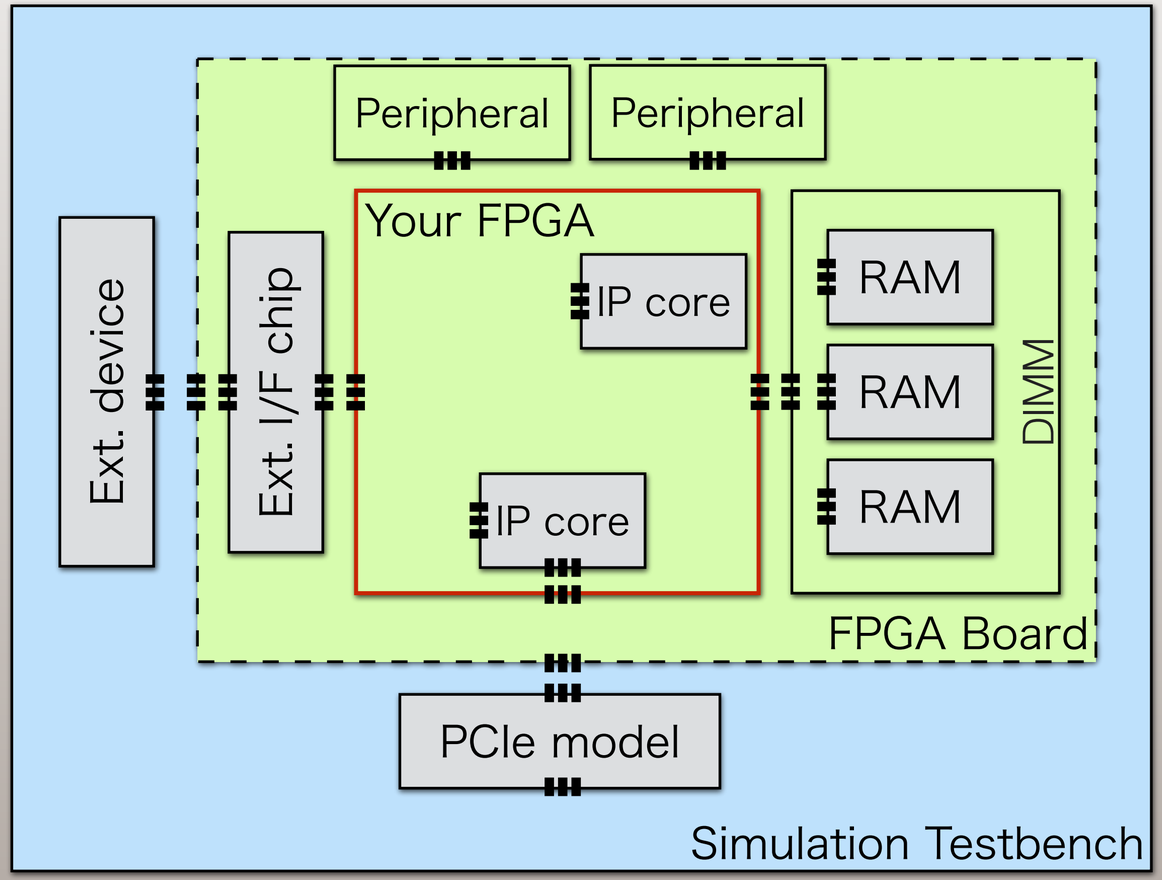
- Basics of Verilog HDL
  - Language to program logic circuit
  - Simulator: to verify behavior in waveform
  - Synthesis tool: generates equivalent circuit in gate-level

Module and port

- In HDL, behavior is described per module basis
  - Module has “ports” to connect outside
    - A port is like a terminal
    - From outside, we can’t see inside a module
  - A module can contain submodules (as blackbox)

Everything as modules

- FPGA design include
  - Your HDL module + IP cores (imported design)
- External device models in testbench



Tools and HDL syntax

- Not all of HDL syntax is “synthesizable”
  - Some syntax is valid only for simulation
    - Used in testbench (we’ll see in later class)
  - At least, FPGA part must be written in “synthesizable” form

Writing HDL

- Verilog HDL or VHDL
  - “Logic circuit in gate level” is possible, but too hard
  - “Writing RTL behavior” is easier and common
    - RTL = Register Transfer Level
  - More recently, high-level synthesis (HLS) to generate RTLs from C/C++/Java is becoming popular

Gate level

- Composed by “NAND”, “Flip-flop” or that kind of…
  - Must be optimized manually: Not realistic for large circuits
  - Or, obtained from RTL by “Logic synthesis”

RTL description

- Do **what** if received this **signal** in this **state**
  - Programmer can give any (meaningful) name to signals
  - Some “software-like” syntax as “if” statement or conditional assignments
  - No details about the final gate-level circuit is required

HDL as a programming language

- HDL is uncommon as a programming language
  - Programming language is usually about software
    - Executed in “up-to-down” manner
  - HDL describes components, continue working all the time
    - No beginning or end in HDL world!

Logic synthesis

- Optimal design for each device technology
  - NAND gates are not almighty
  - Every LSI process and every FPGA have different logic cells
    - Synthesis tools **maps** RTL to the logic cell libraries
    - HDL is (basically) common to all device technologies

How to learn HDL

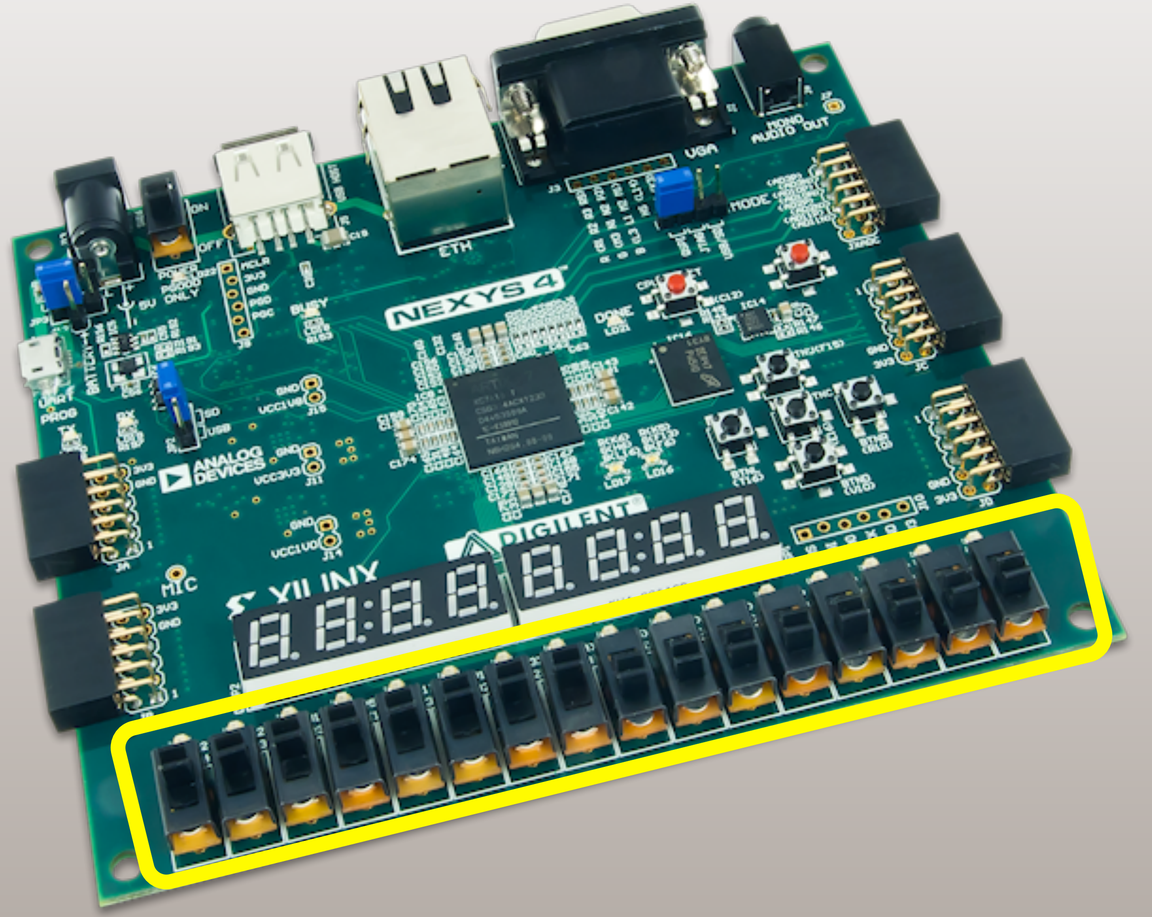
- Syntax: basically same to logic circuit
  - Combinational and sequential logic
- HDL specific stuff:
  - Module hierarchy and simulation
  - … and system organization is much more difficult

