Digital Logic Circuit (SE273 – Fall 2020)

Lecture 6: Hardware Description Languages

Some slides from ICL, Peter Chung

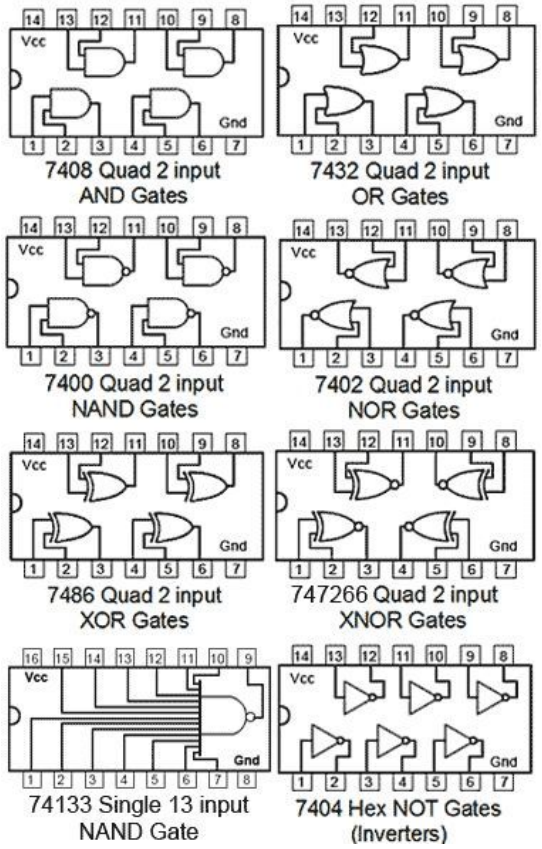
Jaesok Yu, Ph.D. (jaesok.yu@dgist.ac.kr)

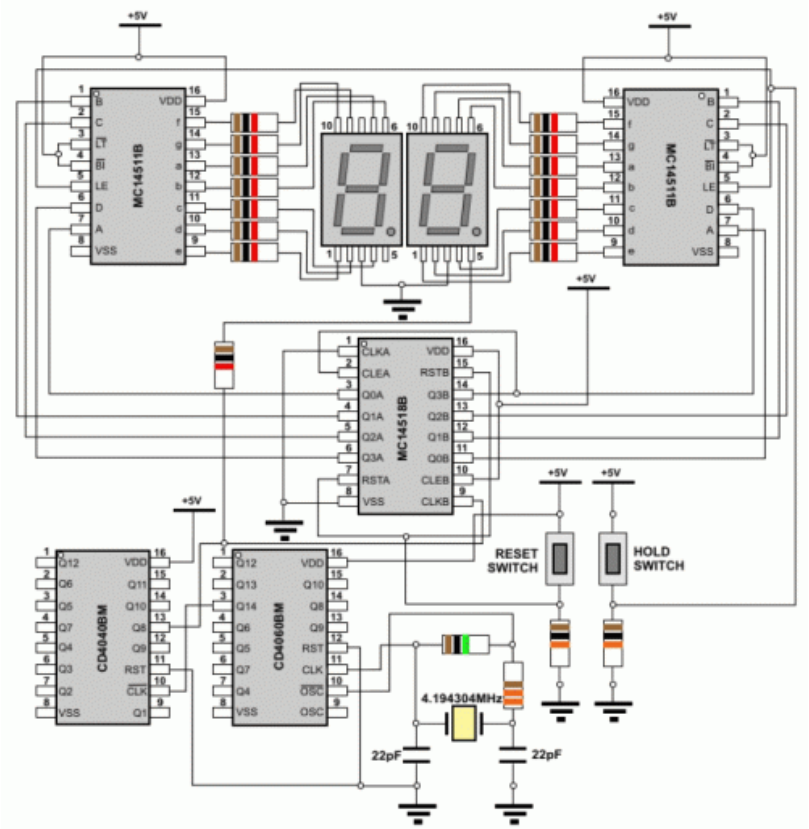
Assistant Professor

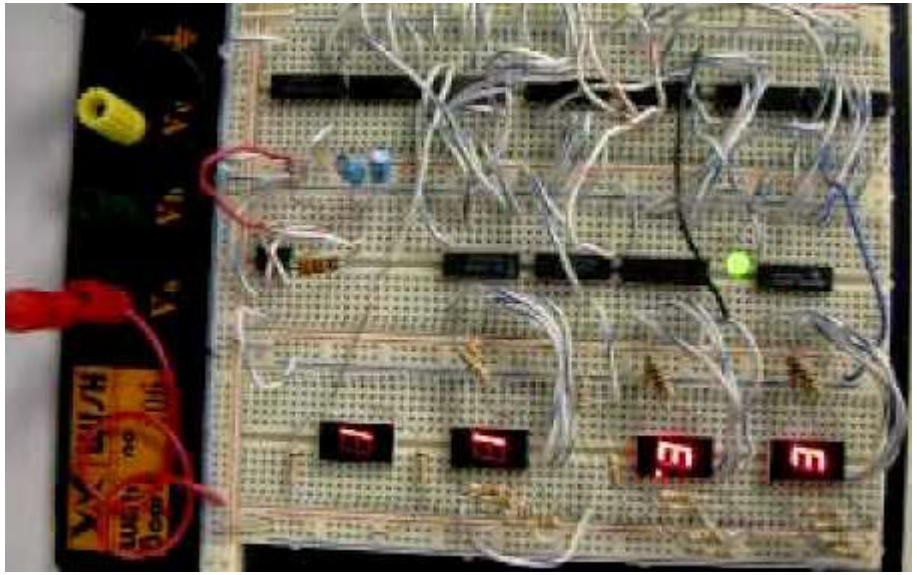
Department of Robotics Engineering, DGIST

Traditional ways of implementing digital circuit

- Discrete logic based on gates or small silicon
- Tedious, Slow, Expensive, Low Efficiency, Low Readability

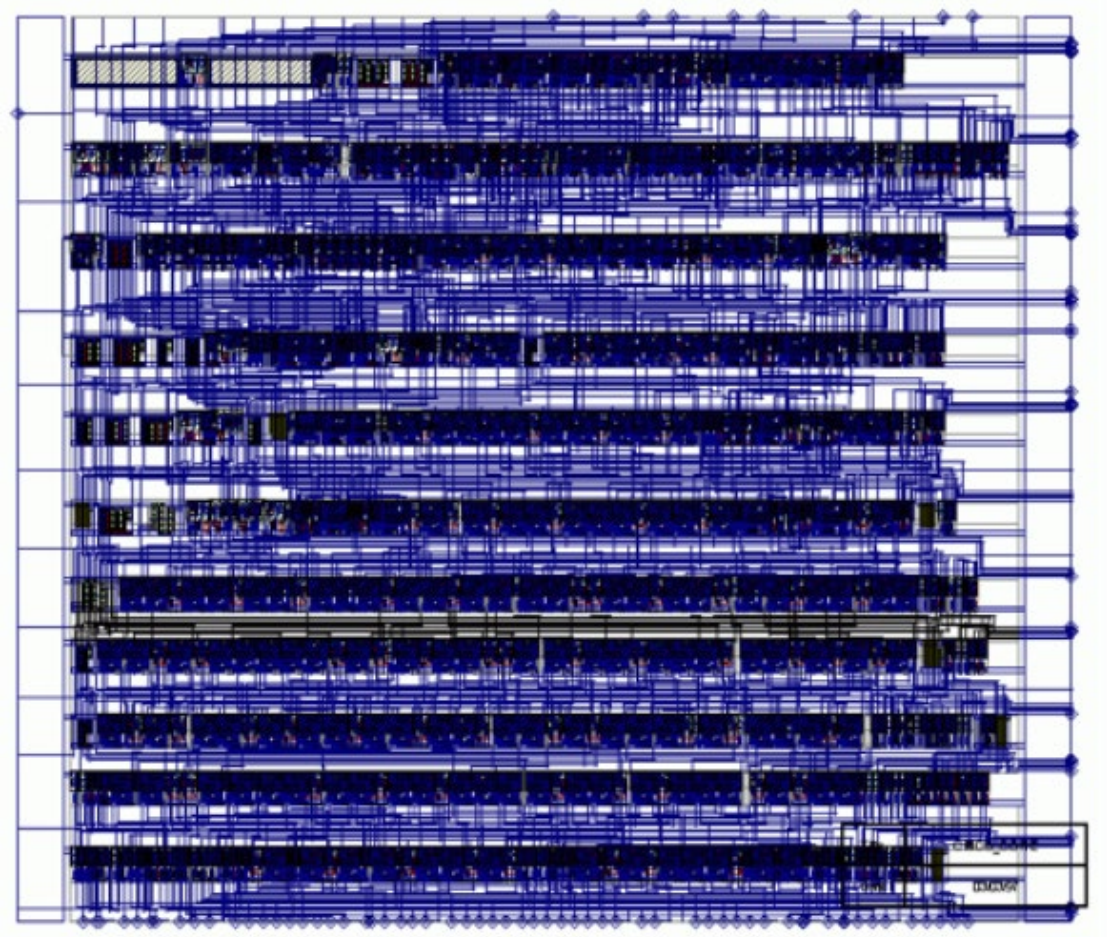






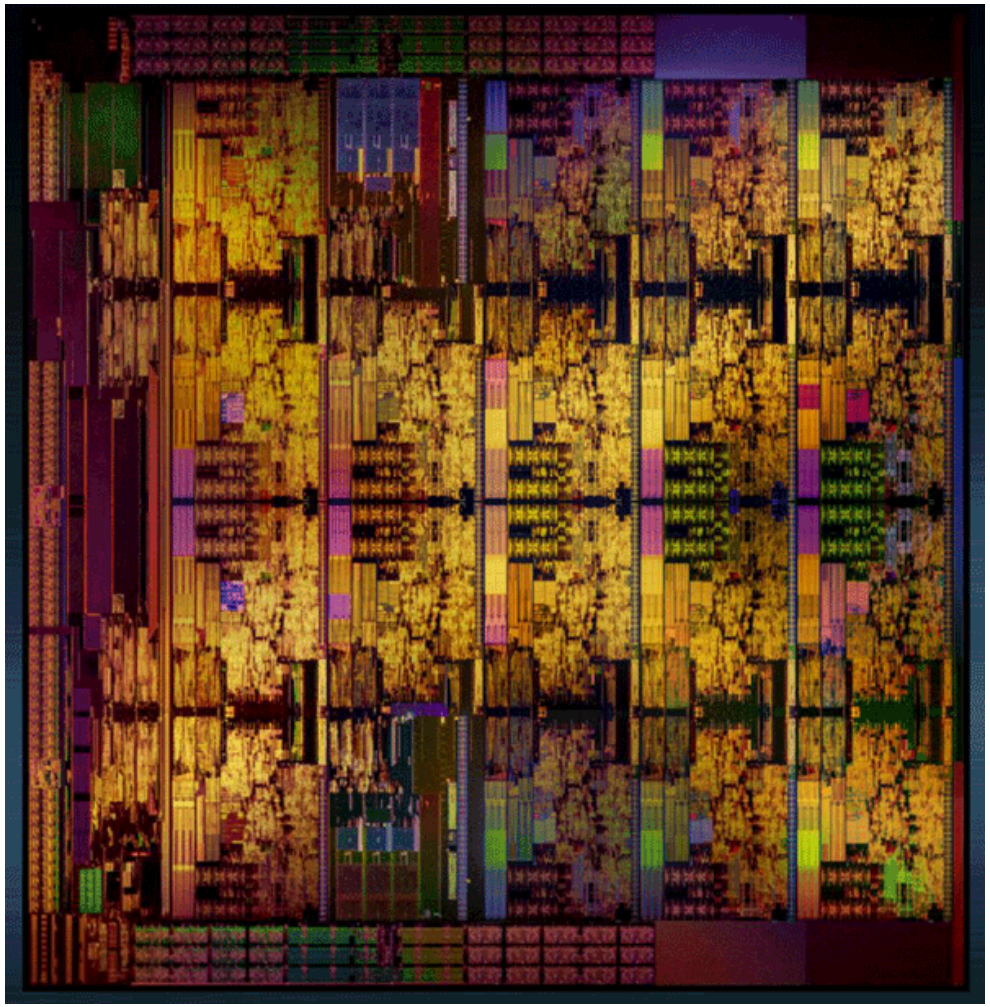
Early Integrated Circuits based on Gate Arrays

- Row of Standard gates – Connect or Disconnect between gates to form customer specific circuits



- Can be full-custom
  - Completely fabricated from scratch
- Can-be semi-custom
  - Customization on the metal layers only
- Once fabricated, the design is fixed.

