Electrical and Computer Engineering

Digital Systems Projects

By Prawat Nagvajara

Synopsis
A set of notes on projects for learning the fundamentals of digital systems – concepts, design and implementation.

I. Introduction to Digital Systems

From a perspective of an electrical and computer engineer (this author) a digital system deals with discrete quantities where measure of time and signal values are whole numbers (natural numbers, integers, fractions and floating-point representation). Digital circuits considered in this discussion are made up of analog circuits where the transistors function as switches realizing a mapping

$$F: x \rightarrow z,\ x \in V^n,\ z \in \{0, 1\}, \qquad (1)$$

where, V^n denotes n-dimensional vector space over {0, 1}. A digital circuit implementing F is called a combinational circuit.

Vector Space over {0, 1}
The vector $x \in V^n$ in (1) belongs in an n-dimensional vector space over {0, 1}, i.e., x is an n-bit vector. The data type commonly used in Hardware Description Language (HDL) is the standard logic vector std_logic_vector which is an array of the std_logic data type.

The std_logic is the data type for digital signals defined in the IEEE 1164 Standard. It is a set of 9 characters {'0',' 1', 'X', 'U', 'Z',' -', 'H', 'L', 'W'} which correspond to 0, 1, cannot be determined (unknown), signal has not been assigned, high impedance, don't care condition, weak 0 value, weak 1 value, and weak unknown value [1],[2]. Design and simulation of digital circuits uses these values. Simulation allows designer to verify the correctness of the design HDL description. The values '0', '1', 'X', 'U', 'Z' are more common in simulation for the correctness of the functional behavior of the design. Character data type constants are between single quotes in languages, e.g. 0 and 1 are integers whereas '0' and '1' are characters. In HDL code '0' and '1' are binary numbers.

No Multiple Drivers
The std_logic is a resolved type. Basically, codes with unresolved type signals cannot contain signals with multiple drivers (e.g., a wire connecting directly to multiple sources). The compiler will flag as error. With the resolved type such as the std_logic multiple-driven signal code will compile. However, synthesis tools generating hardware from an HDL description do not allow multiple drivers. They are design errors. Resolution circuits such as multiplexor resolves multiple drivers by using control signal to select which driver to assign to the destination.

Numbers and std_logic_vector
In VHDL HDL designer declares an n-bit vector signal x by

```
signal x : std_logic_vector(n-1 downto 0);
```

The keyword "downto" means n-1, n-2, …, 0 are indices of the elements starting from the leftmost position down to the rightmost position.

The n-bit vector declared with downto indexing is equivalent to n-bit numbers, unsigned, signed and fraction, having the leftmost position as the

most significant position. For example, data types in C language, such as, int and u16 are signed 32-bit vector and unsigned 16-bit vector. Floating-point numbers, e.g., the float type in C is a 32-bit vector with a floating-point format.

Data in digital systems are enumerable. They are not real numbers (the continuum) like voltage and current in circuit analysis.

Dynamical Systems
For digital systems with dynamics, the states of the system change at discrete points in time and at regular intervals. The contents of the system data storages such as registers and Random Access Memory (RAM) are the states and they get updated at a discrete moments in time, for instance, synchronous to the system clock signal rising edges. The states of the systems are represented by bit vectors stored as contents of registers and memory.

Finite State Machines
Digital system realizes state transition mapping,

$$M: (s(t), x(t)) \rightarrow s(t+1),$$
$$s(t) \in V^n, x(t) \in V^k, t = 0, 1, \ldots \quad (2)$$

where s(t) is the state of the system and x(t) is the input to the system. The next state, s(t+1) is a function of the present state, s(t) and input x(t). M defines the next state mapping.

The output of state machine realizes a mapping

$$O: (s(t), x(t)) \rightarrow z(t),$$
$$s(t) \in V^n, x(t) \in V^k, z(t) \in V^m, t = 0, 1, \ldots \quad (3)$$

where, z(t) is the output of the system, a function of the present state and the input.

The collection {M, s(t), x(t), O, z(t)} forms a dynamical system called a finite state machine (the number of states is finite).

