1 COMPUTER ORGANIZATION AND ARCHITECTURE

Computer architecture embodies two perspectives:

1. Instruction Set Architecture:
   - Functional Behavior - What instructions are (to be) provided
   - Instruction Representation - How instructions are represented in binary
   - Data Representation - How data is (to be) represented.
2. Hardware Organization:
   - Datapath - the components that, collectively, are required to provide the functions specified by an instruction set.
   - Controller: the components that collectively, interpret the machine instructions and determine what subset of components in the datapath should be enabled to perform their various functions.
   - Internal Interface Organization - the interconnections required between the components within the datapath and controller, and the connections between the datapath and controller themselves.
   - External Interface Organization - the interconnections required between the CPU and other components of a computer system, including the memory, and various I/O interfaces and their associated devices.

2 FORMAL SPECIFICATION OF DIGITAL SYSTEMS

A *digital system* is any system that can manipulate discrete elements of information. Given an "informal" description of the system requirements, it is necessary to identify more precisely the following aspects of the digital system before a formal design can be undertaken:

- All possible inputs that might be delivered to the system;
- All possible outputs that might be produced by the system;
- The need for memory to be provided – if so, how much?
- How the outputs will be determined by the inputs and the current contents of memory;
- How memory is updated.

These details can be specified in a number of different ways. To construct a satisfactory system, any such specification must provide a description that satisfies the following requirements:

- There is never more than one possible result for each given input and memory configuration. Such a system is said to be *unambiguous*.

- A result is defined for every possible valid input sequence and memory configuration. In such cases, the system is said to be *complete*.

There are different ways of providing formal specifications that satisfy these requirements. However, all digital systems can be defined in terms of three sets and two functions. Such a definition provides an "abstract" or *Mathematical Model.* Specifically, a digital system can be defined by:

1. An Input Set (I) : The set of values that can be input.
2. An Output Set (O) : Range of values that can occur as outputs.
3. A set of States (S) : The set of all values that can be assigned to the collective memory elements of the machine.
4. An Output Function $(f : S \times I \longrightarrow O)$: The set of transformations performed by the system (NOTE that an output depends on both the input and the current contents of memory).
5. A State Transition Function $(g : S \times I \longrightarrow S)$: defines how memory is changed as a function of the current inputs and what was previously stored in memory.

A digital system is said to be a *combinational system* if its outputs at any instant depend only on the current value of the inputs. That is, no memory is required. Otherwise it is called a *sequential system*

A *combinational system* is defined by:

1. An Input Set (I) : The set of values that can be input.
2. An Output Set (O) : A set of values that can occur as outputs.
3. An Output Function $(h : I \longrightarrow O)$: The set of transformations performed by the system. The function F can be defined by a function table, algebraic expression, algorithm, or other notation that completely and unambiquiously captures the desired behavior (hence the term 'behavioral description')

A digital system is called a *sequential system* if its output at any instant depends on the current value of the inputs **and the current content of all memory elements**, collectively called the *state* of the machine.

EXAMPLE 1.1: A synchronous, positive edge-triggered digital system that generates the sequence 1, 0, 0, 0, 1, 0, 0, 0, 1, ... Each bit value is output on every rising edge of input, C, provided the input, *up*, is asserted.

Input Set, I : The inputs consist of all valid sequences that can be assigned to the pair (C, up). Since the device is only enabled on the positive edge transition of C (denoted by "↑"), there are two possible valid input pairs: $(\uparrow, 0)$ and $(\uparrow, 1)$.

Output Set, O : Only one bit-value is output at a time, so the possible outputs are '0', or '1'.

State Set, S : The purpose of memory is to remember how many times up was asserted (set to '1') since '1' was last output. This can be 0, 1, 2, or 3 times before a '1' must be again output. Therefore $S = \{0, 1, 2, 3\}$.

