TÉCNICO LISBOA

Projecto e Controlo em Lógica Digital

www.lip.pt/~pedjor/PCLD

- Microprocessors
- I/O
- The FPGA

Refs:
Cyclone II device Handbook, Altera corp.
Quartus II Handbook , Altera corp.
DE2 documentation
Verilog HDL, S. Palnitkar, Prentice Hall

Inspired on course 6.111 from MIT * Some support material taken from 6.111 (thank you)

Notes on our VGA

Colors module

```
module Colors(input hcount, vcount
              output [9:0]R, [9:0] G, [9:0] B);

Always @ (*)
              if (hcount < 20)
                            R=
                            G=
                            B=
.....
```

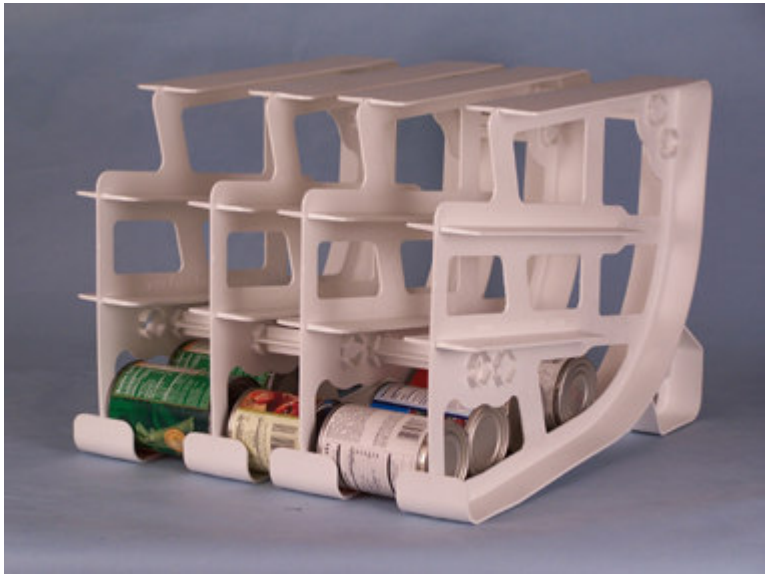
Imagine we have infinite RAM...

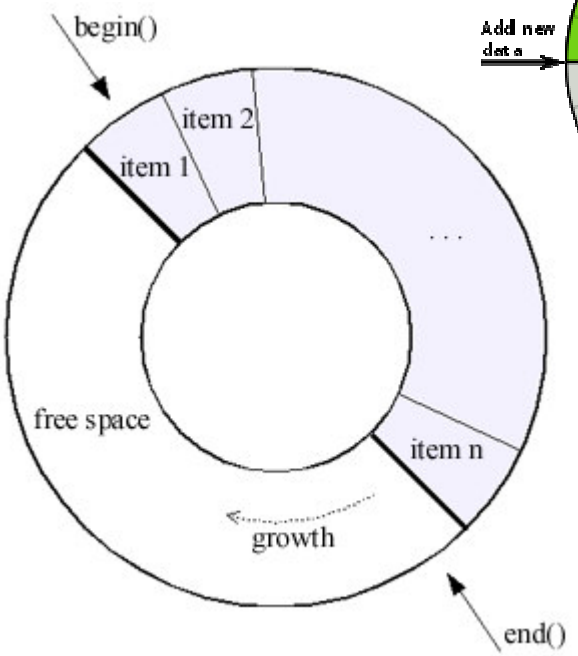
```
module Colors(input hcount, vcount
                                output [9:0]R, [9:0] G, [9:0] B);

Reg [9:0] Rmem [“2^20-1:0”]= {0,0,0,0,1,1,1,,.........}

assign R=Rmem[ {vcount , hcount} ];
```

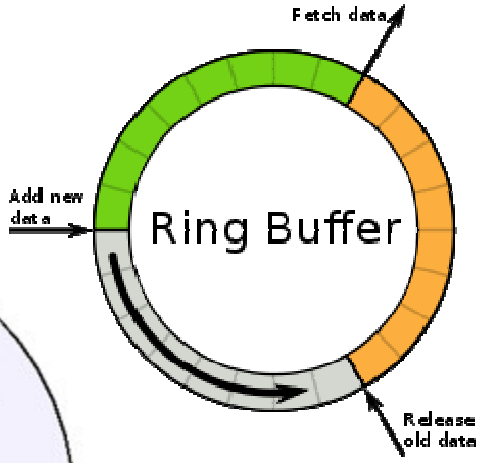
Or, in our application...

```
Reg [7:0] pos [7:0];   //this can be implemented as RAM
Pos={0,0,1,2,3,4,5,4,3,2,1,2,3,4,5,6,7,6,5,4,3,2,1,2,3,4,5,6.....}

If (hcount < 256 && vcount <256)  //define a square to draw data
     if ( vcount == pos [ hcount ]
                RGB = “white”
else “put black”
```

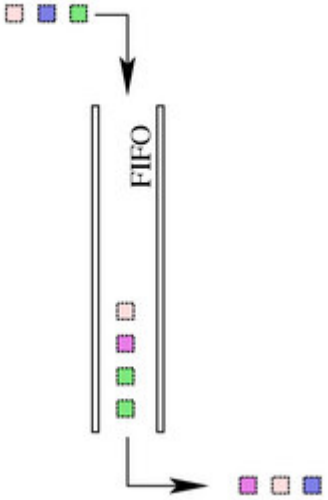
The FIFO vs Ring memory

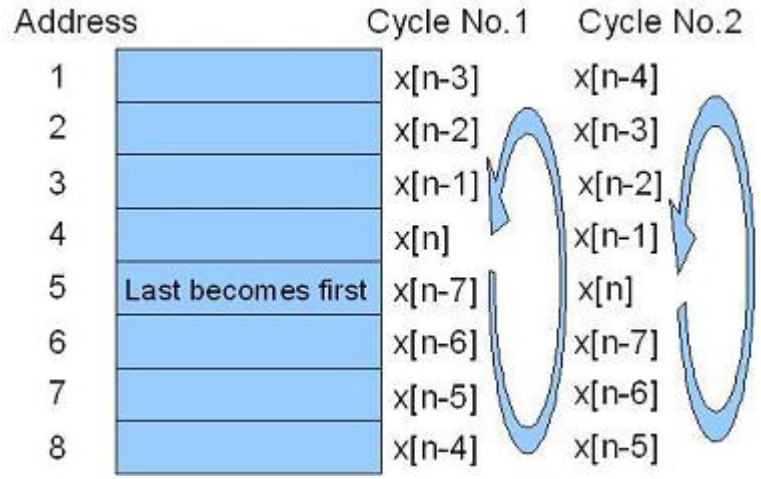






First–in First–out (FIFO)



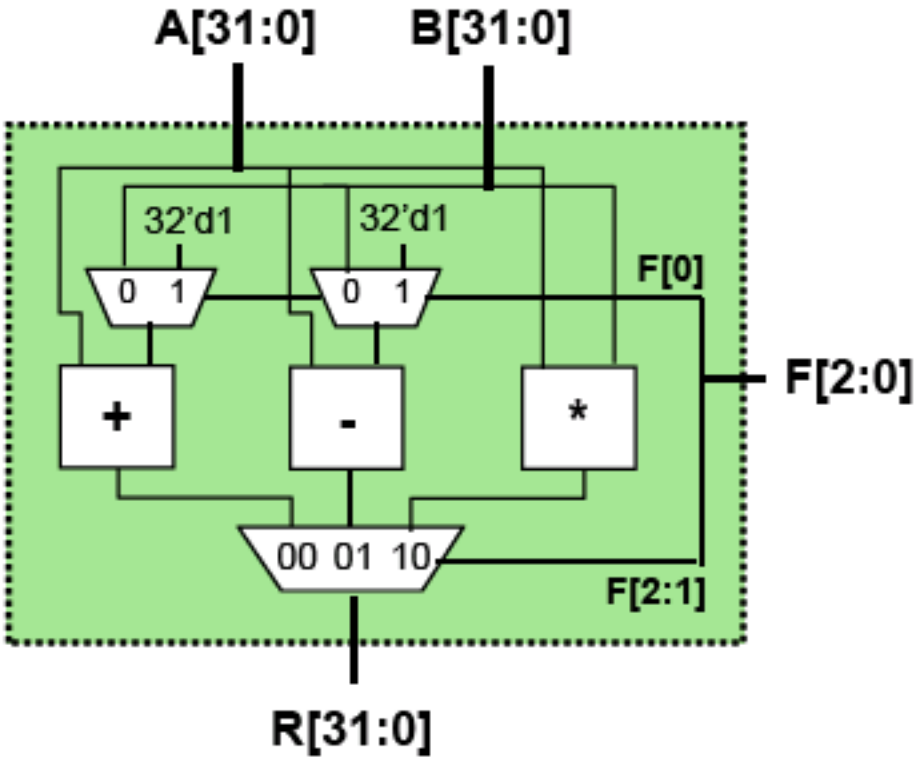


μP

Our ALU ... Or ... A simple CPU

- Modularity is essential to the success of large designs
- A Verilog `module` may contain submodules that are "wired together"
- High-level primitives enable direct synthesis of behavioral descriptions (functions such as additions, subtractions, shifts (<< and >>), etc.

Example: A 32-bit ALU



Function Table

F2	F1	F0	Function
0	0	0	A + B
0	0	1	A + 1
0	1	0	A - B
0	1	1	A - 1
1	0	X	A * B

Modules

2-to-1 MUX

```
module mux32two(i0,i1,sel,out);
input [31:0] i0,i1;
input sel;
output [31:0] out;

assign out = sel ? i1 : i0;

endmodule
```

3-to-1 MUX

```
module mux32three(i0,i1,i2,sel,out);
input [31:0] i0,i1,i2;
input [1:0] sel;
output [31:0] out;
reg [31:0] out;

always @ (i0 or i1 or i2 or sel)
begin
  case (sel)
    2'b00: out = i0;
    2'b01: out = i1;
    2'b10: out = i2;
    default: out = 32'bx;
  endcase
end
endmodule
```

32-bit Adder

```
module add32(i0,i1,sum);
input [31:0] i0,i1;
output [31:0] sum;

assign sum = i0 + i1;

endmodule
```

32-bit Subtracter

```
module sub32(i0,i1,diff);
input [31:0] i0,i1;
output [31:0] diff;

assign diff = i0 - i1;

endmodule
```

16-bit Multiplier

```
module mul16(i0,i1,prod);
input [15:0] i0,i1;
output [31:0] prod;

// this is a magnitude multiplier
// signed arithmetic later
assign prod = i0 * i1;

endmodule
```

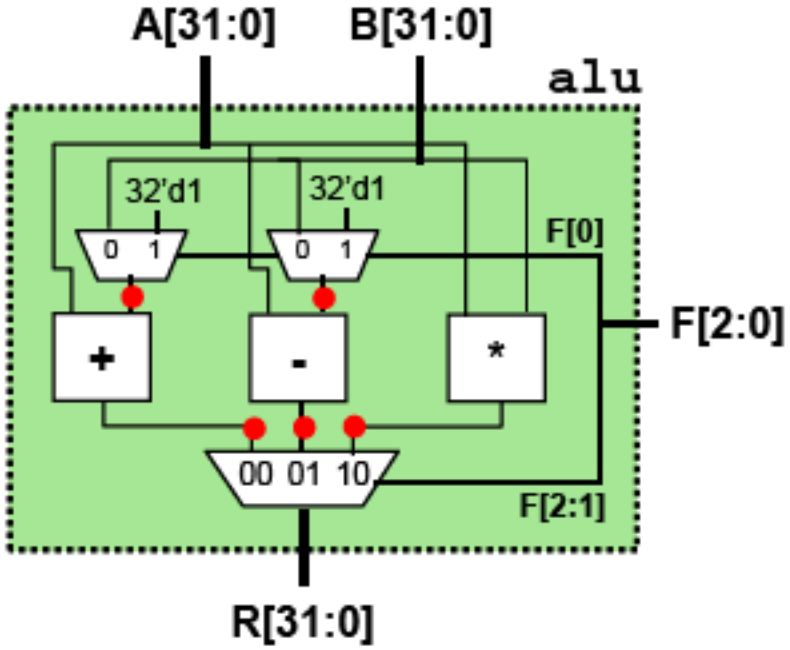
Top-Level: connect the modules

- **Given submodules:**

```
module mux32two(i0,i1,sel,out);
module mux32three(i0,i1,i2,sel,out);
module add32(i0,i1,sum);
module sub32(i0,i1,diff);
module mul16(i0,i1,prod);
```

