Electronic Design Automation
Combination lock state machine design
Using VHDL

By

IBRAHIM MOHAMED ASWAID

جامعة المرقب, كلية التقنية الطبية , قسم هندسة المعدات الطبية , ليبيا

Ibrahim.aswaid@gmail.com

Abstract:

Max+plus II along with VHDL are going to to be used in this research to design a combination lock state machine which represents a mealy sequential cercuit. The design will go through serveral stages.

First, the specifications of the design given in the table will be translated into a state graph.

Second, this state graph will be transformed to a VHDL code by the Text editor in Max+plus II. This code will be compiled first and then a waveform wil be edited. Sumulation is the last stage in this report to analyze and verify wether or not the spcifications are achieved successfully.

1-Introduction:

ICs can hold millions of tiny electrical switches called transistors which are connected with other devices to form electrical circuits. The heart of electronic products is these circuits. As a result, designing and verifying the behaviour of an IC and consequently an electrical circuit needs tremendous effort.

Computer aided design (CAD), computer aided manufacturing (CAM) and computer aided engineering (CAE) have significantly minimized this effort. Electronic design automation (EDA) falls in such packages. This category (EDA) is the software tools that used to design, simulate, analyze and prepare for manufacturing the electronic systems.

VHDL is one of these powerful tools which will be briefly introduced in the next section as it is the core of our program designing report.

(1-1) VHDL:

The behaviour and structure of digital systems can be described by hardware description languages (HDLs), VHDL is one of the most popular languages used in this field to describe and simulate the operation of most of simple to complex digital systems. VHDL stands for VHSIC Hardware Description Language which was standardized by IEEE in 1986 and it is widely used in industry. Several different levels of a digital system can be described in VHDL like behaviour, data flow and structure of the digital system.
Top-down design methodology is the nature of VHDL where the system is first specified, simulated, debugged at a high level. The design can then gradually be refined and finally structurally described to be related to the actual hardware implementation as shown in figure (1-1).

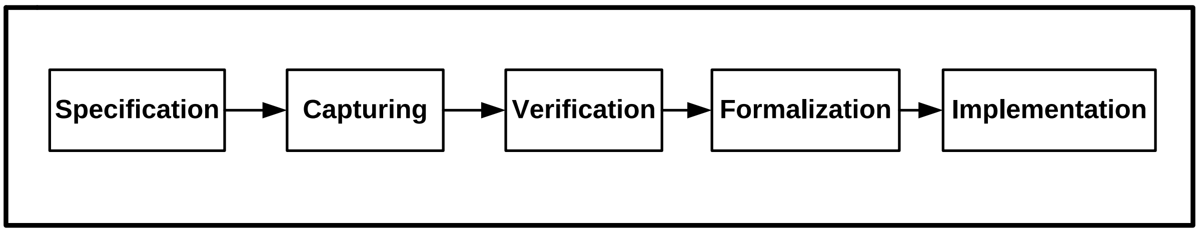

Figure (1-1) Stages of VHDL design.

One of the common tasks carried out by VHDL is designing a synchronous finite state machine which this report is about. The basic models of state machines are Moore and Mealy models.

(1) Moore machine model:

The outputs in this model depend solely on the present state as illustrated in figure (1-2).

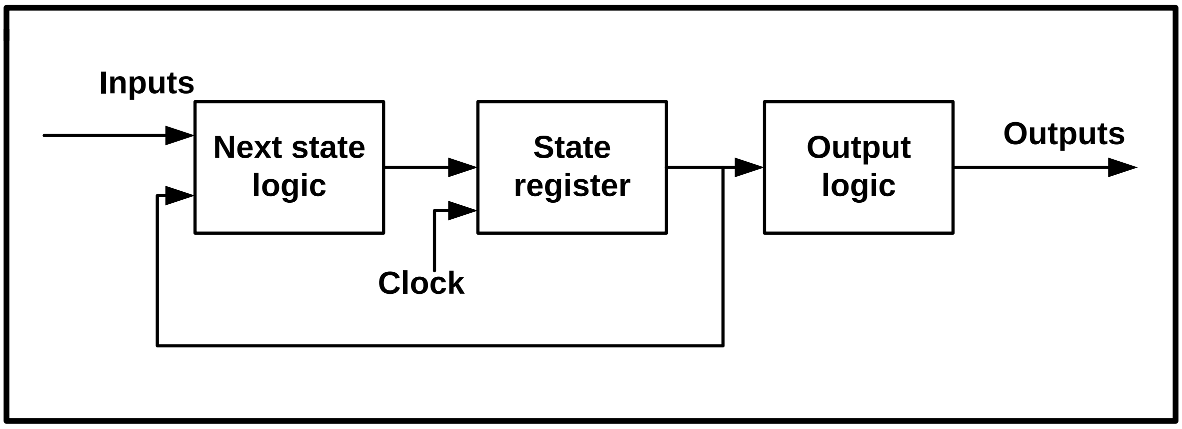

Figure (1-2) Moore state machine.