Schedule

- Episode 1: Module, Combinational circuit and Operators
  - + Design constraints
- Episode 2: Sequential logic and Hierarchical design
- Episode 3: Simple testbench and Running simulation

Goal of the day

- LEDs and slide SWs
  - With logic gates intermediate
  - 16 LEDs and SWs on board
  - Also try using push SW
- First of all, let's see the syntax



Literals

- 123: “Normal” numbers are interpreted as a (32bits) decimal
- 32’h1248abcd : Hexadecimal “1234abcd” of 32bits
  - b: binary, d: decimal, h: hexadecimal
  - Inserting “_” between any columns is allowed for better readability
    - example: 32’h1234_abcd
- Single bit of “1” means “True”, “0” is “False”
  - Vector signal is “False” if all bits are 0

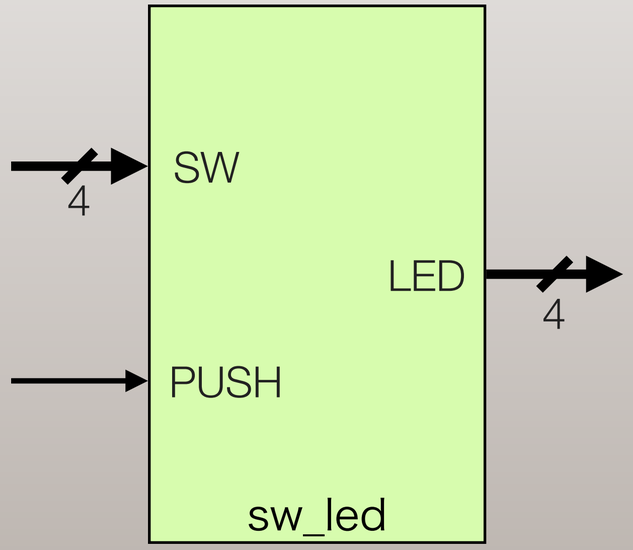
Module and port

- Module definition
  - starts by “module”
  - ends by “endmodule”
- endmodule has no “;”

```
module module_name ( ports );
  . . .

endmodule
```

Module and port

Signal has 4 bits,
LSB is numbered as “0”

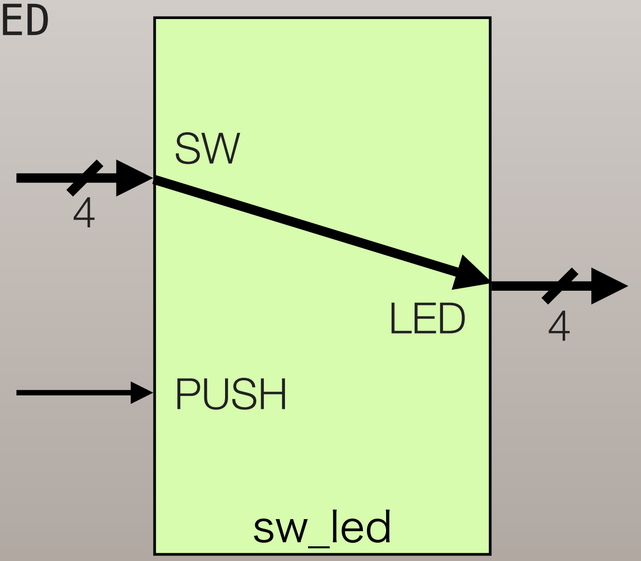



```
module sw_led
  (
   input  wire [3:0] SW,
   input  wire       PUSH,
   output wire [3:0] LED
  );

  . . .

endmodule
```

Continuous assignment

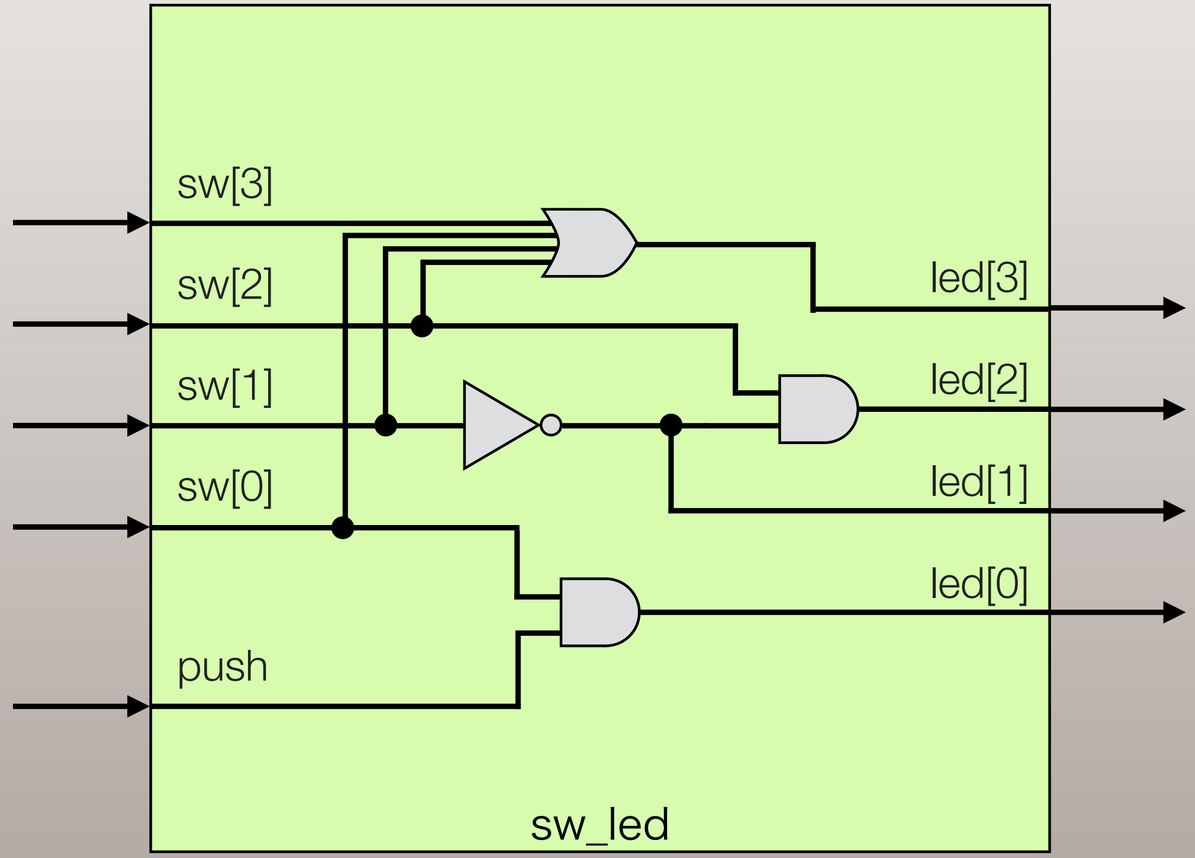
- Assignments in software:
  - Happens when executed
- In hardware:
  - Happens continuously (because it's wired)

```
module sw_led
  (
   input  wire [3:0] SW,
   input  wire       PUSH,
   output wire [3:0] LED
  );

  assign LED = SW;

endmodule
```