- **Declaration of the ALU Module:**

```
module alu(a, b, f, r);
  input [31:0] a, b;
  input [2:0] f;
  output [31:0] r;

  wire [31:0] addmux_out, submux_out;
  wire [31:0] add_out, sub_out, mul_out;

  mux32two   adder_mux(b, 32'd1, f[0], addmux_out);
  mux32two   sub_mux(b, 32'd1, f[0], submux_out);
  add32      our_adder(a, addmux_out, add_out);
  sub32      our_subtracter(a, submux_out, sub_out);
  mul16      our_multiplier(a[15:0], b[15:0], mul_out);
  mux32three output_mux(add_out, sub_out, mul_out, f[2:1], r);

endmodule
```



intermediate output nodes

module names

(unique) instance names

corresponding wires/regs in module alu

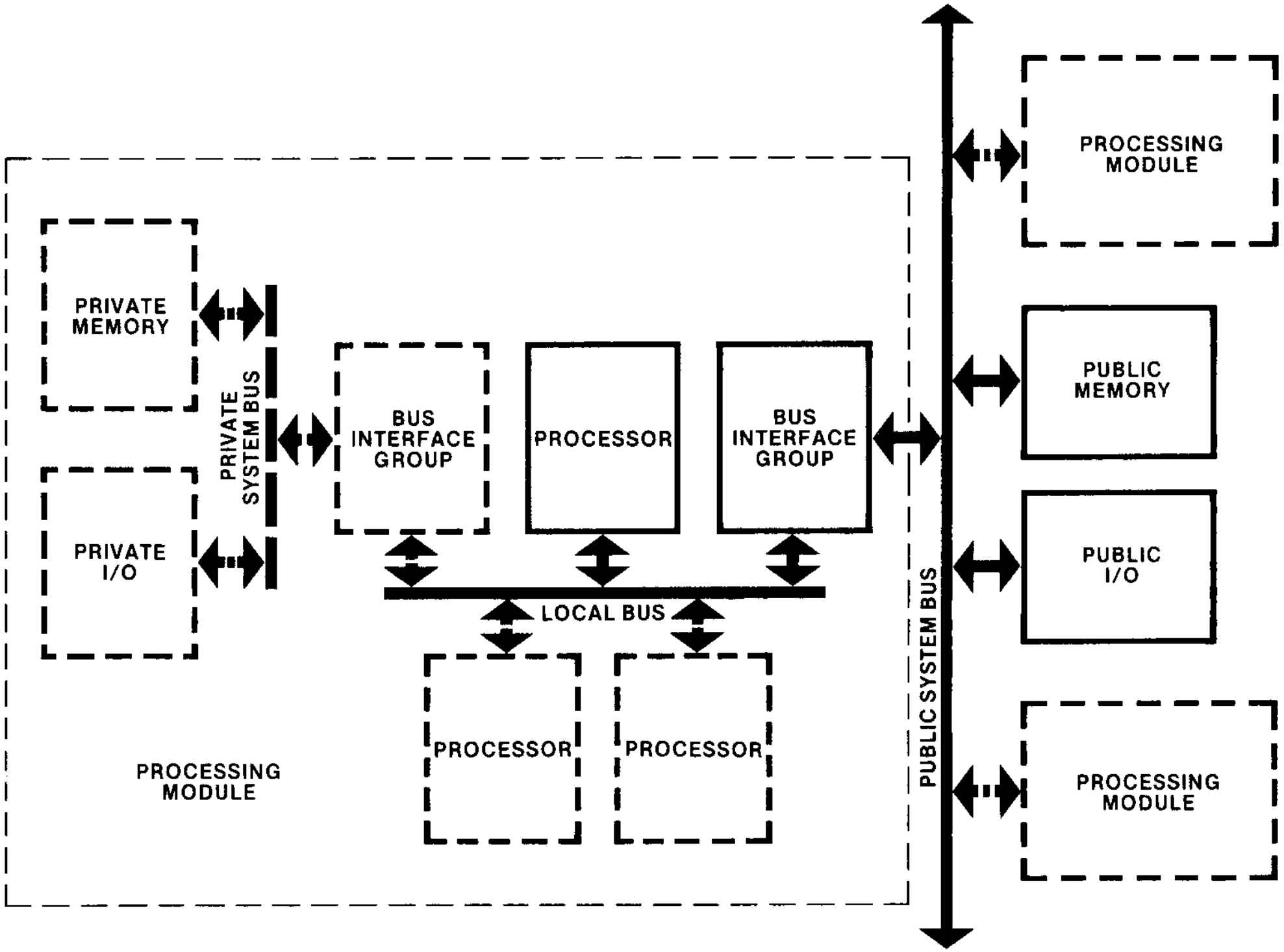
intel

The 8086 Family User's Manual

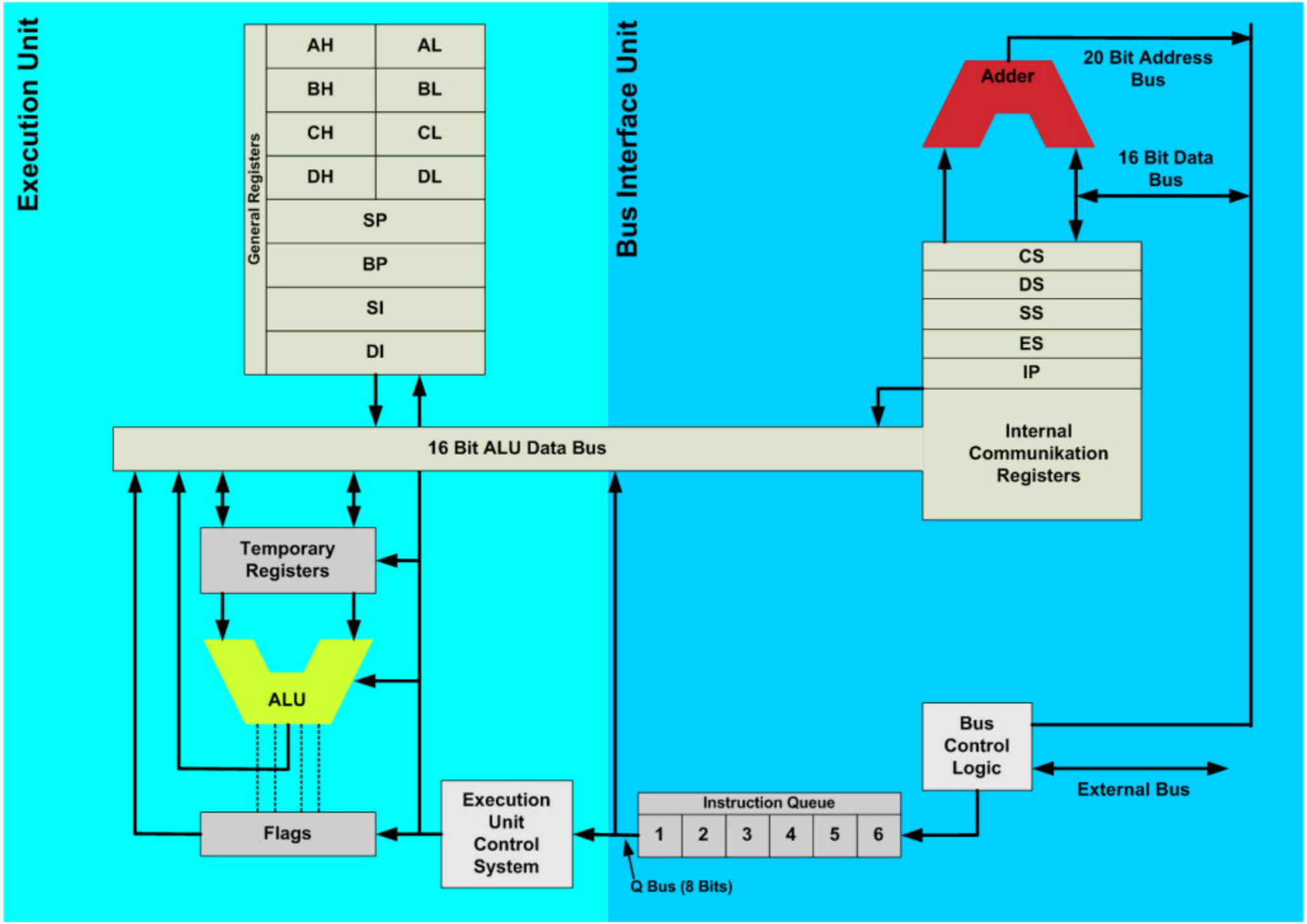
October 1979

9800722-03

The Processor in a computer



Inside the processor



Instructions and Data

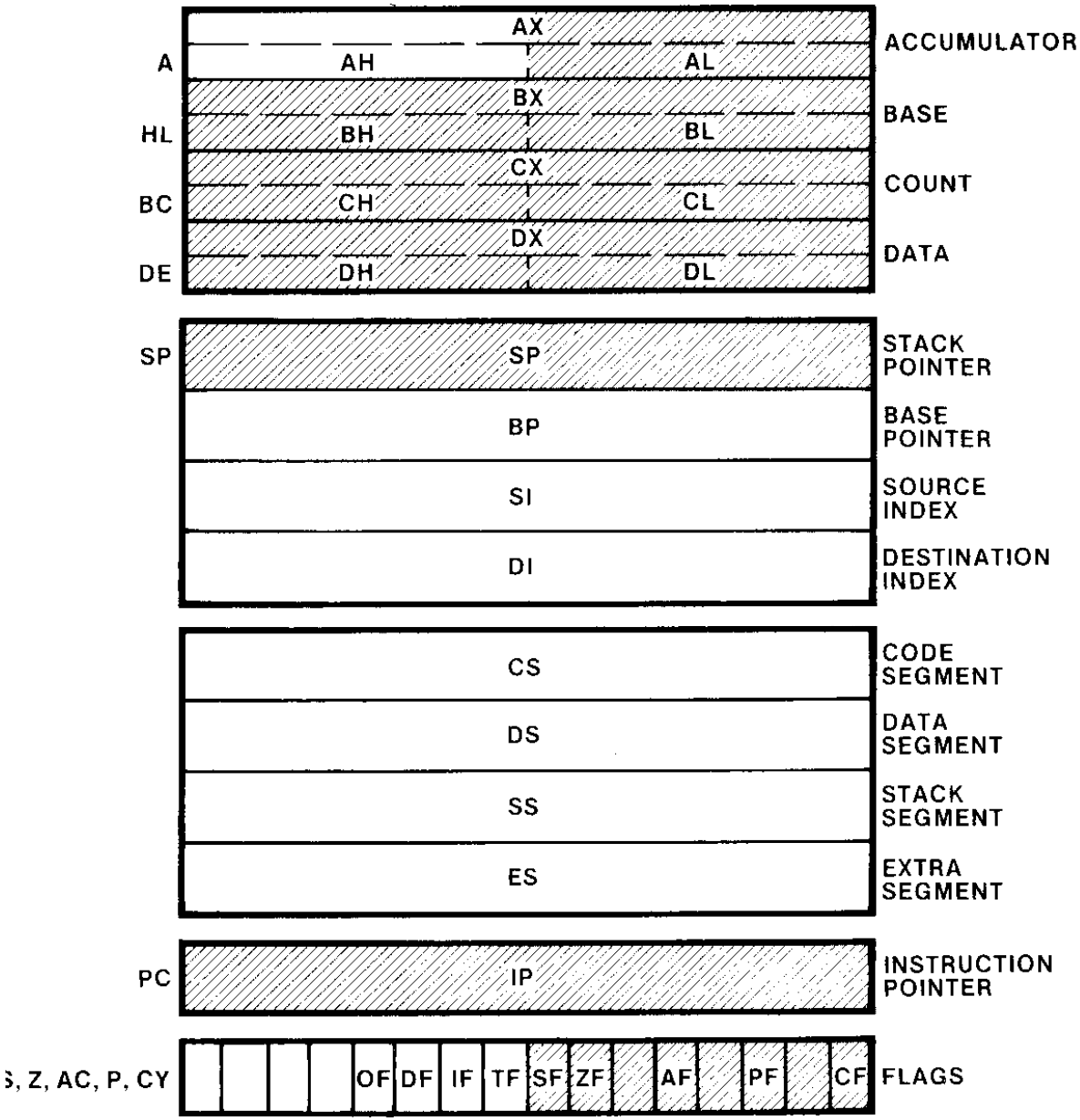
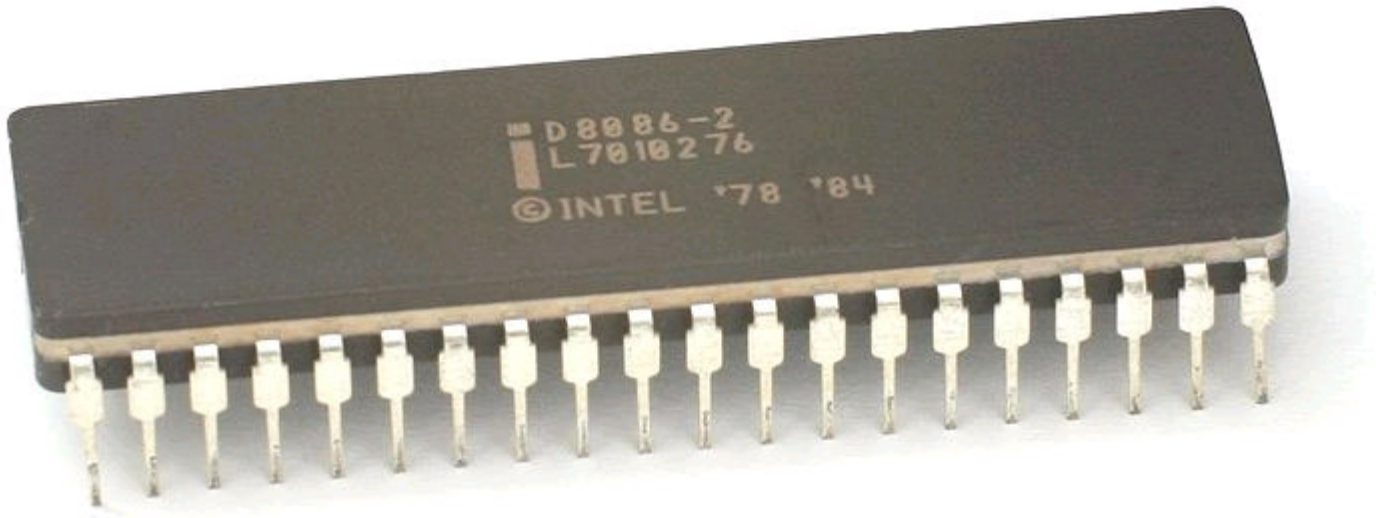