(2) Mealy machine model:

In this model, the outputs depend on both the present state and the present inputs as shown in figure (1-3). This type of modelling will be applied in this report to designing combination lock state machine.

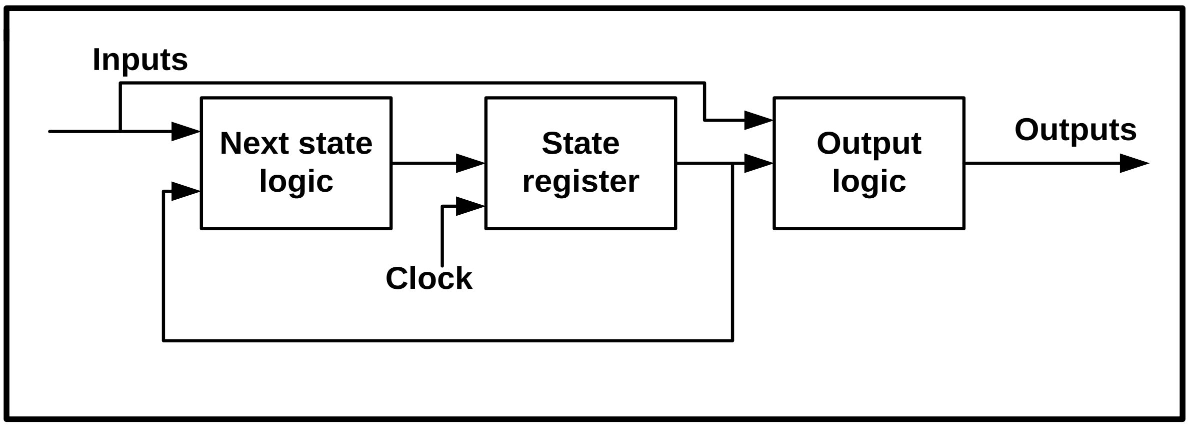

Figure (1-3) Mealy state machine.

This would lead us to glance on the main elements of VHDL to construct such state machines.

(1-1-1) Elements of VHDL:

Many language features in VHDL are designed to facilitate the usage of describing hardware components which are the basis of digital design and in its simplest form consists of an interface specifications and architectural specifications. These components or utilities used for description of components can be grouped by the use of packages. So, packages contain information common to many design units. Such VHDL design units are collected and stored with entities in the libraries which include a library statement and configurations to be constructed. The next figure summarizes the segments of a VHDL code which can be compiled separately and stored in a library.
Also, two domains in VHDL description have to be defined first before we move on to the structure of a VHDL file; they are sequential and concurrent statements.

(1) Sequential statements:

The sequential domain is represented by a process or subprogram that contains sequential statements which, as in other programming languages, are executed in the order in which they appear within the process or subprogram.

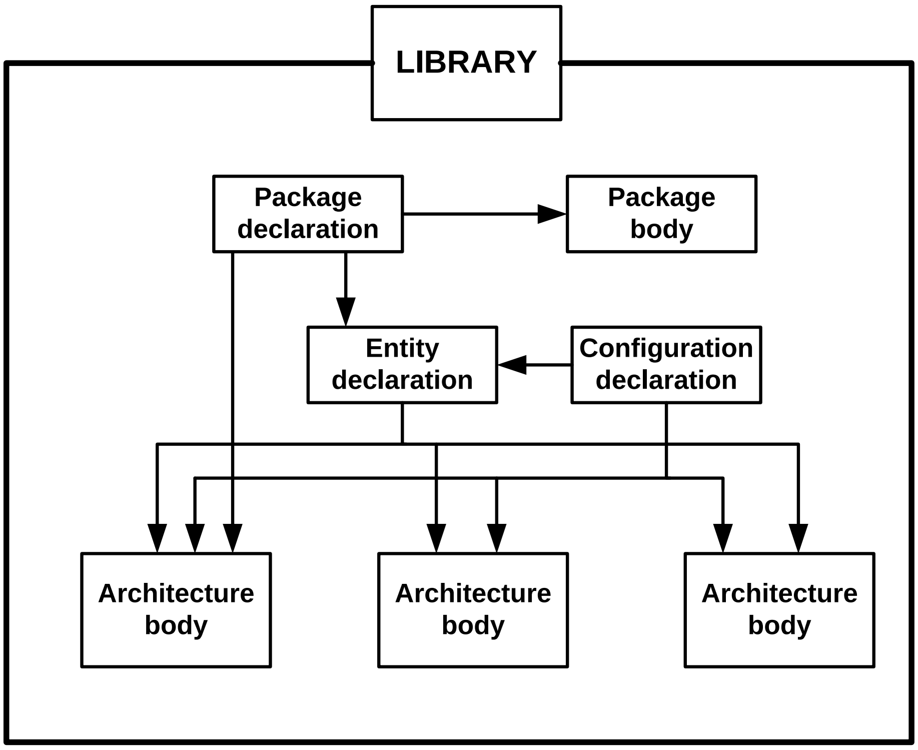

Figure (1-4) Segments of a VHDL code.