Ex1: Basic logic gates

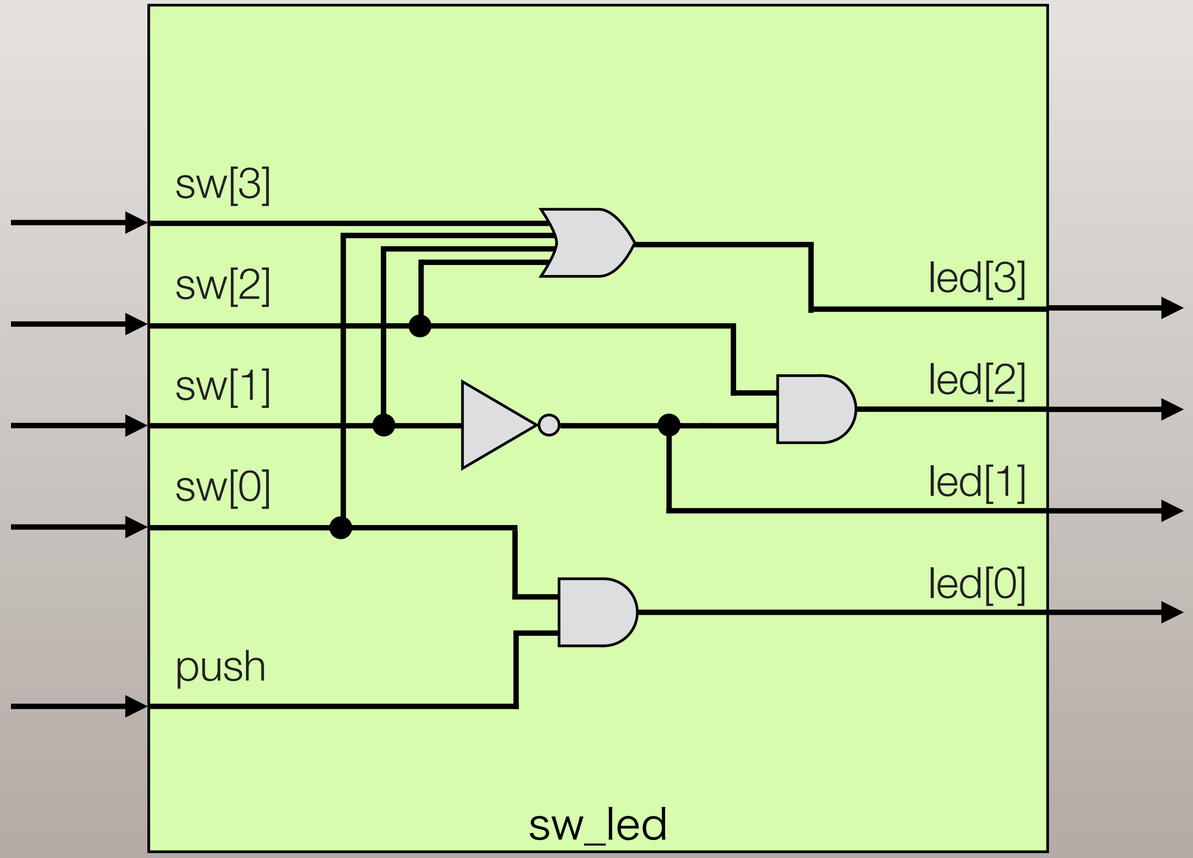
* AND, NOT, OR gates



Logic operators

* AND: “&”, OR: “|”, NOT “~”

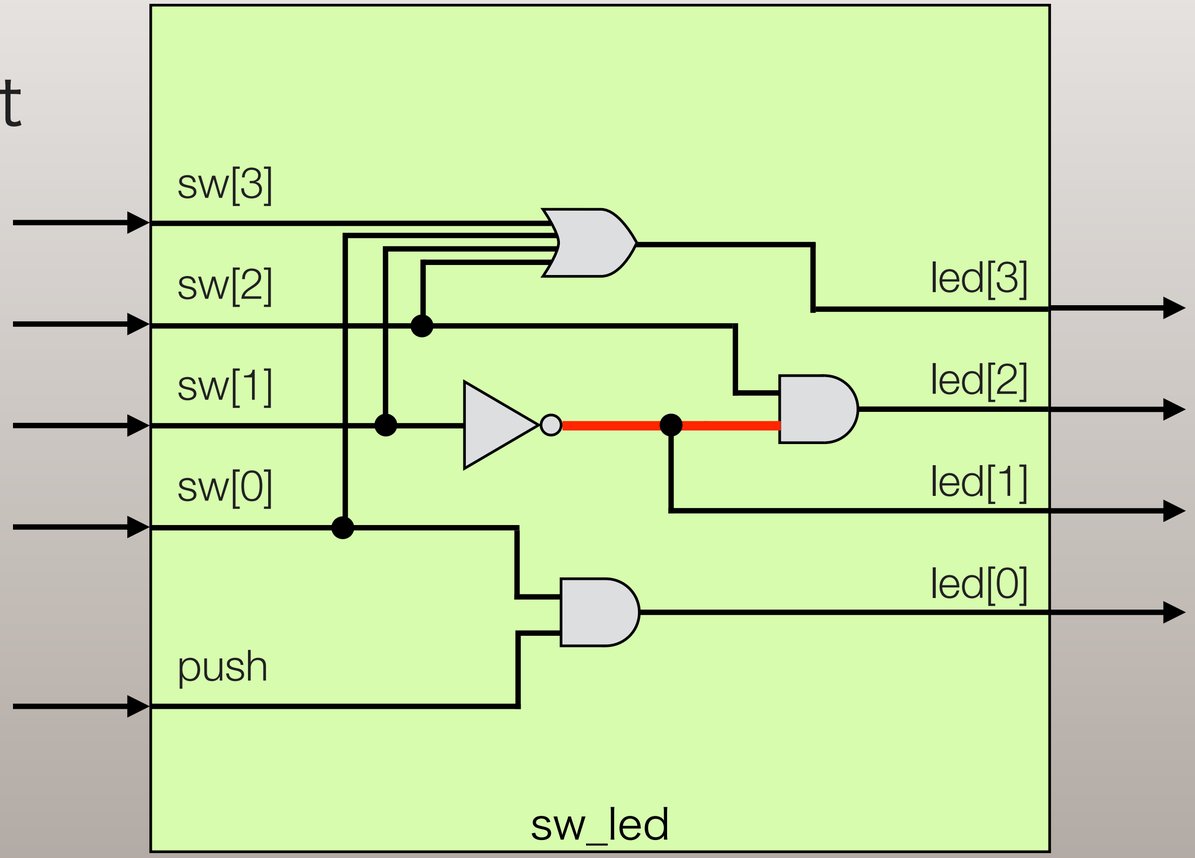
```
module sw_led
  (
   input  wire [3:0] SW,
   input  wire       PUSH,
   output wire [3:0] LED
  );

  assign LED[0] = PUSH & SW[0];
  assign LED[1] = ~SW[1];
  assign LED[2] = LED[1] & SW[2];

endmodule
```



Name internal signals

* “wire” type for assign statement

```
module sw_led
  (
   input  wire [3:0] SW,
   input  wire       PUSH,
   output wire [3:0] LED
  );

  wire   SW1_;
  assign SW1_ = ~SW[1];
  assign LED[1] = SW1_;
  assign LED[2] = SW1_ & SW[2];

endmodule
```