Modern Digital Design – Full custom Integrated Circuits



- Intel Core I9 Die Shot
- >20 billion transistors
- Expensive to design and manufacture
- Highly risky
- Hard to change the design
- Not viable unless the market is huge

- Most applications cannot afford to embark on such a design

> FPGA can be solution.

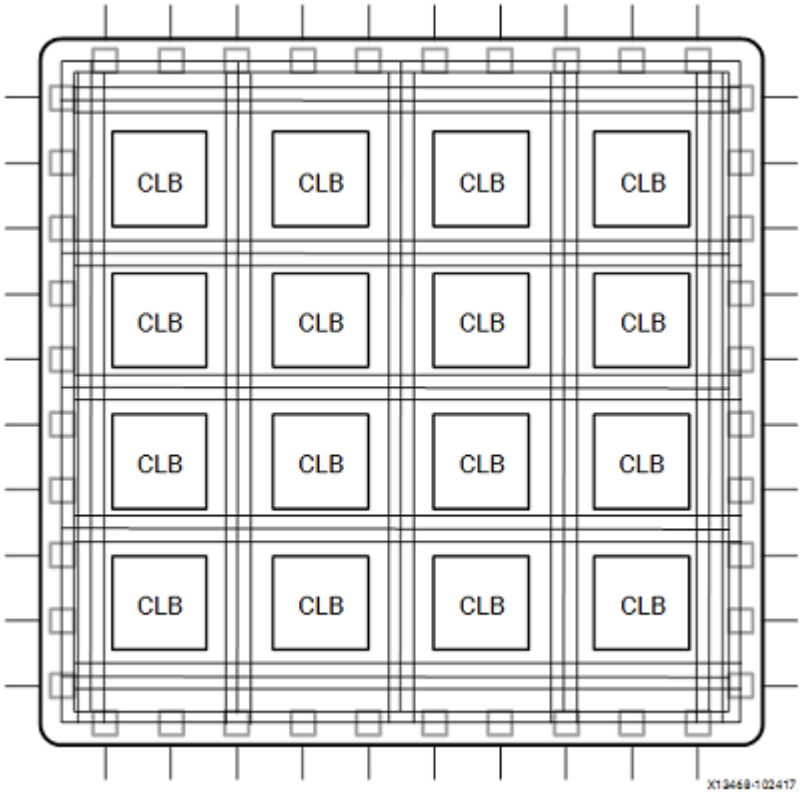
Field Programmable Gate Arrays (FPGAs)

- Combining idea from Programmable Logic Devices (PLDs) and gate arrays
- First introduced by Xilinx (1985)
- **Pros**
  - Fast (Parallelism)
  - Flexibility
  - Power-efficient
  - Low prototyping cost

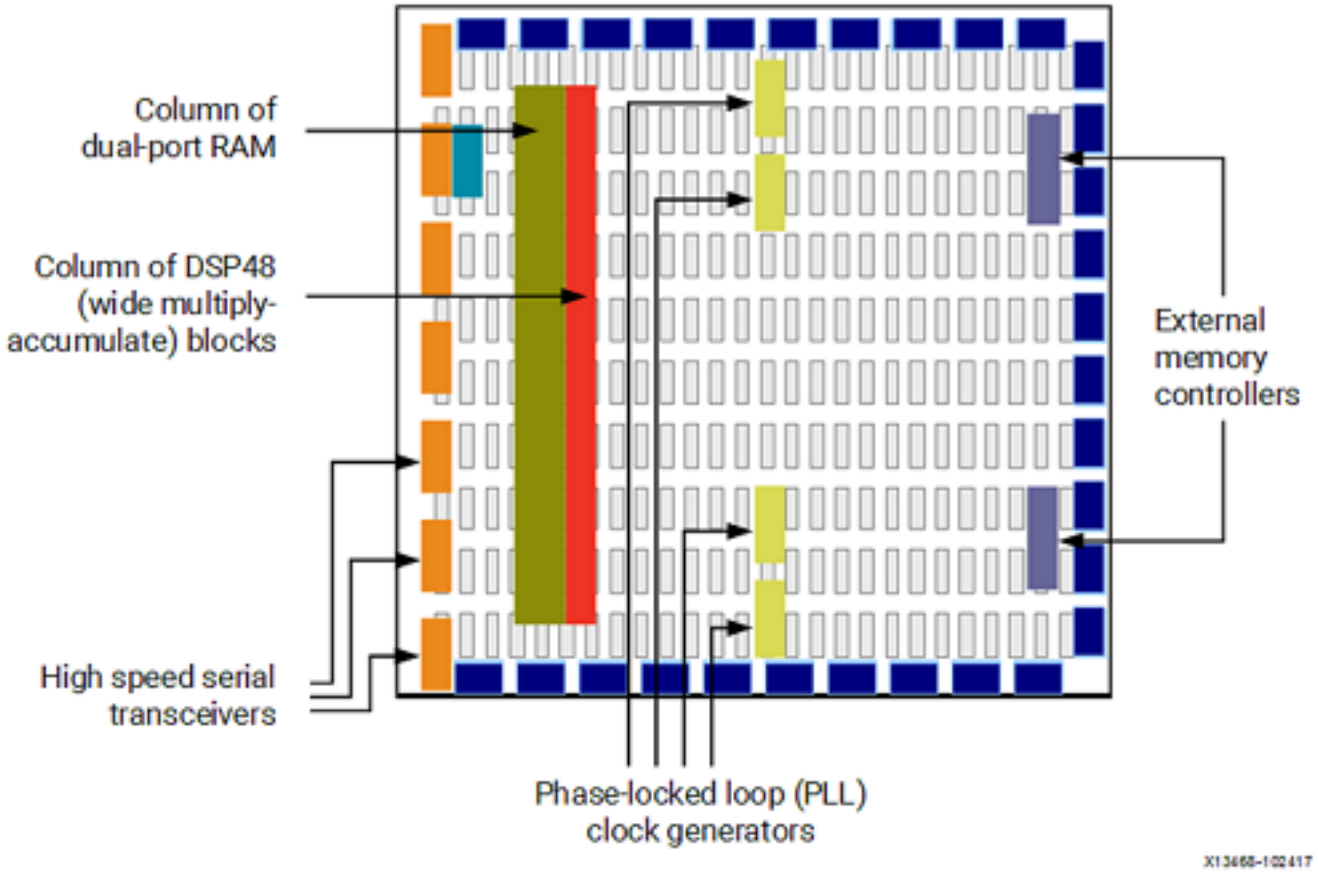
- **Applications**
  - ASIC/DSP prototyping (Simulation > FPGA Hardware > ASIC/DSP Hardware)
  - A.I. accelerator: **FPGA** vs GPGPU
    - Pros: Performance / Power efficiency / Heat
    - Cons: Requiring hardware design knowledge.
    - (Recently, compiler from C language to HDL was released but you still need to know hardware design skill to achieve high performance.)

Field Programmable Gate Array (FPGA)

- Architecture



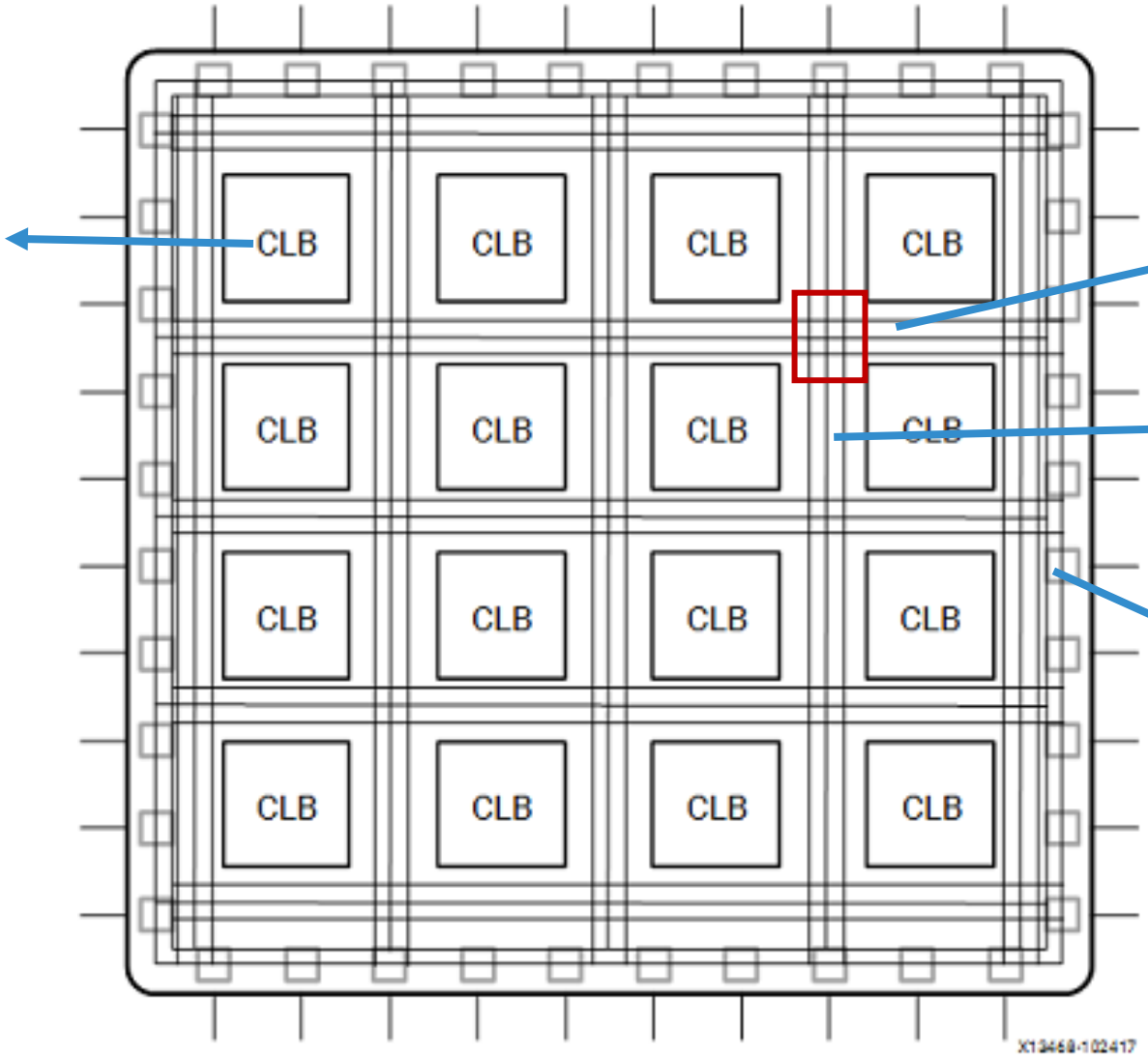
Basic FPGA Architecture



Modern FPGA Architecture

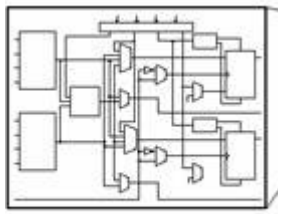
Field Programmable Gate Array (FPGA)