Output Function, $f : S \times I \rightarrow O$: This function can be expressed as a table that lists the output for each combination of state and input value:

	INPUT	
STATE	$(\uparrow, 0)$	$(\uparrow, 1)$
0	1	0
1	0	0
2	0	0
3	0	1

State Transition Function, $g : S \times I \rightarrow S$: This function can be expressed as table a table that lists the next state for each combination of current state and input:

	INPUT	
STATE	$(\uparrow, 0)$	$(\uparrow, 1)$
0	0	1
1	1	2
2	2	3
3	3	0

2.1 Behavioral Descriptions

Mathematical models of digital systems are examples of "behavioral descriptions". A *behavioral description* is a formal representation of a digital system that defines, either explicitly or implicitly, the inputs, outputs, and states and describes the functions performed by the system. It does not provide any information on the internal structure of the system.

Any description that provides information sufficient to identify the three sets and two functions can be considered a behavioral description. For example, a common way to express a behavioral description is in two parts:

- An *entity definition* that defines the input and output sets.
- A *functional specification* that defines the output function, and if memory is involved, the state set and the state-transition function.

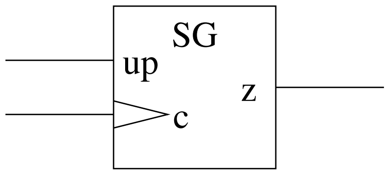
This can be expressed graphically, as follows:

1. An entity definition is often represented by a graphic symbol or 'black-box' diagram, called the "entity", with input and output ports labelled so that it can be used as a component in a logic diagaram.
2. The 'behavior' of the device represented by the entity can be provided in a variety of ways that include:
    - algebraic expressions.
    - function table (set of truth tables) itemizing the observed outputs for every possible input (combinational descriptions only)
    - characteristic / output table.
    - state-transition or state-machine diagram (SMD).

- function select table. This is a table where only the values of the control variables identify each row of the table, and the output is defined by an algebraic expression in terms of the data inputs, rather than by an explicit value.
- an algorithm that describes a computational procedure for obtaining the outputs. This algorithm may be expressed as an state machine diagram (SMD), or in a hardware description language (eg. VHDL).

EXAMPLE 1.2: Example 1.1 expressed as a "graphical representation":

Entity Definition:



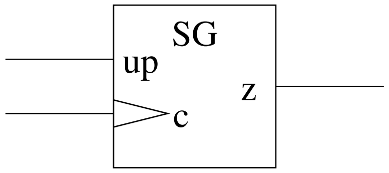
Functional Specification:

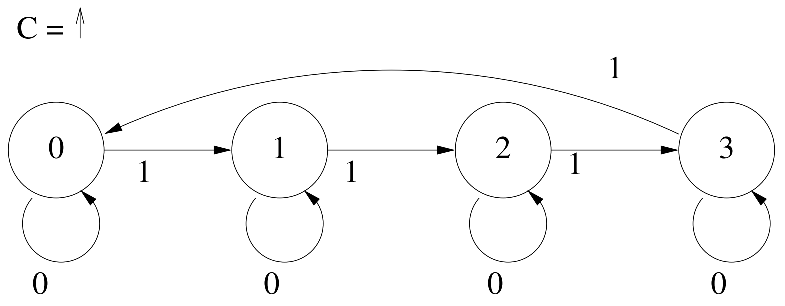
1. As a characteristic / output table :

Current State	C	up	Next State	z
0	↑	0	0	1
0	↑	1	1	0
1	↑	0	1	0
1	↑	1	2	0
2	↑	0	2	0
2	↑	1	3	0
3	↑	0	3	0
3	↑	1	0	1

2. As a State Transition Diagram :

Each transition arrow is labelled by the binary sequences that will result in that particular transition occurring. The transition arrows are also labelled by a binary sequence that defines the output that should occur when the machine enters the state to which the transition arrow points.

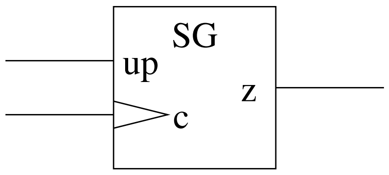




3. As a Function Select Table :

Rather than tabulate all possible assignments of values to the inputs for each state (as is done in the state-transition and characteristic tables), each row identifies descriptively what behavior should be observed for every possible assignment of values to the control inputs only.