Figure 2-10. 8080/8085 Register Subset (Shaded)

... all packed in a DIP



How to give instructions?

1ST BYTE HEX	1ST BYTE BINARY	2ND BYTE	BYTES 3, 4, 5, 6	ASM-86 INSTRUCTION FORMAT	
00	0000 0000	MOD REG R/M	(DISP-LO),(DISP-HI)	ADD	REG8/MEM8,REG8
01	0000 0001	MOD REG R/M	(DISP-LO),(DISP-HI)	ADD	REG16/MEM16,REG16
02	0000 0010	MOD REG R/M	(DISP-LO),(DISP-HI)	ADD	REG8,REG8/MEM8
03	0000 0011	MOD REG R/M	(DISP-LO),(DISP-HI)	ADD	REG16,REG16/MEM16
04	0000 0100	DATA-8		ADD	AL,IMMED8
05	0000 0101	DATA-LO	DATA-HI	ADD	AX,IMMED16
06	0000 0110			PUSH	ES
07	0000 0111			POP	ES

ASSEMBLER IS ALREADY A PROGRAMMING LANGUAGE...

FIRST ABSTRACTION LEVEL!

The Instruction set

Hi \ Lo	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	ADD b,f,r/m	ADD w,f,r/m	ADD b,f,r/m	ADD w,f,r/m	ADD b, ia	ADD w, ia	PUSH ES	POP ES	OR b,f,r/m	OR w,f,r/m	OR b,f,r/m	OR w,f,r/m	OR b,i	OR w,i	PUSH CS	
1	ADC b,f,r/m	ADC w,f,r/m	ADC b,f,r/m	ADC w,f,r/m	ADC b,i	ADC w,i	PUSH SS	POP SS	SBB b,f,r/m	SBB w,f,r/m	SBB b,f,r/m	SBB w,f,r/m	SBB b,i	SBB w,i	PUSH DS	POP DS
2	AND b,f,r/m	AND w,f,r/m	AND b,f,r/m	AND w,f,r/m	AND b,i	AND w,i	SEG =ES	DAA	SUB b,f,r/m	SUB w,f,r/m	SUB b,f,r/m	SUB w,f,r/m	SUB b,i	SUB w,i	SEG =CS	DAS
3	XOR b,f,r/m	XOR w,f,r/m	XOR b,f,r/m	XOR w,f,r/m	XOR b,i	XOR w,i	SEG =SS	AAA	CMP b,f,r/m	CMP w,f,r/m	CMP b,f,r/m	CMP w,f,r/m	CMP b,i	CMP w,i	SEG =DS	AAS
4	INC AX	INC CX	INC DX	INC BX	INC SP	INC BP	INC SI	INC DI	DEC AX	DEC CX	DEC DX	DEC BX	DEC SP	DEC BP	DEC SI	DEC DI
5	PUSH AX	PUSH CX	PUSH DX	PUSH BX	PUSH SP	PUSH BP	PUSH SI	PUSH DI	POP AX	POP CX	POP DX	POP BX	POP SP	POP BP	POP SI	POP DI
6																
7	JO	JNO	JB/ JNAE	JNB/ JAE	JE/ JZ	JNE/ JNZ	JBE/ JNA	JNBE/ JA	JS	JNS	JP/ JPE	JNP/ JPO	JL/ JNGE	JNL/ JGE	JLE/ JNG	JNLE/ JG
8	Immed b,r/m	Immed w,r/m	Immed b,r/m	Immed is,r/m	TEST b,r/m	TEST w,r/m	XCHG b,r/m	XCHG w,r/m	MOV b,f,r/m	MOV w,f,r/m	MOV b,f,r/m	MOV w,f,r/m	**MOV sr,f,r/m**	LEA	**MOV sr,f,r/m**	POP r/m
9	XCHG AX	XCHG CX	XCHG DX	XCHG BX	XCHG SP	XCHG BP	XCHG SI	XCHG DI	CBW	CWD	CALL I,d	WAIT	PUSHF	POPF	SAHF	LAHF
A	MOV m → AL	MOV m → AX	MOV AL → m	MOV AX → m	**MOVS**	**MOVS**	**CMPS**	**CMPS**	**TEST b,I,a**	**TEST w,i,a**	**STOS**	**STOS**	**LODS**	**LODS**	**SCAS**	**SCAS**
B	MOV i → AL	MOV i → CL	MOV i → DL	MOV i → BL	MOV i → AH	MOV i → CH	MOV i → DH	MOV i → BH	MOV i → AX	MOV i → CX	MOV i → DX	MOV i → BX	MOV i → SP	MOV i → BP	MOV i → SI	MOV i → DI
C			RET, (i+SP)	RET	LES	LDS	MOV b,i,r/m	MOV w,i,r/m			RET, I,(i+SP)	RET I	INT Type 3	INT (Any)	INTO	IRET
D	Shift b	Shift w	Shift b,v	Shift w,v	AAM	AAD		XLAT	ESC 0	ESC 1	ESC 2	ESC 3	ESC 4	ESC 5	ESC 6	ESC 7
E	LOOPNZ/ LOOPNE	LOOPZ/ LOOPE	LOOP	JCXZ	**IN b**	**IN w**	**OUT b**	**OUT w**	CALL d	JMP d	JMP I,d	JMP si,d	**IN v,b**	**IN v,w**	**OUT v,b**	**OUT v,w**
F	LOCK		REP	REP z	HLT	CMC	Grp 1 b,r/m	Grp 1 w,r/m	CLC	STC	CLI	STI	CLD	STD	Grp 2 b,r/m	Grp 2 w,r/m

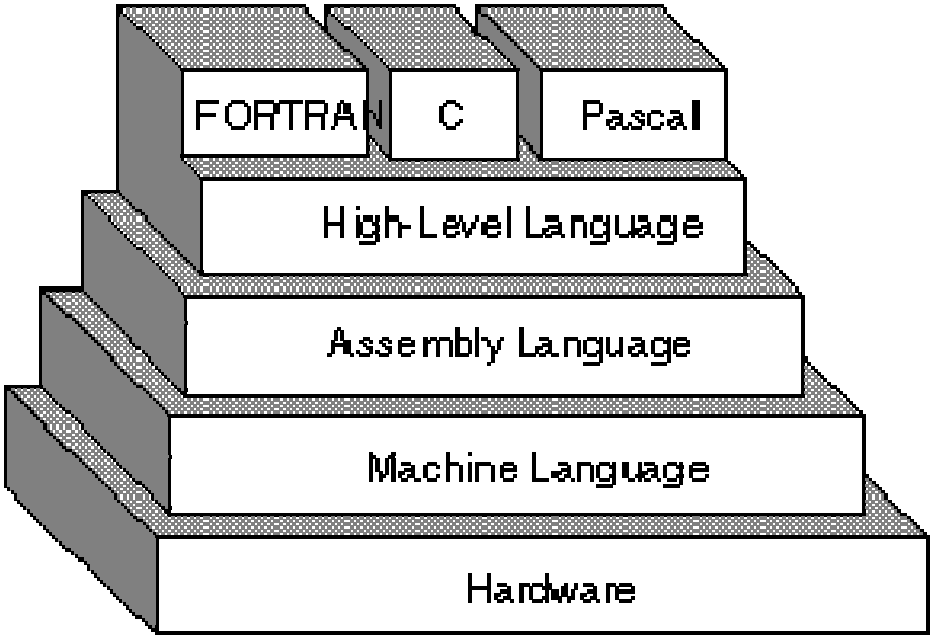
The definition of Instructions... Two examples

ADD	**ADD** destination,source Addition			**Flags** O D I T S Z A P C X X X X X X
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3	—	2	ADD CX, DX
register, memory	9 + EA	1	2-4	ADD DI, [BX].ALPHA
memory, register	16 + EA	2	2-4	ADD TEMP, CL
register, immediate	4	—	3-4	ADD CL, 2
memory, immediate	17 + EA	2	3-6	ADD ALPHA, 2
accumulator, immediate	4	—	2-3	ADD AX, 200

AND	**AND** destination,source Logical and			**Flags** O D I T S Z A P C 0 X X U X 0
Operands	**Clocks**	**Transfers***	**Bytes**	**Coding Example**
register, register	3	—	2	AND AL,BL
register, memory	9 + EA	1	2-4	AND CX,FLAG_WORD
memory, register	16 + EA	2	2-4	AND ASCII [DI],AL
register, immediate	4	—	3-4	AND CX,0F0H
memory, immediate	17 + EA	2	3-6	AND BETA, 01H
accumulator, immediate	4	—	2-3	AND AX, 01010000B