Registers store data which are the system state variables. They operate synchronously with a clock signal rising-edges. A clock signal is a periodic square wave having high voltage value (e.g., 3.3 V) for half of the period and low voltage value (0 V known as digital ground) half of the period. The reciprocal of the period is the frequency in Hz. For example, with 1GHz clock signal the period is 1ns. Note that, in 1ns light travels about a foot or 30cm. The registers store or load their new input vectors on the clock rising-edge, in other words, they synch to the "clock beats".

Digital Artifacts of Analog Electronics
There are infinitely many voltage amplitudes between the high voltage (e.g., 3.3V) and the low voltage (0V), but the meaningful voltage values in the digital circuit representing '0' and '1' binary numbers are those which are close to the low and high voltages. How close the signal values to the ideal high value or low value, to be a valid '1' or '0' depends on the ability to which the signals turn on or off the transistors (switches) they are driving.

Today's (2015) digital systems (combinational circuits and storages) typically comprise of Field Effect Transistor (FET) switches. Claude Shannon in "A Symbolic Analysis of Relay and Switching Circuits," 1937 Master of Science thesis at the Massachusetts Institute of Technology, showed an analysis of electrical relay switch network (circuits) using the algebra of logic developed by Gorge Boole in 1854 "An Investigation of the Laws of Thoughts." Reader may find an Intro IT article on Boole and Shannon informative [4].

It from Bit
In 1947 Shannon's later work "A Mathematical Theory of Communication" started a study of Information Theory. Information is a foundation of Digital Systems.

Shannon proposed the entropy

$$H = -\Sigma P_i \text{ Log } P_i, \quad (4)$$

as a measure of the amount of information contained in a source emitting messages. The unit of the entropy is in bit per symbol, or just bit (for a transmitted message is in bit per second). It is the average amount of uncertainty per symbol, where P_i is the probability that event i will happen, the summation is over all events in the probability space. In communication the events are sequences of symbols of finite length (signals) generated by the source.

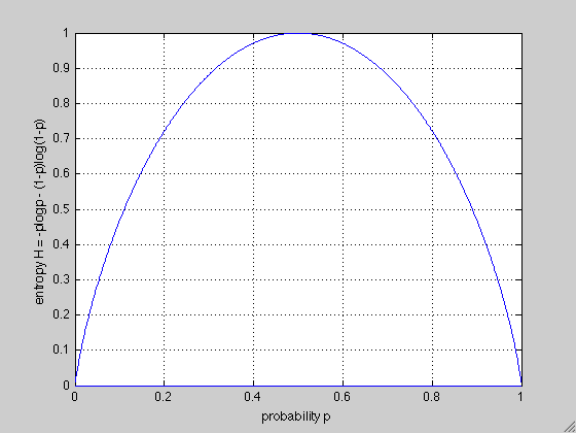

Fig. 1 Entropy H of one-bit Message (data)

Consider two-event probability space, that is, one-bit message. Let x, $x \in \{0,1\}$ be a random variable, with probability $P_0 = Pr[x=0] = 1-p$ and $P_1 = Pr[x=1] = p$. The entropy H is

$$H = P_0 + P_1 = -(1-p)\log(1-p) - p\log p \qquad (5)$$

H is maximum when p=0.5 and 0 when p=0 and p=1 (Fig. 1). One bit of information is maximum when the datum is random, in other words, maximum degree of disorder or surprises in terms of data transmission. The message x has zero information when p=0 or p=1 because the message is deterministic and need not be transmitted or an emitting source.

Information entropy is similar to the thermodynamic entropy, Boltzmann Entropy $S=k\log W$ ($-\Sigma P_i \text{ Log } P_i$). "S=klogW" was carved on Bozeman's headstone in Vienna. The entropy quantifies the Second Law of Thermodynamic which gives the direction of time. The entropy quantifies the amount of information the foundation of information theory, communication theory and digital systems. Information is the first ingredient, then space, mass and energy. "It from bit" Charles Archibald Wheeler, physicist and a pioneer in quantum information theory (Box 1).