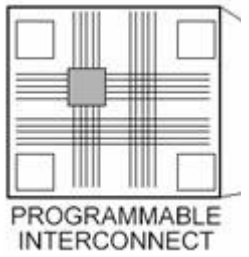
- Architecture



Configurable Logic Block (CLB)

- LUT + Flip-flops

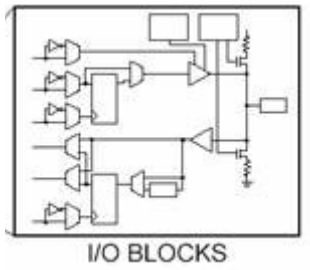




Switching Matrix

Routing Channels

I/O Pad

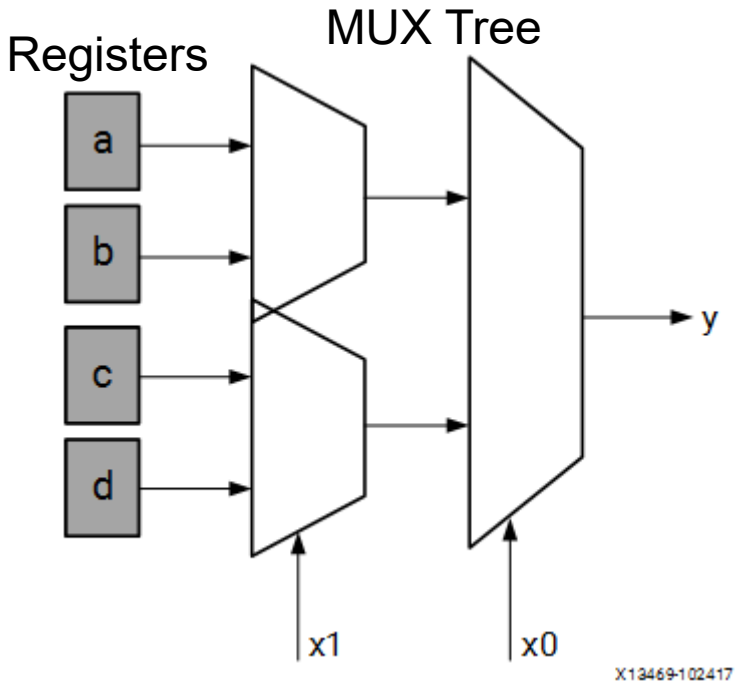


Field Programmable Gate Array (FPGA)

- Look-up table (LUT) - This element performs logic operations.
- Flip-Flop (FF) - This register element stores the result of the LUT.
- Wires - These elements connect elements to one another.
- Input/Output (I/O) pads - These physical ports get data in and out of the FPGA.

Look-up Table (LUT)

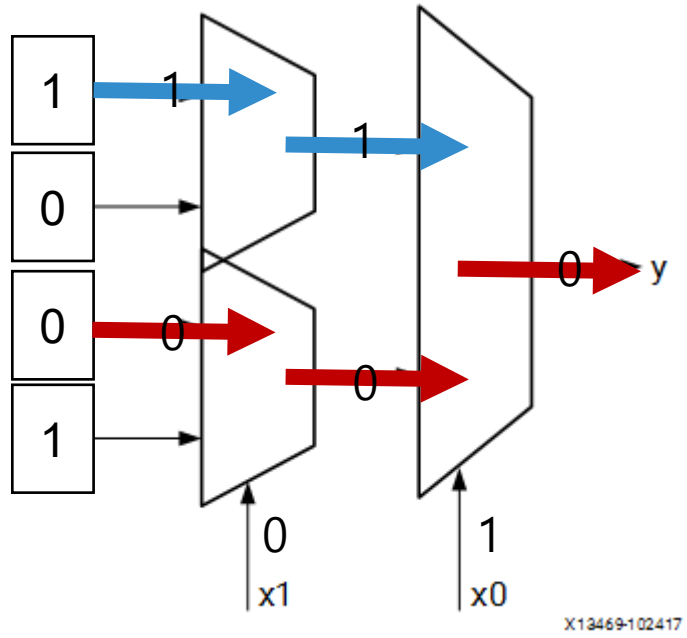
- LUT: The basic building block of an FPGA and is capable of implementing any logic function of N Boolean variables. (In modern Xilinx FPGA, N = 6)
- Truth Table! (Download a BITSTREAM, which is not a PROGRAM)
- NOTE: Discrete logic gates do not actually exist inside of an FPGA!



2-input LUT Structure

x0	x1	y
0	0	1
0	1	0
1	0	0
1	1	1

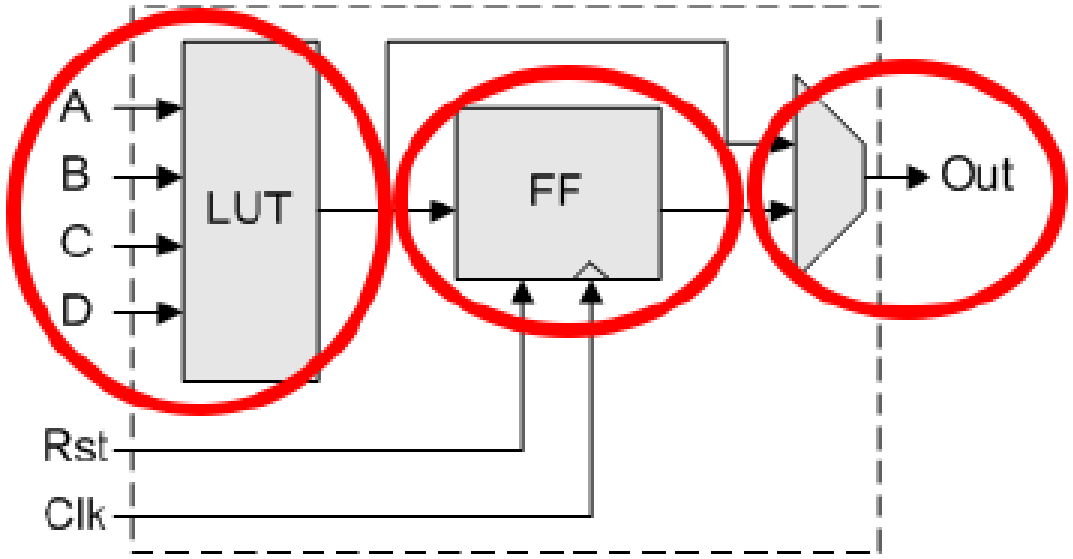
XNOR Truth Table



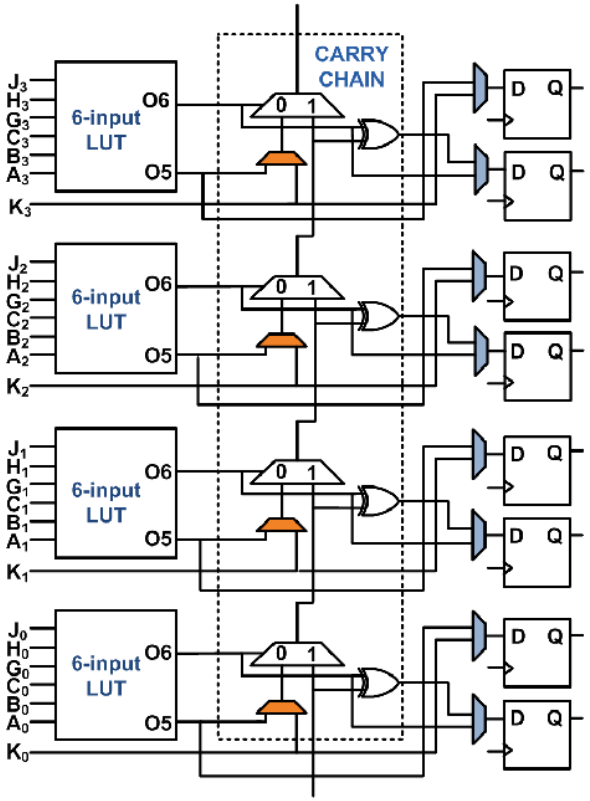
ex) LUT implementation of XNOR gate

Configurable Logic Block (CLB)

- Based around Look-up Tables
- Optional D-Flipflop at the output of the LUT
- Special Circuit for cascading logic blocks (ex, carry-chain)