Programming a microprocessor

Programming a microprocessor



Hardware

- CPU
- Devices

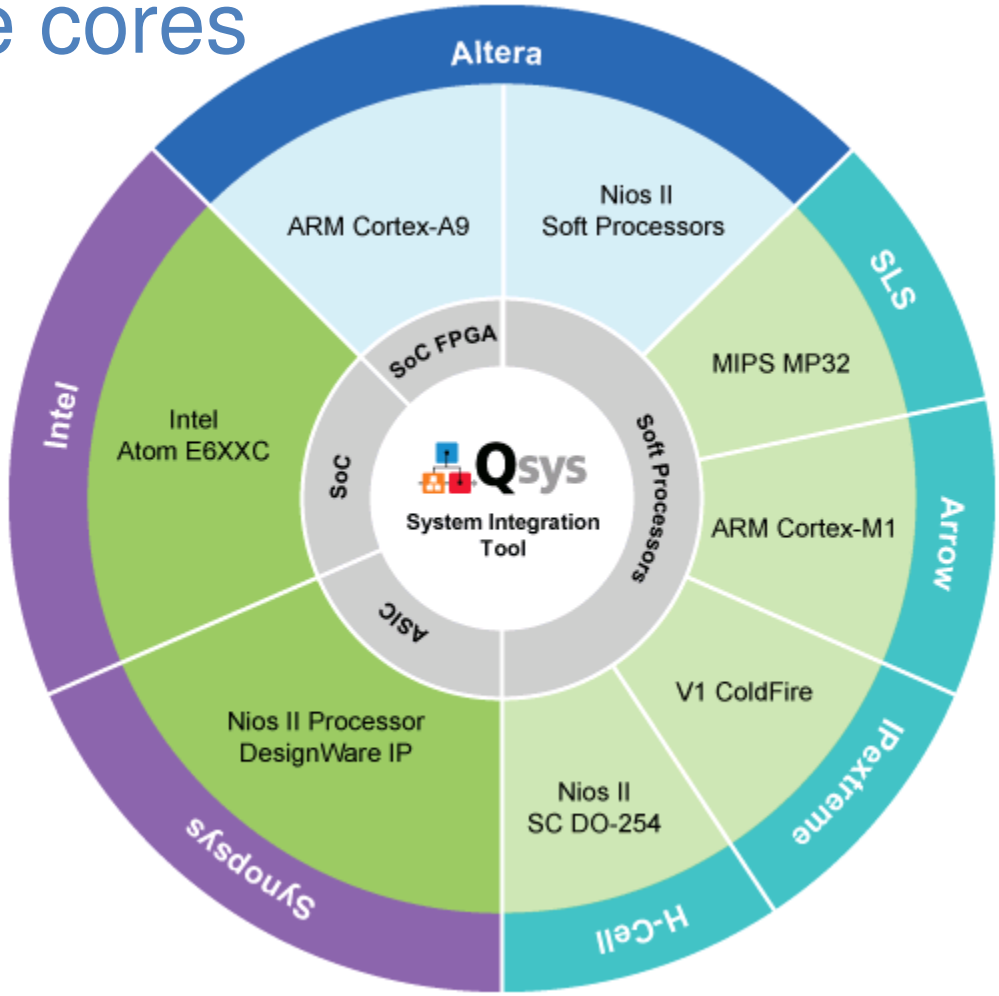
Operating System

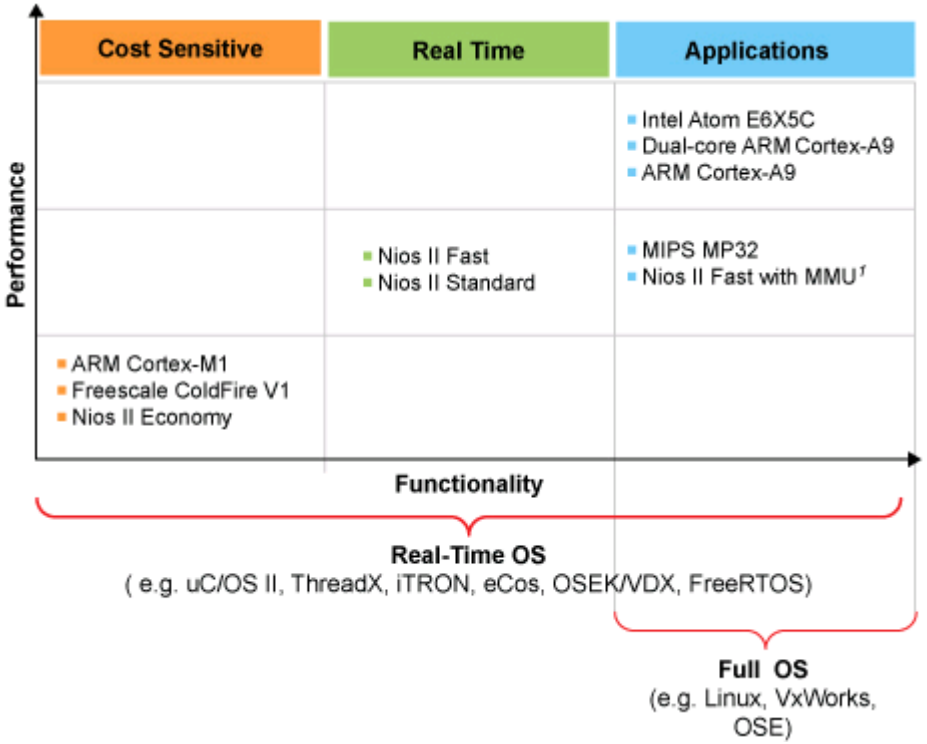
- Drivers
- API
- Programing language

Program

- User Application

Available cores





http://www.altera.com/devices/processor/additional/ipm-index.jsp

The ALTERA “standard”

Nios® II

Lots of tools and tutorials…
e.g. NIOS IDE (Integrated Development Environment)
DE2 demonstrations

Tools – SOPC builder

http://www.altera.com/education/demonstrations/sopc-builder/sopc-builder-demo.html

ftp://ftp.altera.com/up/pub/Tutorials/DE2/Computer_Organization/tut_sopc_introduction_verilog.pdf

The SOPC Builder is a tool used in conjuction with the Quartus II CAD software. It allows the user to easily create a system based on the Nios II processor, by simply selecting the desired functional units and specifying their parameters.

There are other choices of µ-processors to implements. E.g.: mips
Operating systems can be used. E.g. µ-Clinux™

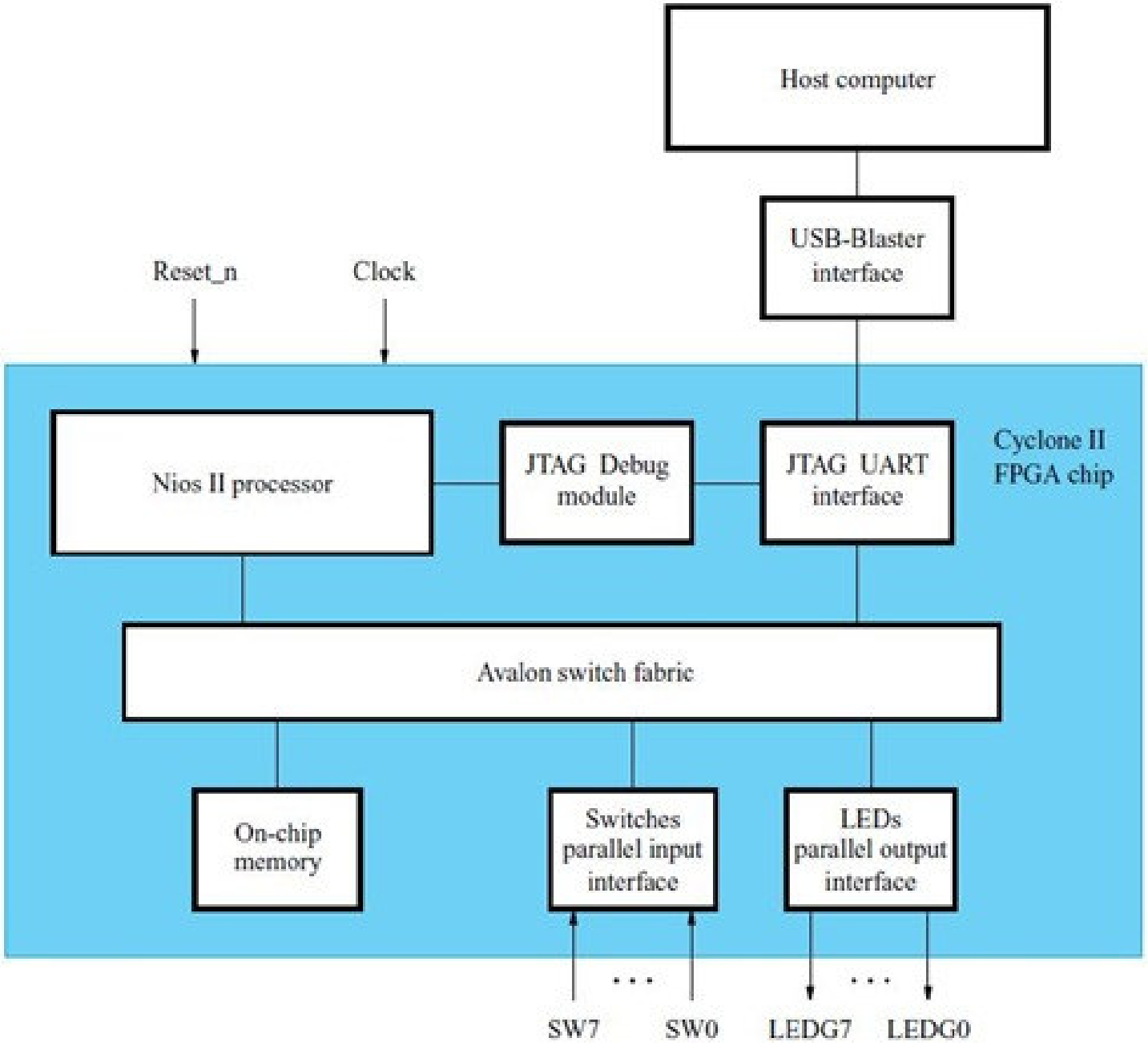
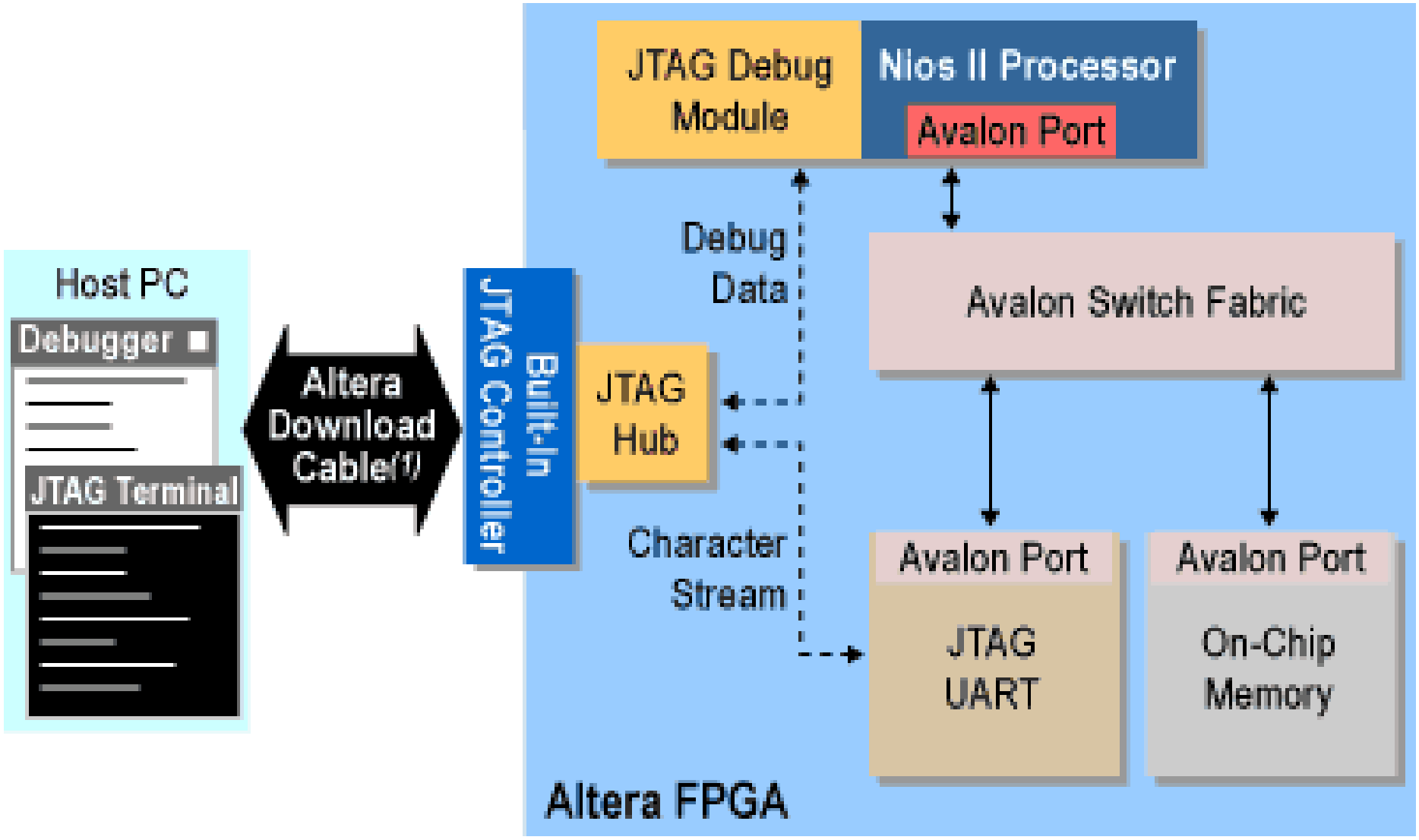


Figure 2. A simple example of a Nios II system.