Name internal signals

- Assignment can be done within wire declaration

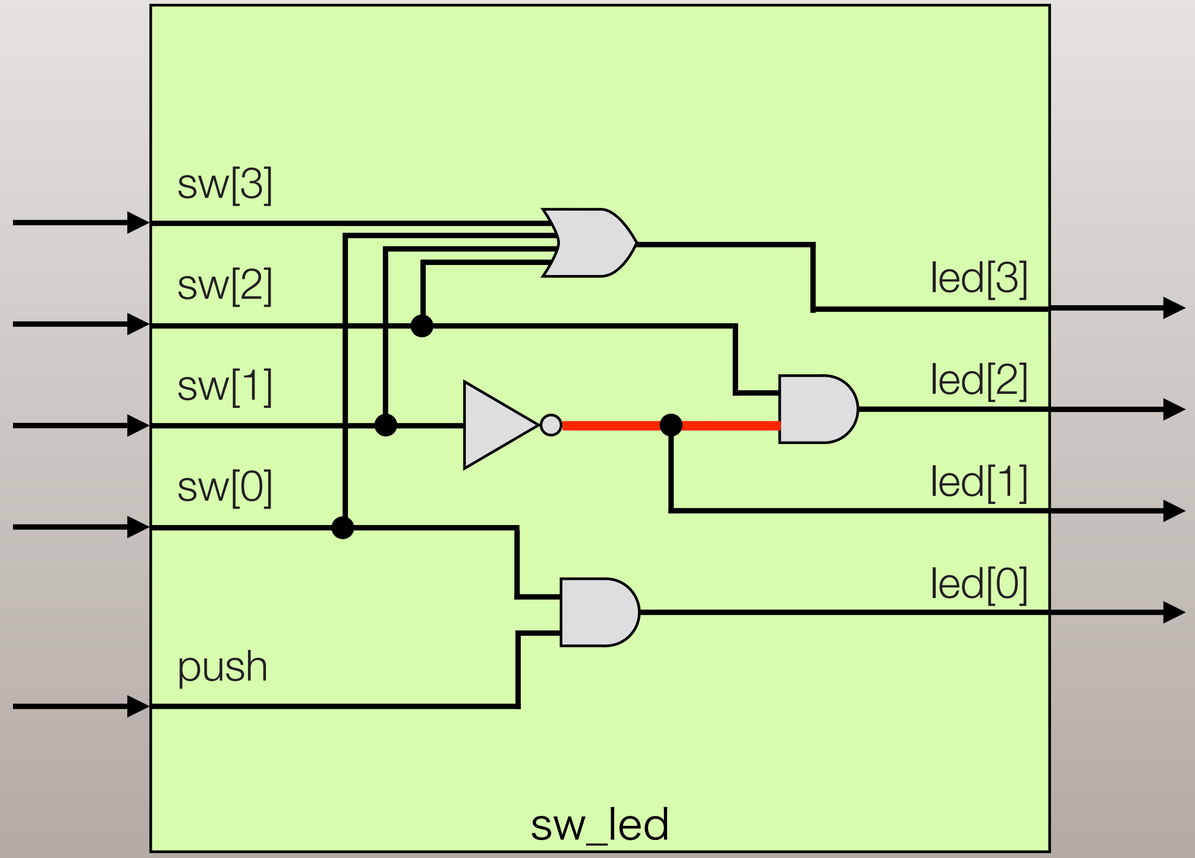
```
module sw_led
  (
   input  wire [3:0] SW,
   input  wire       PUSH,
   output wire [3:0] LED
  );

  wire   SW1_;
  assign SW1_ = ~SW[1];
  assign LED[1] = SW1_;
  assign LED[2] = SW1_ & SW[2];

endmodule
```

```
module sw_led
  (
   input  wire [3:0] SW,
   input  wire       PUSH,
   output wire [3:0] LED
  );

  wire   SW1_ = ~SW[1];
  assign LED[1] = SW1_;
  assign LED[2] = SW1_ & SW[2];

endmodule
```

Reduction operator

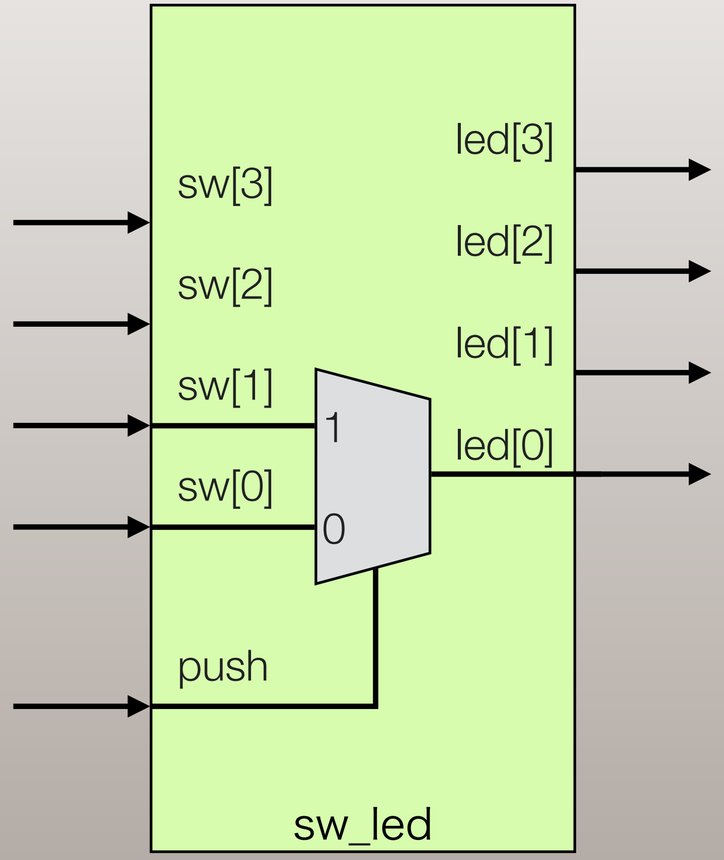
- LED[3]: Simplify or'ing call
  - Operator before signal

```
module sw_led
  (
   input  wire [3:0] SW,
   input  wire       PUSH,
   output wire [3:0] LED
  );

 assign LED[3] = |SW;

endmodule
```



Ex 2 : Multiplexer

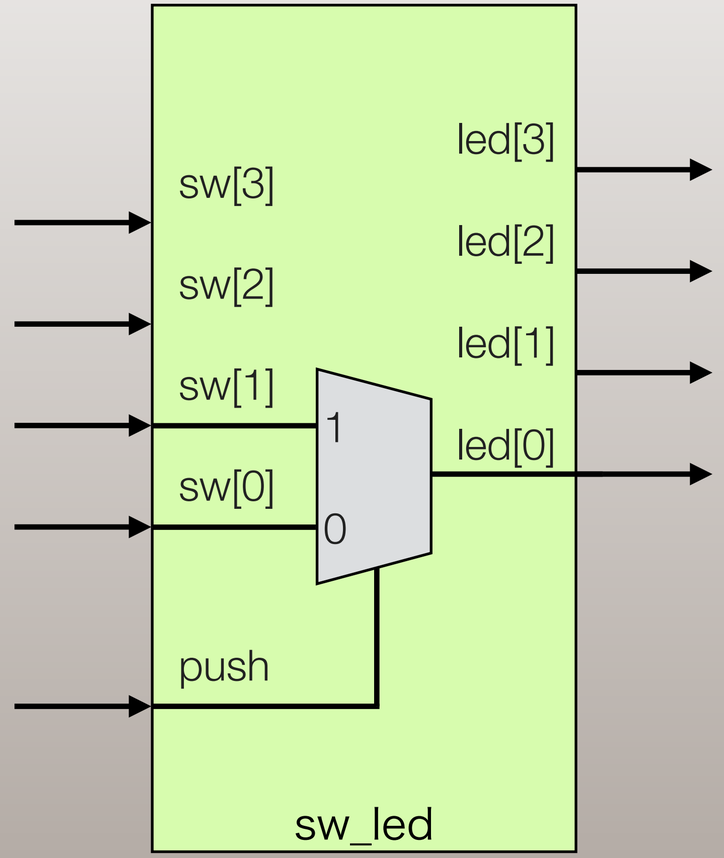
- 2-input MUX
  - Not simple with logic operators
- Conditional assignment
  - assign X = (cond) ? value : default_val;
  - Don’t forget default value