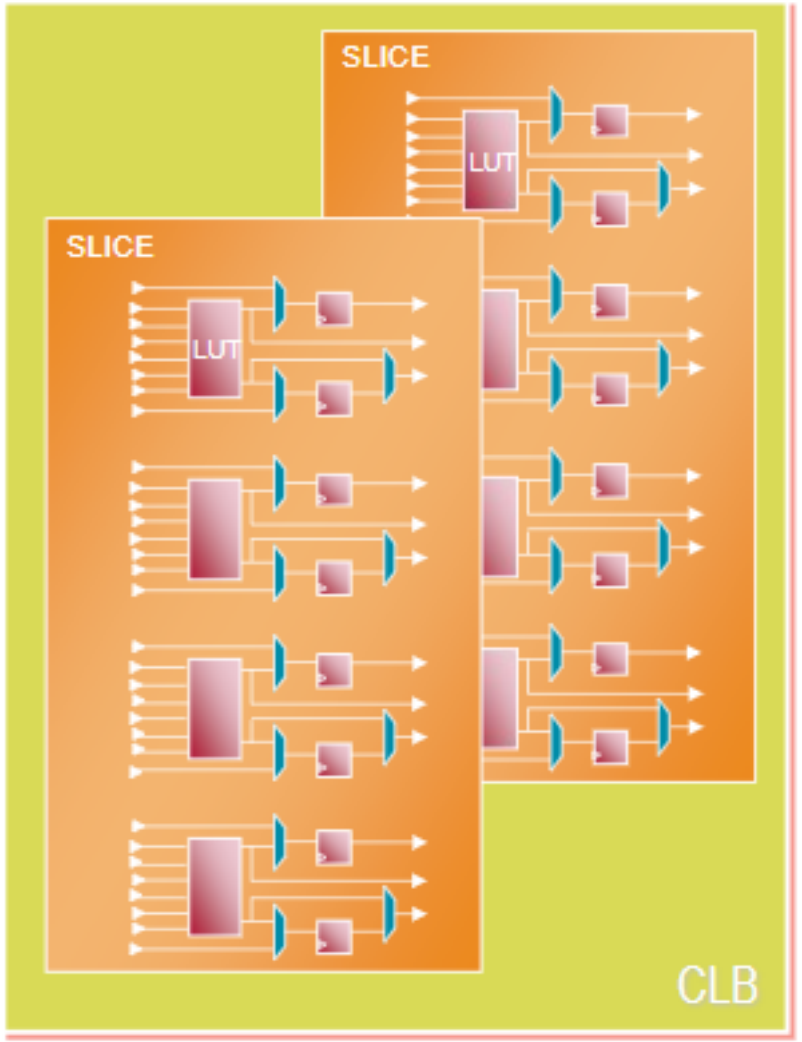


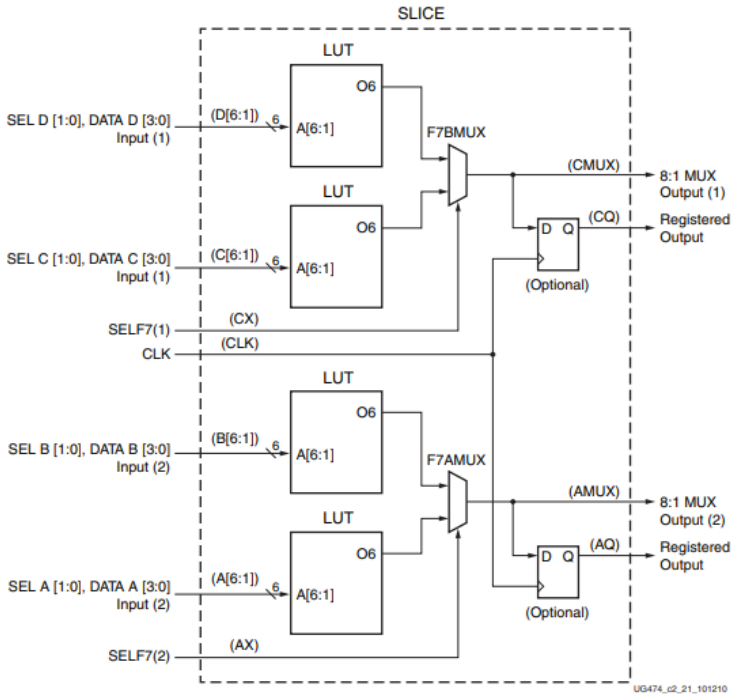
CLB with 4-input LUT and D-flipflop



Configurable Logic Block (CLB)

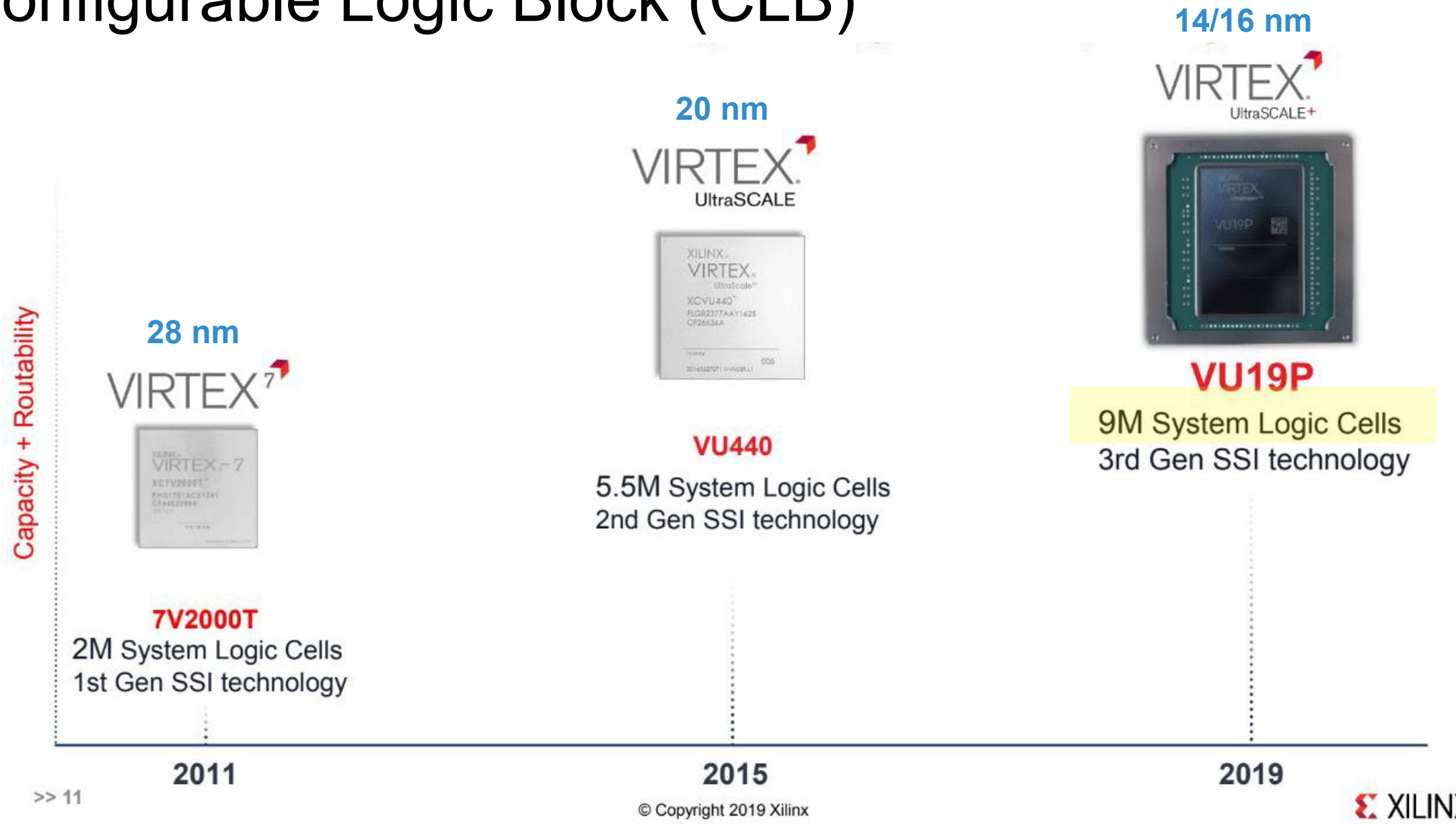
- **Two side-by-side slices per CLB**
  - Slice_M are memory-capable
  - Slice_L are logic and carry only
- **Four 6-input LUTs per slice**
  - Consistent with previous architectures
  - Single LUT in Slice_M can be a 32-bit shift register or 64 x 1 RAM
- **Two flip-flops per LUT**
  - Excellent for heavily pipelined designs





XILINX ALL PROGRAMMABLE.

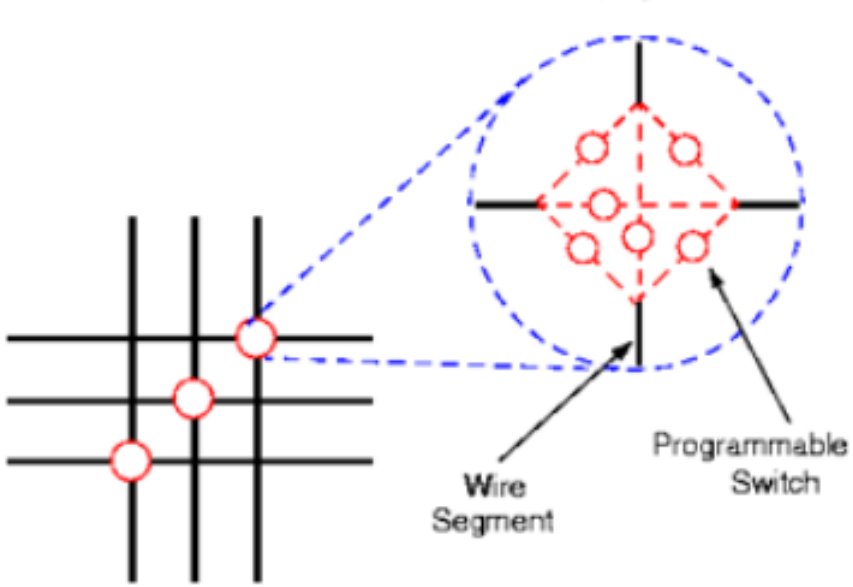
Configurable Logic Block (CLB)

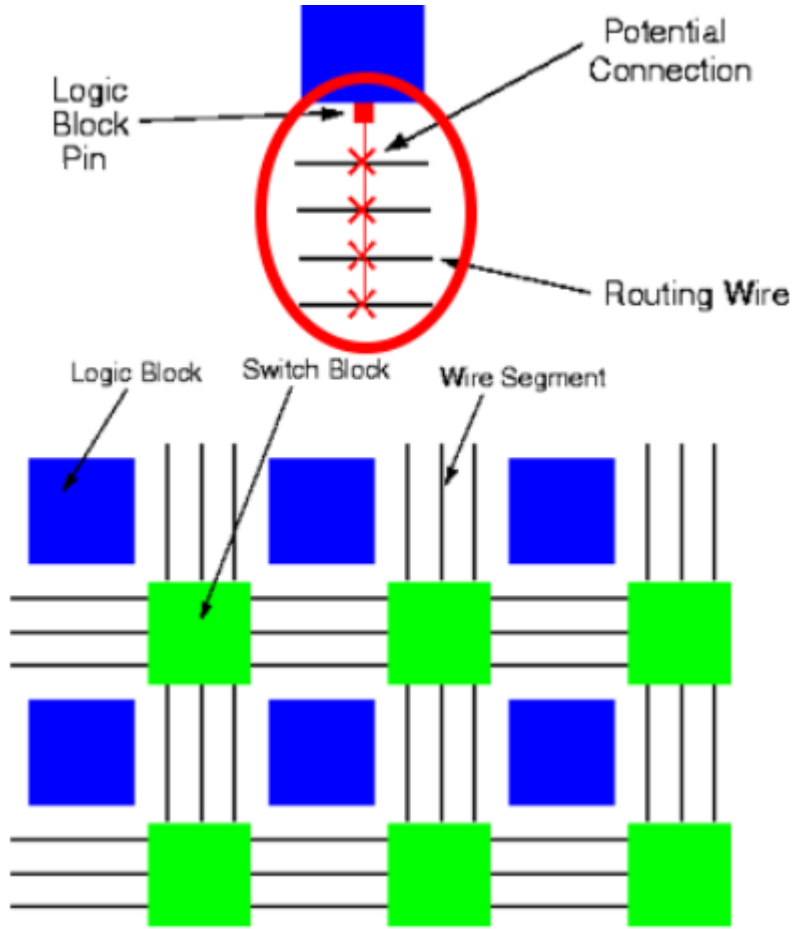


XILINX

Programmable Routing (Switching Matrix)