> "*It from bit* symbolizes the idea that every item of the physical world has at bottom — at a very deep bottom, in most instances — an immaterial source and explanation; that what we call reality arises in the last analysis from the posing of yes-no questions and the registering of equipment-evoked responses; in short, that all things physical are information-theoretic in origin and this in a *participatory universe*"

Box 1. *It from bit,* Charles Archibald Wheeler 1989 [5]

Restoring Logic

Implementation of FET switching circuit typically uses the restoring logic for which the outputs of logic blocks always connect to the power rail or ground. The time duration for the states of the switches to change between off-state and on-state depends on the amount of time to charge or discharge the gate capacitors of the load FET transistors to which outputs are driving.

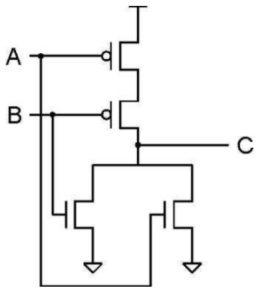

Fig. 2 CMOS 2-Input NOR Logic Block

Figure 2 shows a Complementary Metal Oxide Silicon (CMOS) circuit of a two-input NOR-gate. A transistor in digital circuits is a three-terminals device (source, gate and drain terminals). It operates as a switch where the gate voltage controls whether the transistor is conducting current between the source terminal and the ground terminal. In Fig. 2, the input A is connected to the gate terminal of a p-type transistor (the top structure) whose source terminal is connected to the V_{DD} (e.g., 3.3V power rail) and the drain terminal is connected to the source terminal of another p-type transistor. The input A is also connected to the gate terminal of the n-type transistor (the bottom structure) whose drain terminal is connected to the output terminal and the drain terminal of the p-type transistor. The source terminal of the n-type transistor is connected to ground. The connections for the input B are similar to the input A.

The p-type transistor conducts the current when the voltage at the gate terminal (gate voltage) is 0V and open otherwise. It is the opposite for the n-type transistor. The V_{DD} represents logic '1' and 0V represents logic '0'. The logic expressions for the output C are as follows:

In the top structure, the output C is connected to V_{DD} (logic '1') when not A and not B is true, that is,

$$C = (\text{not } A) \text{ and } (\text{not } B) \qquad (4)$$

In the bottom structure, the output C is connected ground when A or B is true, that is,

$$\text{not } C = A \text{ or } B \qquad (5)$$

By De Morgan's law, (1) and (2) are equal and the two expressions are consistent. The top structure supplies the V_{DD} when the output is '1' and the bottom structure supplies the 0V when the output is '0' logic. The output logic is,

$$C = A \text{ nor } B. \qquad (6)$$

The structure is complementary of p-type and n-type MOS FETs describing logic expression and its negation. The p-type FETs - top structure provides the connection (path) between the logic block outputs to power rail. The n-type FETs - bottom structure provides connection between the logic block outputs of ground. The outputs are connected to either the power (logic '1') or ground (logic '0') restoring the signals. Because the p-type and n-type structures are the complementary (duality) the output is either '1' or '0' - power or ground, hence, the name restoring logic.

Registering of the Equipment

Registers are basic building blocks in digital systems. They are storages of the states of the systems. They are synchronous to the discrete time steps (i.e., the clock signal rising-edges).

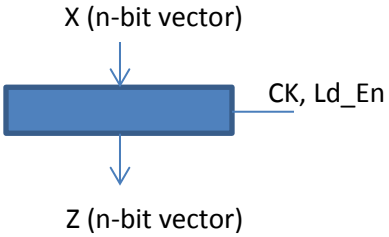

Fig. 3 Register Block Diagram

A register (Fig. 3) stores an n-bit vector as its content. The output Z is an n-bit vector wired to the storage elements (e.g., flip-flops). The input X becomes a new content, that is, X is loaded into the register, synchronous to the clock rising

edge and when the Load Enable (Ld_En) control signal is a '1'.

A VHDL description consists of the entity declaration and the architecture declaration and body. Box 2 shows VHDL entity declaration of the register in Fig. 3.

```
Library ieee;
use std_logic_1164.all;

entity reg is
generic (n: natural);
port (
x: in   std_logic_vector(n-1 downto 0);
z: out std_logic_vector(n-1 downto 0);
ck, Ld_en: in std_logic
);
end reg;
```

Box 2. Resister Entity Declaration

The code uses the ieee library and the standard logic package (std_logic_1164). The code first declares the generic parameter, in this case the number of bits n in the vector. This is similar to an expression such as $x \in V_n$ over {0, 1}, where n is a natural number. Natural is a data type in VHDL (see references on VHDL data types, e.g., [3] Chapter 3). The code next declares the input and output ports with their data types which are the ieee standards for bit vectors and bit. The input ck is the clock signal. The input ld_en is the load enable signal. The register assigns x to its content at the clock rising edge and the load enable is a '1'.