(2) Concurrent statements:

Concurrent behaviour of a circuit can be defined by VHDL concurrent statements which their execution order is not affected by their appearance order inside the architecture body. Consequently, concurrent domain is represented by an architecture which includes processes, concurrent procedure calls, concurrent signal assignments and component instantiations.
Simple comparison is provided in the next table.

Modelling style	Concurrent	Sequential
Location	Inside architecture	Inside process
Example statement	Process, component instance, concurrent signal assignment	If, for, switch-case, signal assignment

(1-1-2)Basic structure of a VHDL file:

(1) Header:

The header contains the libraries which consist of commonly used packages and entities to be used in the VHDL files. The library and package of a certain program can be referenced using ***LIBRARY*** and ***USE*** statements respectively as follow:

```
LIBRARY  ieee;
USE  ieee.std_logic_1164.All;
```

Where (ieee) is the library and (std_logic_1164) is the package and both of them will be used in this designing report.

(2) Entity declaration:

The input and output ports of the design are defined in the entity declaration where the entity statement describes the ‘package’ or ‘pinout’, or how the circuit interacts with the outside world.

```
ENTITY  name of the design IS
List of the inputs.
List of the outputs.
END name of the design;
```

where the name of the entity must be the same as the VHDL file name.

(3) Ports:

Ports provide channels of communication between the component and its environment and each port must have a name, direction and a type.

- Port directions are: ***IN, OUT, INOUT, BUFFER.***
- Port types are: ***BIT, BIT_VECTOR.***

(4) The architecture:

The hard part of VHDL coding is the architecture body of the design where a particular function of a design can be described in many different ways. The architecture statements describe the inner workings of the circuit or the system and the functions that the system actually performs.

```
***ARCHITECTURE*** _ name of the architecture ***OF*** name of the entity
***IS***
List of internal signals.
List of components.
***BEGIN***
Architecture body.
***END*** name of the architecture.
```

(5) Comments:

Comments can be made to VHDL code to make it legible and easy to understand. The comments start with two hyphens and considered as support to understanding certain statements in the code. These comments are not part of the code, so the VHDL compiler will ignore them.

(1-2) Overview of MAX+PLUS **II :**

Max+plus **II** is the software we are using in this report which supports VHDL and stands for Multiple Array matriX Programmable Logic User System. Max+plus **II** is easy to learn and easy to use. Mainly, it provids three types of basic design enteries:

(1) Graphic editor.
(2) Text editor.
(3) Waveform editor.

Text editor and waveform editor will be used in this report as well as the compilation and simulation features.

(1-2-1) Text editor

This is where the VHDL code is composed and edited.

(1-2-2) Compiler:

Compiler performs most tasks automatically where it reads the design and makes it ready for programming, simulation and timing analysis. Errors and warnings report is produced at the end of the compilation process.

(1-2-3) Waveform editor:

Waveform editor enables the designer to create and edit waveform design files and input signals. Simulation results and design analysis can both be carried out with waveform editor.

(1-2-4) Simulator:

Testing the logical operation and internal timing of the system design are performed in the simulator.

These features will be applied in the forthcoming section to design a combination lock state machine using VHDL.

....

(2) Combination lock state machine design:

A combination lock state machine will be modelled in this report which will activate an unlock output when a certain binary sequence is received.
A clocked synchronous state machine will be designed as illustrated in figure (2-1).

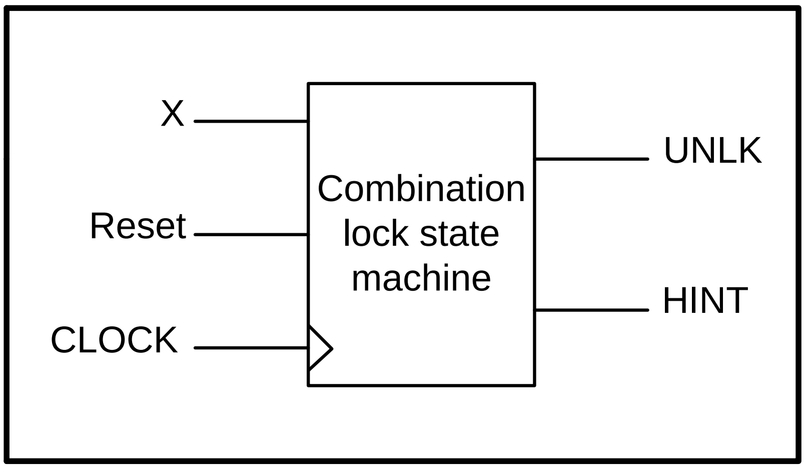

Figure (2-1) Combination lock state machine.

