(Micro) processor, computer, controller

- ***CPU***: is a unit that fetches and processes a set of general-purpose instructions.

- ***Microprocessor***: is a CPU on a single chip. It *may* also have other units (e.g. caches, floating point processing arithmetic faster processing of instructions. (Intel 4004)

- ***Microcomputer***: when a microprocessor + I/O + memory+ etc are put together to form a small computer for applications like data collection, or control application.

- ***Microcontroller (MCU)***: a microcomputer on a single chip. It brings together the microprocessor core and a rich collection of peripherals and I/O capability. (TMS1000)

The Processing Unit – CMPE310 Review

Microcontroller (MCU)*

- A microcontroller is a single chip unit which, though having limited computational capabilities, possesses enhanced input-output capabilities and a number of on-chip functional units
- Particularly suited for use in embedded systems for real-time control applications with on-chip program memory and devices
- Common peripherals include serial communication devices, timers, counters, pulse width modulators, analog-to-digital and digital-to-analog convertors
- Enables single-chip system implementation and hence smaller and lower-cost products
- Examples: Motorola 68HC11xx, HC12xx, HC16xx, Intel 8051, 80251, PIC 16F84, PIC18, ARM9, ARM7, *Atmel AVR* etc.

* Courtesy of Dr. Patel, CMPE 310, Introduction lecture, slide 8.

Wikipedia definition:

A microcontroller (sometimes abbreviated µC, uC or MCU) is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals. Program memory in the form of NOR flash or OTP ROM is also often included on chip, as well as a typically small amount of RAM. Microcontrollers are designed for embedded applications, in contrast to the microprocessors used in personal computers or other general purpose applications.

Microcontrollers are used in automatically controlled products and devices, such as automobile engine control systems, implantable medical devices, remote controls, office machines, appliances, power tools, toys and other embedded systems. By reducing the size and cost compared to a design that uses a separate microprocessor, memory, and input/output devices, microcontrollers make it economical to digitally control even more devices and processes. Mixed signal microcontrollers are common, integrating analog components needed to control non-digital electronic systems.

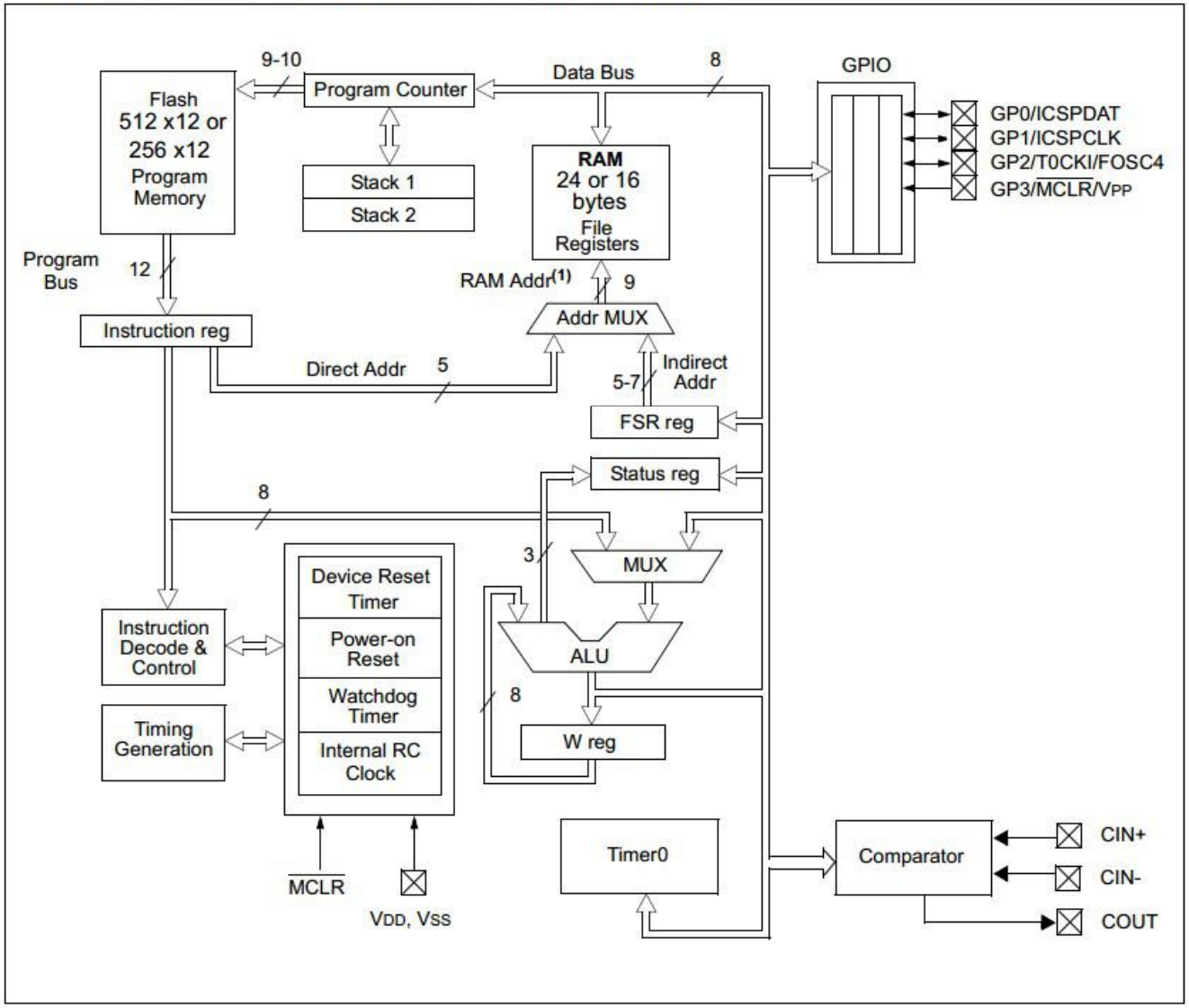
Some microcontrollers may use four-bit words and operate at clock rate frequencies as low as 4 kHz, for low power consumption (single-digit milliwatts or microwatts). They will generally have the ability to retain functionality while waiting for an event such as a button press or other interrupt; power consumption while sleeping (CPU clock and most peripherals off) may be just nanowatts, making many of them well suited for long lasting battery applications. Other microcontrollers may serve performance-critical roles, where they may need to act more like a digital signal processor (DSP), with higher clock speeds and power consumption.


Microchip - 8-bit **Microcontroller**
$0.68
Mouser Electr...