Ex 2 : Multiplexer

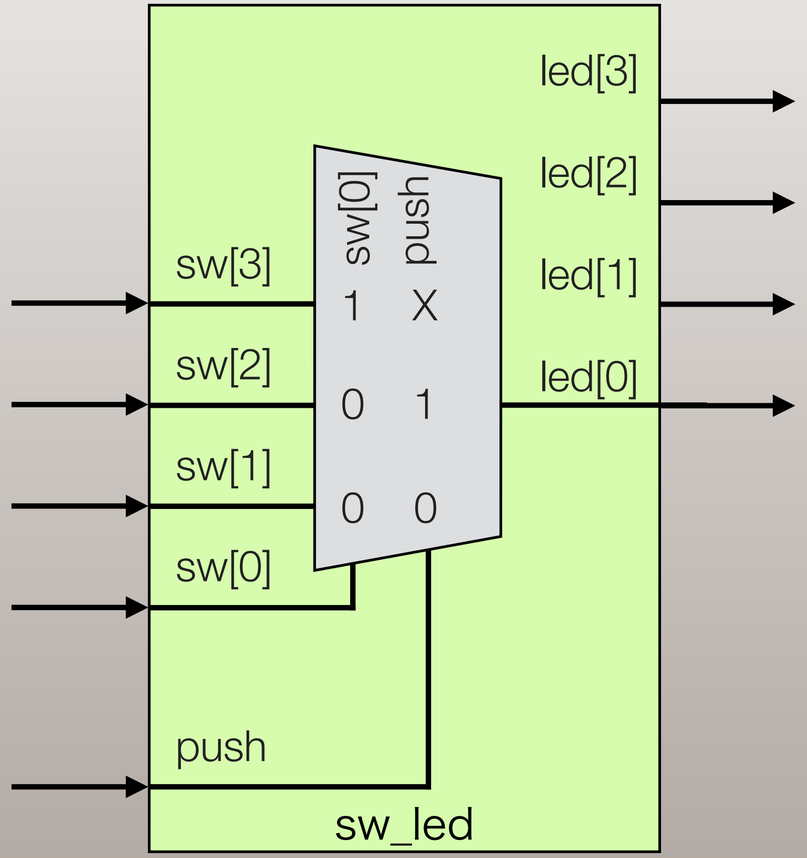
- Conditional assignment
  - assign X = (cond) ? val : default_val;

```
module sw_led
  (
   input  wire [3:0] SW,
   input  wire       PUSH,
   output wire [3:0] LED
  );

 assign LED[0] = PUSH ? SW[1] : SW[0];

endmodule
```



Ex2.5: Multi-input MUX

* 2bit select signal



Signal decomposition and concatenation

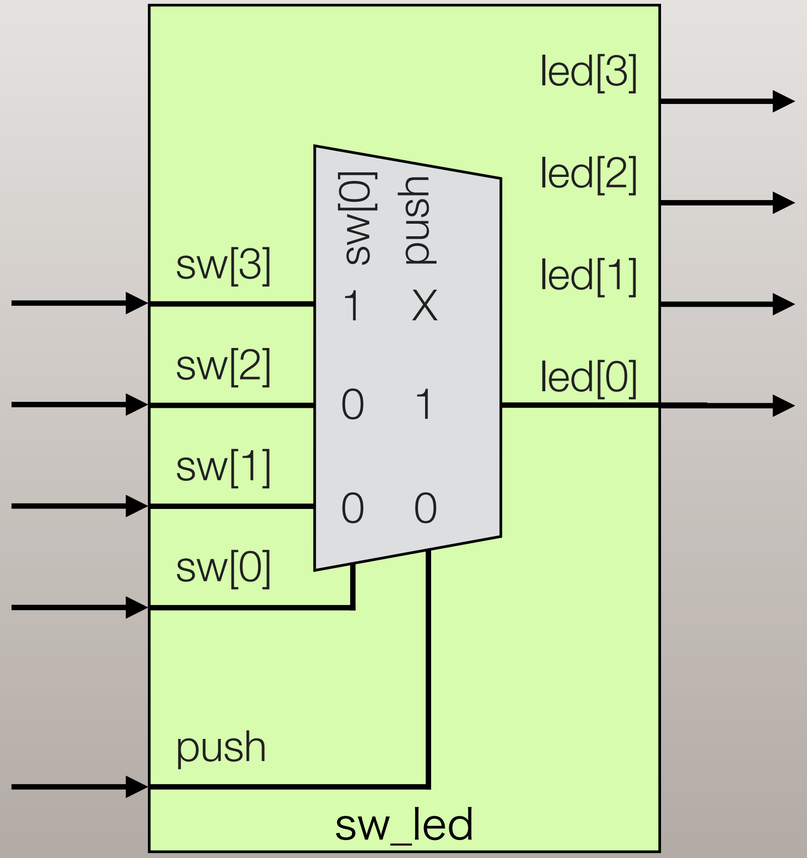
- [] to extract bit(s)
- {} to concatenate
  - New “bundled” signal
  - Duplication:
    {4{X}} is { X, X, X, X }

```
module sw_led
  (
   input  wire [3:0] SW,
   input  wire       PUSH,
   output wire [3:0] LED
  );

 wire [1:0] SEL;
 assign SEL = { SW[0], PUSH };


endmodule
```

Ex2.5: Multi-input MUX

* Possible with conditional assignment

```
module sw_led
  (
   input  wire [3:0] SW,
   input  wire       PUSH,
   output wire [3:0] LED
  );

 wire [1:0] SEL;
 assign SEL = { SW[0], PUSH };
 assign LED[0] = (SEL==2'b00) ? SW[1] :
                 (SEL==2'b01) ? SW[2] :
                 SW[3];

endmodule
```



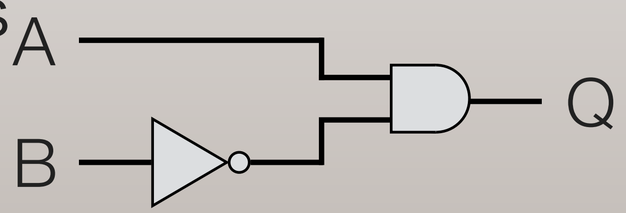
Conditional operations

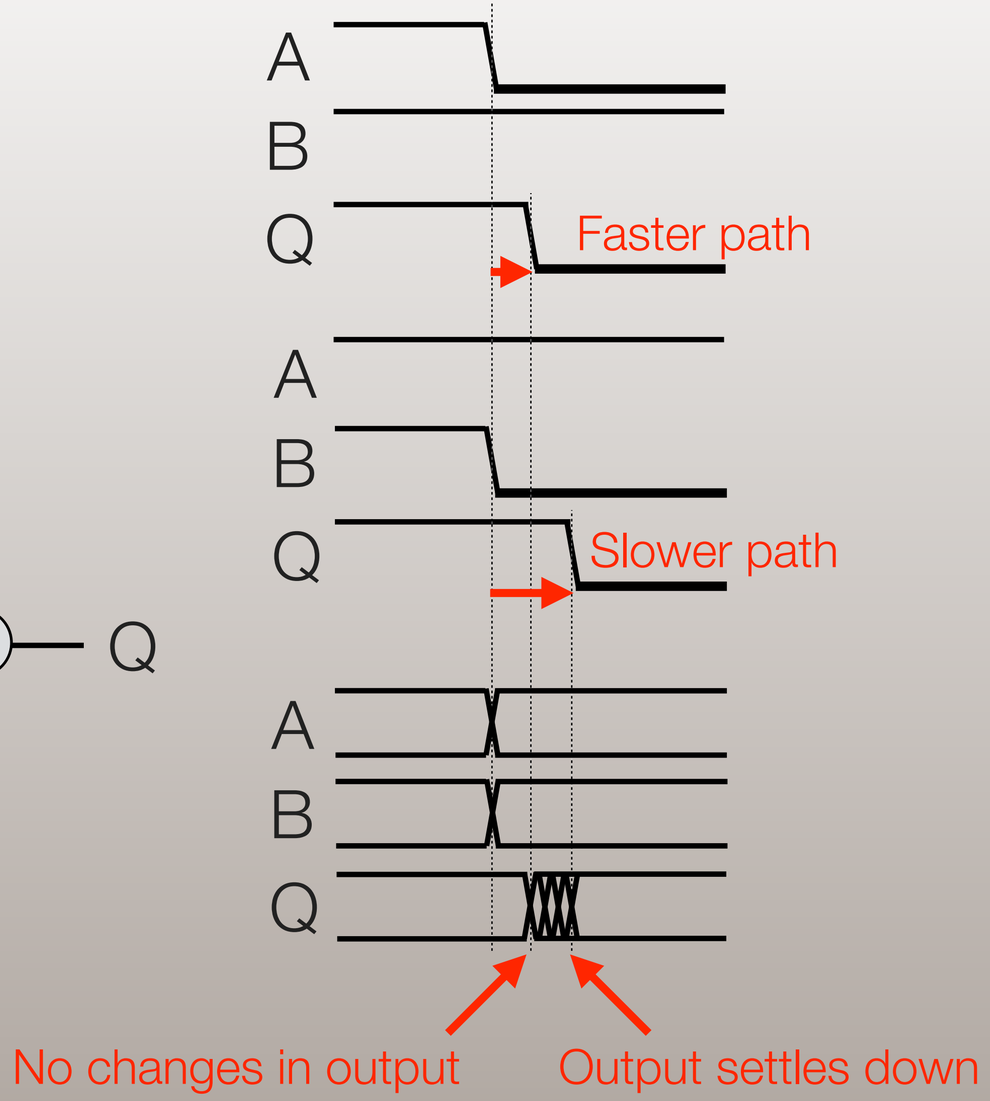
- Think about circuit!
  - Don’t care is “x”
- For correctness:
  - Write default value
  - Check consistency of conditions

```
// Using don’t care makes circuit smaller
assign X = ( A == 3’b1xx ) ? 3 :
           ( A == 3’b01x ) ? 2 :
           ( A == 3’b001 ) ? 1 : 0;

// Adder is synthesized: slow and large
assign X = ( A < 100 ) ? B : C;

// With priority: slow
assign X = COND1 ? ( COND2 ? A : B ) : C;

// Not mutually exclusive: will be broken
assign X = ( A[3:2] == 2’b10 ) ? 1 :
           ( A[1:0] == 2’b11 ) ? 2 : 3;
```