The architecture declaration and body contain the description of the register behavior (Box 3).

The architecture declaration (Box 3) contains an n-bit vector signal "temp" declaration. Temp will represent storage (e.g., flip-flops).

Architecture body starts after the "begin" and ends with "end Behavioral". "Behavioral" is a name used in this code any names would do.

```
architecture Behavioral of reg is

-- declare signal as storage
signal temp:
std_logic_vector(n-1 downto 0);

begin
-- synchronous process
process(CK)
begin
if ck='1' and ck'event then
  if ld_en = '1' then
     Temp <= x;
  end if;
end if;
end process;
-- wire temp to output port
z <= temp;
end Behavioral;
```

Box 3. Architecture Behavioral Description

The architecture body in Box 3 contains one process describing the register and one concurrent statement describing the register content (signal temp) to the output port signal z wiring. Upper or lower characters are interchangeable (not case sensitive). Comments are marked by double hyphens.

The process is a piece of code (thread) that represents a hardware module. Architecture typically contains many modules described using processes. They interact via signals. A process has a declaration area and body marked by "begin" and ends with "end process".

Designer codes a process by "process" followed by a list of signals inside the parentheses called "sensitivity list", (need not declare any prototypes). Execution of processes is event-driven. An event – a change in the value on one of the signals in the sensitivity list triggers execution of the code in the body starting from "begin" line by line serially until "end process" and the execution suspends until an event happens.

When the execution reaches “end process;” the process updates the signals with their new values. Signal assignments use an arrow “<=” where the less-than symbol is the arrowhead and the equal symbol is the shaft, setting the values of left-hand-side signals to be later substituted. Assignment of each signal can happen once during the interpretation from “begin” to “end process”, thus, when the process suspends, substitution of the signals old values to new values are concurrent.

In Box 3 there is only one signal in the sensitivity list of the process, the clock signal. Clock signal drives synchronous hardware in the system. The interpretation is serial. The execution begins when ck is an event. If ck is a one and ck is an event (ck’event returns TRUE) then if ld_en is a ‘1’ then temp gets the value of x when the process suspends.

The attribute ck’event reads ck tick event returns true if ck has just changed its value from ‘0’ to ‘1’ or from ‘1’ to ‘0’. The expression ck = ’1’ and ck’event evaluates to TRUE when a rising edge had just occurred. The ck’event is redundant since the clock signal is in the sensitivity list. It is used to indicate that the type of storage is an edge sensitive, e.g., flip-flops, as opposed to a level-sensitive latch.

```
process(ck)
begin
If ck = ‘1’ and ck’event then
    If reset = ‘1’ then
      Temp <= (others => ‘0’);
    Else
      if ld_en = '1' then
        Temp <= x;
      end if;
    End if;
End if;
End process;
```

Box 4. Reset Condition “if – then – else”

The “if ck=’1’ and ck’event then … end if;” is called a “clock fence”. There are similar phrases like a “reset fence” where a design has inside a clock fence a description for reset condition. For example, a process in Box 4 describes a synchronous reset condition which assigns the content of the register to a vector of all ‘0’s else the register operates as described in Box 3. The signal reset is an addition port in the entity declared as an std_logic along with the ports ck and ld_en.

VHDL syntax for assigning individual bits in a vector (array or string of characters) consists of individual assignments of the bit position followed by “=>” and the bit value, e.g., (n-1 => ‘1’, 2 => ‘1’, others => ‘0’) describes an n-bit vector with the (n-1) and 2 bits assigned to ‘1’ and the other bits to ‘0’, that is, “100…0100” (e.g., see Chapter 3 of [3]). Hence, (others => ‘0’) is equal to a vector of all ‘0’s.