JTAG Debug Module



Tools – Mega Wizard

ftp://ftp.altera.com/up/pub/Tutorials/DE2/Digital_Logic/tut_lpms_verilog.pdf

Tools – SOPC builder

http://www.altera.com/education/demonstrations/sopc-builder/sopc-builder-demo.html

ftp://ftp.altera.com/up/pub/Tutorials/DE2/Computer_Organization/tut_sopc_introduction_verilog.pdf

The SOPC Builder is a tool used in conjuction with the Quartus II CAD software. It allows the user to easily
create a system based on the Nios II processor, by simply selecting the desired functional units and specifying
their parameters.

Tools – QSys

QSys is currently replacing SOPC builder in newer versions of Altera's software.
Some Quartus versions will ask you to use this new tool!

I/O

Interfaces to the outside world

I/O - parallel

Parallel protocols are tipically easy and fast…

However the resources alocated are big (pins)

Tipycally the protocol language is very easy.
Time constraints easy to deal with

e.g.: GPIO

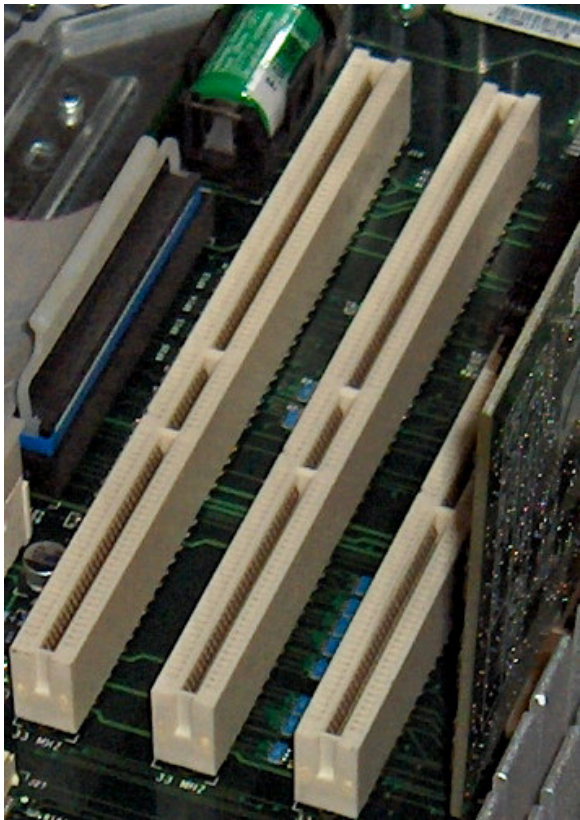
I/O - parallel

Parallel protocols are tipically easy and fast…

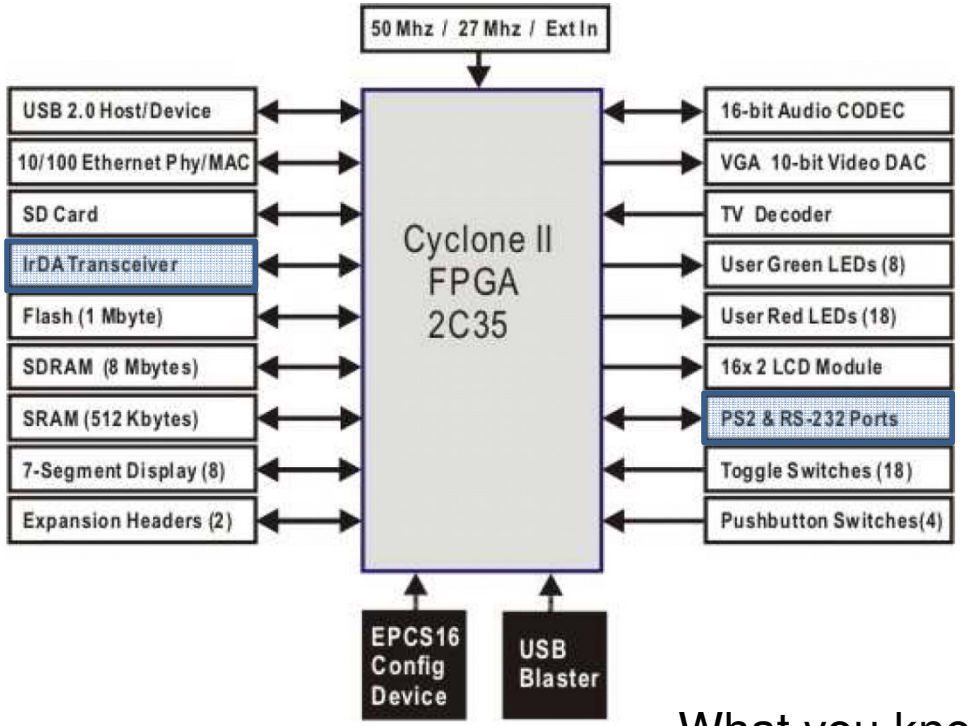
However the resources alocated are big (pins)

Tipycally the protocol language is very easy.
Time constraints easy to deal with

e.g.: The PCI protocol



I/O - serial



Readily available:
IrDA, RS232, PS2

You can build some other…
I2C - Inter-Integrated Circuit
Serial Peripheral Interface (SPI)
…

What you kneed to know?

- Pinout
- Clocking requirements
- Protocol (the language)
- The commands of the peripherical you are communicating with

- A zillion standards
  - Asynchronous (no explicit clock) vs. Synchronous (CLK line in addition to DATA line).
  - Recent trend to reduce signaling voltages: save power, reduce transition times
  - Control/low-bandwidth Interfaces: SPI, I²C, 1-Wire, PS/2, AC97
  - Networking: RS232, Ethernet, T1, Sonet
  - Computer Peripherals: USB, FireWire, Fiber Channel, Infiniband, SATA, Serial Attached SCSI

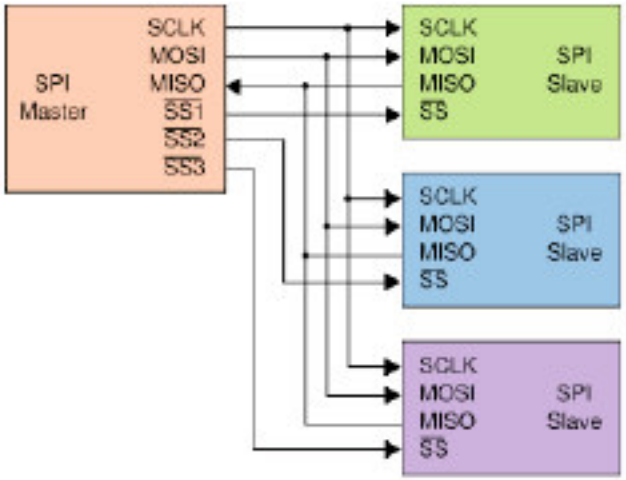
RS232 – The serial port

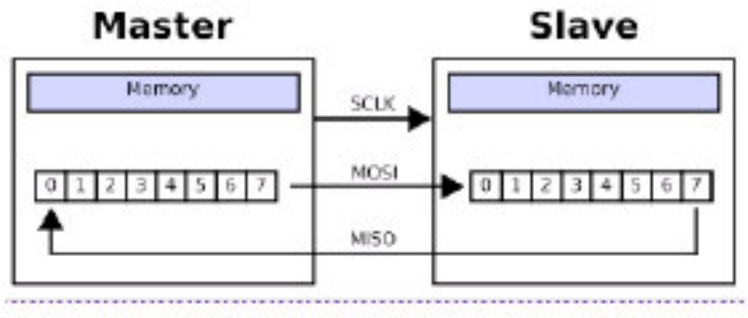
- Characteristics
  - Large voltages => special interface chips
    (1/mark: -12V to -3V, 0/space: 3V to 12V)
  - Separate xmit and rcv wires: full duplex
  - Slow transmission rates (1 bit time = 1 baud); most interfaces support standardized baud rates: 1200, 2400, 4800, 9600, 19.2K, 38.4K, 57.6K, 115.2K
  - Format
    - Wire is held at 1/mark when idle
    - Start bit (1 bit of "0" at start of transmission)
    - Data bits (LSB first, can be 5 to 8 bits of data)
    - Parity bit (none, even, odd)
    - Stop bits (1, 1.5 or 2 bits of 1/mark at end of symbol)
    - Most common 8-N-1: eight data bits, no parity, one stop bit

Sending: easy – Just compose your signal respecting time constraints
Receiving: Need to oversample and understand where the bits are… and check the protocol format: start, stop, parity

SPI (Serial Peripheral Interface)

- Simple, 3-wire interface + devices selects
  - SCLK generated by master (1-70MHz). Assert data on one edge, sample data on the other. Default state of SCLK and assignment of edges is often programmable.
  - Master Out Slave In (MOSI) data shifted out of master register into slave register
  - Master In Slave Out (MISO) data shifted out of slave register and into master register
  - Selects (usually active low) determine which device is active. Assertion often triggers an action in the slave, so master waits some predetermined time then shifts data.

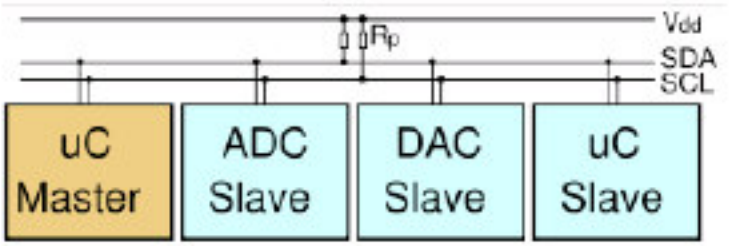


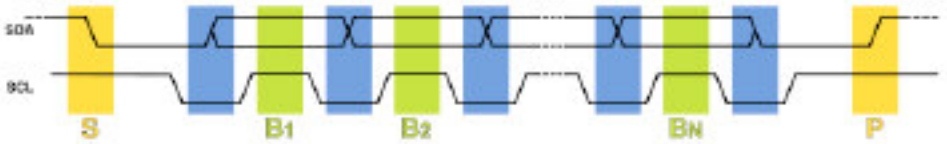


Figures from Wikipedia

I²C (Inter-Integrated Circuit)

- 2 open-drain wires (SCL = clock, SDA = data)
- Multiple-master, each transmission addresses a particular device, many devices have many different sub-addresses (internal registers)
- Format (all addresses/data send MSB first):
  - Sender: Start [S] bit (SDA↓ while SCL high)
  - Sender: One or more 8-bit data packets, each followed by 1-bit ACK
    - Data changed when SCL low, sampled at SCL↑
    - Receiver: Active-low ACK generated after each data packet
  - Sender: Stop [P] bit (SDA↑ while SCL high)
- SCL and SDA have pullup resistors, senders only drive low, go high -impedance to let pullups make line high (so multiple drivers okay!)
  - Receiver can hold SCL low to stretch clock timing, sender must wait until SCL goes high before moving to next bit.
  - Multiple senders can contend using SDA for arbitration





Just two examples in a PC

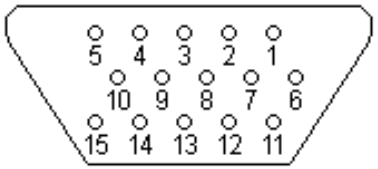
Sensors like thermometers in the motherboard communicate using I2C

Most PCs have a “hidden I2C Port”

VGA DB15 connector pinout

The pin layout of the **VGA** interface connector is shown in the figure below. Three pins are used to carry the three basic **RGB** color signals *red*, *green* and *blue* and two pins carry the horizontal and vertical *sync* signal. The red, green and blue signal lines have their own ground return line. The picture shows the **VGA DDC2** connector including the **I2C SLC** clock and **SDA** data lines for exchanging digital data between the video controller and the display.

