

Sistemas Digitales

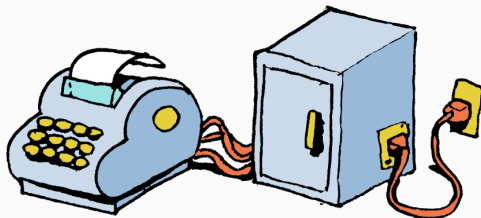
Introducción

Primer Cuatrimestre 2024

Sistemas Digitales
DC - UBA

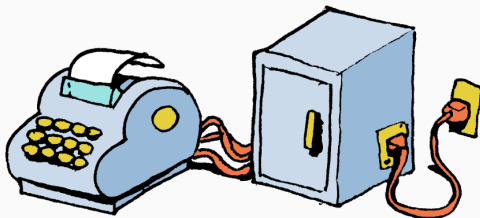
Motivación

El objetivo de Sistemas Digitales es que puedan comprender los principios de diseño y funcionamiento de:



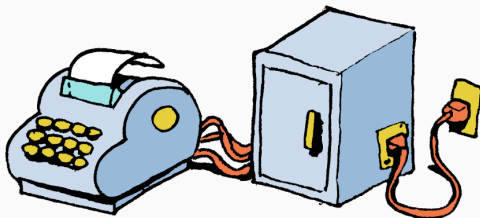
El objetivo de Sistemas Digitales es que puedan comprender los principios de diseño y funcionamiento de:

- **La arquitectura de una computadora**, o sea, cómo se programa en el lenguaje que la máquina interpreta.



El objetivo de Sistemas Digitales es que puedan comprender los principios de diseño y funcionamiento de:

- **La arquitectura de una computadora**, o sea, cómo se programa en el lenguaje que la máquina interpreta.
- **Su microarquitectura**, o sea, cómo se construye una computadora que pueda interpretar un programa.



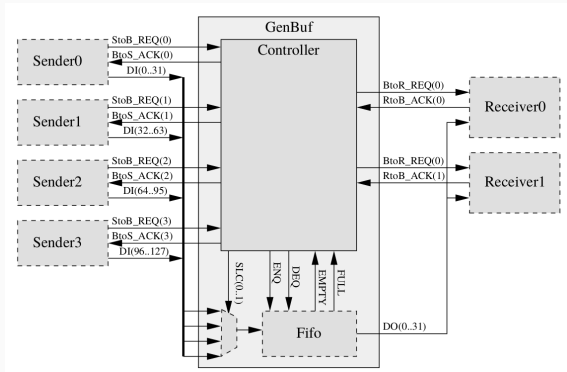
Hardware - su práctica

Al igual que el software, la práctica del diseño y la implementación del hardware depende del siguiente grupo de actividades:

Diseño → Especificación → Implementación → Validación → Verificación

Del diseño a la implementación tenemos:

Diseño Descripción esquemática de los componentes de un sistema y cómo se conectan sus interfaces.



Del diseño a la implementación tenemos:

Especificación Descripción del comportamiento esperado en términos de presunciones y garantías en términos de sus señales y estado (memoria).

G1	$\forall i : \text{always} (\text{StoS_REQ}(i) \rightarrow \text{eventually! BtoS_ACK}(i))$ $\forall i : \text{always} (\neg \text{StoS_REQ}(i) \rightarrow \text{eventually! } \neg \text{BtoS_ACK}(i))$
G2	$\forall i : \text{always} (\text{rose}(\text{StoS_REQ}(i)) \rightarrow \neg \text{BtoS_ACK}(i))$
G3	$\forall i : \text{always} (\text{rose}(\text{BtoS_ACK}(i)) \rightarrow \text{prev}(\text{StoS_REQ}(i)))$
G4	$\forall i : \text{always} ((\text{BtoS_ACK}(i) \wedge \text{StoS_REQ}(i)) \rightarrow \text{next! BtoS_ACK}(i))$
A1	$\forall i : \text{always} (\text{StoS_REQ}(i) \wedge \neg \text{BtoS_ACK}(i) \rightarrow \text{next! StoS_REQ}(i))$ $\forall i : \text{always} (\text{BtoS_ACK}(i) \rightarrow \text{next! } \neg \text{StoS_REQ}(i))$
G5	$\forall i \forall i' \neq i : \text{always} \neg (\text{BtoS_ACK}(i) \wedge \text{BtoS_ACK}(i'))$
A2	$\forall j : \text{always} (\text{BtoR_REQ}(j) \rightarrow \text{eventually! RtoB_ACK}(j))$ $\forall j : \text{always} (\neg \text{BtoR_REQ}(j) \rightarrow \text{next! } \neg \text{RtoB_ACK}(j))$
A3	$\forall j : \text{always} (\text{BtoR_REQ}(j) \wedge \text{RtoB_ACK}(j) \rightarrow \text{next! RtoB_ACK}(j))$
A4	$\forall j : \text{always} (\text{RtoB_ACK}(j) \rightarrow \text{prev}(\text{BtoR_REQ}(j)))$