- Between rows and columns of logic blocks are wiring channels
- Programmable – a logic block pin can be connected to one of many wiring tracks through programmable switch
- Each wire segment can be connected in one of many ways

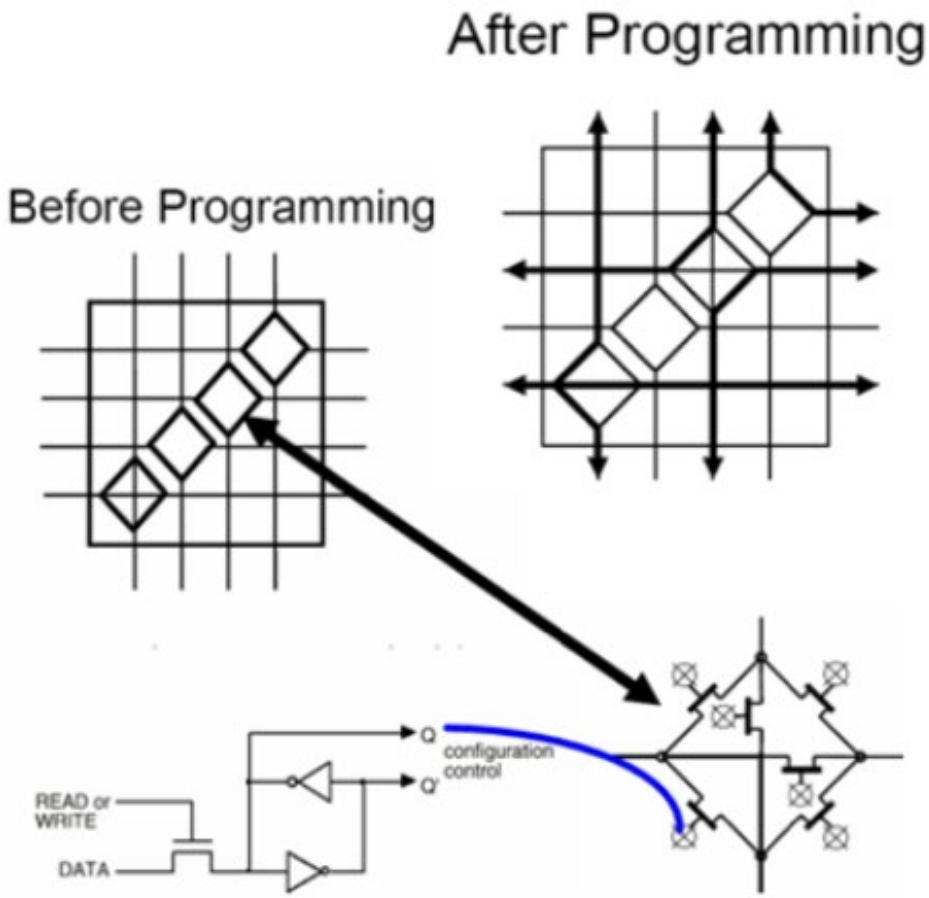




Programmable Routing (Switching Matrix)

- At each interconnect site, there is a transistor switch which is default OFF (not conducting)
- Each switch is controlled by the output of a 1-bit configuration register
- Configuring the routing is simply to put a ‘1’ or ‘0’ in this register to control the routing switches

- Configuration is volatile
- “Bitstream” is either stored on local flash memory or download via computer
- Configuration happens on Power-up



Modern FPGA

Kintex-7 FPGA Feature Summary

Table 6: **Kintex-7 FPGA Feature Summary by Device**

Device	Logic Cells	Configurable Logic Blocks (CLBs)		DSP Slices(2)	Block RAM Blocks(3)			CMTs(4)	PCIe(5)	GTXs	XADC Blocks	Total I/O Banks(6)	Max User I/O(7)
		Slices(1)	Max Distributed RAM (Kb)		18 Kb	36 Kb	Max (Kb)						
XC7K70T	65,600	10,250	838	240	270	135	4,860	6	1	8	1	6	300
XC7K160T	162,240	25,350	2,188	600	650	325	11,700	8	1	8	1	8	400
XC7K325T	326,080	50,950	4,000	840	890	445	16,020	10	1	16	1	10	500
XC7K355T	356,160	55,650	5,088	1,440	1,430	715	25,740	6	1	24	1	6	300
XC7K410T	406,720	63,550	5,663	1,540	1,590	795	28,620	10	1	16	1	10	500
XC7K420T	416,960	65,150	5,938	1,680	1,670	835	30,060	8	1	32	1	8	400
XC7K480T	477,760	74,650	6,788	1,920	1,910	955	34,380	8	1	32	1	8	400

Notes:
1. Each 7 series FPGA slice contains four LUTs and eight flip-flops; only some slices can use their LUTs as distributed RAM or SRLs.
2. Each DSP slice contains a pre-adder, a 25 x 18 multiplier, an adder, and an accumulator.
3. Block RAMs are fundamentally 36 Kb in size; each block can also be used as two independent 18 Kb blocks.
4. Each CMT contains one MMCM and one PLL.
5. Kintex-7 FPGA Interface Blocks for PCI Express support up to x8 Gen 2.
6. Does not include configuration Bank 0.
7. This number does not include GTX transceivers.

Table 7: **Kintex-7 FPGA Device-Package Combinations and Maximum I/Os**

Package(1)	FBG484			FBG676(2)			FFG676(2)			FBG900(3)			FFG900(3)			FFG901			FFG1156		
Size (mm)	23 x 23			27 x 27			27 x 27			31 x 31			31 x 31			31 x 31			35 x 35		
Ball Pitch (mm)	1.0			1.0			1.0			1.0			1.0			1.0			1.0		
Device	GTX(4)	I/O		GTX(4)	I/O		GTX	I/O		GTX(4)	I/O		GTX	I/O		GTX	I/O		GTX	I/O	
		HR(5)	HP(6)		HR(5)	HP(6)		HR(5)	HP(6)		HR(5)	HP(6)		HR(5)	HP(6)		HR(5)	HP(6)		HR(5)	HP(6)
XC7K70T	4	185	100	8	200	100															
XC7K160T	4	185	100	8	250	150	8	250	150												
XC7K325T				8	250	150	8	250	150	16	350	150	16	350	150						
XC7K355T																24	300	0			
XC7K410T				8	250	150	8	250	150	16	350	150	16	350	150						
XC7K420T																28	380	0	32	400	0
XC7K480T																28	380	0	32	400	0

Notes:
1. All packages listed are Pb-free (FBG, FFG with exemption 15). Some packages are available in Pb option.
2. Devices in FBG676 and FFG676 are footprint compatible.
3. Devices in FBG900 and FFG900 are footprint compatible.
4. GTX transceivers in FB packages support the following maximum data rates: 10.3Gb/s in FBG484; 6.6Gb/s in FBG676 and FBG900. Refer to *Kintex-7 FPGAs Data Sheet: DC and AC Switching Characteristics* (DS182) for details.
5. HR = High-range I/O with support for I/O voltage from 1.2V to 3.3V.
6. HP = High-performance I/O with support for I/O voltage from 1.2V to 1.8V.

Hardware Description Languages (HDLs)

Verilog:

```
1
2
3  always @((S0,S1), A, B, C, D)
4        case ((S0,S1))
5           2'b00: Y = A;
6           2'b01: Y = B;
7           2'b10: Y = C;
8           2'b11: Y = D;
9        endcase
10
```

Verilog	VHDL
ASIC Designs	FPGA Designs
Weakly Typed	Strongly Typed
Low Verbosity	High Verbosity
Partially Deterministic	Very Deterministic
More “C” like	Non “C” like

VHDL:

```
2  process ((S0,S1),A,B,C,D)
3  begin
4     case (S0,S1), is
5        when "00" => Y <= A;
6        when "01" => Y <= B;
7        when "10" => Y <= C;
8        when "11" => Y <= D;
9        when others => Y <= A;
10    end case;
11 end process;
```

IEEE standard (1364)
(https://standards.ieee.org/standard/1364-2005.html)

IEEE standard (1076)
(https://standards.ieee.org/standard/1076-2019.html)

Originated from Cadence (Industry)

Originated from USA DoD

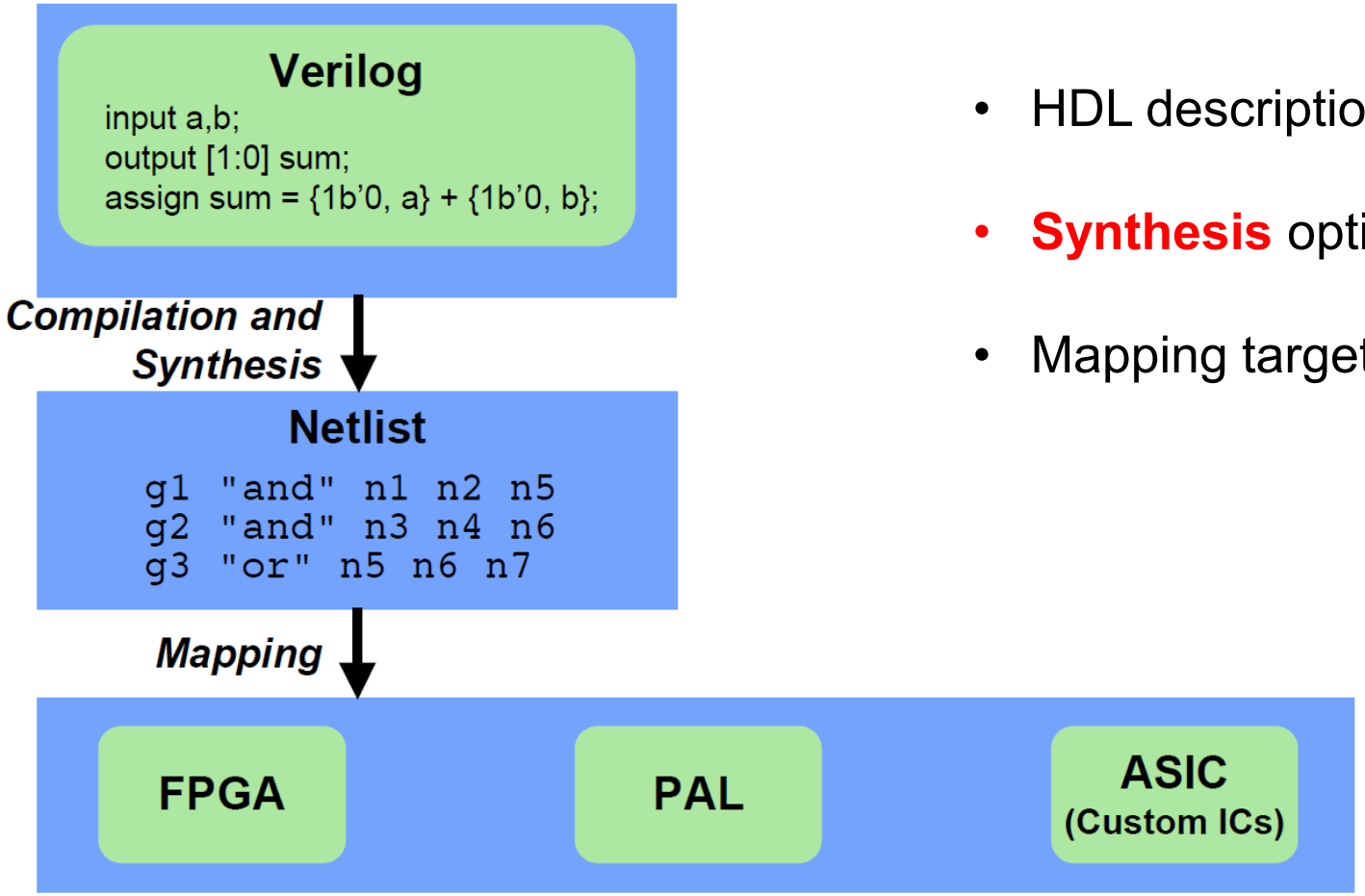
Other HDL..