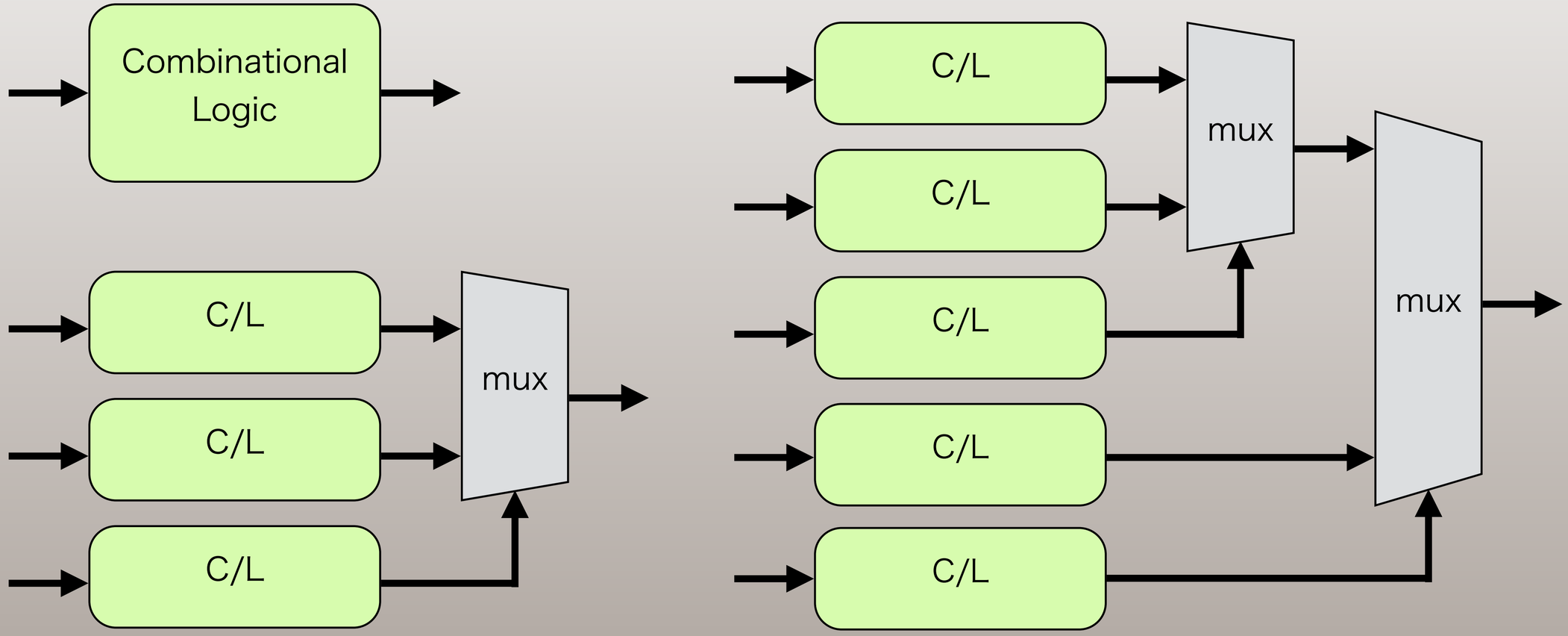
Gates and Delays

- Input / Output transitions
  - Gate and wire delay
  - Different input causes different delay time
- Contamination delay and Propagation delay





Think about resulting circuit



Condition affects on # inputs of MUX and # MUX stages

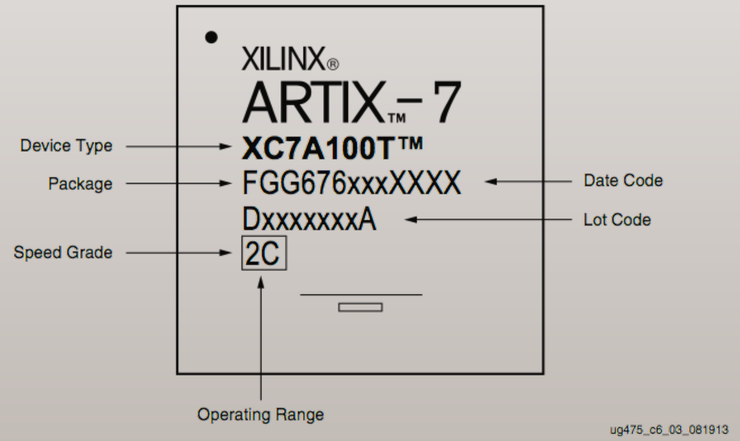
All above is about “Internals”

- Thinking about “inside” of the FPGA is not sufficient
  - Let’s see signal pins of the FPGAs