MICROCHIP
PIC10F206-I/P
$0.38
Newark

Microcontroller Example Block Diagram

FIGURE 3-2: PIC10F204/206 BLOCK DIAGRAM



Microcontroller Example Instruction Set

http://www.mouser.com/ds/2/268/41239A-80396.pdf

TABLE 10-1: OPCODE FIELD DESCRIPTIONS

Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 0 or 1) The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
d	Destination select; d = 0 (store result in W) d = 1 (store result in file register 'f') Default is d = 1
label	Label name
TOS	Top-of-Stack
PC	Program Counter
WDT	Watchdog Timer counter
TO	Time-out bit
PD	Power-down bit
dest	Destination, either the W register or the specified register file location
[ ]	Options
( )	Contents
→	Assigned to
< >	Register bit field
∈	In the set of
italics	User defined term (font is courier)

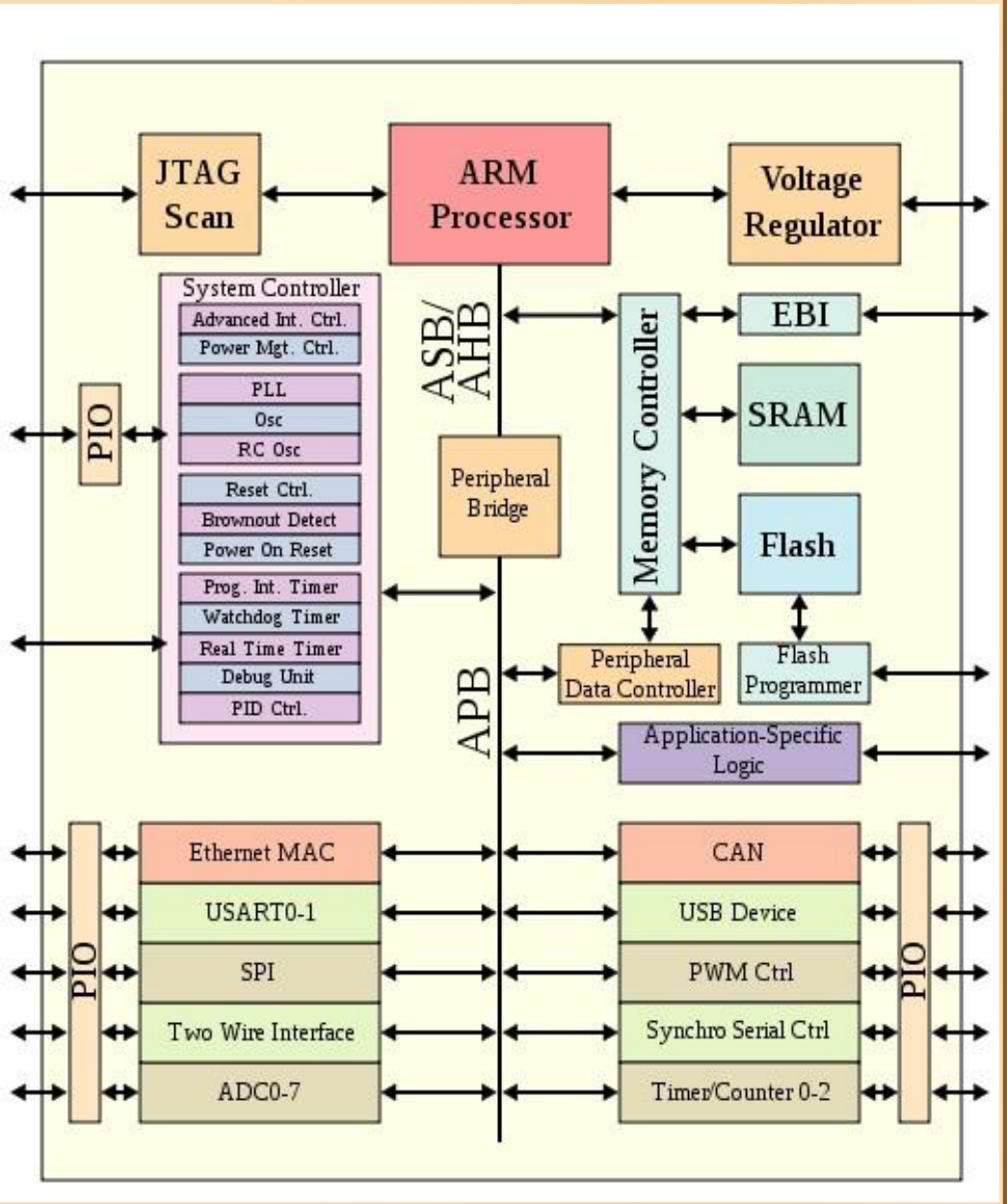
TABLE 10-2: INSTRUCTION SET SUMMARY

Mnemonic, Operands		Description	Cycles	12-Bit Opcode MSb		LSb	Status Affected
ADDWF	f, d	Add W and f	1	0001	11df	ffff	C, DC, Z
ANDWF	f, d	AND W with f	1	0001	01df	ffff	Z
CLRF	f	Clear f	1	0000	011f	ffff	Z
CLRW	–	Clear W	1	0000	0100	0000	Z
COMF	f, d	Complement f	1	0010	01df	ffff	Z
DECF	f, d	Decrement f	1	0000	11df	ffff	Z
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	0010	11df	ffff	None
INCF	f, d	Increment f	1	0010	10df	ffff	Z
INCFSZ	f, d	Increment f, Skip if 0	1(2)	0011	11df	ffff	None
IORWF	f, d	Inclusive OR W with f	1	0001	00df	ffff	Z
MOVF	f, d	Move f	1	0010	00df	ffff	Z
MOVWF	f	Move W to f	1	0000	001f	ffff	None
NOP	–	No Operation	1	0000	0000	0000	None
RLF	f, d	Rotate left f through Carry	1	0011	01df	ffff	C
RRF	f, d	Rotate right f through Carry	1	0011	00df	ffff	C
SUBWF	f, d	Subtract W from f	1	0000	10df	ffff	C, DC, Z
SWAPF	f, d	Swap f	1	0011	10df	ffff	None
XORWF	f, d	Exclusive OR W with f	1	0001	10df	ffff	Z
		BIT-ORIENTED FILE REGISTER OPERATIONS					
BCF	f, b	Bit Clear f	1	0100	bbbf	ffff	None
BSF	f, b	Bit Set f	1	0101	bbbf	ffff	None
BTFSC	f, b	Bit Test f, Skip if Clear	1(2)	0110	bbbf	ffff	None
BTFSS	f, b	Bit Test f, Skip if Set	1(2)	0111	bbbf	ffff	None
		LITERAL AND CONTROL OPERATIONS					
ANDLW	k	AND literal with W	1	1110	kkkk	kkkk	Z
CALL	k	Call Subroutine	2	1001	kkkk	kkkk	None
CLRWDT		Clear Watchdog Timer	1	0000	0000	0100	$\overline{TO}$, $\overline{PD}$
GOTO	k	Unconditional branch	2	101k	kkkk	kkkk	None
IORLW	k	Inclusive OR literal with W	1	1101	kkkk	kkkk	Z
MOVLW	k	Move literal to W	1	1100	kkkk	kkkk	None
OPTION	–	Load Option register	1	0000	0000	0010	None
RETLW	k	Return, place Literal in W	2	1000	kkkk	kkkk	None
SLEEP	–	Go into Standby mode	1	0000	0000	0011	$\overline{TO}$, $\overline{PD}$
TRIS	f	Load TRIS register	1	0000	0000	0fff	None
XORLW	k	Exclusive OR literal to W	1	1111	kkkk	kkkk	Z