Del diseño a la implementación tenemos:

Descripción de comportamiento Una descripción del comportamiento del sistema en un lenguaje apropiado (HDL) de la que puede sintetizarse una implementación funcional.

SystemVerilog

```
module maindec(input  logic [6:0] op,
               output logic [1:0] ResultSrc,
               output logic      MemWrite,
               output logic      Branch, ALUSrc,
               output logic      RegWrite, Jump,
               output logic [1:0] ImmSrc,
               output logic [1:0] ALUOp);

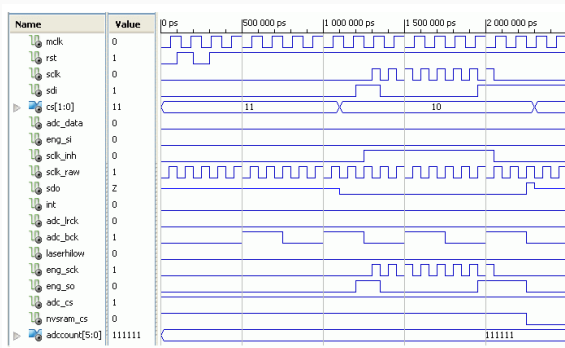
    logic [10:0] controls;

    assign {RegWrite, ImmSrc, ALUSrc, MemWrite,
           ResultSrc, Branch, ALUOp, Jump} = controls;

    always_comb
    case(op)
        // RegWrite_ImmSrc_ALUSrc_MemWrite_ResultSrc_Branch_ALUOp_Jump
        7'b0000011: controls = 11'b1_00_1_0_01_0_00_0; // lw
        7'b0100011: controls = 11'b0_01_1_1_00_0_00_0; // sw
        7'b0110011: controls = 11'b1_xx_0_0_00_0_10_0; // R-type
        7'b1100011: controls = 11'b0_10_0_0_00_1_01_0; // beq
        7'b0010011: controls = 11'b1_00_1_0_00_0_10_0; // I-type ALU
        7'b1101111: controls = 11'b1_11_0_0_10_0_00_1; // jal
        default:    controls = 11'bx_xx_x_x_xx_x_xx_x; // ???
    endcase
endmodule
```

Y en las etapas post implementativas:

Validación Conjunto de pruebas no exhaustivas que prueban el comportamiento de la especificación y/o implementación



Y en las etapas post implementativas:

Verificación Prueba de propiedades formales (con garantías basadas en algún mecanismo matemático) de la especificación y/o implementación

```
import "DwyerPatterns.spectra"

module ElevatorUnrealizable

type Floors = Int(0..3);
sys Floors elevatorLocation;
env Floors request;

gar startOnGroundFloor:
  elevatorLocation=0;

gar moveOneFloorAtATime:
  G (next(elevatorLocation) = elevatorLocation+1) |
    (next(elevatorLocation) = elevatorLocation-1);

gar eventuallyHandleOpenRequests:
  pRespondsToS(request != elevatorLocation,
               request=elevatorLocation);
```

En la práctica de la materia vamos a concentrarnos en las etapas de **diseño** y **implementación**, la primera al estudiar las arquitecturas existentes, sus motivaciones y prácticas comunes, y la segunda al construir un procesador de 32 bits sobre una arquitectura RISC V utilizando un lenguaje de descripción de hardware (HDL) llamado SystemVerilog. La ejecución de tests sobre los componentes con los que vamos a trabajar servirán como una instancia de **validación**.