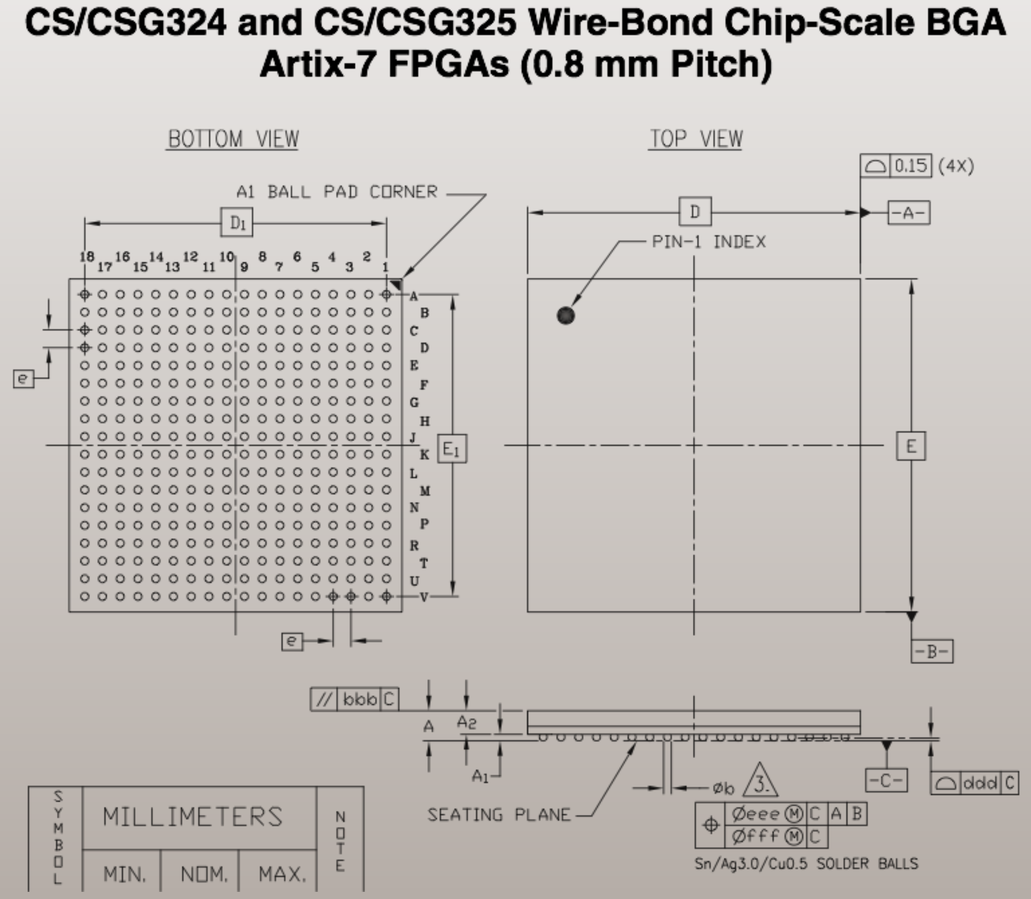
FPGA is general-purpose

- Some flexibility of using signal pins
  - “Dedicated” pins such as power supplies and startup
  - “User I/O” pins that are fully programmable
  - “Multi-purpose” pins for user I/O but some special capability such as clock inputs