Wikipedia ARM entry header:

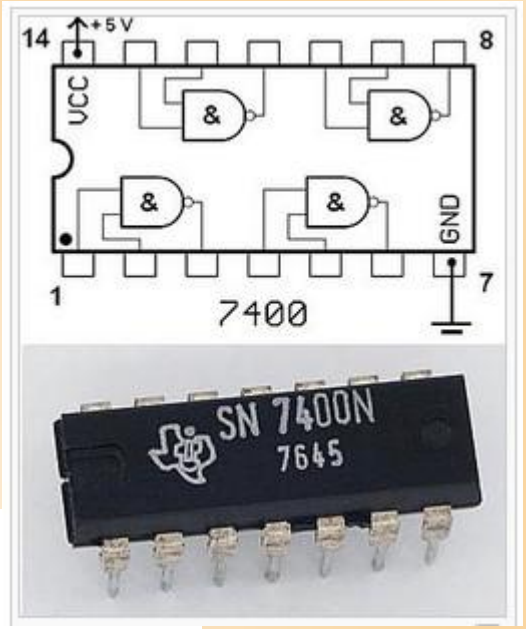
ARM is a family of instruction set architectures for computer processors based on a reduced instruction set computing (RISC) architecture developed by British company ARM Holdings.

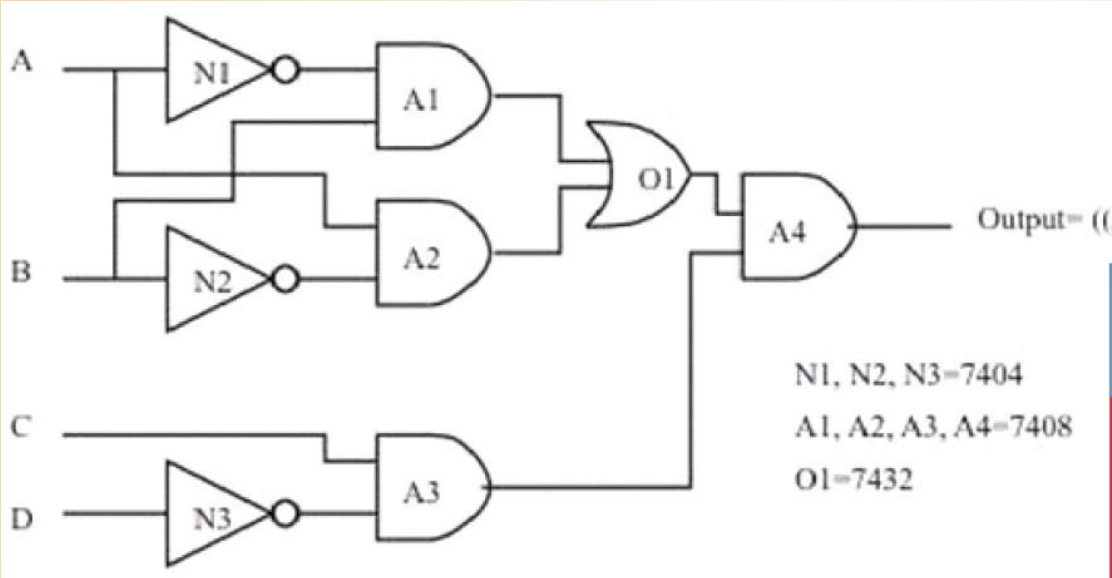
A RISC-based computer design approach means ARM processors require significantly fewer transistors than typical CISC x86 processors in most personal computers. This approach reduces costs, heat and power use. These are desirable traits for light, portable, battery-powered devices—including smartphones, laptops, tablet and notepad computers, and other embedded systems. A simpler design facilitates more efficient multi-core CPUs and higher core counts at lower cost, providing improved energy efficiency for servers



Alternatives for MCU

- ***Alt 1: discrete ICs*** Dedicated digital circuit from
  - Might use various ICs for various functions (AND, OR, XOR,..)
  - Example utilization: 57%