The shown above state machine consists of one input ***X*** and two outputs ***UNLK*** and ***HINT***. To activate the ***UNLK*** output, two conditions must be verified:

(1) ***X*** input must be ***0*** and the sequence of inputs received on ***X*** at the preceding seven clock ticks was ***0110111***.
(2) The ***HINT*** output should be ***1*** if and only if the current value of ***X*** is the correct one to move the machine closer to being in the unlocked state (with ***UNLK=1***).

Clearly, it is a Mealy machine in which the ***UNLK*** output depends on both the past history of inputs and ***X***'s current value. Also, ***HINT*** depends on both the state and the current ***X*** (indeed, if the current ***X*** produces ***HINT***=***0***, then the clued-in user will want to change ***X*** before the click tick).

(2-1) State and output table:

The combination lock state machine specifications are represented in the following table:

Sequence of inputs	Current state	X (input)					
		0			1		
		Next state	UNLK	HINT	Next state	UNLK	HINT
Got zip	A	B	0	1	A	0	0
Got 0	B	B	0	0	C	0	1
Got 01	C	B	0	0	D	0	1
Got 011	D	E	0	1	A	0	0
Got 0110	E	B	0	0	F	0	1
Got 01101	F	B	0	0	G	0	1
Got 011011	G	E	0	0	H	0	1
Got 0110111	H	B	1	1	A	0	0

From the table, it is apparent that to move from a state to the correct next state, ***HINT*** must be ***1*** (condition ***2***). Initially, it is assumed that there are no inputs received in the required sequence. ***0*** is the first input in the sequence that moves state ***A*** to state ***B***, otherwise will stay in state ***A***. Next, to move from state ***B*** to state ***C***, ***1*** must be received; else it will be in the same state (***B***). Generally, we move on gradually and successfully to the next state (from ***A*** to ***H***) if the correct sequence of inputs are received, otherwise we go back to ***A*** or ***B*** with an exception in state ***G*** in which we go back to ***E*** instead of ***B*** in the event of receiving wrong input. Finally, receiving the required seven clock ticks (***0110111***) in state ***H*** will activate the ***UNLK*** output which will be set to ***1***. All transitions occur on the rising edge of the clock except for when the ***Reset*** is activated. In this case, transition to state ***A*** will occur immediately regardless of any other circumstances of the system.
These specifications will be translated into a state graph in the next section.

(2-2) State graph:

State graph is the first step in modelling the state machine and it is a graphical description of the state machine that gives better understanding of the design and shows the design in an easier way.

Figure (2-2) depicts the state graph of the combination lock machine. The circles represent the states (***A*** to ***H***), while the arrowed lines show the paths of the transitions from state to state. Alongside each line, there is a Boolean expression which consists of three numbers, the first number is the input (***X***) and the other two numbers are the ***UNLK*** and ***HINT*** respectively. Each green line expresses the closeness to the unlocked state (***UNLK=1***) while the red line indicates on receiving a wrong input and the current state will go back to a previous state instead of moving on closer.