To illustrate, Example 1.1 can be defined as follows:



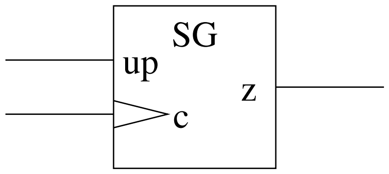
Ctrl Inputs		
C	up	Function
↑	0	$state^+ \leftarrow state$ if $state^+ = 0$ then $z = 1$ else $z = 0$
↑	1	$state^+ \leftarrow (state + 1)$ mod 4 if $state^+ = 0$ then $z = 1$ else $z = 0$

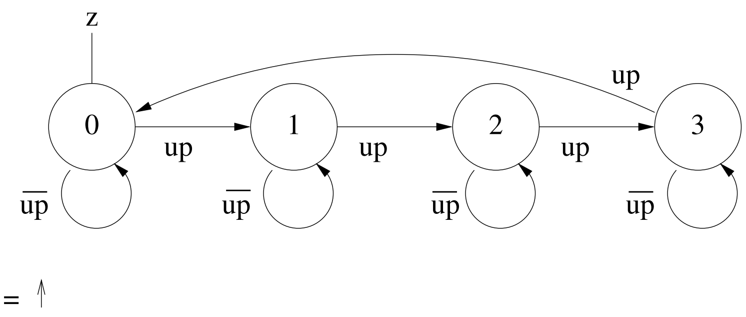
4. As a State Machine Diagram (SMD) :

SMD's are really just state transition diagrams where additional labels, called "Asserted Outputs" , are attached to the states, and where the inputs are defined by Boolean Expressions, called "Transition Conditions", rather than by binary sequences.

- Transition Condition: A Boolean function of the input variables with the property that it is "true" only for any assignment of values to the input variables that results in the transition along the "arrow" that is being labelled.
- Asserted Outputs: A list of all output variables that are to be asserted (that is, set to '1'). By inference, those variables not included in the list are assumed to be set to '0'. As well the list includes any register transfers that are to take place in the datapath while the machine is in the state being labelled.

Example 1.1 can be expressed as follows:





5. In hardware description language (HDL) :

A hardware description language is one where the entity definition and the functional specification can be expressed in formal textual notation, much as algorithms can be expressed in programming languages.

Example 1.1 can be expressed in the hardware description language, VHDL, as follows:

```
entity SG is
   port(C, up : in std_logic;
        z : out std_logic);
end SG;

architecture behav of SG is
begin
   process is
   variable state : integer := 0;
   begin
      -- State Transition Function:
      if C = '1' then
         if up = '1' then
            state := (state + 1) mod 4;
         end if;
      end if;
      -- Output Function
      if state = 0 then z <= '1';
      else z <= '0';
      end if;
      wait on C;
   end process;
end behav;
```

3 VHDL: A NOTATION SYSTEM FOR HARDWARE MODELS

VHDL is a language for describing electronic systems, either in terms of their behavior (hence called behavioral models) or in terms of their components and their interconnections (and so called structural models).

VHDL is the abbreviation for *VHSIC Hardware Description Language*, where VHSIC stands for 'Very High Speed Integrated Circuit.' Some of the features of VHDL include:

- The ability to specify structural descriptions;
- The ability to specify behavioral descriptions;
- The ability to simulate systems so described before fabrication, thus reducing the time and costs of development;
- The ability to construct designs that can be simulated using components whose structural descriptions have not been determined, thus permitting earlier strategic design decisions to be made.
- The ability to provide specifications that can guide fabricating hardware for the actual construction of physical components corresponding to the the structural description provided.

4 MODELING DIGITAL SYSTEMS IN VHDL

4.1 Basic VHDL Concepts

1. Every digital system is defined by an `entity` declaration and an `architecture` declaration. The body of the entity declaration consists of a `port` statement that identifies the inputs and outputs.