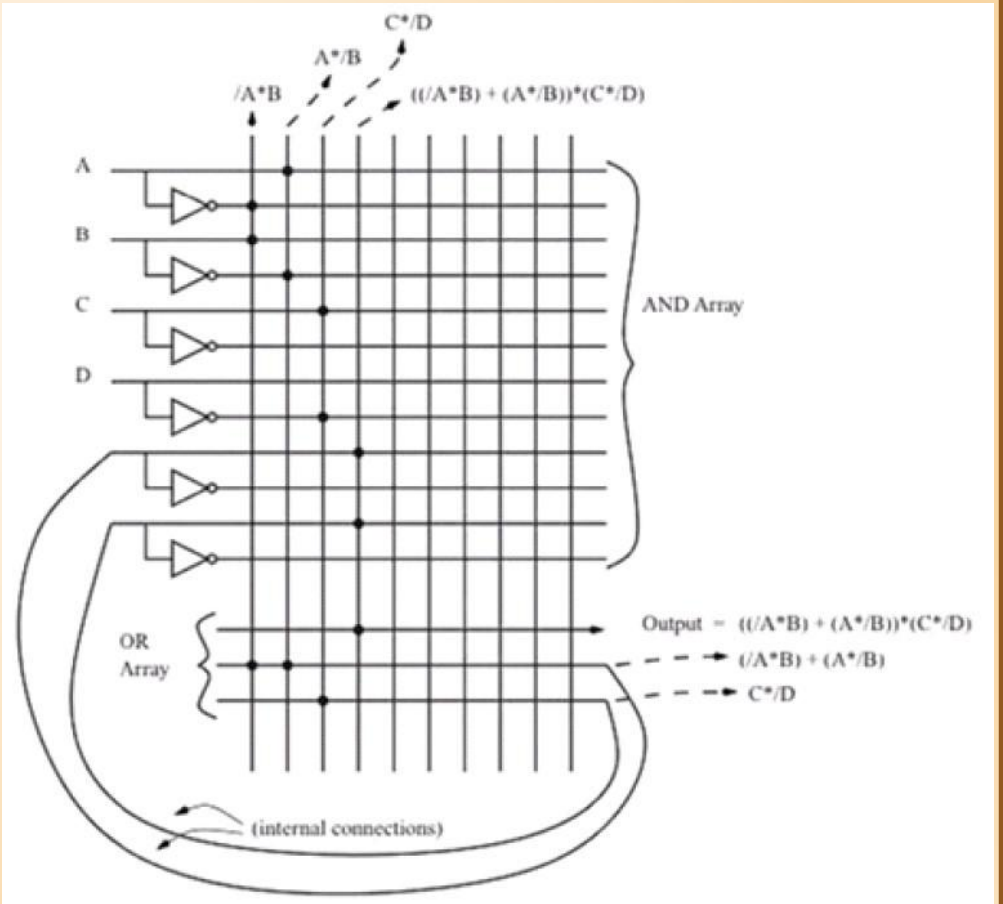




	IC	Gates	Used gates
Not	7404	6	3
AND	7408	4	
OR	7432	4	1
Sum	3	18	8

Alternatives for MCU

- **Alt 2: PLD(Programmable logic device)** PLD contains various logic functions. User selects which blocks (a fraction, not 100% utilized) is needed and how they are connected. Results in more compact system compared to dedicated digital circuit
  - One IC, 120 gates, we used 12 (8% utilization)
  - Actual 16L8 IC has more gates than 120

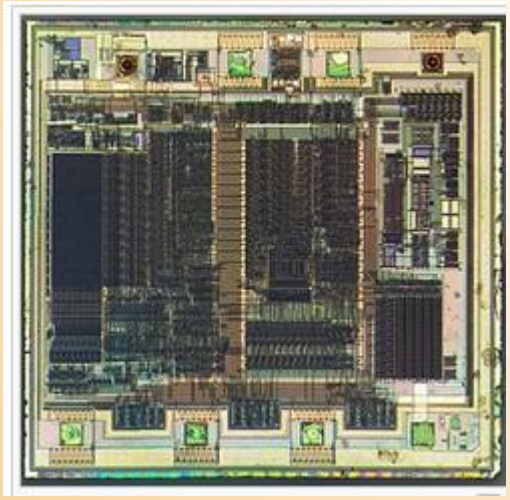


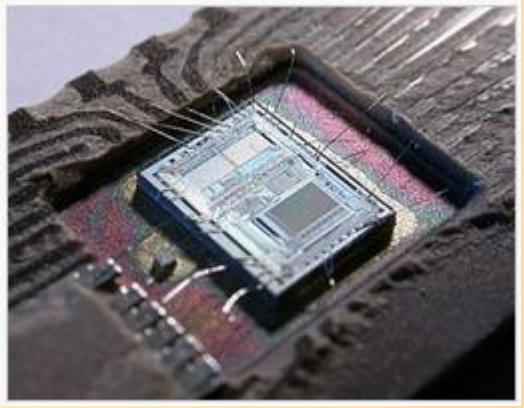
- **FPGAs** – see slides 11 - 18:

Alternatives for MCU

- **Alt 3: ASIC (Application Specific Integrated Circuit)s**

  Specific optimized implementation.
  - One IC, 12 gates, 100% utilization!!!
  - We probably are wasting a lot of die area on the chip level if we only implement this circuit.





Why MCU then?

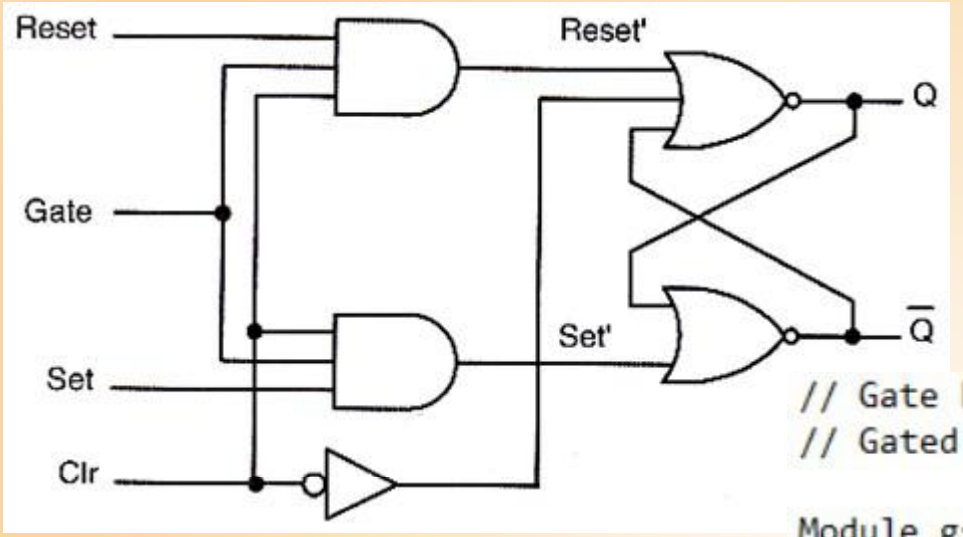
- Application development effort is limited to software development and validation (easier, faster and cheaper than Hardware design, test and verification)
  - Small overhead when upgrading the system with small changes
  - Inventory of small number of components
  - Non-Recurring Engineering (NRE) costs are amortized amongst all the users of a particular MCU architecture

Should we always use MCU?