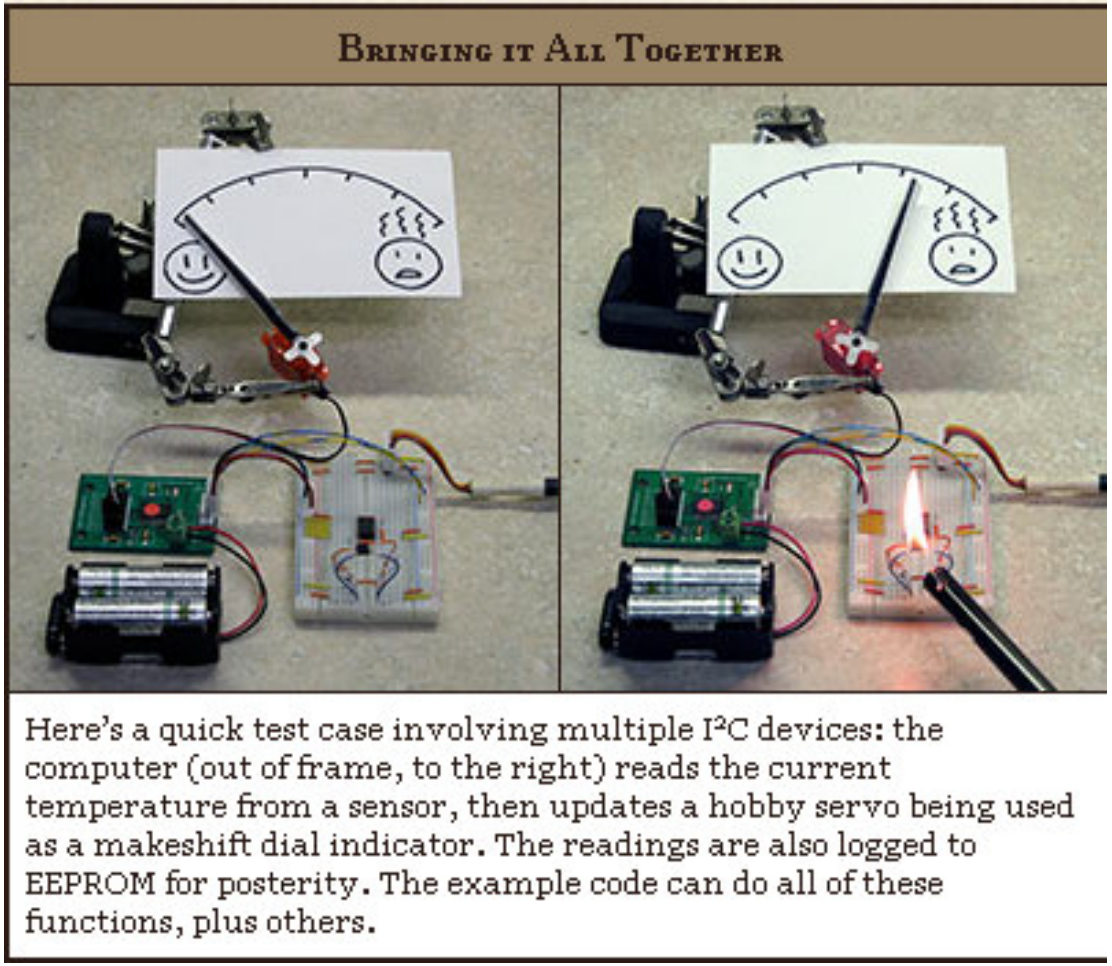
VGA connector pinout



Pin	Name	Function
1	RED	Red video
2	GREEN	Green video
3	BLUE	Blue video
4	n/c	not connected
5	GND	Signal ground
6	RED_RTN	Red ground
7	GREEN_RTN	Green ground
8	BLUE_RTN	Blue ground
9	VDC	5 VDC supply (fused)
10	GND	Signal ground
11	n/c	not connected
12	SDA	DDC / I2C data
13	HSYNC	Horizontal sync
14	VSYNC	Vertical sync
15	SCL	DDC / I2C clock

THE 25¢ I²C ADAPTER

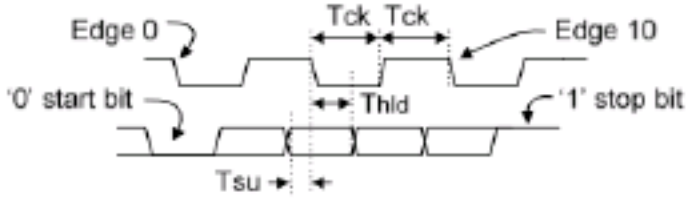
(http://www.paintyourdragon.com/?p=43)



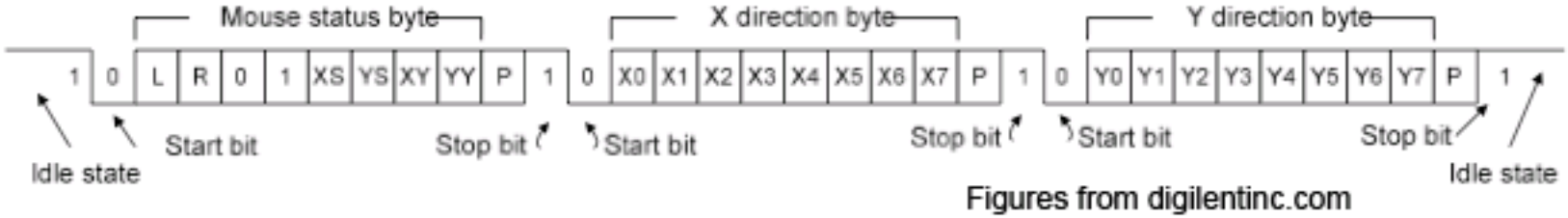
Here's a quick test case involving multiple I²C devices: the computer (out of frame, to the right) reads the current temperature from a sensor, then updates a hobby servo being used as a makeshift dial indicator. The readings are also logged to EEPROM for posterity. The example code can do all of these functions, plus others.

PS/2 Keyboard/Mouse Interface

- 2-wire interface (CLK, DATA), bidirectional transmission of serial data at 10-16kHz
- Format
  - Device generates CLK, but host can request-to-send by holding CLK low for 100us
  - DATA and CLK idle at "1", CLK starts when there's a transmission. DATA changes on CLK↑, sampled on CLK↓
  - 11-bit packets: one start bit of "0", 8 data bits (LSB first), odd parity bit, one stop bit of "1".
  - Keyboards send scan codes (not ASCII!) for each press, 8'hF0 followed by scan code for each release
  - Mice send button status, Δx and Δy of movement since last transmission



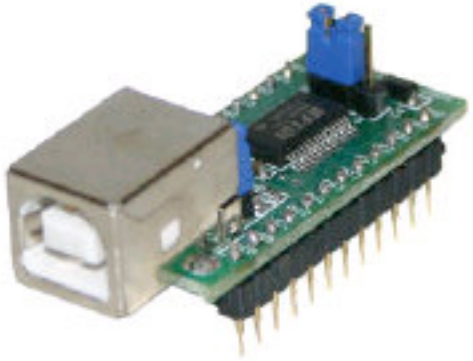
Symbol	Parameter	Min	Max
T_{CK}	Clock time	30us	50us
T_{SU}	Data-to-clock setup time	5us	25us
T_{HLD}	Clock-to-data hold time	5us	25us

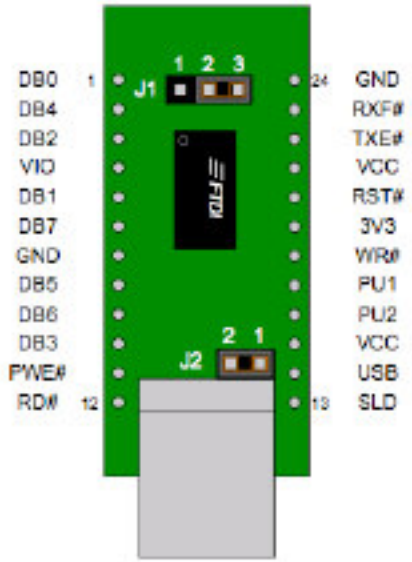


Figures from digilentinc.com

USB (Universal Serial Bus)

- 2-wire (D+,D-) for high-speed, bidirectional polled transmission between master and addressable endpoints in multiple devices. Full speed (12Mbps) and High speed (480Mbps) data rates.
- Multi-level tiered-star topology (127 devices, including hubs)
- FTDI UM245R USB-to-FIFO module for bidirectional data transfer using a handshake protocol, also asynchronous "bit -bang" mode with selectable baud rates.
  - 24-pin DIP module, wire to user pins





Figures from ftdi.com

I/O - Analog

PWM – Pulse Width Modulation

```
wire [7:0] H_PWM;
wire [3:0] bright;
assign bright=SW[3:0];
reg [3:0] PWM_count;
always @ (posedge CLOCK_50)
            PWM_count<=PWM_count+1;
assign H_PWM=(bright>PWM_count)? 7'hFF : 7'h00;
```

I/O – Digital ⇔ Analog

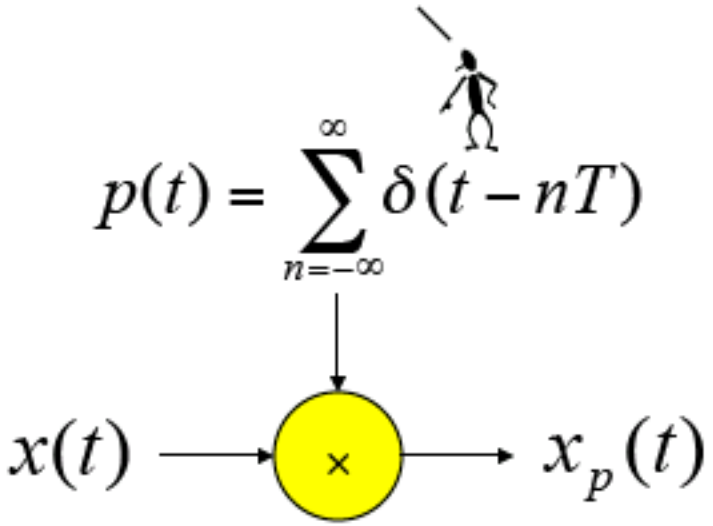
FPGAs are digital devices!
They can't generate or receive analog signals.

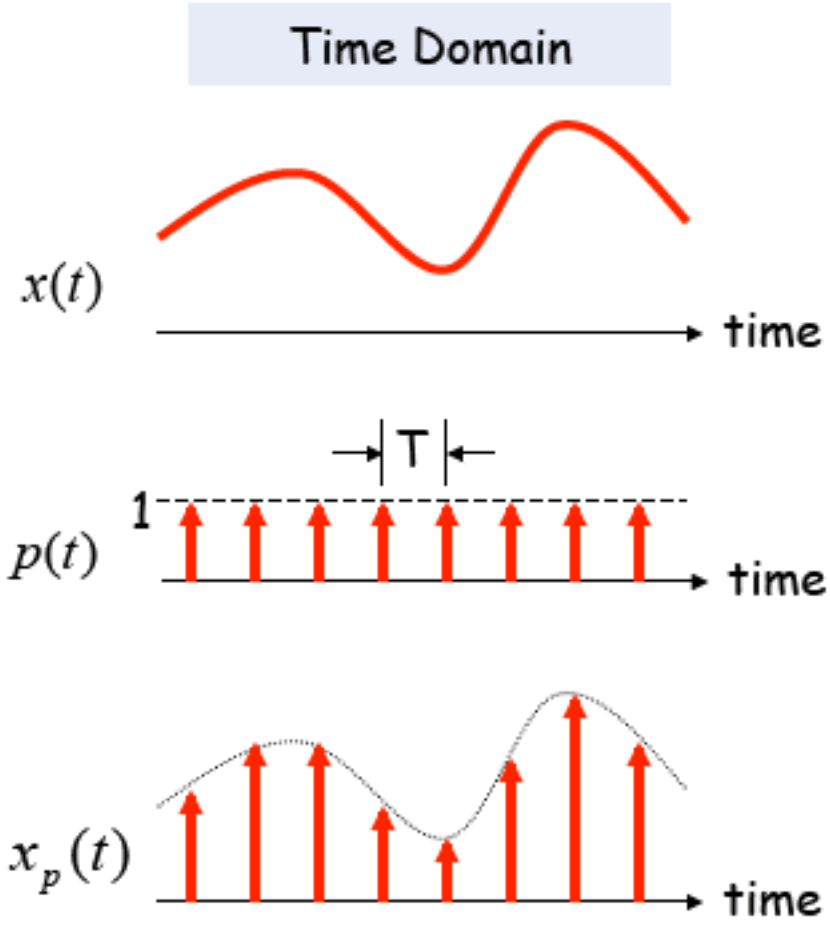
You need to use a DAC or ADC to interface analog devices!