- *ABEL (Advanced Boolean Expression Language)*
- *AHDL (Altera HDL)*
- *JHDL (Java HDL)*
- *MyHDL (Python-based HDL).*
- *Etc.*

V = Very-High-Speed Integrated Circuits

HDLs and Synthesis

- Hardware Description Language (HDL) is a convenient, device-independent representation of digital logic



- HDL description is compiled into a **netlist**
- **Synthesis** optimizes the logic
- Mapping targets a specific hardware platform

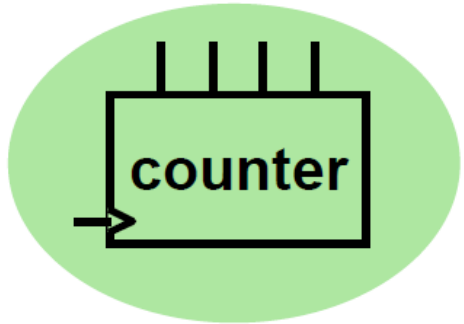
Synthesis and Mapping

- Infer macros: choose FPGA macros that efficiently implement various parts of the HDL code

```
...
always @ (posedge clk)
begin
    count <= count + 1;
end
...
```

HDL Code

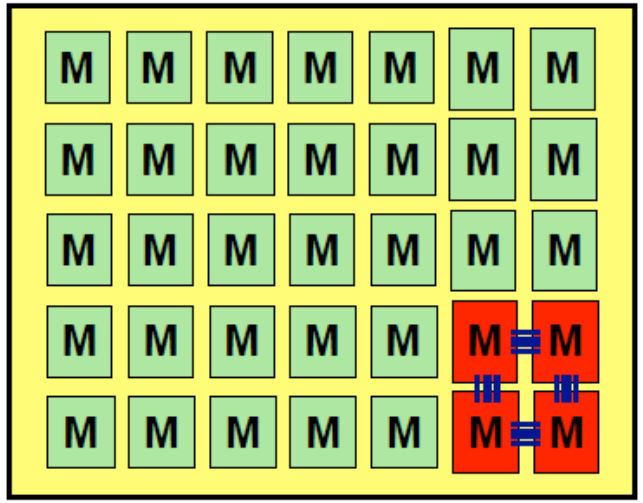
"This section of code looks like a counter. My FPGA has some of those..."



Inferred Macro

Synthesis and Mapping

- Place-and-Route: with area and/or speed in mind, choose the needed macros by location and route the interconnect

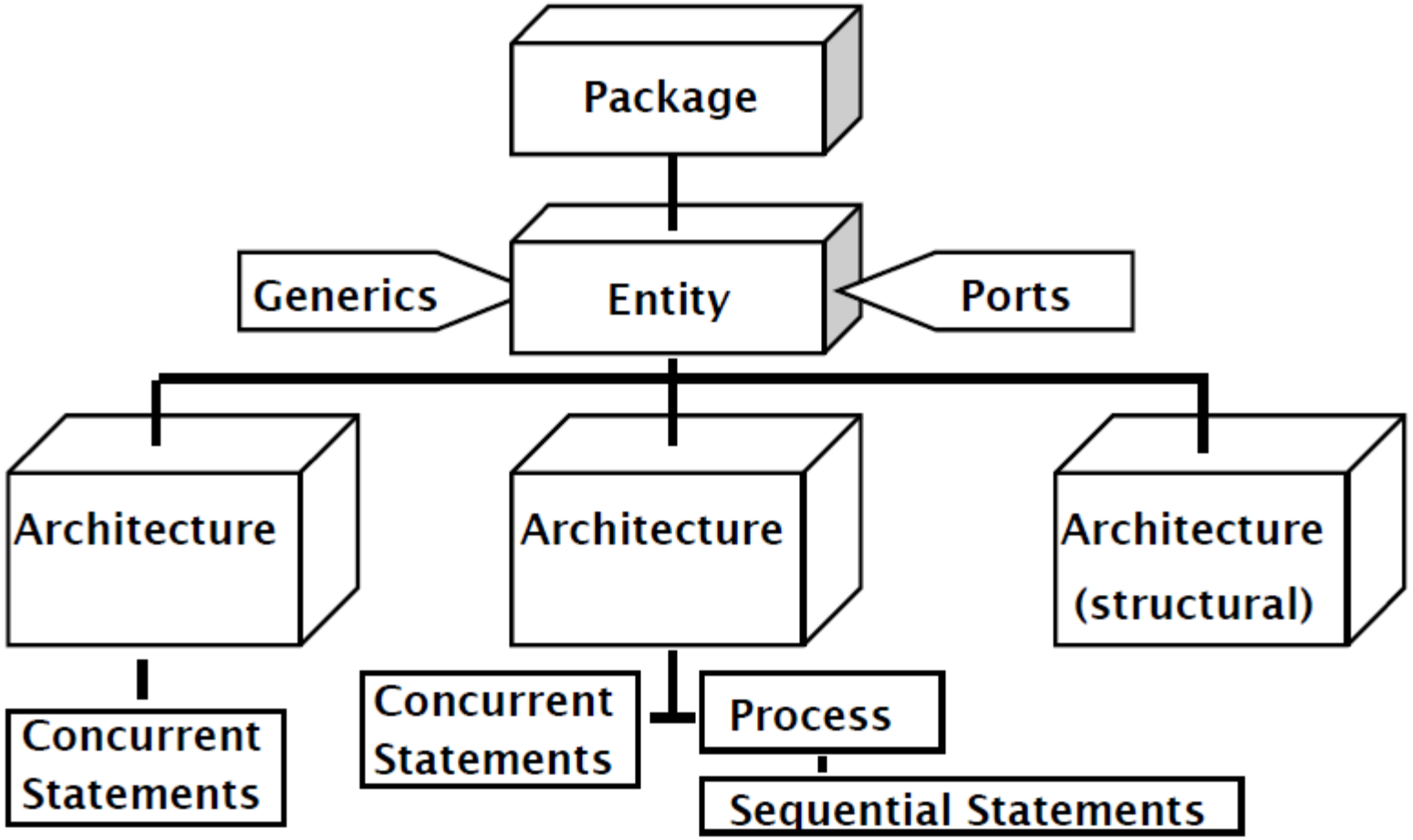


"This design only uses 10% of the FPGA. Let's use the macros in one corner to minimize the distance between blocks."

VHDL

- VHDL = VHSIC Hardware Description Language
  - VHSIC = Very High-Speed Integrated Circuit
- Developed in 1980's by US DOD
- ANSI/IEEE Standard 1076
- VHDL descriptions can be synthesized and implemented in programmable logic
  - NOTE: Synthesis tools can accept only subset of VHDL
- Some useful rules
  - VHDL is not case sensitive
  - Identifier must start with a letter
  - All statements end with a semi-colon
  - Comments precede with (--)
  - "<= " - signal assignment
  - ":=" - variable assignment
  - Signals have a one-time value set associated to it at any time.

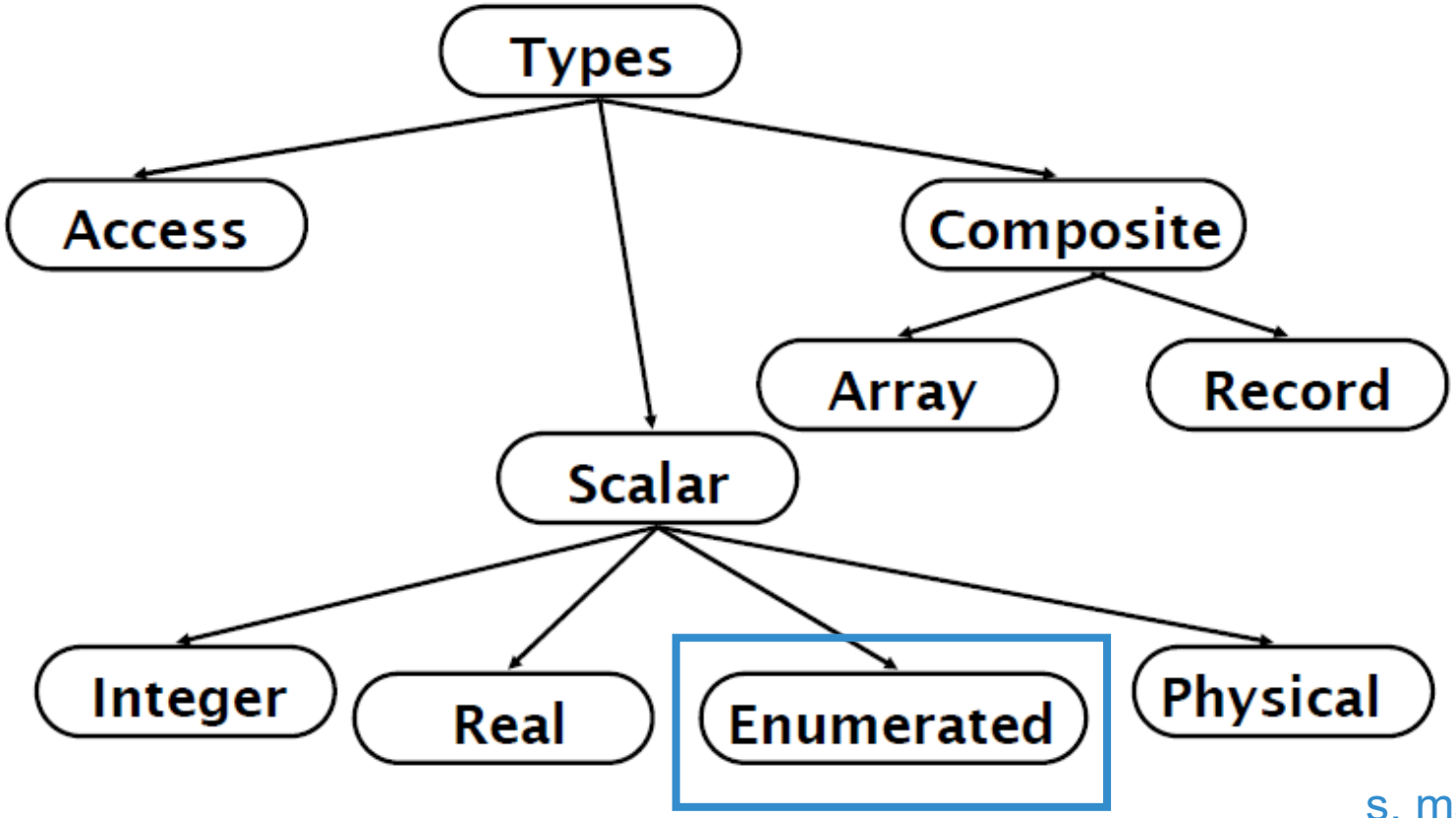
VHDL Hierarchy



VHDL structure 1: Library Packages

- **'std_logic_1164' package**
  - **std_logic (BIT)**
  - **std_logic_vector (MULTI-BITS)**
  - **integer**
- 'numeric_std' package
  - Mathematical operation & Signed/Unsigned
- 'textio' and 'std_logic_textio'
  - Used in testbench to write and read the data to the file
- Others
  - Fixed point & floating point…