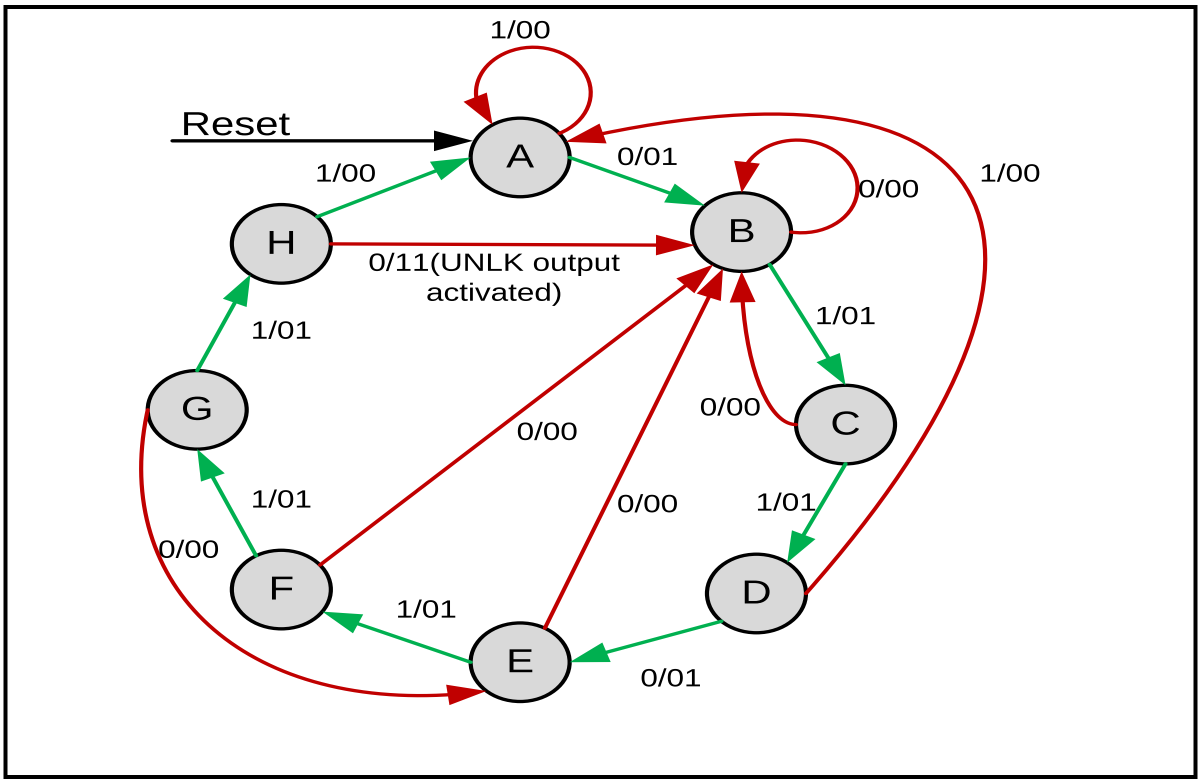

Figure (2-2) State graph for the combination lock state machine.

This wealth of information state graph will be transformed into a VHDL code in the next section.

(2-3) VHDL code:

After transforming the specifications of the combination lock state machine from a mere table into a state graph, it is the step now to be translated into a VHDL code. This is the hardest part where thoughts should be well-organized prior starting to include all cases and actions of both the external interface and the internal operation. ***Text Editor*** in Max+plus **II** was used to construct the next code which models the combination lock state machine.

```
..........................................................................................................................
Title         : Combination lock state machine.
File name : ASWAID.
```

Auther : Ibrahim Aswaid.

....

```
LIBRARY  ieee;                          -- The library is ieee.
USE ieee.std_logic_1164.ALL;    -- The package is std_logic_1164.
 ENTITY ASWAID IS                   -- Entity declaration.
PORT(                                       -- Starting of inputs and outputs declarations.
clk :  IN bit;                            -- Clock input.
reset: IN bit;                            -- Reset input.
x:     IN bit;                              -- X input.
unlk : OUT bit;                          -- Unlock output.
hint : OUT  bit                           -- Hint output.
);
END ASWAID;                           -- This is the end of the entity.
 ARCHITECTURE ARCH OF ASWAID IS -- This is the beginning of one architecture associated with the entity.
TYPE STATE_TYPE IS (A, B, C, D, E, F, G, H); -- States (A to H) enumeration  type.
 SIGNAL state: STATE_TYPE;   -- Define signal (states A to H) for waveform output, the initial     state is A.
 BEGIN                                    -- The beginning of the architecture body.
PROCESS (clk, reset)              --The sensitivity list consists of clock and reset.
 BEGIN                                    -- The beginning of the process.
 IF reset = '0' THEN               -- Active low reset which interrupts the sequence of the states to be state A.
  state <= a;
ELSIF clk' EVENT AND clk = '1' THEN -- States only change at the rising edge of the clock.
 CASE state IS        -- Defines the current state.
  WHEN a =>          -- when the current state is A.
  IF x ='0' THEN     --This is the first evaluation of the input X which will select one or more of the enclosed     sequences.
  state <= b;           -- This is the correct next state.
 hint <='1';              -- Hint=1 is always the clue of receiving a correct input.
ELSIF x='1' THEN -- X=1 is a wrong input according to the specifications.
state <= a;             -- The current state will not change.
 hint <= '0';            -- Hint=0 is always the clue of receiving a wrong input.
END IF;
WHEN b =>          -- when current state is B.
IF x='1' THEN
state <= c;           -- To move to the next correct state which is C, input X must be 1
hint <= '1';           -- As long as the correct transition is achieved, the Hint is always 1.
ELSIF x='0' THEN
state <= b;
hint <= '0';
END IF;
WHEN c =>
IF x='1' THEN
state <= d;
hint <= '1';
ELSIF x='0' THEN
```

```
state <= b;
hint <= '0';
END IF;
WHEN d =>
IF x='0' THEN
state <= e;
hint <= '1';
ELSIF x='1'THEN
state <= a;
hint <= '0';
END IF;
WHEN e =>
IF x='1' THEN
state <= f;
hint <= '1';
ELSIF x='0' THEN
state <= b;
hint <= '0';
END IF;
WHEN f =>
IF x='1' THEN
state <= g;
hint <= '1';
ELSIF x='0' THEN
state <= b;
 hint <= '0';
 END IF;
 WHEN g =>
 IF x='1' THEN
state <= h; -- This is the final state that activates (with X=0) the UNK output which
indicate   receiving the correct sequence of inputs.
hint <= '1';
ELSIF x='0' THEN
state <= e;
hint <= '0';
END IF;
WHEN h =>
IF x='0' THEN
state <= b;
hint <= '1';
ELSIF x='1' THEN
state <= a;
hint <= '0';
END IF;
END CASE; -- This is the end of the CASE which will produce the final current state.
END IF;
END PROCESS;--This is the end of the sequential assignment statements of the
process.
```

```
unlk <= '1' WHEN state = h AND (x = '0') ELSE '0'; -- UNLK output will be 1 (active) only if the current state is H and the input is 0.
END ARCH;    -- End of the architecture.
```

(2-4) Analysis of the code:

First of all, the colours indicate different categories where the blue indicates the key words (reserved words), the purple colour indicates the explanatory comments and the rest of the code is in the normal black.