Discrete Time

Let's use an impulse train to sample a continuous-time function at a regular interval T:

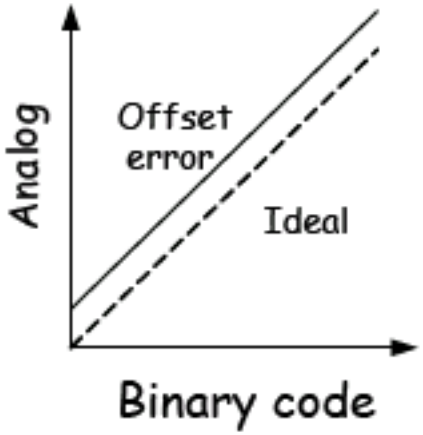
δ(x) is a narrow impulse at x=0, where $\int_{-\infty}^{\infty} f(t)\delta(t-a)dt = f(a)$



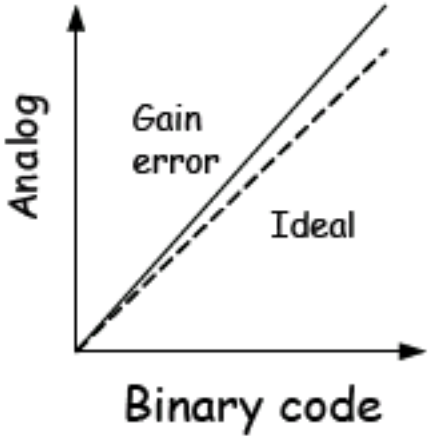


Non-idealities in Data Conversion

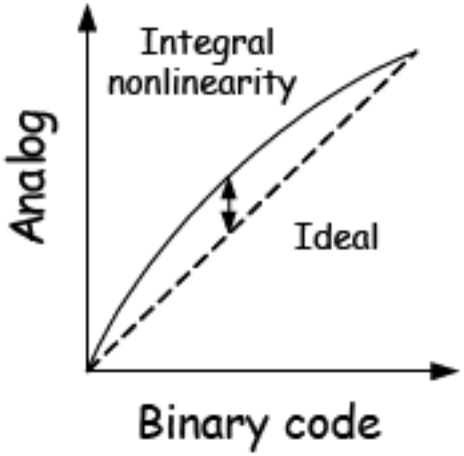
Offset – a constant voltage offset that appears at the output when the digital input is 0



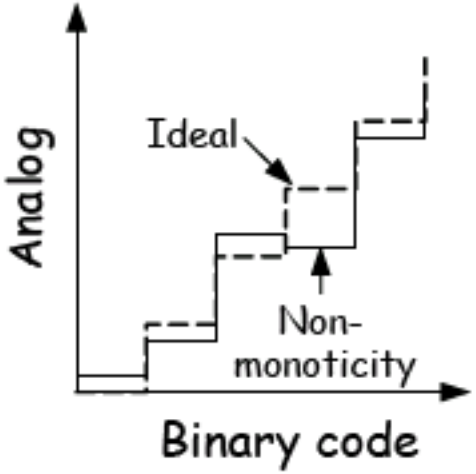
Gain error – deviation of slope from ideal value of 1

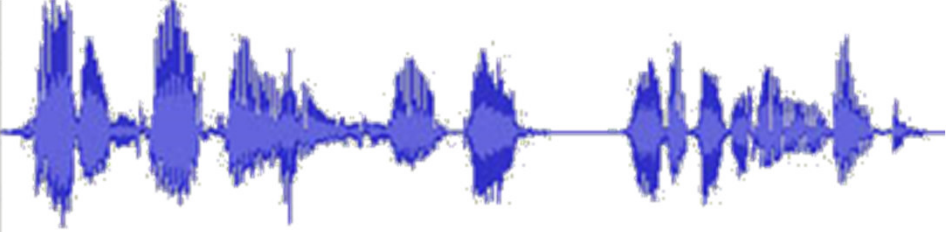


Integral Nonlinearity – maximum deviation from the ideal analog output voltage



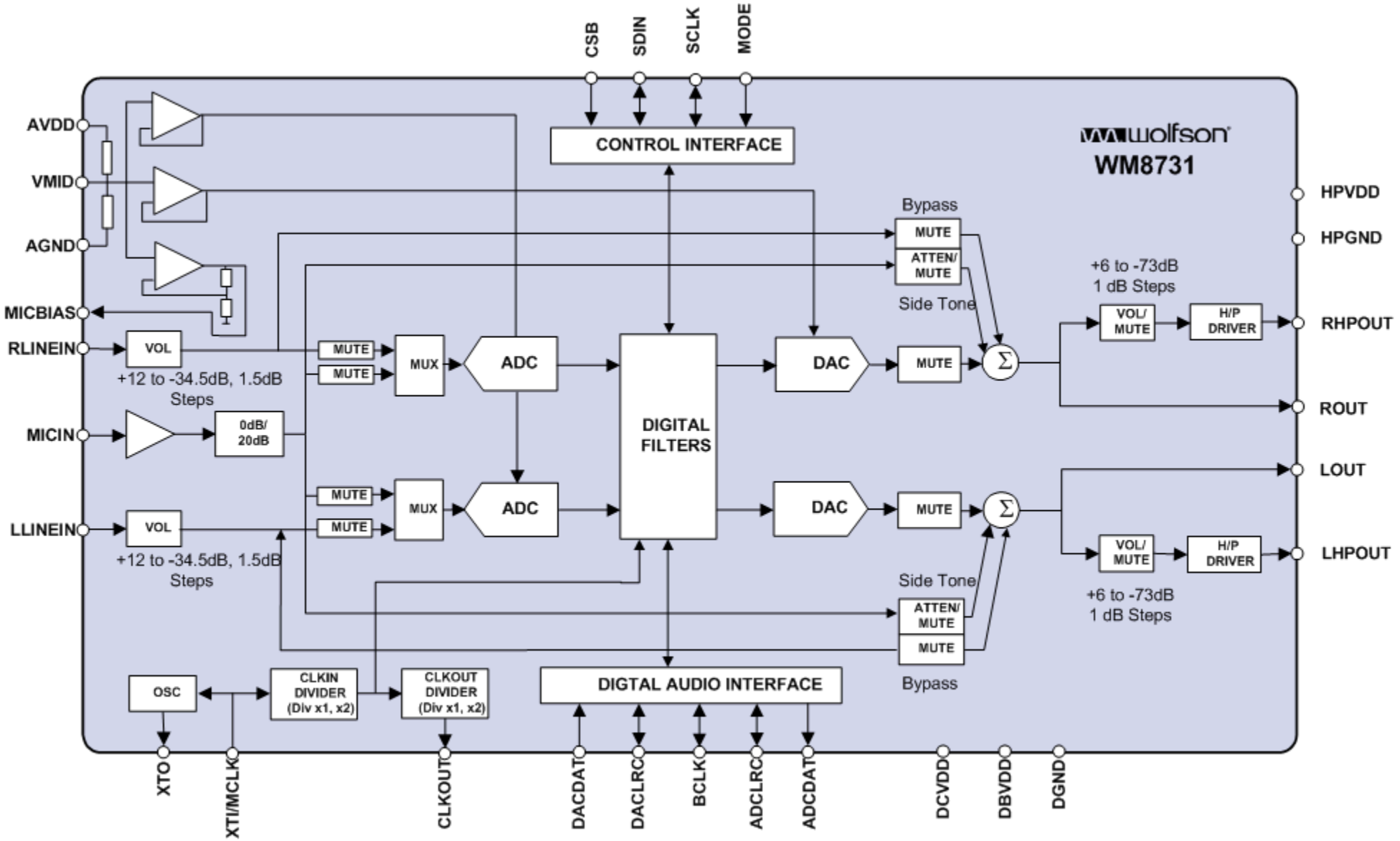
Differential nonlinearity – the largest increment in analog output for a 1-bit change



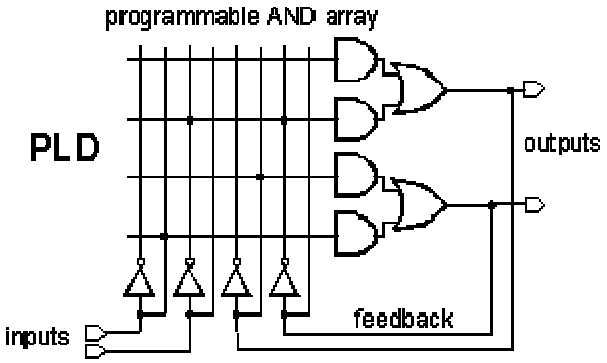


Sound (analog signal)

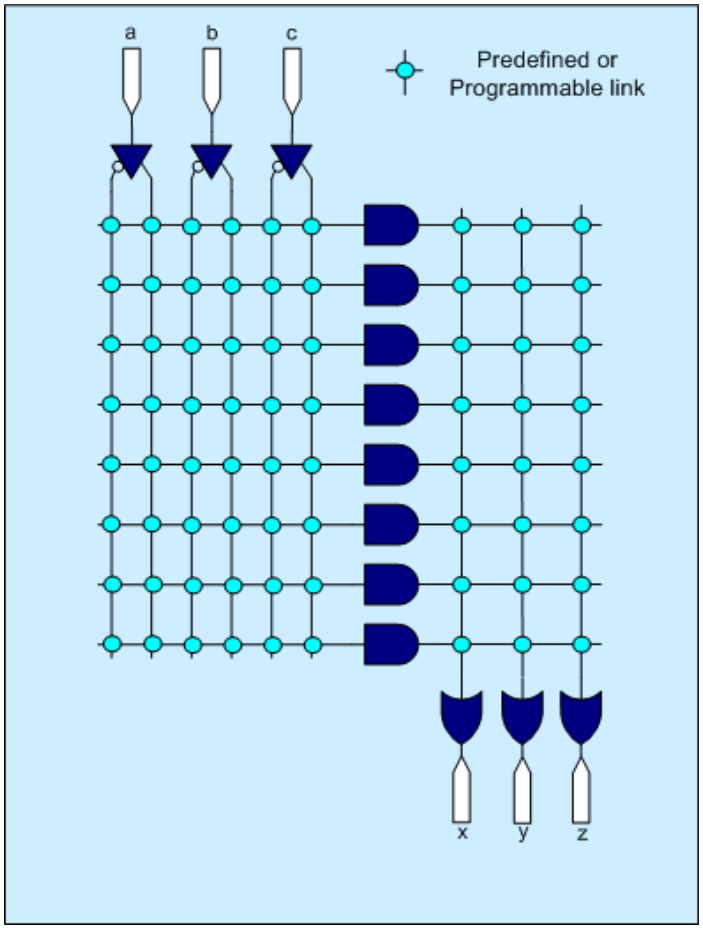
Means ADC/DAC



Finally...
What is an FPGA



PLD Architecture



Architecture of CPLD :