The register described in Box 2, 3 and 4 involve a generic parameter n that is not bound to any value. To use copies of the n-bit register as components of another entity, designer instantiates the copies, binds (maps) a value to n and wires (maps) actual signals.

Design Hierarchy

The following design illustrates the concept of designing with imported components.

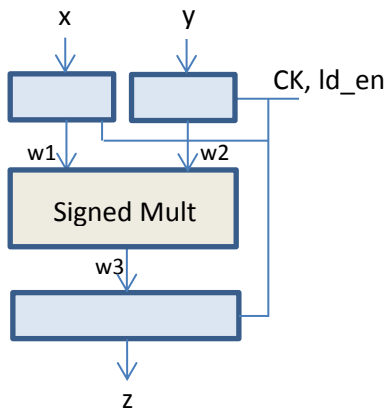

Fig. 4 Registers and Combination Circuit

The design shown in Fig. 4 comprises three copies of the register (Fig. 3) and a process describing the signed multiplier. A VHDL code for the design is as follows.

```
package reg_mult_pack is
constant n: natural := 4;
end reg_mult_pack;
```

```
----------------------------------------
-- Company: Drexel ECE
-- Engineer: Prawat
-- Module Name: reg_mult, signed multiplier
-- Description: inputs x, y buffered in Rx, Ry
-- output Rx*Ry buffered and wired to output z
----------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
-- reg_mult_pack package defines
-- constant n: natural := 4;
use work.reg_mult_pack.all;

entity reg_mult is
Port (
  x,y: in  std_logic_vector(n-1 downto 0);
rx,ry: out std_logic_vector(n-1 downto 0);
    z: out std_logic_vector(2*n - 1 downto 0);
ck,en: in  std_logic;
btnL,btnR: in std_logic
);
end reg_mult;
architecture Behavioral of reg_mult is
-- n-bit register
-- for input registers and
-- output register
component reg
generic (n: natural);
Port ( ck, ld_en: in std_logic;
 x: in  std_logic_vector(n-1 downto 0);
 z: out std_logic_vector(n-1 downto 0));
end component;

-- wires input registers outputs
signal w1, w2: std_logic_vector(n-1 downto 0);

-- signal for the multiply result and
-- input to the output register
signal w3: std_logic_vector(2*n-1 downto 0);

-- single step clock
signal ck_step: std_logic;
```

```
begin
-- instantiate registers
x_reg: reg generic map(n)
port map(x => x, z => w1,
        ck => ck_step, ld_en => en);
y_reg: reg generic map(n)
port map(x => y, z => w2,
        ck => ck_step, ld_en => en);
z_reg: reg generic map(2*n)
port map(x => w3, z => z,
        ck => ck_step, ld_en => en);

-- signed multiply module
process(w1,w2)
begin
w3 <= signed(w1) * signed(w2);
end process;

-- wiring w1 and w2 to Rx, Ry outputs
rx <= w1;
ry <= w2;

-- single step clock pulse (debounce)
-- btn0 to enter and btn1 to reset
-- ck_step signal drives components
process(ck)
type state is (not_rdy, rdy, pulse);
variable ns: state;
begin
if ck='1' and ck'event then
  case ns is
   when not_rdy => ck_step <= '0';
   if btnR = '1' then ns := rdy; end if;
   when rdy => ck_step <= '0';
   if btnL = '1' then ns := pulse; end if;
   when pulse => ck_step <= '1';
   ns := not_rdy;
   when others => null;
  end case;
end if;
end process;

end Behavioral;
```

The code starts with a package declaration containing a constant n equal to 4. The tool compiles this package into the default work library for the project. In the entity declaration reg_mult uses the package, “use work.reg_mult_pack.all”.

The library used is the ieee and packages are the 1164 standard, the standard logic arith and the standard logic signed for arithmetic functions and signed vectors (numbers).

The reg_mult entity consists of 2 input operands, x, y, and output z. The length of z is twice the operands length since z is x*y. The outputs rx and ry are the contents of the input registers. Last the entity has the ports btn0 and btn1 which are to be connected to push buttons on Field Programmable Gate Array (FPGA) prototype board (e.g., the Basys 3, Digilent Inc. [6]) during the implementation and verification.