Data Type



std_logic/std_logic_vector
boolean, bit, etc..

s, ms, us, ns, ps

Enumerated type

- type std_logic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-'); *Represent "Wire"*
  - 1: Logic 1
  - 0: Logic 0
  - U: Uninitialized
  - X: Unknown
  - Z: High-Z
  - W: Weak signal, (can't tell if 0 or 1)
  - L: Weak 0, pull down
  - H: Weak 1, pull up
  - -: Don't care
- type std_ulogic is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
- type state is (STATE1, STATE2, STATE 3,..);
- type boolean is (false,true); / type bit is ('0', '1');

Signal assignments

- constant a: integer := 523;

- signal b: std_logic_vector(11 downto 0);

```
b <= “000000010010”;
b <= B”000000010010”;
b <= B”0000_0001_0010”;
b <= X”012”;
b <= O”0022”;
```

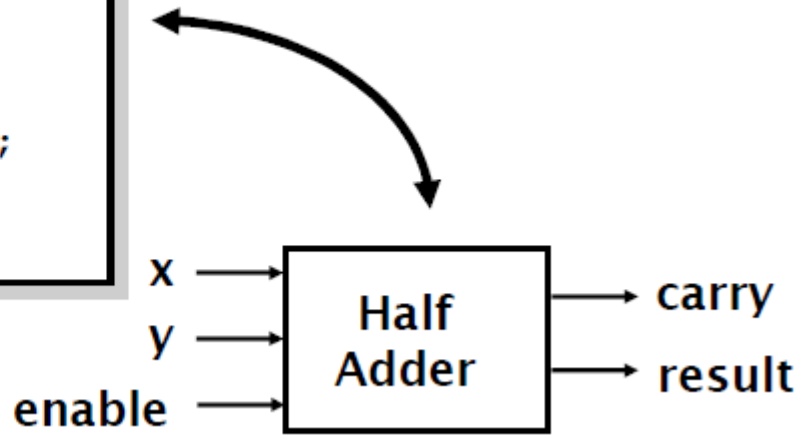
Built-in operators

- Logic operators
  - AND, OR, NAND, NOR, XOR, XNOR (XNOR in VHDL'93 only!!)
- Relational operators
  - =, /=, <, <=, >, >=
- Addition operators
  - +, -, &
- Multiplication operators (DON'T USE IN THIS TERM PROJECT)
  - *, /, mod, rem
- Miscellaneous operators
  - **, abs, not

VHDL structure 2: Entity Declaration

- An entity declaration describes the interface of the component.
- PORT clause indicates input and output ports.
- An entity can be thought of as a symbol for a component.

```
ENTITY half_adder IS

    PORT( x, y, enable: IN bit;
          carry, result: OUT bit);

END half_adder;
```



VHDL structure 3: Port Declaration

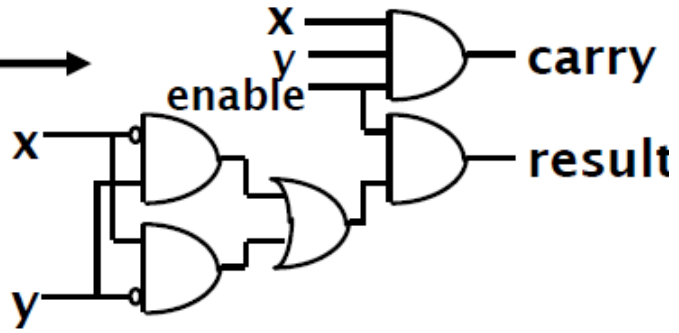
- PORT declaration establishes the interface of the object to the outside world.
- Three parts of the PORT declaration
  - Name
    - Any identifier that is not a reserved word.
  - Mode
    - In, Out, Inout, Buffer
  - Data type
    - Any declared or predefined datatype.
- Sample PORT declaration syntax:

```
ENTITY test IS
    PORT( name : mode data_type);
END test;
```

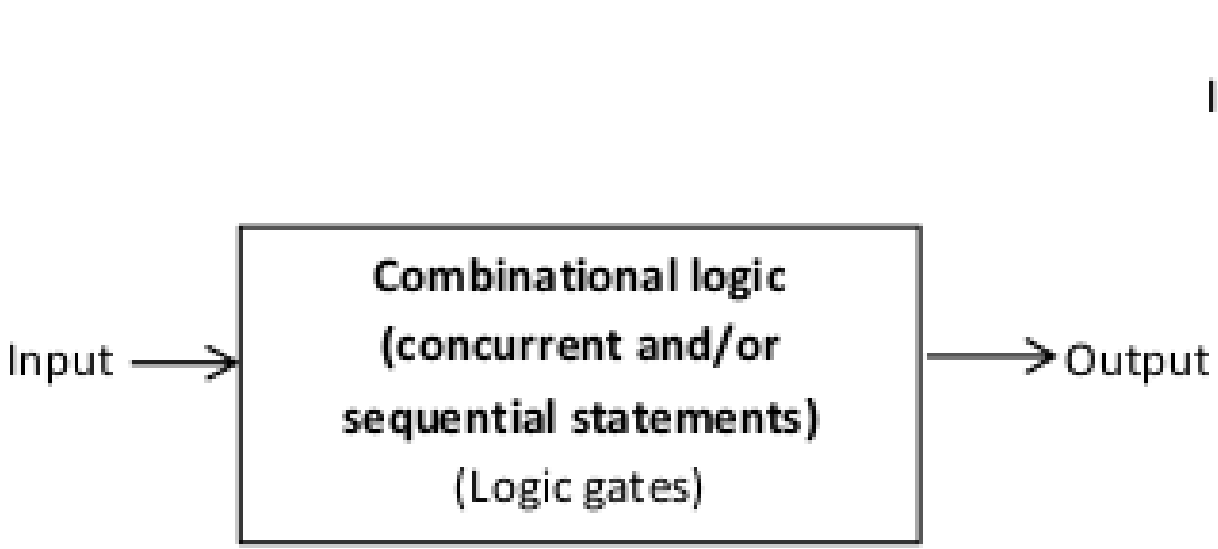
VHDL structure 4: Architecture Declaration

- Architecture declarations describe the operation of the component.
- Many architectures may exist for one entity, but only one may be active at a time.
- An architecture is similar to a schematic of the component.

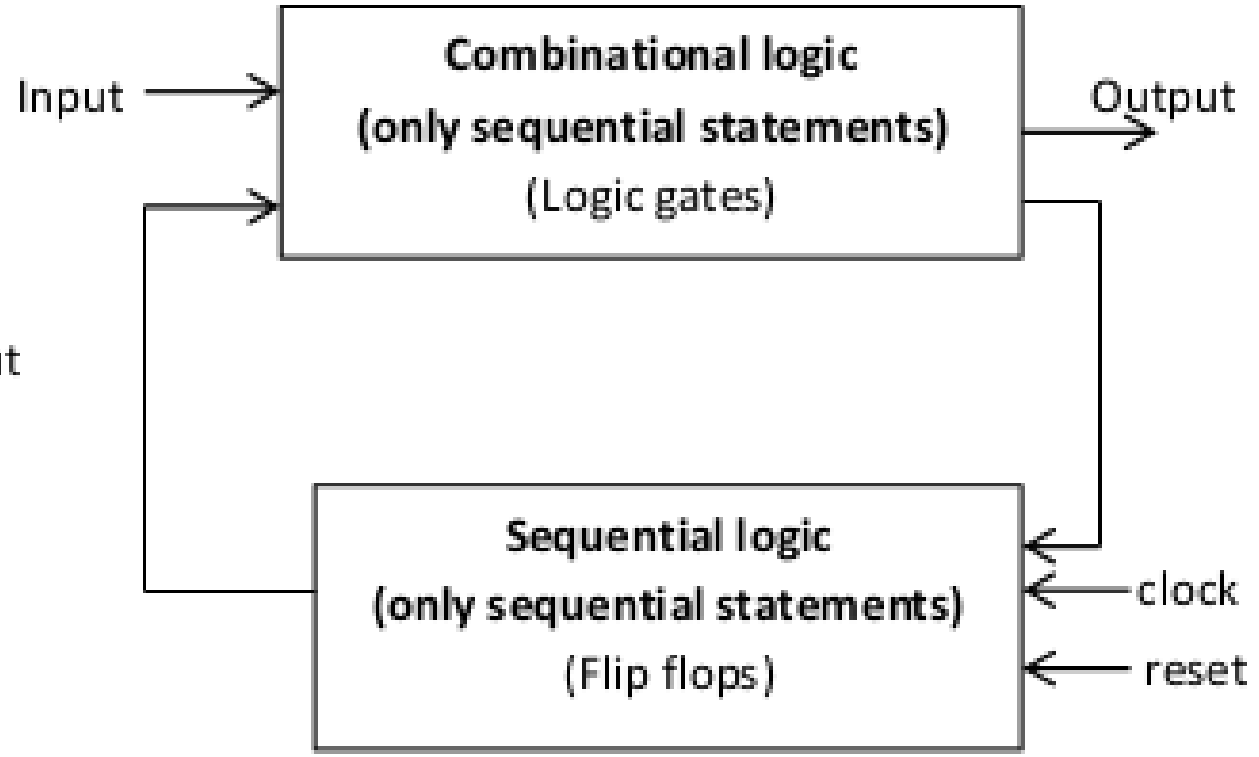
```
ARCHITECTURE behave OF half_adder IS
BEGIN
    PROCESS (enable, x, y)
    BEGIN
      IF (enable = '1') THEN
        result <= x XOR y;
        carry  <= x AND y;
      ELSE
        carry  <=  '0';
        result <=  '0';
      END IF;
    END PROCESS;
END behave;
```