- No, it depends to application. e.g. if high speed is needed, FPGA can be a choice - or even higher speed is needed, ASIC will be the choice.

  http://www.eetimes.com/design/industrial-control/4016917/A-tradeoff-between-microcontroller-DSP-FPGA-and-ASIC-technologies

Structural Verilog – appendix A in your text




```
// Gate Level Model
// Gated SR Latch with clear

Module gsrLatch(q, qnot, sg, rg, clr,enab);
        input sg, rg, clr, enab;
        output q, qnot;
        parameter delay0 = 2;

// Build the gating logic
        not n0(nclr, clr);
        and and0(rL, rg, clr, enab);
        and and1(sL, sg, clr, enag);

// Build the basic RS latch
        nor #delay0 n0(q, rL, nclr, qnot);
        nor #delay0 n1(qnot, sL, q);
endmodule
```

Behavioral Verilog – appendix A in your text

```
// Behavioral Level Model
// Gated SR Latch
module gsrLatch(q, qnot, s, r, clr, enab);
    input s, r, enab, clr;
    output q, qnot;

    reg q, qnot;

    always@ (~clr or enab)
    begin
        if(~clr)
        begin
            q = 1'b0;
            qnot = 1'b1;
        end

        else
        begin
            if (s & ~r)
            begin
                q <= s;
                qnot <= r;
            end
            else if (~s & r)
            begin
                q <= s;
                qnot <= r;
            end
        end
    end
endmodule
```

```
// Use two SR Latches in a master slave
// configuration to build a flip-flop

module srmsff(q, qnot, s, r, clk, clr);
    input s, r, clk, clr;
    output q, qnot;

    gsrLatch master(qm, qmnot, s, r, clr, clk);
    gsrLatch slave(q, qnot, qm, qmnot, clr, ~clk);
endmodule
```

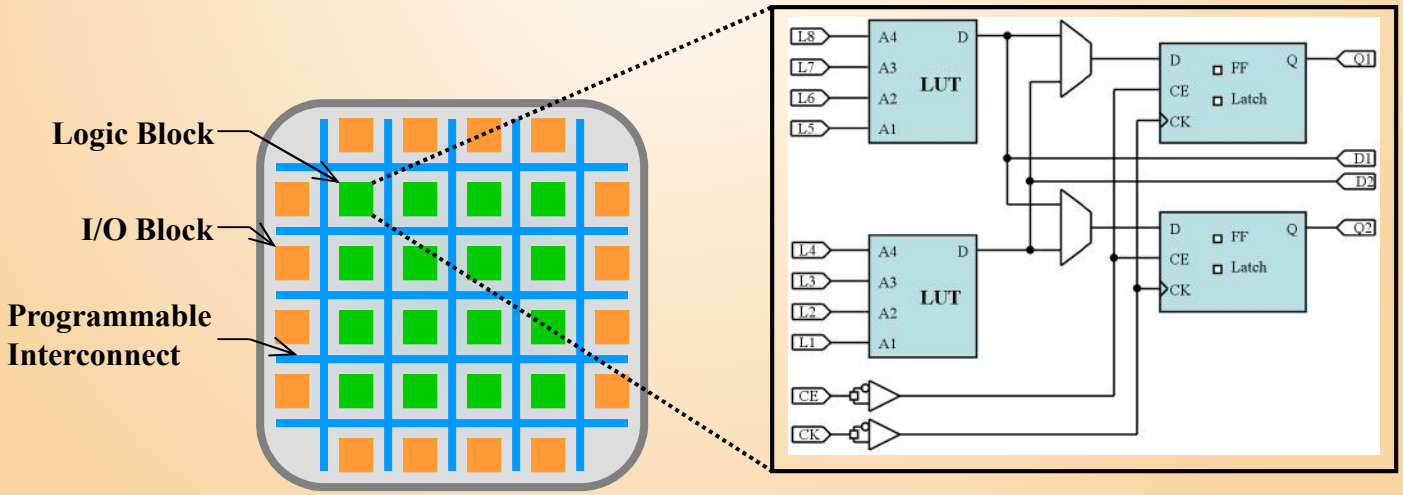
```
// Build a synchronous two bit binary up counter
// using master slave SR flip-flops

module TwoBitCntr(qA, qB, clr, clk);
    input clr, clk;
    output qA, qB;

    reg sA, rA;
    wire qA, qAnot, qB;

    always@(posedge clk)
    begin
        sA = qAnot & qB;
        rA = qA & qB;
    end

    srmsff FFB(qB, qBnot, qBnot, qB, clk, clr);
    srmsff FFA(qA, qAnot, sA, rA, clk, clr);
endmodule
```

Figure A.28 Behavioral Models of the S R Latch, Master–Slave Flip-Flop, and 2-Bit Binary Up Counter

FPGA Design Process

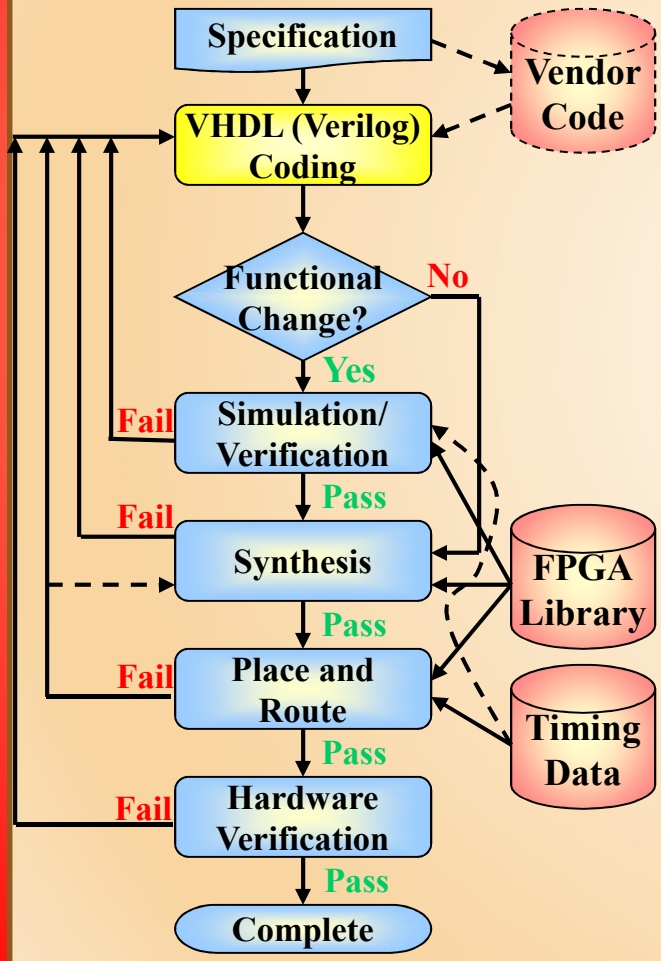
FPGAs may be fuse, RAM or flash-RAM based devices (fuses are programmed only once, RAM based allow multiple programming cycles)
There are three key parts of its structure:

- **I/O blocks** form a ring around the outer edge of the part. Each provides individually selectable input, output, or bi-directional access to the exterior pins.
- **Logic blocks** (CLBs – Configurable Logic Blocks) are typically comprised of a Logic Lookup Table (LUT) and a Flip-Flop. Additional logic features, like carry-look-ahead may be added.
- **Programmable Interconnect** is a set of logic enabled connections throughout the chip; and includes clocking blocks.