Recordemos que queremos diseñar e implementar un procesador.
Para el alcance de la materia vamos a analizar:

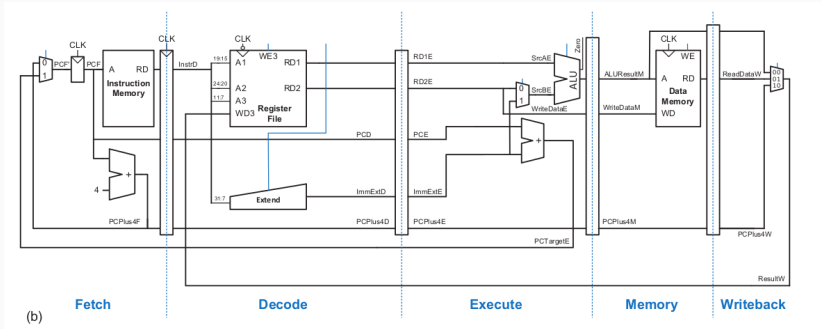
Recordemos que queremos diseñar e implementar un procesador.
Para el alcance de la materia vamos a analizar:

- **La arquitectura:** Set de instrucciones, registros, memoria (la interfaz a la que accede quien vaya a programar el procesador).

Recordemos que queremos diseñar e implementar un procesador.
Para el alcance de la materia vamos a analizar:

- **La arquitectura:** Set de instrucciones, registros, memoria (la interfaz a la que accede quien vaya a programar el procesador).
- **La microarquitectura:** Implementación de los componentes, camino de datos, lógica de control (implementación en un soporte electrónico).

op	funct3	funct7	Type	Instruction	Description	Operation
0000011 (3)	000	-	I	lb rd, imm(rs1)	load byte	rd = SignExt([Address] _{7:0})
0000011 (3)	001	-	I	lh rd, imm(rs1)	load half	rd = SignExt([Address] _{15:0})
0000011 (3)	010	-	I	lw rd, imm(rs1)	load word	rd = [Address] _{31:0}
0000011 (3)	100	-	I	lbu rd, imm(rs1)	load byte unsigned	rd = ZeroExt([Address] _{7:0})
0000011 (3)	101	-	I	lhu rd, imm(rs1)	load half unsigned	rd = ZeroExt([Address] _{15:0})
0010011 (19)	000	-	I	addi rd, rs1, imm	add immediate	rd = rs1 + SignExt(imm)
0010011 (19)	001	0000000	I	slli rd, rs1, uimm	shift left logical immediate	rd = rs1 << uimm
0010011 (19)	010	-	I	slti rd, rs1, imm	set less than immediate	rd = (rs1 < SignExt(imm))
0010011 (19)	011	-	I	sltiu rd, rs1, imm	set less than imm. unsigned	rd = (rs1 < SignExt(imm))
0010011 (19)	100	-	I	xori rd, rs1, imm	xor immediate	rd = rs1 ^ SignExt(imm)
0010011 (19)	101	0000000	I	srii rd, rs1, uimm	shift right logical immediate	rd = rs1 >> uimm
0010011 (19)	101	0100000	I	srafi rd, rs1, uimm	shift right arithmetic imm.	rd = rs1 >>> uimm
0010011 (19)	110	-	I	ori rd, rs1, imm	or immediate	rd = rs1 SignExt(imm)
0010011 (19)	111	-	I	andi rd, rs1, imm	and immediate	rd = rs1 & SignExt(imm)
0010111 (23)	-	-	U	uiopcd rd, upimm	add upper immediate to PC	rd = {upimm, 12'b0} + PC
0100011 (35)	000	-	S	sb rs2, imm(rs1)	store byte	[Address] _{7:0} = rs2 _{7:0}
0100011 (35)	001	-	S	sh rs2, imm(rs1)	store half	[Address] _{15:0} = rs2 _{15:0}
0100011 (35)	010	-	S	sw rs2, imm(rs1)	store word	[Address] _{31:0} = rs2
0110011 (51)	000	0000000	R	add rd, rs1, rs2	add	rd = rs1 + rs2
0110011 (51)	000	0100000	R	sub rd, rs1, rs2	sub	rd = rs1 - rs2
0110011 (51)	001	0000000	R	sll rd, rs1, rs2	shift left logical	rd = rs1 << rs2 _{4:0}
0110011 (51)	010	0000000	R	slt rd, rs1, rs2	set less than	rd = (rs1 < rs2)
0110011 (51)	011	0000000	R	sltu rd, rs1, rs2	set less than unsigned	rd = (rs1 < rs2)
0110011 (51)	100	0000000	R	xor rd, rs1, rs2	xor	rd = rs1 ^ rs2
0110011 (51)	101	0000000	R	srii rd, rs1, rs2	shift right logical	rd = rs1 >>> rs2 _{4:0}
0110011 (51)	101	0100000	R	sra rd, rs1, rs2	shift right arithmetic	rd = rs1 >>> rs2 _{4:0}
0110011 (51)	110	0000000	R	or rd, rs1, rs2	or	rd = rs1 rs2
0110011 (51)	111	0000000	R	and rd, rs1, rs2	and	rd = rs1 & rs2
0110111 (55)	-	-	U	lui rd, upimm	load upper immediate	rd = {upimm, 12'b0}
1100011 (99)	000	-	B	beq rs1, rs2, label	branch if =	if (rs1 == rs2) PC = BTA
1100011 (99)	001	-	B	bne rs1, rs2, label	branch if ≠	if (rs1 ≠ rs2) PC = BTA
1100011 (99)	100	-	B	blt rs1, rs2, label	branch if <	if (rs1 < rs2) PC = BTA
1100011 (99)	101	-	B	bge rs1, rs2, label	branch if ≥	if (rs1 ≥ rs2) PC = BTA
1100011 (99)	110	-	B	bltu rs1, rs2, label	branch if < unsigned	if (rs1 < rs2) PC = BTA
1100011 (99)	111	-	B	bgeu rs1, rs2, label	branch if ≥ unsigned	if (rs1 ≥ rs2) PC = BTA
1100111 (103)	000	-	I	jair rd, rs1, imm	jump and link register	PC = rs1 + SignExt(imm), rd = PC + 4
1101111 (111)	-	-	J	jal rd, label	jump and link	PC = JTA, rd = PC + 4