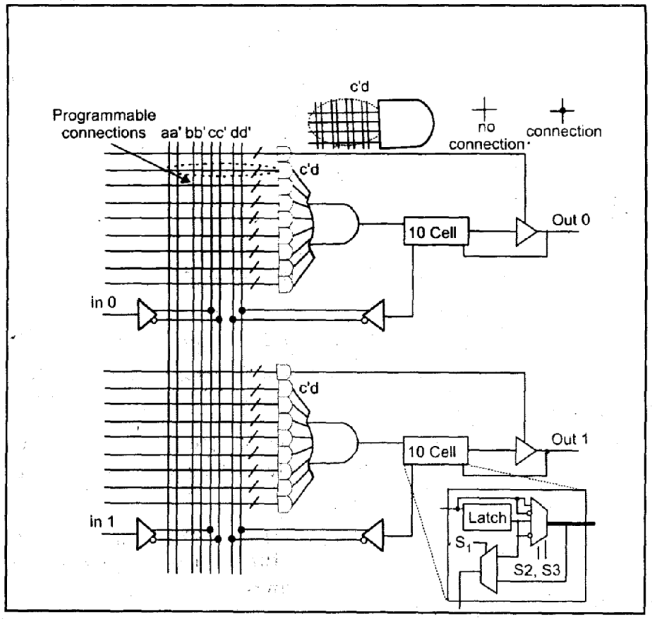
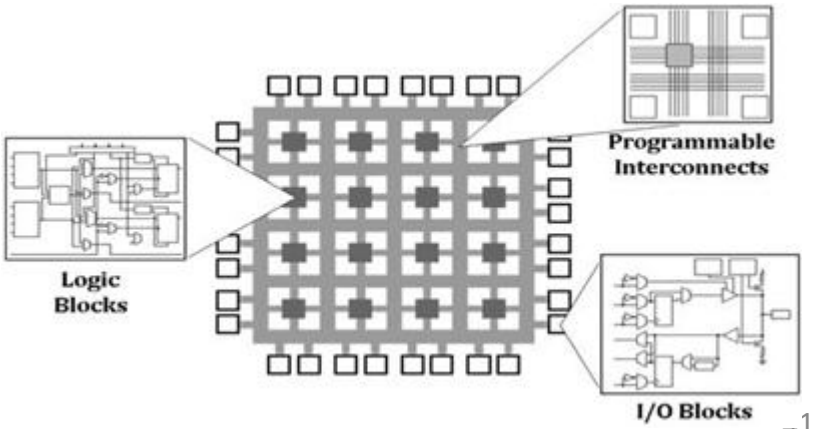
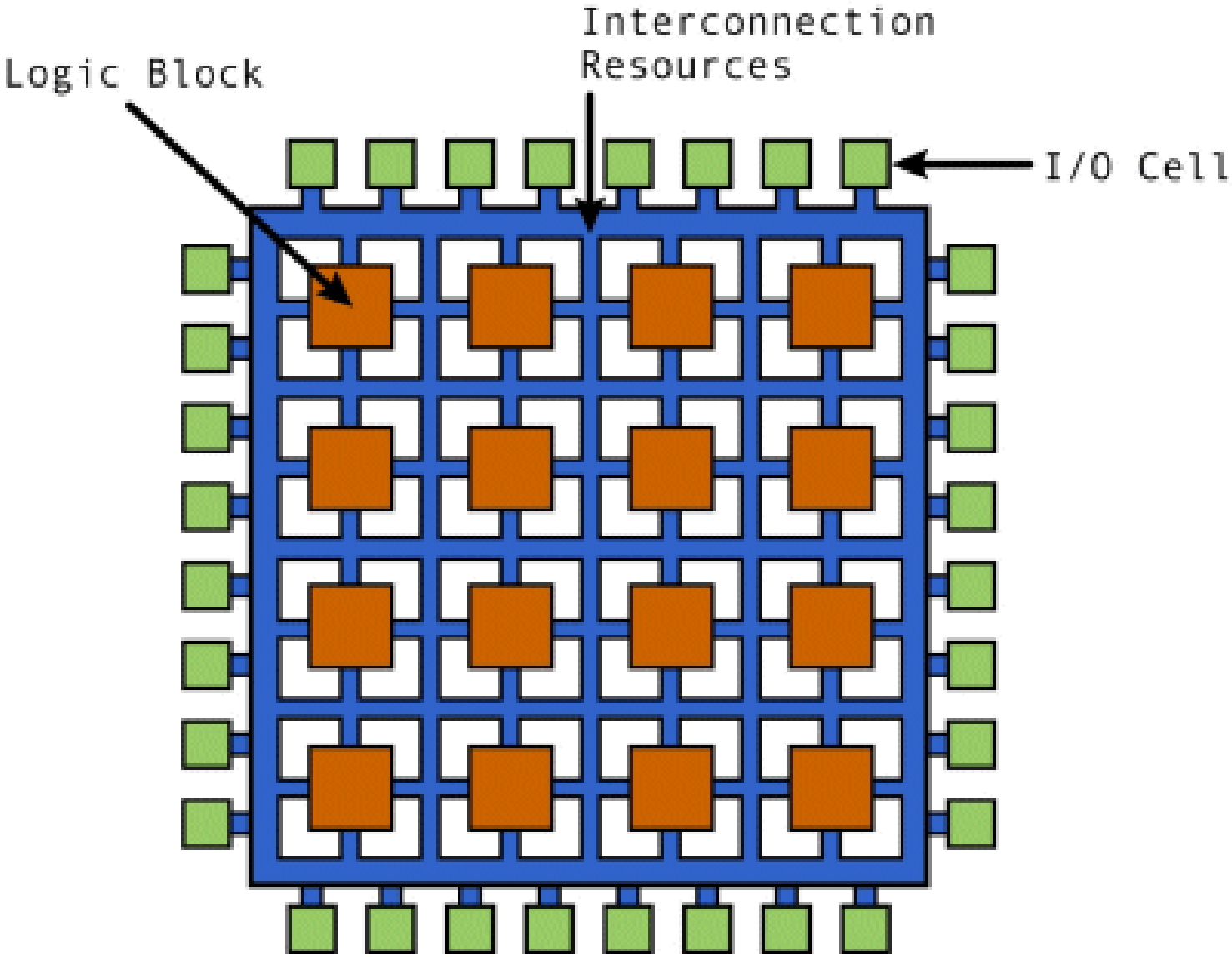


Fig. Simple architecture of a CPLD.



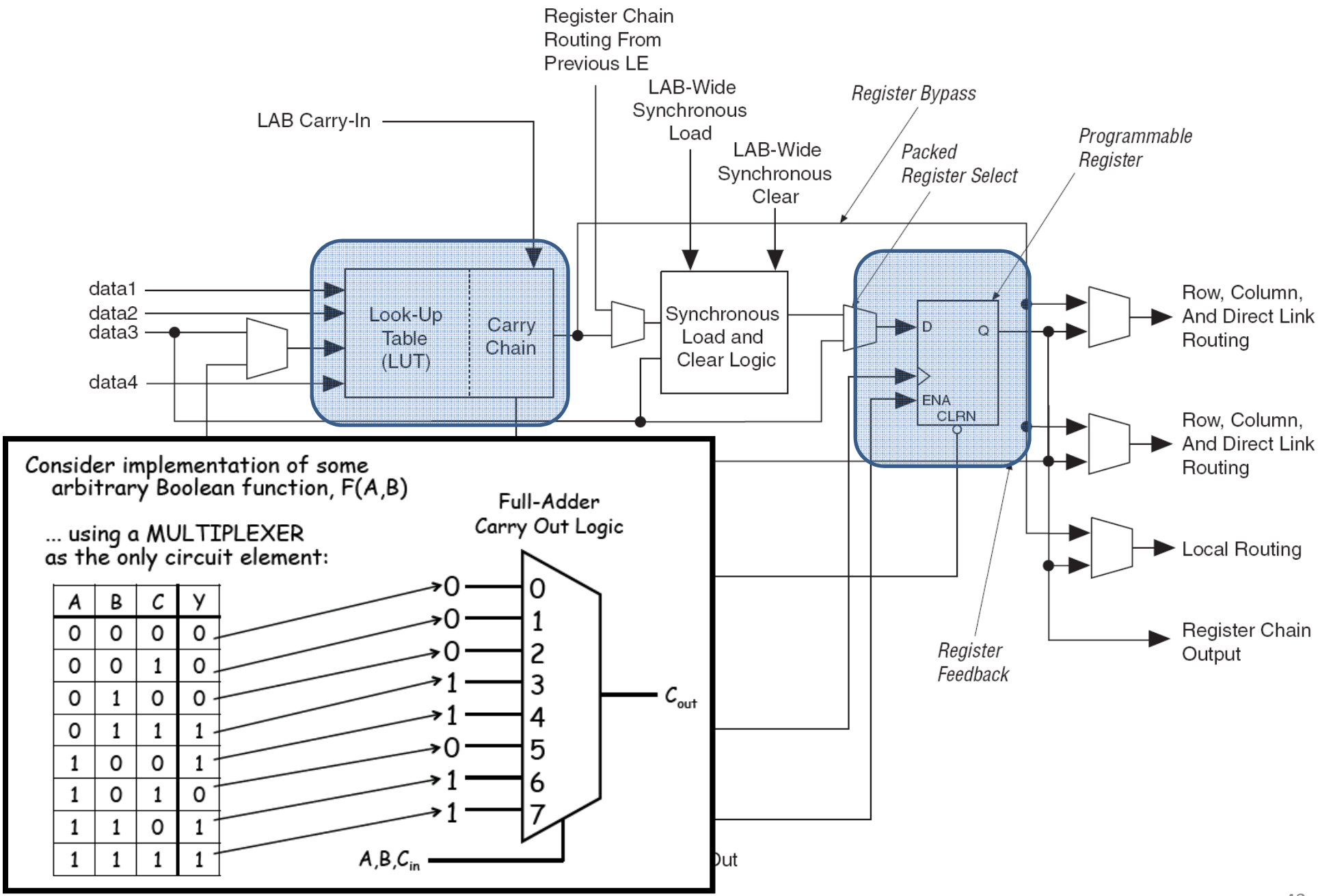
FPGA

Field Programmable Gate Array:
Set of Chips in a bread board.
Control:
Chips functions
Connections



Logic elements

Figure 2–2. Cyclone II LE



HDL

Hardware Description Language

A Tale of Two HDLs

VHDL

ADA-like verbose syntax, lots of redundancy (which can be good!)

Extensible types and simulation engine. Logic representations are not built in and have evolved with time (IEEE-1164).

Design is composed of entities each of which can have multiple architectures. A configuration chooses what architecture is used for a given instance of an entity.

Behavioral, dataflow and structural modeling. Synthesizable subset...

Harder to learn and use, not technology-specific, DoD mandate

Verilog

C-like concise syntax

Built-in types and logic representations. Oddly, this led to slightly incompatible simulators from different vendors.

Design is composed of modules.

Behavioral, dataflow and structural modeling. Synthesizable subset...

Easy to learn and use, fast simulation, good for hardware design

Programação de microprocessadores
(assembly, c++, etc.)

Dar ao "gnomo" uma lista (consecutiva) de operações a realizar

1 Máquina executa várias tarefas consecutivas

Programação de Lógica Digital
(FPGAs+HDL)

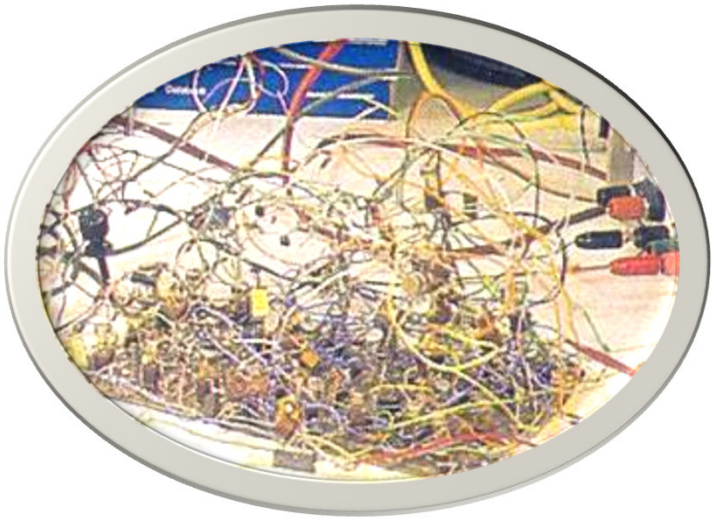
Dar ao gnomo uma lista de "objectos" para construir

Várias Máquinas executam várias tarefas em paralelo

Em linguagens de alto nível por vezes confundem-se

Remember:

In FPGAs there isn't:
Do this, after this, then that..



\+ HDL code


STOP