Front and back of the chip

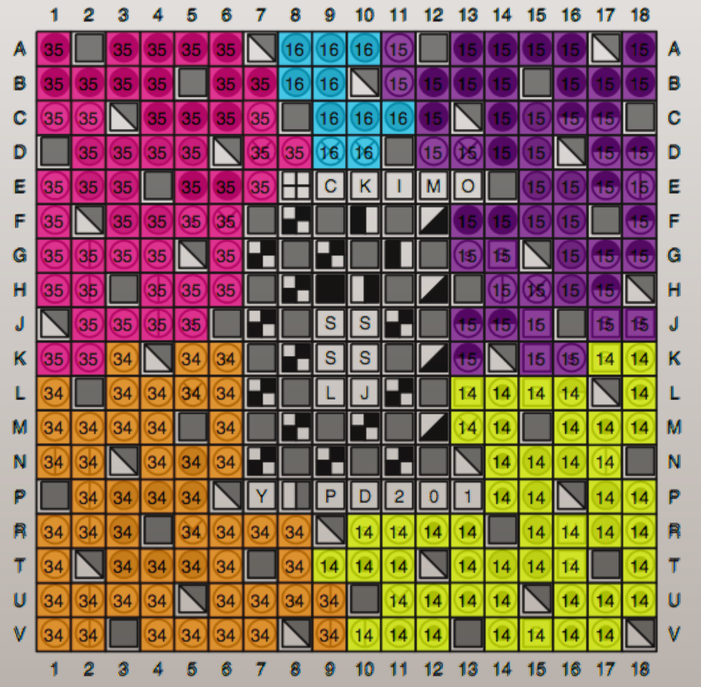


Xilinx UG475より

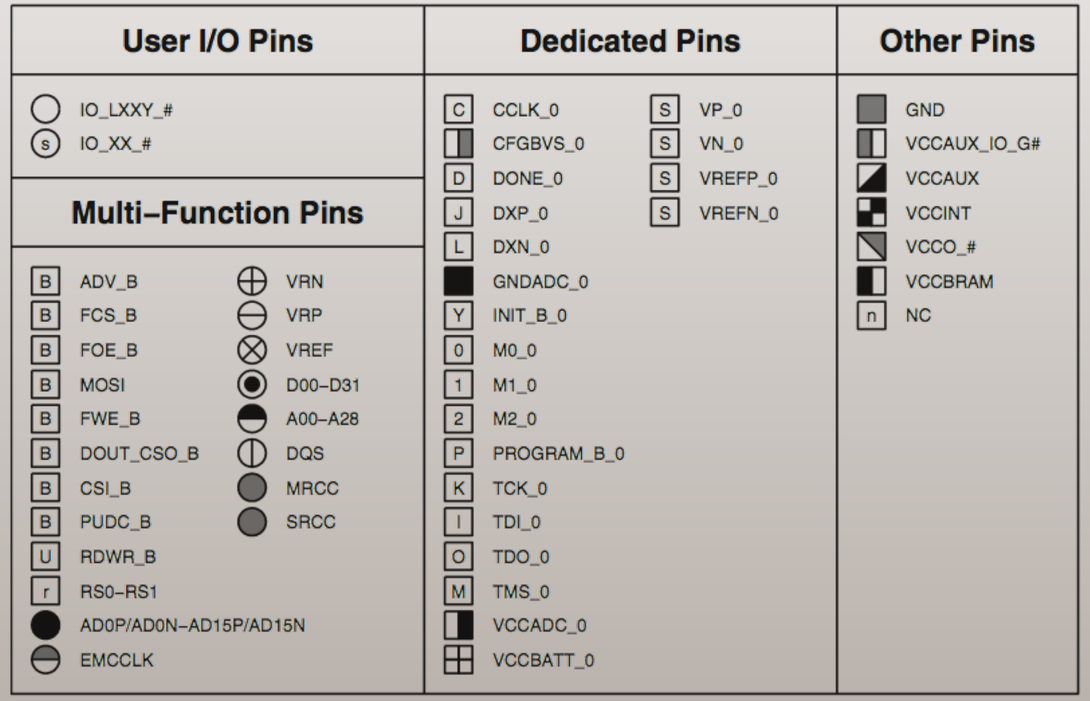


BGA (Ball Grid Array) is popular: pins are regularly arranged solder balls

Pin maps



ug475_c3_306_081211



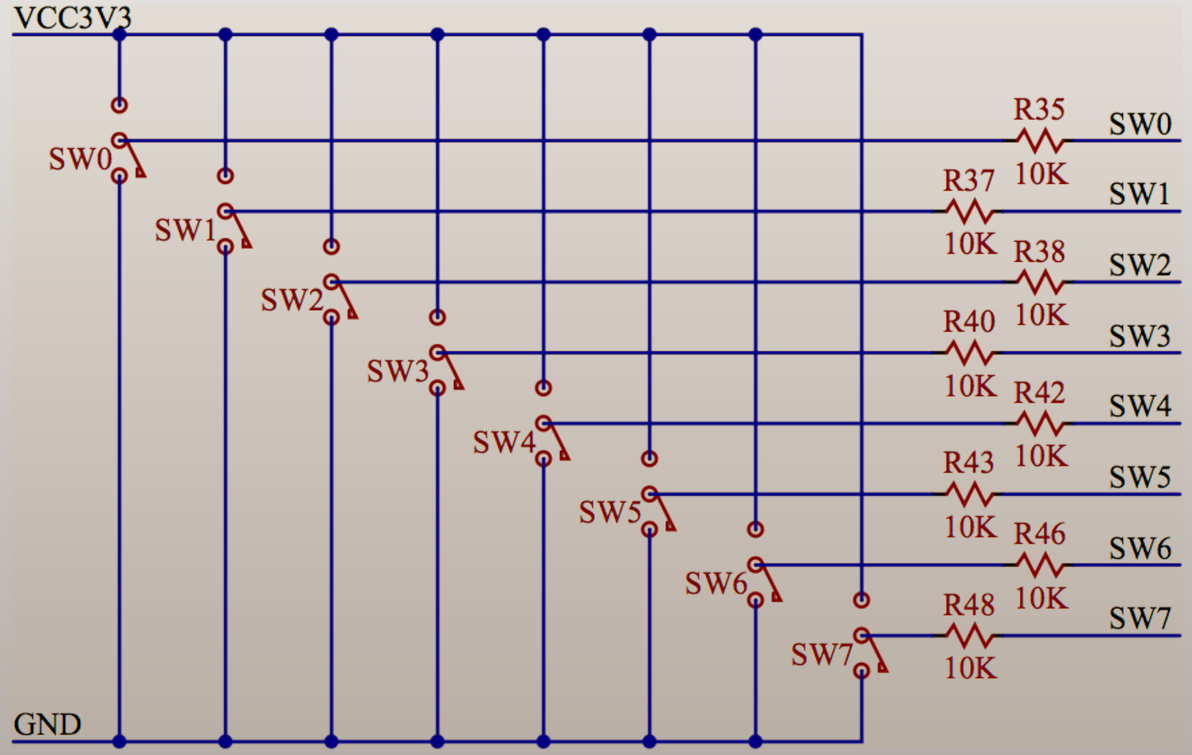
ug475_c3_305_090711

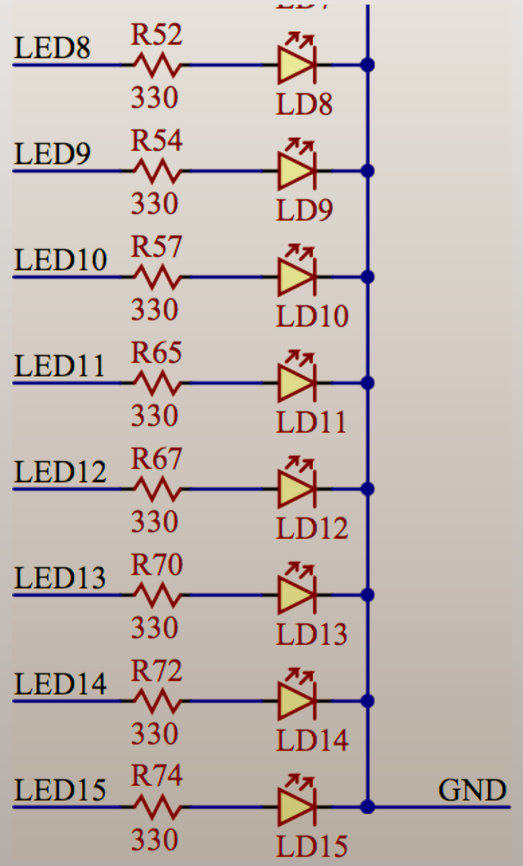
Colored are user I/O pins, different colors are I/O banks

Pin arrangement and designer

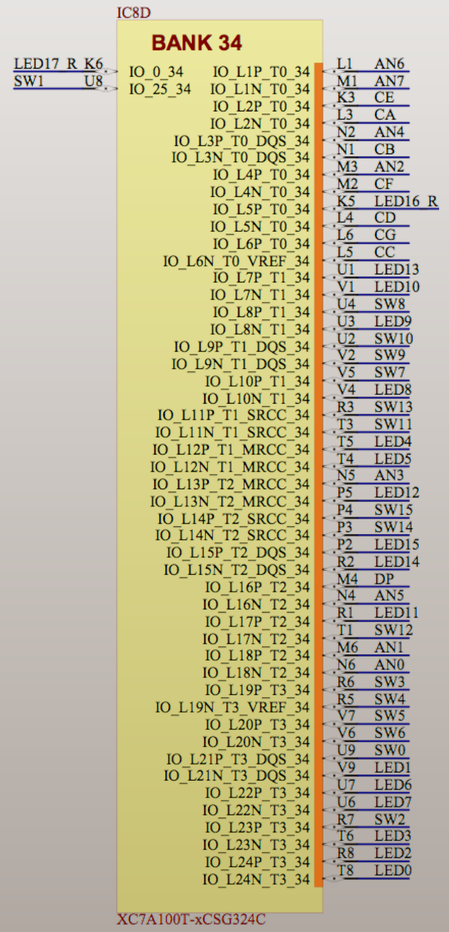
- Required to read the data sheet carefully if:
  - You’re designing FPGA board
  - Connecting something to board’s GPIO (General Purpose I/O)
- Otherwise, we have to just know “what is connected to which pin”
  - Using off-the-shelf board is (mostly) this

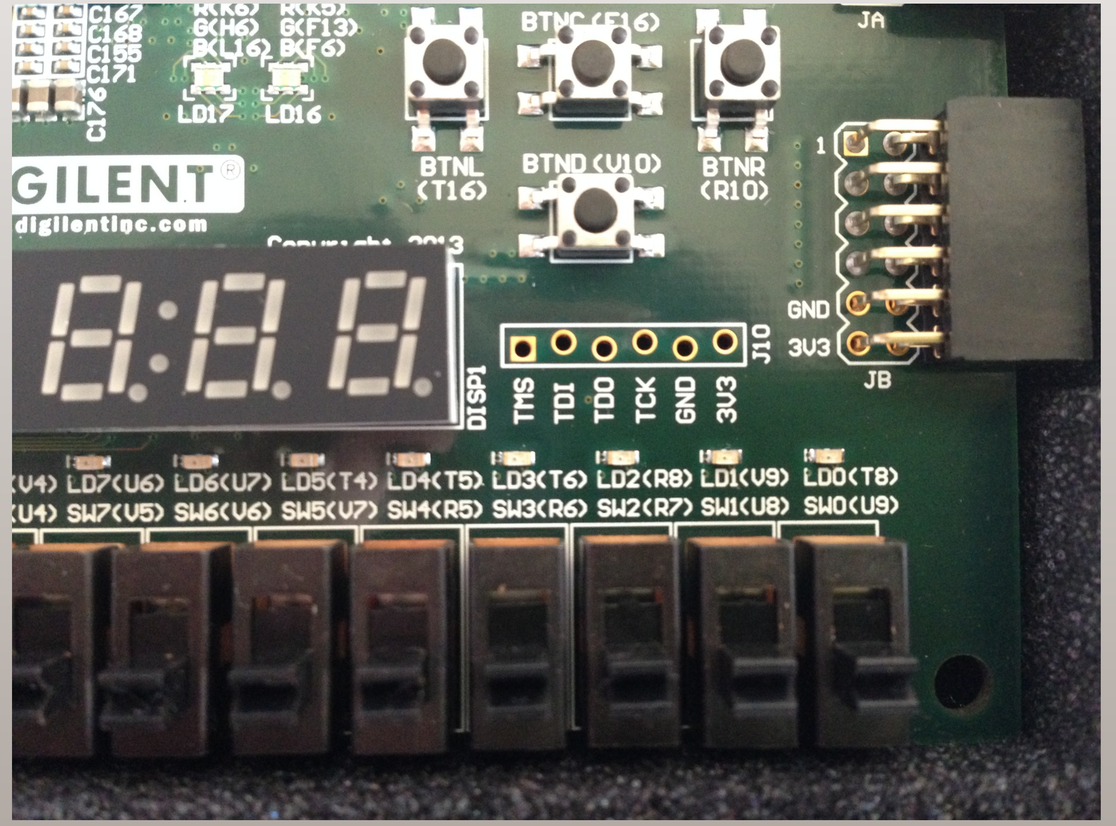
Pin assignment: Schematics





Pin assignemnt: Schematics





Constraints

- Pin assignments are not described by HDL
  - Given to CAD tools as “implementation constraints”
    - Independent file from HDL
  - Pin number, voltage, clock frequency, etc.

XDC constraints (Vivado)

```
module sw_led
  (
   input  [3:0] SW,
   input        PUSH );

set_property PACKAGE_PIN U9 [get_ports SW [0]]
set_property PACKAGE_PIN U8 [get_ports SW [1]]
set_property PACKAGE_PIN R7 [get_ports SW [2]]
set_property PACKAGE_PIN R6 [get_ports SW [3]]
set_property IOSTANDARD LVCMOS33 [get_ports {SW[*]}]

set_property PACKAGE_PIN E16 [get_ports PUSH]
set_property IOSTANDARD LVCMOS33 [get_ports PUSH]
```

UCF constraints (ISE: obsoleted)

```
module sw_led
  (
   input  [3:0] SW,
   input        PUSH );

NET “SW[0]” LOC = U9;
NET “SW[1]” LOC = U8;
NET “SW[2]” LOC = R7;
NET “SW[3]” LOC = R6;
NET “SW[*]” IOSTANDARD = LVCMOS33;

NET “PUSH” LOC = E16 | IOSTANDARD = LVCMOS33;
```