Header description is arbitrarily the first partition of the code where (ieee) is the library and (std_logic_1164) is the package.

Inputs and outputs of the design are declared in the entity which named (***ASWAID***). All inputs and outputs are of type (BIT).

The second partition of the code is the architecture which is named (***ARCH***). The first two lines of the architecture declarations define the signal called ***state*** that can have any one of the values (***A*** to ***H***) and this signal is initialized to ***A*** as the default initialization is the leftmost element in the enumeration list. The architecture body starts from the reserved word ***BEGIN*** which will then announce the start of the ***PROCESS.***

PROCESS is a concurrent statement that defines the sequential behaviour of the design through a sequence of sequential statements (signal assignment statements). (***CLK, Reset***) is the sensitivity list, and the process executes whenever any signal in the sensitivity list changes. Also, it is very important to know that sequential statements are executed in the order in which they appear in the process (sequentially).

Therefore, the process begins with two main sequential statements that define the ***CLK*** and the ***Reset***.

Reset is active low '***0***' and according to the specifications, at any time ***Reset*** goes to '***0***', the sequence of the states will be interrupted and will go back to state '***A***'. The expression ***CLK'event*** is true whenever the signal ***CLK*** changes from '***0***' to '***1***' (at the rising edge of the clock). Thus, states ***A*** to ***H*** change only at the rising edge of the clock.

CASE statement selects execution one of the sequential statements that follow it to give the current state. This is the partition of the code where the state graph is transformed into a VHDL code with the exception of ***ULK*** output which is dependent on ***X*** at seven clock ticks is ***0110111*** which meets ***H*** state and the current value of ***X=0***. This means that ***ULK*** output waits only for the state ***H*** to show up and on parallel with the current input ***X*** is ***0***, and this is another interpretation of the given ***ULK*** specifications. Thus, ***ULK*** was excluded from the sequential statements of the ***CASE*** statement to brief the steps of the code.

According to the state graph, in the event of state '***A***', as an example, if '***X***' input is '***0***', then '***HINT***' will change to '***1***' and state '***A***' will move on to state '***B***', otherwise state '***A***' will not change and '***HINT***' will be '***0***' (condition 2).

Thus, the code was constructed to contain all the given specifications to be executed sequentially.
The code then was saved with the same name as the entity (***ASWAID***).

(2-5) Compilation of the code:

Upon completion, the code had to be compiled to check the syntax and other errors. Successfully, the code was compiled with no errors and one warning and the code thus ready for simulation.

(2-5) Waveform entry and simulation:

Input vectors can be created and edited through the ***Waveform Editor*** where the generated nodes are inserted into the waveform using the (.snf) file.
The clock (***CLK***) was set to ***100ns*** and the ***Reset*** was set to '***1***' first to check the functionality of the whole code. Also, to check the unlocking state, the correct set of inputs of '***X***' which is '***0110111'*** was edited and the resultant waveform graph is as shown in figure (2-3).

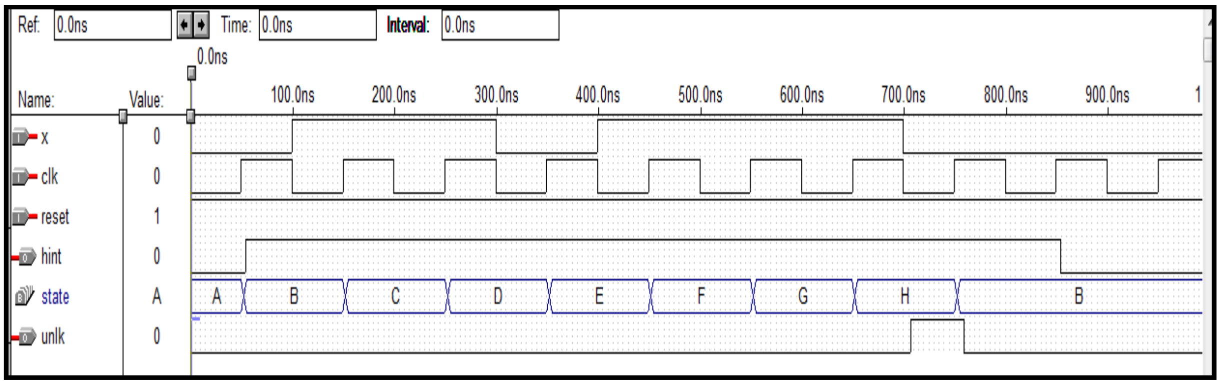

Figure (2-3) resultant waveform of correct sequence of inputs

Figure (2-3) shows a waveform graph representing correct full sets of inputs ***0110111***. In order to verify that the resultant graph carries good news about the design, it has to be detailed as follow.
It can be noted from the figure that the initial state is ***A*** and ***0*** is the first input we are seeking to move from state ***A*** to state ***B***. Transition to state ***B*** was occurred because of two conditions:

(1) Correct input, ***0***, was received.
(2) The clock was at the rising edge.