Contenidos de la materia

Los contenidos que vamos a presentar durante las clases son los siguientes:

Lógica Combinatoria y Secuencial

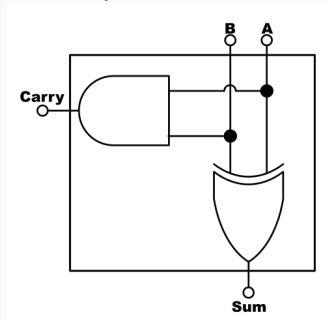
Diseño de un set de instrucciones

Arquitectura

Lenguajes de descripción de hardware

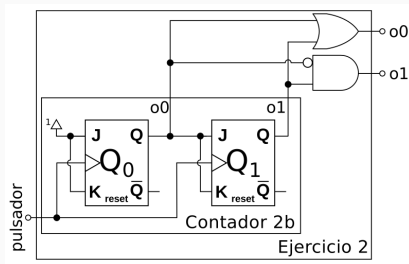
Microarquitectura

Lógica Combinatoria presenta los principios fundamentales para construir circuitos que implementen en un soporte electrónico la



semántica de la lógica proposicional

Lógica Secuencial introduce los elementos básicos para mantener el valor de un dato a lo largo del tiempo, los mecanismos de sincronización de circuitos y junto con estos la capacidad y técnicas que nos permiten descomponer e implementar un cómputo complejo en una secuencia de pasos atómicos



La **arquitectura** va a definir qué instrucciones va a poder ejecutar el procesador, sobre qué registros vamos a poder trabajar y cómo se accede a la memoria.

Los **lenguajes de descripción de hardware** son la forma en la que se describe hardware a nivel industrial y vamos a utilizarlos para construir nuestro procesador.

La **microprogramación** va a describir la forma en que nuestros componentes síncronos interactúan para implementar las operaciones descritas en el set de instrucciones previamente definido.