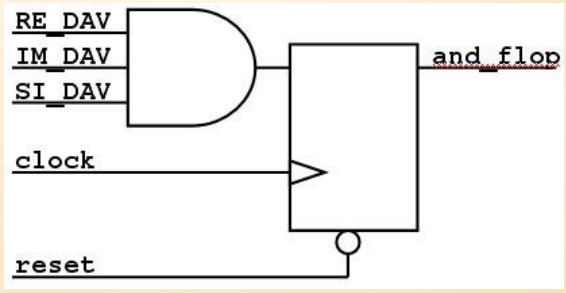


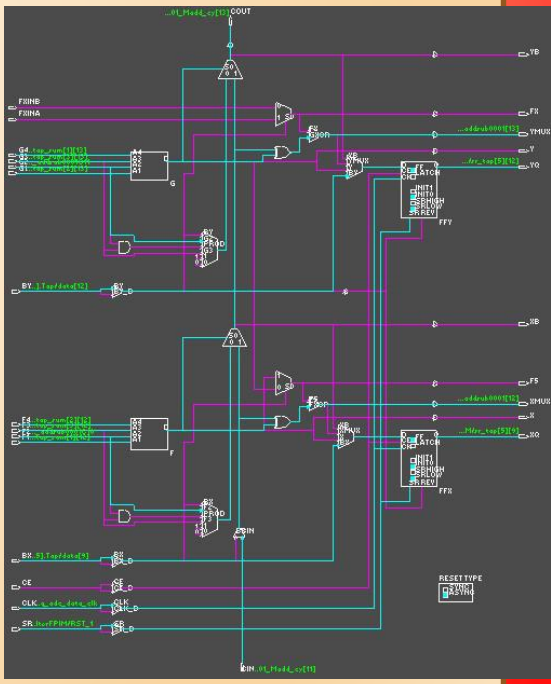
FPGA Design Process

VHDL Coding



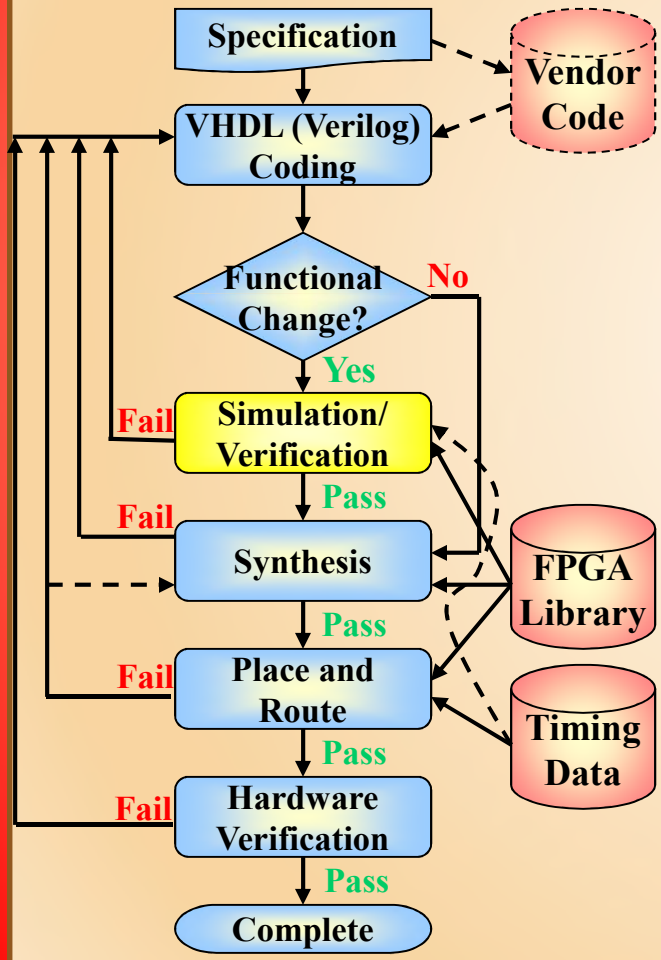

```
and_flop_example : process (clock, reset)
  begin
    if (reset = '0') then
      and_flop <= '0';
    elsif (rising_edge(clock)) then
      and_flop <= RE_DAV and IM_DAV and SI_DAV;
    end if;
  end process and_flop_example;
```