Propagation delay time is noticeable here where to move from any state to the next, a ***2.5ns*** delay has to be passed after the rising edge of the clock.

Next, ***HINT*** changed to ***1*** after ***2ns*** (propagation delay) as an indication of receiving a correct input and consequently correct transition and as long as the transition from state to state is correct, ***HINT*** will always stay at ***1***.

Transition from state ***B*** to state ***C*** was obtained because of the correct input and ***HINT*** remained at ***1***.

Correct transitions from state ***A*** to state ***H*** were achieved successfully as a result of receiving sequence of correct inputs ***0110111***.

Activating the ***UNLK*** output was also achieved successfully after ***7.5ns*** of receiving the correct sequence of inputs ***0110111*** and the current input ***X*** is **0** and the current state is ***H.*** This propagation delay is longer than the previous delays (***2.5ns*** and ***2ns***) because the relevant statement in the code is outside the clocked process and the code has to be evaluated again to see what the next value of **X**, if it is ***0 UNLK*** will be set to ***1***.

Finally and according to the specifications of the design, state ***H*** moved to state ***B*** at the rising edge of the clock and ***UNLK*** output deactivated to ***0*** after ***7ns*** propagation delay time of the transition from ***H*** to ***B***. So it can be said, at this point of analysis that the function of the circuit was achieved successfully.

Also, the design has to be tested in case of entering a wrong sequence of inputs. Randomly, ***X*** input was edited in the waveform editor to be (***0100101***) and the resultant graph is as shown in figure (2-4).

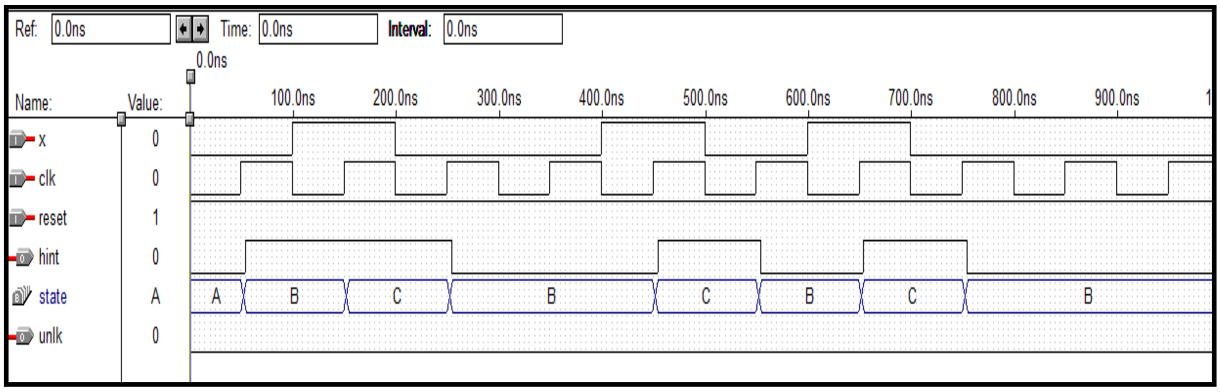

Figure (2-4) resultant waveform of wrong sequence of inputs

The transitions in figure (2-4) verify the individual statement of the code correctly, but the figure shows wrong sequence of transitions because of the wrong sequence of inputs and consequently unlocking will not be achieved. ***HINT***, however, is still doing its job of indicating wrong or correct transition.

The next graph, figure (2-5), shows the exception case (state ***G***) which moves to state ***E*** instead of ***B*** or ***A*** in the case of receiving wrong input ***X=0***.