Combinational logic vs Sequential Logic



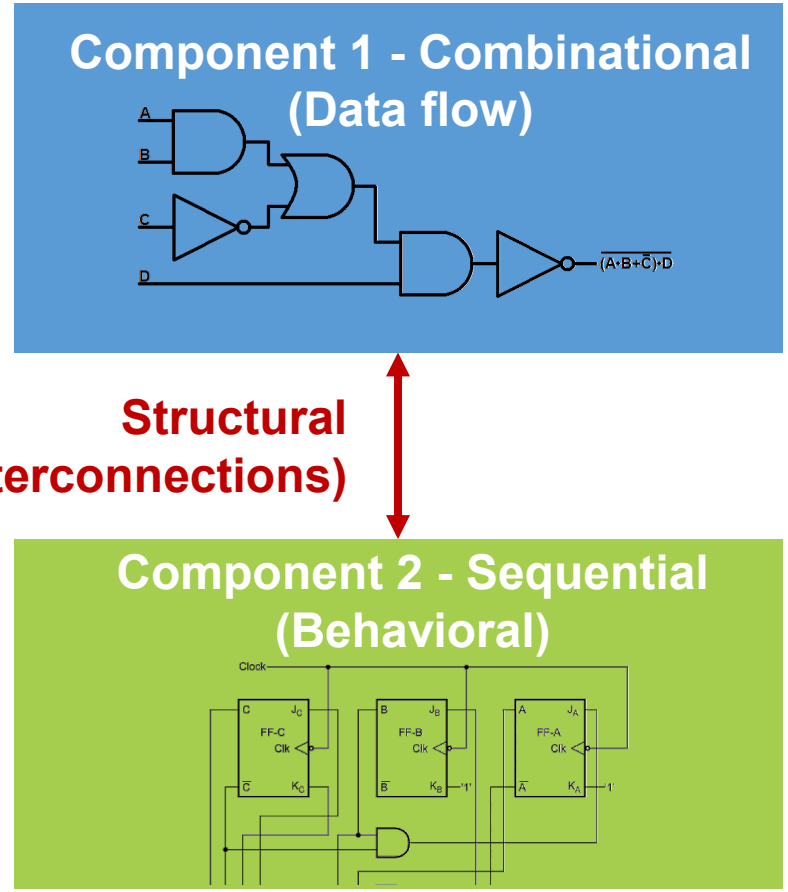
a) Combination design



b) Sequential design

Modeling Styles

- There are three modeling styles:
  - Dataflow (Combinational)
    - LOGIC
  - Behavioral (Sequential)
    - Only PROCESS statement
  - Structural
    - Interconnection between components



Modeling Styles

- Register-Transfer Level (RTL) Design
  - When designing digital integrated circuits with a HDL, the designs are usually engineered at a higher level of abstraction than transistor or logic gate level
  - But, a gate-level logic implementation (RTL design) is sometimes preferred.
  - This level describes the logic in terms of registers and the Boolean equations for the combinational logic between registers
  - For a combinational system there are no registers and the RTL logic consists only of combinational logic

Modeling Styles

Design	Statement	VHDL
Sequential (Flip-flops and Logic gates)	Sequential statements only	if
		case
		loop
		wait
Combinational (Logic gates Only)	Concurrent and Sequential	not/and/or/nand/nor/xor/xnor
		when-else
		with-select
		generate

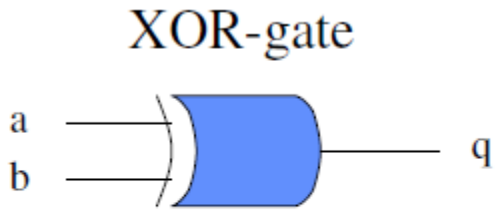
Concurrent vs Sequential Statements in VHDL

- Two different types of execution: sequential and concurrent.
- Different types of execution are useful for modeling of real hardware.
  - Supports various levels of abstraction.

- Sequential statements view hardware from a “programmer” approach.

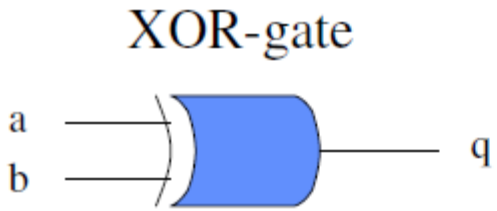
- Concurrent statements are order-independent and asynchronous.

Example: XOR-gate in Sequential Style




```
process(a,b)
begin
        if (a/=b) then
                q <= '1';
        else
                q <= '0';
        end if;
end process;
```

Example: XOR-gate in Dataflow Style

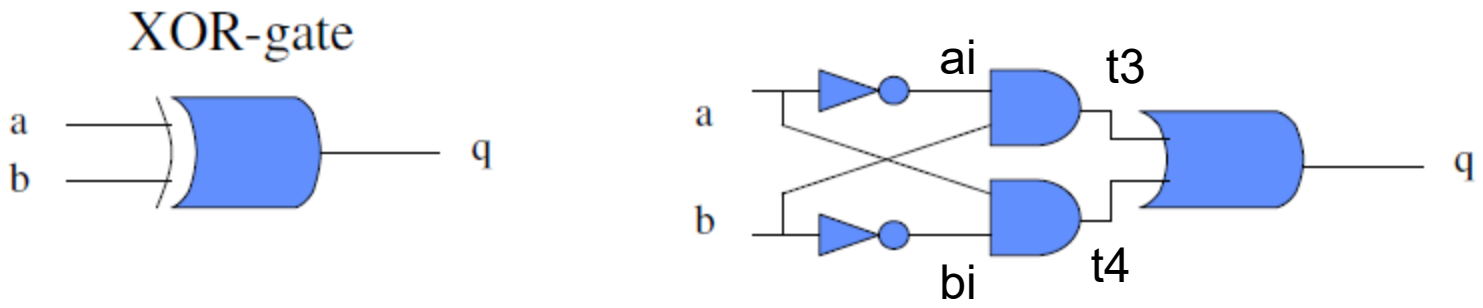



```
q <= a xor b;
```

Or in behavioral data flow style:

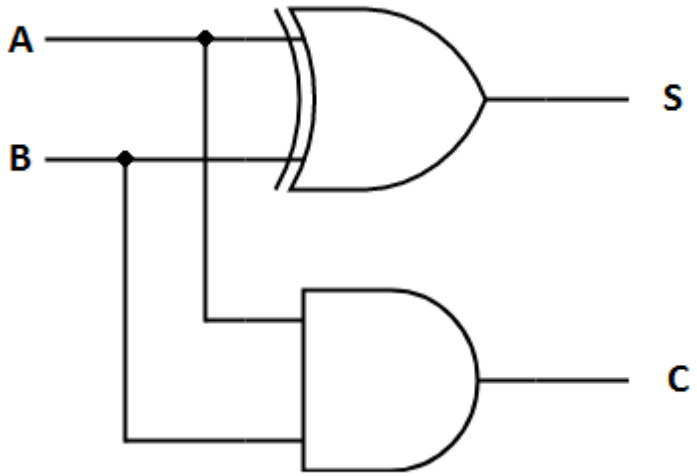
```
q <= '1' when a/=b else '0';
```

Example: XOR-gate in Structural Style



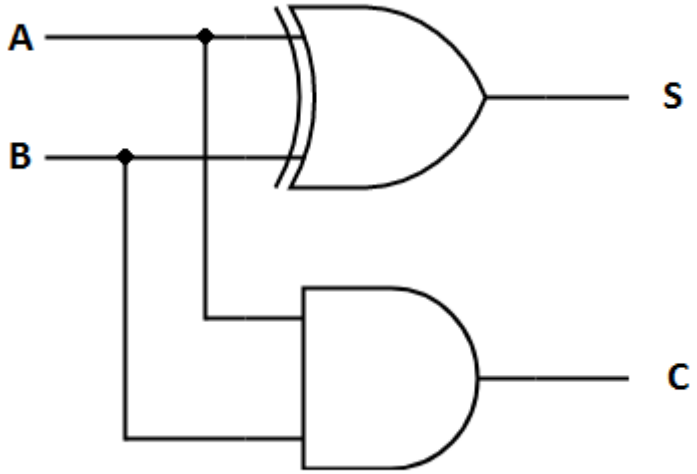

```
u1: inverter port map (a,ai);

u2: inverter port map (b,bi);

u3: and_gate port map (ai,b,t3);

u4: and_gate port map (bi,a,t4);

u5: or_gate port map (t3,t4,q);
```

Example: Half-adder in Sequential Style



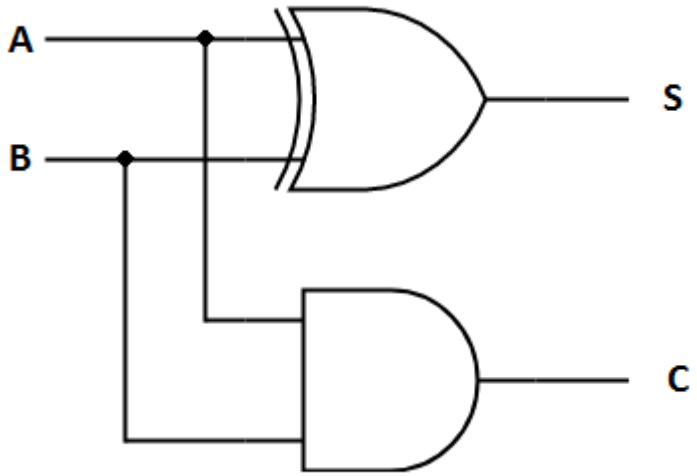

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity half_adder is
  port (a, b: in std_logic;
    sum, carry_out: out std_logic);
  end half_adder;

architecture behavior of half_adder is
  begin
    ha: process (a, b)
    begin
      if a = '1' then
        sum <= not b;
        carry_out <= b;
      else
        sum <= b;
        carry_out <= '0';
      end if;
    end process ha;

end behavior;
```

Example: XOR-gate in Dataflow Style




```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity half_adder is
  port (a, b: in std_logic;
    sum, carry_out: out std_logic);
  end half_adder;

architecture dataflow of half_adder is
  begin
  sum <= a xor b;
  carry_out <= a and b;
end dataflow;
```

Example: XOR-gate in Structural Style




```
u1: xor_gate port map (a, b, sum);
u2: and_gate port map (a, b, carry_out);
```

```
library ieee;
use ieee.std_logic_1164.all;

entity half_adder is                        -- Entity declaration for half adder
  port (a, b: in std_logic;
    sum, carry_out: out std_logic);
end half_adder;

architecture structure of half_adder is     -- Architecture body for half adder

  component xor_gate                        -- xor component declaration
    port (i1, i2: in std_logic;
      o1: out std_logic);
  end component;

  component and_gate                        -- and component declaration
    port (i1, i2: in std_logic;
      o1: out std_logic);
  end component;

begin
   u1: xor_gate port map (i1 => a, i2 => b, o1 => sum);
   u2: and_gate port map (i1 => a, i2 => b, o1 => carry_out);
-- We can also use Positional Association
--    => u1: xor_gate port map (a, b, sum);
--    => u2: and_gate port map (a, b, carry_out);
end structure;
```

Helpful websites

FPGA designs with VHDL

- https://vhdlguide.readthedocs.io/en/latest/index.html

VHDL Tutorial, University of Pennsylvania (Jan Van der Spiegel)

- https://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html