   For the sequence generator example above, the corresponding entity definition is:

   ```
   entity SG is
      port(C, up : in std_logic;
            z : out std_logic);
   end SG;
   ```

2. There are two kinds of variables:

   - *Ordinary variables*: provide labels for computed values. They are introduced through "`variable`" declaration statements. The symbol "`state`" in the example above is an instance of an ordinary variable.
   - *Signal variables*: model ports and signal lines of a digital system. One way they are introduced is through the `port` statement of an entity definition. The symbols "`up`" , "`c`", and"`z`" in the entity definition above are examples of signal variables.

3. The data type of signal and ordinary variables are defined in their declaration statements. Among the valid scalar types are "`bit`", "`boolean`", "`natural`", and "`integer`". Binary sequences, called "`bit_vector`"s can also be declared.

4. There are two kinds of assignment statements:

   - *Ordinary assignments*: are made using the operator symbol "`:=`". The value of the expression on the right-hand side is immediately assigned to the ordeinary variable on the left-hand side,.

   - *Signal assignments*: are made using the signal assignment operator "`<=`". The value of the expression on the right-hand side is NOT immediately assigned to the signal variable on the left-hand side. Rather, an "event" is established for completing this assignment at some time in the future.

5. "`Time`" is a valid data type that is used to represent the dynamic properties of a digital system. In particular, "propagation delay" and "latency" are two terms used in conjunction with the dynamic behavior of digital systems:

   Propagation Delay, t_{pd} : The time required for the output of a digital system to be affected by a change in input to that system.

   Latency : The minimum time required to guarantee that a system can complete its task regardless of the input.

   The "latency" of a digital system is its longest propagation delay.

   VHDL provides the following time units:

	time	unit
1 sec	1000 ms	(milliseconds)
1 ms	1000 us	(microseconds)
1 us	1000 ns	(nanoseconds)
1 ns	1000 ps	(pecoseconds)
1 ps	1000 fs	(feptoseconds)

4.2 The std_logic Data Type

Since signal lines can exhibit other electrical conditions than logic-0 (0v) and logic-1 (5v), such as 'uninitialized', 'high impedance', 'weak 0 (~1v)', 'weak 1 (~4v)', etc., the data type `bit` is not a realistic choice for modelling the signal values in digital systems. It is therefore useful to adopt a 9 value data type called *std_ulogic*, that includes the following symbols as values: {0, 1, X, U, Z, W, L, H, -}. Since it is usually undesirable to define digital systems with 'don't care' values on its signal lines, the 8 valued data type *std_logic* is more commonly used and includes all the symbols except '-'. A VHDL model for which the signal value '-' is eliminated, is said to be *resolved.*

The values that a variable of type `std_logic` can assume can be expressed as character constants. These are denoted by enclosing the appropriate symbol in single quotes.

The data type, *std_logic_vector*, is used to represent sequences of `std_logic` values. When declared, a variable with type `std_logic_vector` must include a range that specifies the bit-position of the most significant bit (MSB) and the least significant bit (LSB).

For example the following declaration defines `word` to be an ordinary variable whose type is a 32-bit std_logic_vector, with the MSB = 31, and the LSB = 0:

variable word : std_logic_vector(31 downto 0);

4.3 Libraries

In order for the logical functions, `and`, `or`, `not`, `xor`,..., to be applicable to `std_logic` values as well as `bit` values (i.e, '0' and '1'), the definitions of these functions must be overloaded. This is achieved by importing overloaded definitions from the pre-defined `IEEE` library, and indicating that the `std_logic_1164` definitions are to be used. This is accomplished by including the following statements at the top of your source file:

```
library IEEE;
use IEEE.std_logic_1164.all;
```

The "sequence generator (SG) example above was written with the "std_logic" data type declarations for the input and output ports.

NOTE 1: Each entity is normally defined in its own source file, and the filename should be the same as the entity name. When compiled, unless otherwise directed, these entities are placed in a default library called `work`. To access these previously compiled definitions, the following directive should also be included at the top of your source file:

```
use work.all;
```

NOTE 2: Most VHDL compilers and simulators assume the existence of the 'work' library and so the 'library work' declaration not required.