Estructura de la materia

Las clases van a dividirse de la siguiente manera:

Clases expositivas donde vamos a presentar los temas previamente mencionados.

Ejercicios en el campus que van a tener que completar luego de las clases expositivas (tienen una semana para hacerlo).

Talleres a partir de los cuales vamos a ir definiendo y probando, en grupos de tres personas y en un lenguaje visual (Logisim) o de especificación de hardware (HDL), los componentes necesarios para construir nuestro microprocesador

Un evaluación escrita donde van a tener que diseñar, analizar y/o implementar algunos elementos de una arquitectura o microarquitectura.

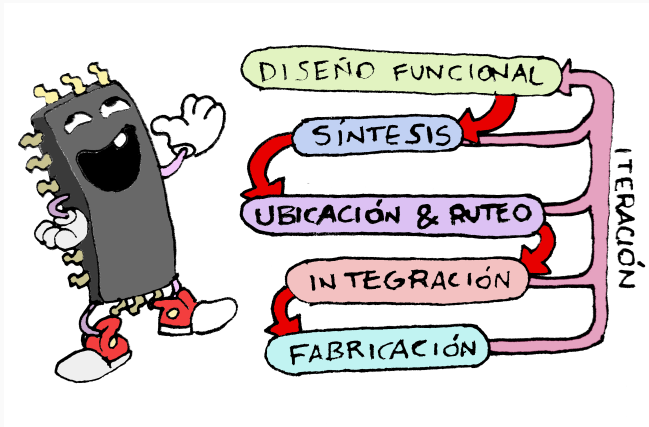
Las clases son de asistencia obligatoria y para justificar una ausencia deberán presentar la documentación relacionada. Para aprobar la materia deben:

Completar los ejercicios en el campus con la mitad o más de las preguntas respondidas correctamente.

Aprobar los talleres presentados en los laboratorios entregándolos en el día en que se presentan o, a lo sumo, en la próxima fecha de taller.

Aprobar la evaluación de cierre en la instancia inicial o de recuperatorio. Con un puntaje igual o mayor a 8 pueden promocionar la materia.

HDL



Del diseño a la implementación tenemos:
Un lenguaje de descripción
de hardware, o HDL por
sus siglas en inglés
(hardware description
language) es un lenguaje
que describe la estructura
y el comportamiento de un
circuito digital. Los dos
lenguajes más utilizados
en la industria son VHDL
y Verilog. En Sistemas
Digitales vamos a usar
Verilog.

SystemVerilog

```
module maindec(input  logic [6:0] op,
                output logic [1:0] ResultSrc,
                output logic      MemWrite,
                output logic      Branch, ALUSrc,
                output logic      RegWrite, Jump,
                output logic [1:0] ImmSrc,
                output logic [1:0] ALUOp);
    logic [10:0] controls;

    assign {RegWrite, ImmSrc, ALUSrc, MemWrite,
           ResultSrc, Branch, ALUOp, Jump} = controls;

    always_comb
        case(op)
            // RegWrite_ImmSrc_ALUSrc_MemWrite_ResultSrc_Branch_ALUOp_Jump
            7'b0000011: controls = 11'b1_00_1_0_01_0_00_0; // lw
            7'b0100011: controls = 11'b0_01_1_1_00_0_00_0; // sw
            7'b0110011: controls = 11'b1_xx_0_0_00_0_10_0; // R-type
            7'b1100011: controls = 11'b0_10_0_0_00_1_01_0; // beq
            7'b0010011: controls = 11'b1_00_1_0_00_0_10_0; // I-type ALU
            7'b1101111: controls = 11'b1_11_0_0_10_0_00_1; // jal
            default:    controls = 11'bx_xx_x_x_xx_x_xx_x; // ???
        endcase
endmodule
```

Cierre

En esta presentación vimos:

Motivación Necesidad de modelar, implementar y computar.

Hardware - su práctica Diseño, especificación, implementación, validación y verificación.

Contenidos de la materia Lógica combinatoria y secuencial, arquitectura, HDL, microarquitectura.

Estructura de la materia Clases, talleres, evaluación.

HDL Fases de fabricación del hardware.