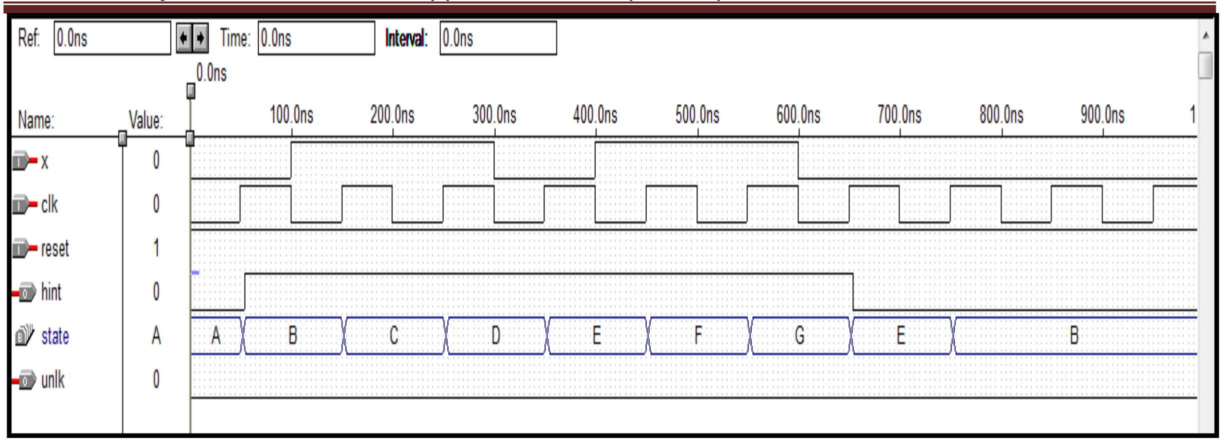

Figure (2-5) The exceptional transition from G to E.

Reset has to be tested before concluding this report. The next waveform graph shows the influence of the ***Reset*** on the transitions. Activating the ***Reset*** at low value ***0*** will interrupt the sequence of the transitions at any time ***Reset*** is activated and will force this sequence to be in state ***A*** after ***3.5ns*** propagation delay time. This transition to state ***A*** does not depend on the rising edge of the clock because the statement of the ***Reset*** statement is not among the clocked statements. This case is depicted in the next figure (2-6).

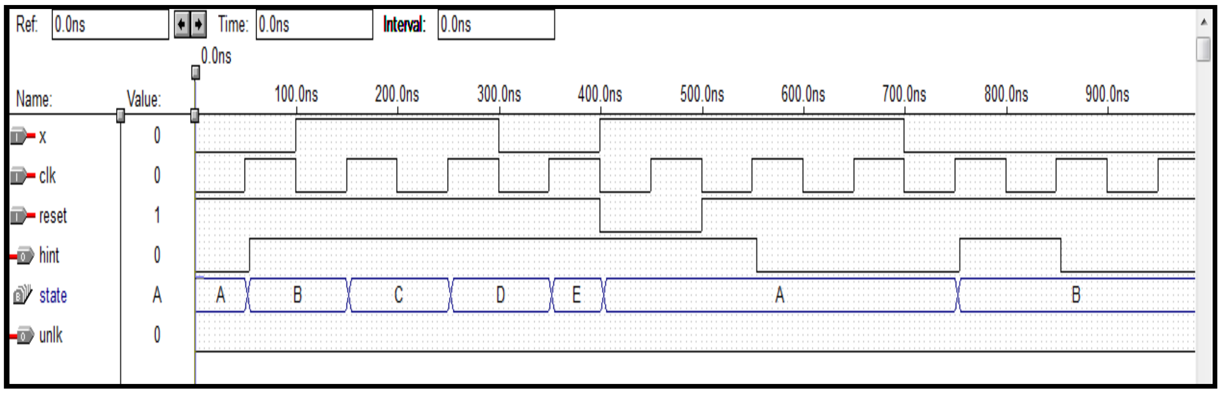

Figure (2-6) Activation of the Reset.

(3) Conclusion:

Designing modern IC's and PCB's would be an impossible task without the aid of electronic design automation (EDA). VHDL is one of EDA tools which was used in this report with Max+Plus II designing environment to design a combination lock state machine.

A VHDL code was constructed, compiled and simulated. Simulation results were perfect and all the specifications were achieved correctly through the analysis of the waveforms.

Generally speaking, Max+plus II is an interesting software that broadens the thoughts in designing such program as the program can be written in many different ways and I personally had a great time in this design.

Referenes:

(1) Dr. D. Xu (2011). Lectures Notes. Teesside University: School of Science and Engineering.

(**1**). Martin, G.& Scheffer, L. (2006) *Electronic Design Automation for Integrated Circuits Handbook*. Publisher: CRC Press. ISBN: 0849330963

(**2**). Jansen, D. (2003) *The Electronic Design Automation Handbook Book*. Publisher: Springer Netherlands, ISBN: 1402075022

(**3**). Mark, D. (2004) *Essential Electronic Design Automation (EDA)*. ISBN: 0131828290

(**4**). Kaufmann, M. (2002) *The Designer's Guide to VHDL*. 2nd ed, USA. ISBN 1-55860-270-4

(**5**). Barry, W. (1992) *Digital system design, 2nd ed,* Prentice Hall, ISBN:013220286

(**6**). John,P. & Bob, M. & Anand, P. (2010) Introduction to VHDL and MAX+plus II. Available at: http://www.cs.ucla.edu/Logic_Design/vhdlintro.html (Accessed: 9th April 2010).

(**7**). Altera Corporation. MAX+plus II Tutorial. Available at: http://www.altera.com/literature/manual/81_gs3.pdf.