Next, the architecture declaration phase contains component reg declaration (prototype) consistent with Box 2. It also contains 4 signal declarations w1, w2, w3 and ck_step.

The body of the architecture contains three instantiations x_reg, y_reg and z_reg. The statements x_reg: reg …, y_reg: reg …, and z_reg: reg …, bind the instances to the reg component code. The generic map binds the constant n defined in the mult_reg_pack package to the generic parameter declared in reg entity (Box 2). The port map maps by associations of the formal signal (left hand side) to the actual signal (right hand side). Signals w1, w2 and w3 glue these instances like hardware wires connecting the sub-components. The inputs to reg_x and reg_y are wired from the input ports and the output of reg_z is wired to the output port of the mult_reg entity (see Fig. 4).

The hardware module which calculates the input operands, buffered in the registers, is a process statement with the sensitivity list containing w1 and w2. The body of the process is an assignment of the multiplication of vectors w1 and w2 to vector w3. Vectors w1 and w2 are cast as signed. The operator * from standard logic signed package, returns the multiplication as signed number in a vector representation, for example, (others => ‘1’) * (others => ‘1’) is equal to (0=>’1’, others => ‘0’), that is, (-1) * (-1) equals 1.

There are two concurrent statements for wiring w1 and w2 to the output ports rx and ry used in the verification of the implemented hardware on FPGA board. An implementation on FPGA board can connect the ports rx, ry and z to the on board LEDs; x, y and en to the switches and; btn0 and btn1 to the push buttons.

The last module of the design is a process describing a state machine with 3 states declared as an enumeration type (see Chapter 3 of [3] and [7]), rdy, not_rdy, and pulse.

The state transitions (M in Equation 2) are; not_rdy → rdy, if btnR = ‘1’; rdy → pulse, if btnL = ’1’, and pulse → not_rdy, on the next clock beat unconditionally. The output function (O in Equation 3) is; when ns is not_rdy, ck_step <= ‘0’; when ns is rdy, ck_step <= ‘0’; and when ns is pulse, ck_step <= ‘1’; that is a single pulse (with one clock-period duration) is generated when btnL is pressed, and user must first reset the mechanism by pressing btnR, in order to generate another pulse. Figure 5 shows the state diagram.

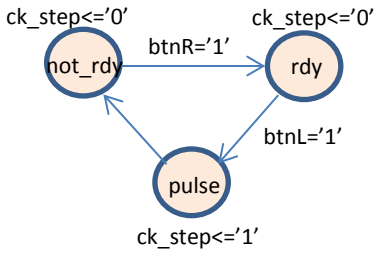

Fig. 5 State Diagram

The next section will cover more on state machines and HDL styles for describing them.

Conclusions

Digital systems concepts introduced in this section include discrete nature of digital systems which is an artifact of analog electronic switches, bits – quantitative measure of

information, combinational circuits, finite state machines, VHDL, events driven process statements, hierarchical design, registers and state machine VHDL description.

A set of tutorials accompanying this section includes VHDL and implementation on Field Programmable Gate Array of decoder and multiplexer combinational circuits, clock divider sequential circuits and registers. The tutorials provide the concepts and practices in coding VHDL, FPGA implementation and verifications.

A project register bank provides an introduction to HDL design and implementation on FPGA. The design comprises four 4-bit register and seven-segment displays showing the register contents. User can selectively load new register contents via switches and push buttons. The project introduces design stages approach in coping with design complexity.

In the next section reader will design a state machine for controlling a procedure in a calculation. Reader will simulate the VHDL design using simulation tool to verify correctness prior to implementation.

References

1. en.wikipedia.org/wiki/IEEE_1164
2. www.thecodingforums.com/threads/what-is-the-difference-between-the-types-std_logic-and-std_ulogic.538967/
3. J. Bhasker, *A VHDL Primer,* Englewood Cliffs, NJ: Prentice Hall, 1995
4. http://www.intosaiitaudit.org/intoit_articles/18p54top59.pdf
5. http://strangewondrous.net/browse/author/w/wheeler+john+archibald
6. http://www.digilentinc.com/Products/Detail.cfm?Prod=BASYS3
7. http://www.vhdl.renerta.com/mobile/source/vhd00026.htm