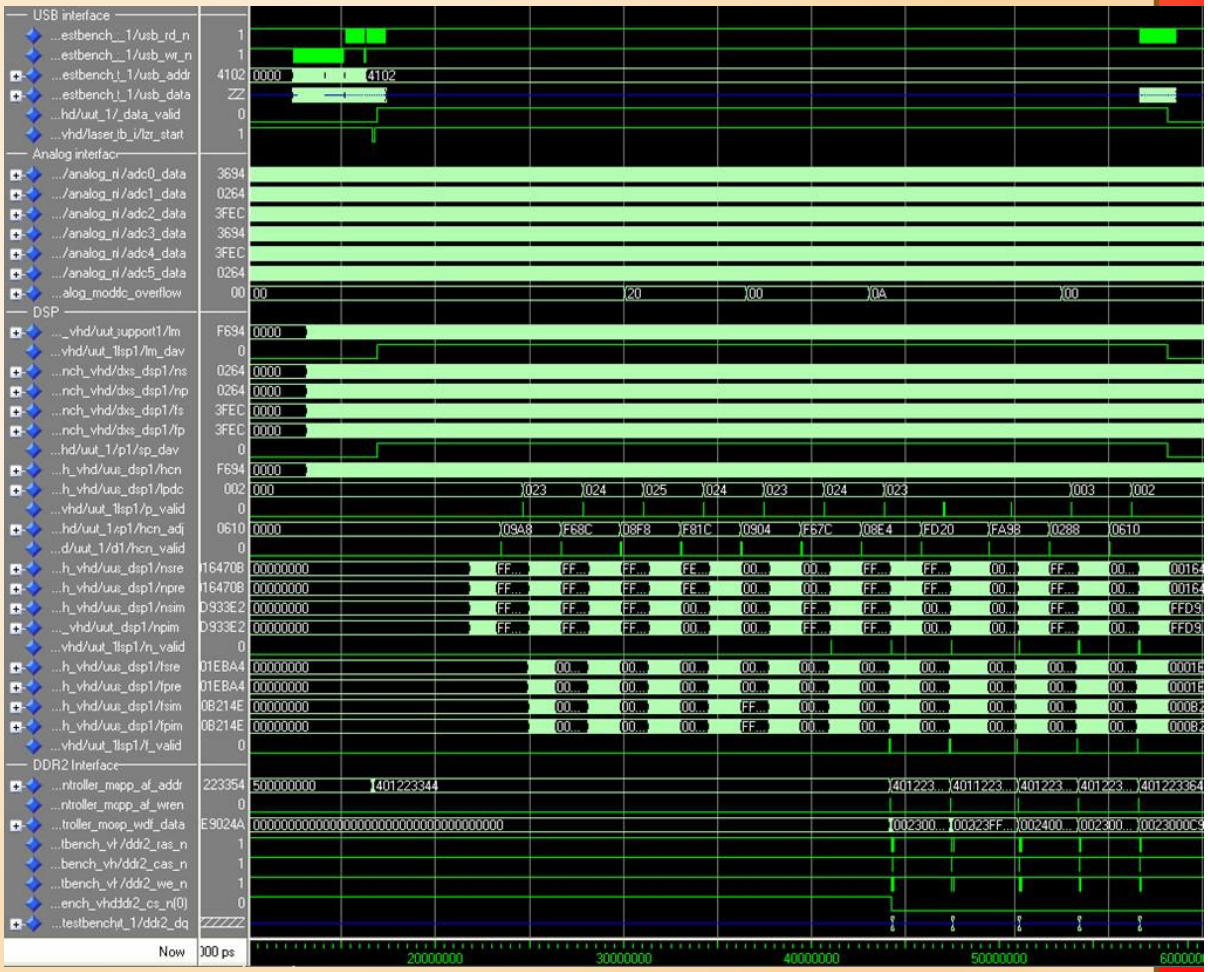




FPGA Design Process

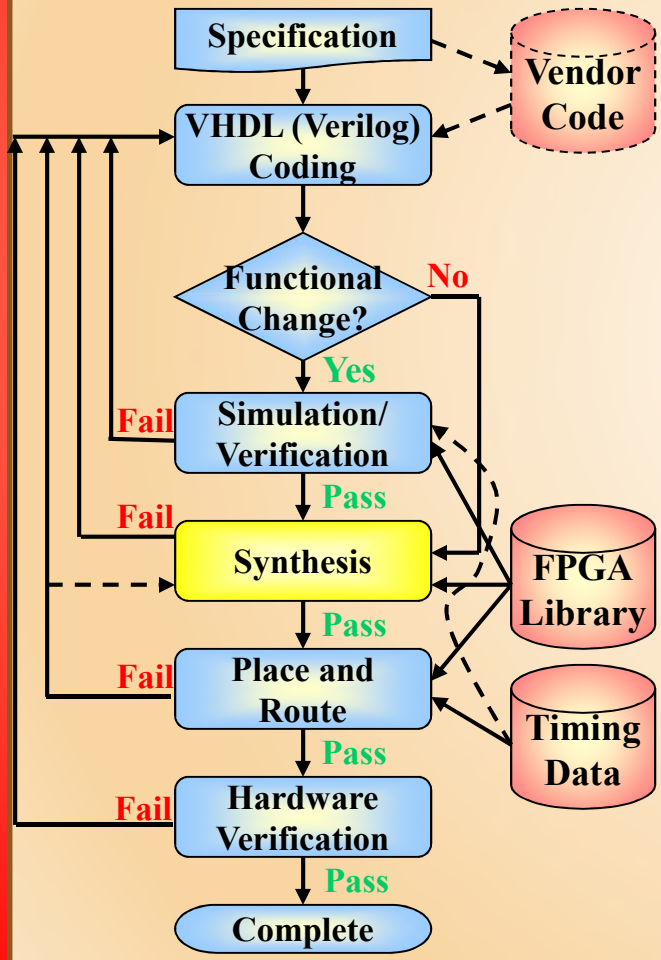
Simulation/ Verification

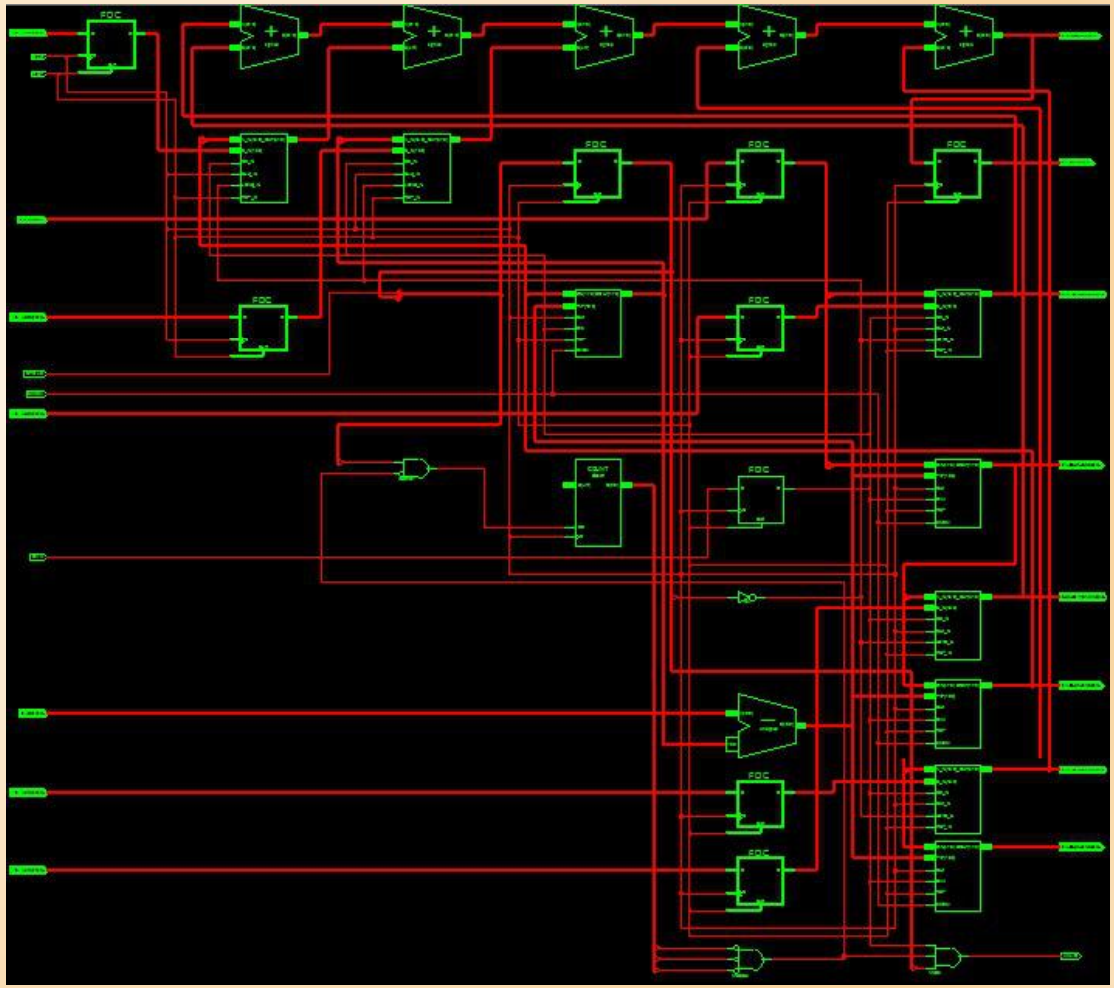




FPGA Design Process

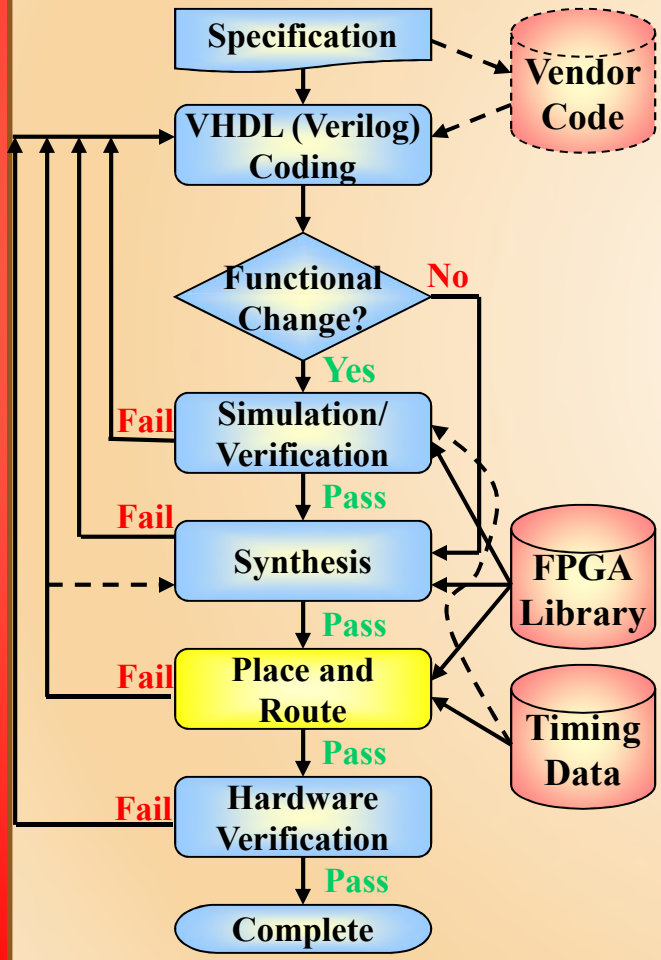
Synthesis: creating Electronics from Code.

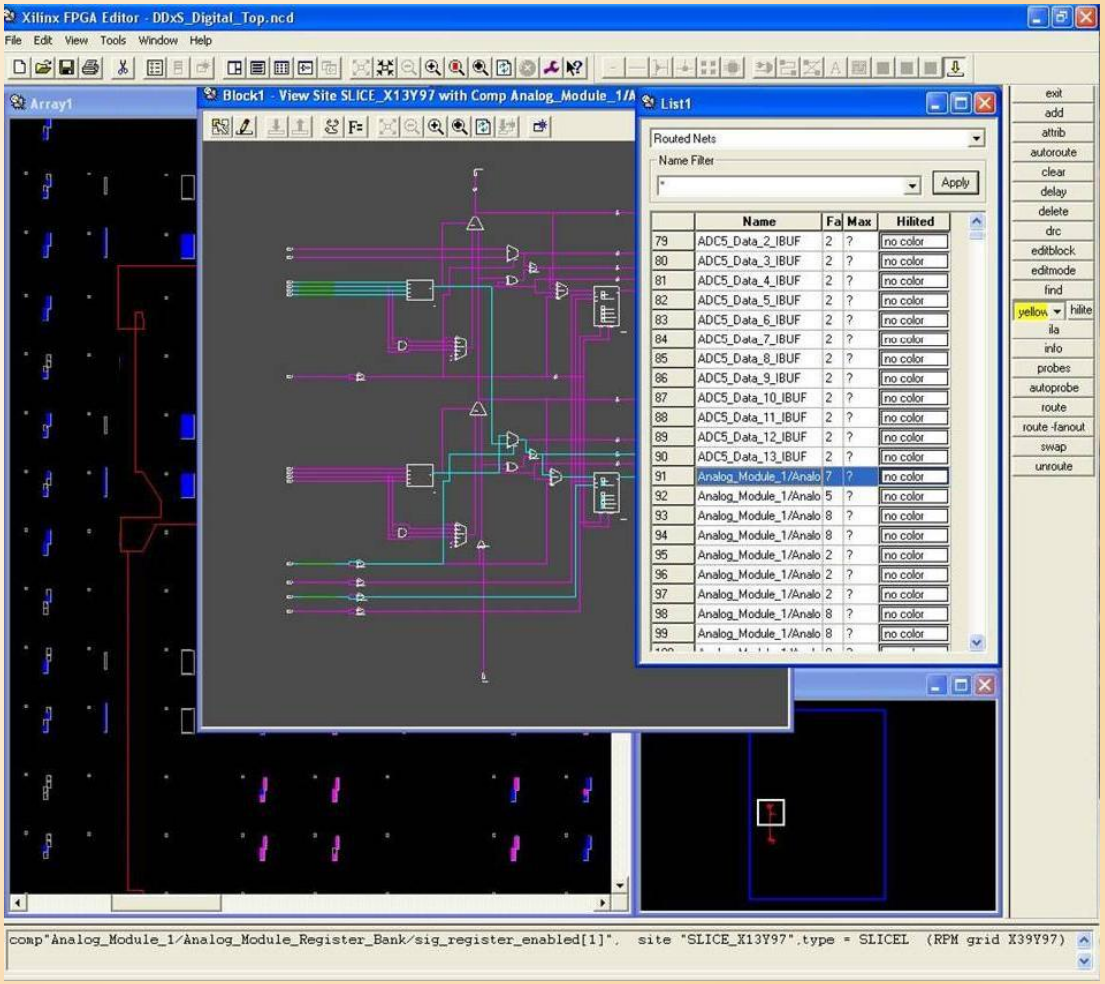




FPGA Design Process

Place and Route - Applying results to FPGA resources





FPGA Design Process

Hardware Verification: