CHAPTER-1

INTRODUCTION OF VLSI

1.1. Introduction

Very-large-scale integration (VLSI) is the process of creating integrated circuits by combining thousands of transistor-based circuits into a single chip. VLSI began in the 1970s when complex semiconductor and communication technologies were being developed. The microprocessor is a VLSI device. The term is no longer as common as it once was, as chips have increased in complexity into the hundreds of millions of transistors.

1.2. Overview

The first semiconductor chips held one transistor each. Subsequent advances added more and more transistors, and, as a consequence, more individual functions or systems were integrated over time. The first integrated circuits held only a few devices, perhaps as many as ten diodes, transistors, resistors and capacitors, making it possible to fabricate one or more logic gates on a single device. Now known retrospectively as "small-scale integration" (SSI), improvements in technique led to devices with hundreds of logic gates, known as large-scale integration (LSI), i.e. systems with at least a thousand logic gates. Current technology has moved far past this mark and today's microprocessors have many millions of gates and hundreds of millions of individual transistors.

At one time, there was an effort to name and calibrate various levels of large-scale integration above VLSI. Terms like Ultra-large-scale Integration (ULSI) were used. But the huge number of gates and transistors available on common devices has rendered such fine distinctions moot. Terms suggesting greater than VLSI levels of integration are no longer in widespread use. Even VLSI is now somewhat quaint, given the common assumption that all microprocessors are VLSI or better.

As of early 2008, billion-transistor processors are commercially available, an example of which is Intel's Montecito Itanium chip. This is expected to become more commonplace as semiconductor fabrication moves from the current generation of 65 nm processes to the next 45 nm generations (while experiencing new challenges such as increased variation across process corners). Another notable example is NVIDIA’s 280 series GPU.

This microprocessor is unique in the fact that its 1.4 Billion-transistor count, capable of a teraflop of performance, is almost entirely dedicated to logic (Itanium's transistor count is largely due to the 24MB L3 cache). Current designs, as opposed to the earliest devices, use extensive design automation and automated logic synthesis to lay out the transistors, enabling higher levels of complexity in the resulting logic functionality. Certain high-performance logic blocks like the SRAM cell, however, are still designed by hand to ensure the highest efficiency (sometimes by bending or breaking established design rules to obtain the last bit of performance by trading stability).

1.3. What is VLSI?

VLSI stands for "Very Large Scale Integration". This is the field, which involves packing more and more logic devices into smaller and smaller areas.

- Simply we say Integrated circuit is many transistors on one chip.
- Design/manufacturing of extremely small, complex circuitry using modified semiconductor material
- Integrated circuit (IC) may contain millions of transistors, each a few mm in size
- Applications wide ranging: most electronic logic devices.

1.4. History of Scale Integration

- Late 40s Transistor invented at Bell Labs
- Late 50s First IC (JK-FF by Jack Kilby at TI)
- Early 60s Small Scale Integration (SSI)
    - 10s of transistors on a chip
- Late 60s Medium Scale Integration (MSI)
    - 100s of transistors on a chip
- Early 70s Large Scale Integration (LSI)
    - 1000s of transistor on a chip
- Early 80s VLSI
    - 10,000s of transistors on a chip (later 100,000s & now 1,000,000s)
- Ultra LSI is sometimes used for 1,000,000s

- SSI - Small-Scale Integration (0-102)
- MSI - Medium-Scale Integration (102ˆ-103)
- LSI - Large-Scale Integration (103-105)
- VLSI - Very Large-Scale Integration (105-107)
- ULSI - Ultra Large-Scale Integration (>=107)

1.5. Advantages of ICs over discrete components

While we will concentrate on integrated circuits, the properties of integrated circuits-what we can and cannot efficiently put in an integrated circuit-largely determine the architecture of the entire system. Integrated circuits improve system characteristics in several critical ways. ICs have three key advantages over digital circuits built from discrete components:

- **Size**: Integrated circuits are much smaller-both transistors and wires are shrunk to micrometer sizes, compared to the millimeter or centimeter scales of discrete components. Small size leads to advantages in speed and power consumption, since smaller components have smaller parasitic resistances, capacitances, and inductances.

- **Speed**: Signals can be switched between logic 0 and logic 1 much quicker within a chip than they can between chips. Communication within a chip can occur hundreds of times faster than communication between chips on a printed circuit board. The high speed of circuits on-chip is due to their small size-smaller components and wires have smaller parasitic capacitances to slow down the signal.

- **Power consumption:** Logic operations within a chip also take much less power. Once again, lower power consumption is largely due to the small size of circuits on the chip-smaller parasitic capacitances and resistances require less power to drive them.

1.6. VLSI and systems

These advantages of integrated circuits translate into advantages at the system level:

Smaller physical size: Smallness is often an advantage in itself-consider portable televisions or handheld cellular telephones.

Lower power consumption: Replacing a handful of standard parts with a single chip reduces total power consumption. Reducing power consumption has a ripple effect on the rest of the system: a smaller, cheaper power supply can be used; since less power consumption means less heat, a fan may no longer be necessary; a simpler cabinet with less shielding for electromagnetic shielding may be feasible, too.

Reduced cost: Reducing the number of components, the power supply requirements, cabinet costs, and so on, will inevitably reduce system cost. The ripple effect of integration is such that the cost of a system built from custom ICs can be less, even though the individual ICs cost more than the standard parts they replace.

Understanding why integrated circuit technology has such profound influence on the design of digital systems requires understanding both the technology of IC manufacturing and the economics of ICs and digital systems.

1.7. Applications

- ✓ Electronic system in cars.
- ✓ Digital electronics control VCRs
- ✓ Transaction processing system, ATM
- ✓ Personal computers and Workstations
- ✓ Medical electronic systems etc….

Electronic systems now perform a wide variety of tasks in daily life. Electronic systems in some cases have replaced mechanisms that operated mechanically, hydraulically, or by other means; electronics are usually smaller, more flexible, and easier to service. In other cases electronic systems have created totally new applications. Electronic systems perform a variety of tasks, some of them visible, some more hidden:

- Personal entertainment systems such as portable MP3 players and DVD players perform sophisticated algorithms with remarkably little energy.

- Control fuel injection systems, adjust suspensions to varying terrain, and perform the control functions required for anti-lock braking (ABS) systems.

- Digital electronics compress and decompress video, even at high-definition data rates, on-the-fly in consumer electronics.

- Low-cost terminals for Web browsing still require sophisticated electronics, despite their dedicated function.

- Personal computers and workstations provide word-processing, financial analysis, and games. Computers include both central processing units (CPUs) and special-purpose hardware for disk access, faster screen display, *etc*.

- Medical electronic systems measure bodily functions and perform complex processing algorithms to warn about unusual conditions. The availability of these complex systems, far from overwhelming consumers, only creates demand for even more complex systems.
- The growing sophistication of applications continually pushes the design and manufacturing of integrated circuits and electronic systems to new levels of complexity. And perhaps the most amazing characteristic of this collection of systems is its variety-as systems become more complex, we build not a few general-purpose computers but an ever-wider range of special-purpose systems. Our ability to do so is a testament to our growing mastery of both integrated circuit manufacturing and design, but the increasing demands of customers continue to test the limits of design and manufacturing.

CHAPTER - 2
VERILOG HDL

Verilog HDL is a hardware description language that can be used to model a digital system at many levels of abstraction ranging from the algorithmic-level to the gate-level to the switch-level. The complexity of the digital system being modeled could vary from that of a simple gate to a complete electronic digital system, or anything in between. The digital system can be described hierarchically and timing can be explicitly modeled within the same description.

The Verilog HDL language includes capabilities to describe the behavior-al nature of a design, the dataflow nature of a design, a design's structural composition, delays and a waveform generation mechanism including aspects of response monitoring and verification, all modeled using one single language. In addition, the language provides a programming language interface through which the internals of a design can be accessed during simulation including the control of a simulation run.

The language not only defines the syntax but also defines very clear simulation semantics for each language construct. Therefore, models written in this language can be verified using a Verilog simulator. The language inherits many of its operator symbols and constructs from the C programming language. Verilog HDL provides an extensive range of modeling capabilities, some of which are quite difficult to comprehend initially. However, a core subset of the language is quite easy to learn and use. This is sufficient to model most applications.

2.1 History:

The Verilog HDL language was first developed by Gateway Design Automation in 1983 as hardware are modeling language for their simulator product, At that time, was a propnetary language. Because of the popularity of the, simulator product, Verilog HDL gained acceptance as a usable and practical language by a number of designers. In an effort to increase the popularity of the language, the language was placed in the public domain in 1990. Open Verilog International (OVI) was formed to promote Verilog. In 1992 OVI decided to pursue standardization of Verilog HDL as an IEEE standard. This effort was successful and the language became an IEEE standard in 1995. The complete standard is described in the Verilog hardware description language reference manual. The standard is called std 1364-1995.

2.2 Major Capabilities:

Listed below are the major capabilities of the Verilog hardware description:

- Primitive logic gates, such as and, or and nand, are built-in into the language.
- Flexibility of creating a user-defined primitive (UDP). Such a primitive could either be a combinational logic primitive or a sequential logic primitive.
- Switch-level modeling primitive gates, such as pmos and nmos, are also built-in into the language.
- Explicit language constructs are provided for specifying pin-to-pin delays, path delays and timing checks of a design.
- A design can be modeled in three different styles or in a mixed style. These styles are: behavioral style - modeled using procedural constructs; dataflow style - modeled using continuous assignments; and structural style - modeled using gate and module instantiations.
- There are two data types in Verilog HDL; the net data type and the register data type. The net type represents a physical connection between structural elements while a register type represents an abstract data storage element.
- Fig.2.1 shows the mixed-level modeling capability of Verilog HDL, that is, in one design; each module may be modeled at a different level.

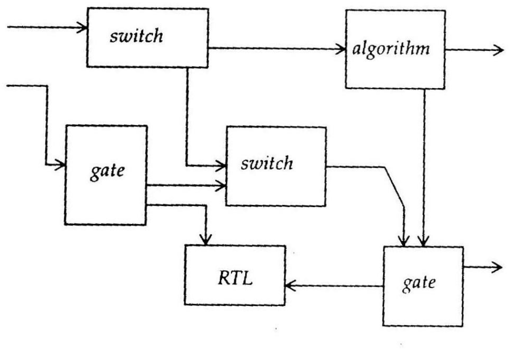


Fig.2.1 Mixed level modeling

- Verilog HDL also has built-in logic functions such as & (bitwise-and) and I (bitwise-or).
- High-level programming language constructs such as condition- all case statements, and loops are available in the language.
- Notion of concurrency and time can be explicitly modeled.
- Powerful file read and write capabilities fare provided.
- The language is non-deterministic under certain situations, that is, a model may produce different results on different simulators; for example, the ordering of events on an event queue is not defined by the standard.

2.3 SYNTHESIS:

Synthesis is the process of constructing a gate level net list from a register-transfer level model of a circuit described in Verilog HDL. Fig.2.2 shows such a process. A synthesis system may as an intermediate step, generate a net list that is comprised of register-transfer level blocks such as flip-flops, arithmetic-logic-units, and multiplexers, interconnected by wires. In such a case, a second program called the RTL module builder is necessary. The purpose of this builder is to build, or acquire from a library of predefined components, each of the required RTL blocks in the user-specified target technology.

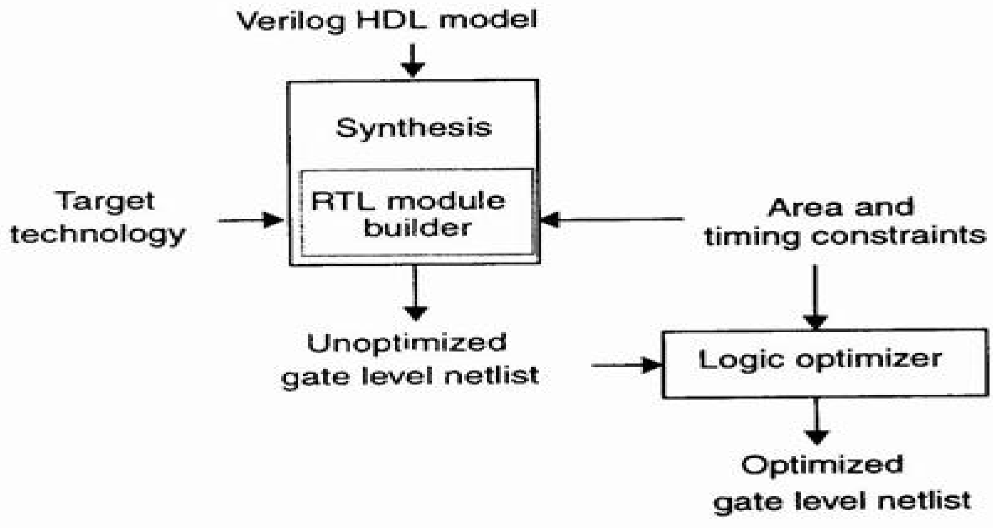

Verilog HDL model
Synthesis
RTL module builder
Target technology
Area and timing constraints
Unoptimized gate level netlist
Logic optimizer
Optimized gate level netlist

Fig.2.2 Synthesis process

Having produced a gate level net list, a logic optimizer reads in the net list and optimizes the circuit for the user-specified area and timing constraints. The module builder for appropriate selection or generation of RTL blocks may also use these area and timing constraints. In this book, we assume that the target net list is at the gate level. The logic gates used in the synthesized net lists are described in Appendix B. The module building and logic optimization phases are not described in this book.

The above figure shows the basic elements of Verilog HDL and the elements used in hardware. A mapping mechanism or a construction mechanism has to be provided that translates the Verilog HDL elements into their corresponding hardware elements as shown in Fig.2.3

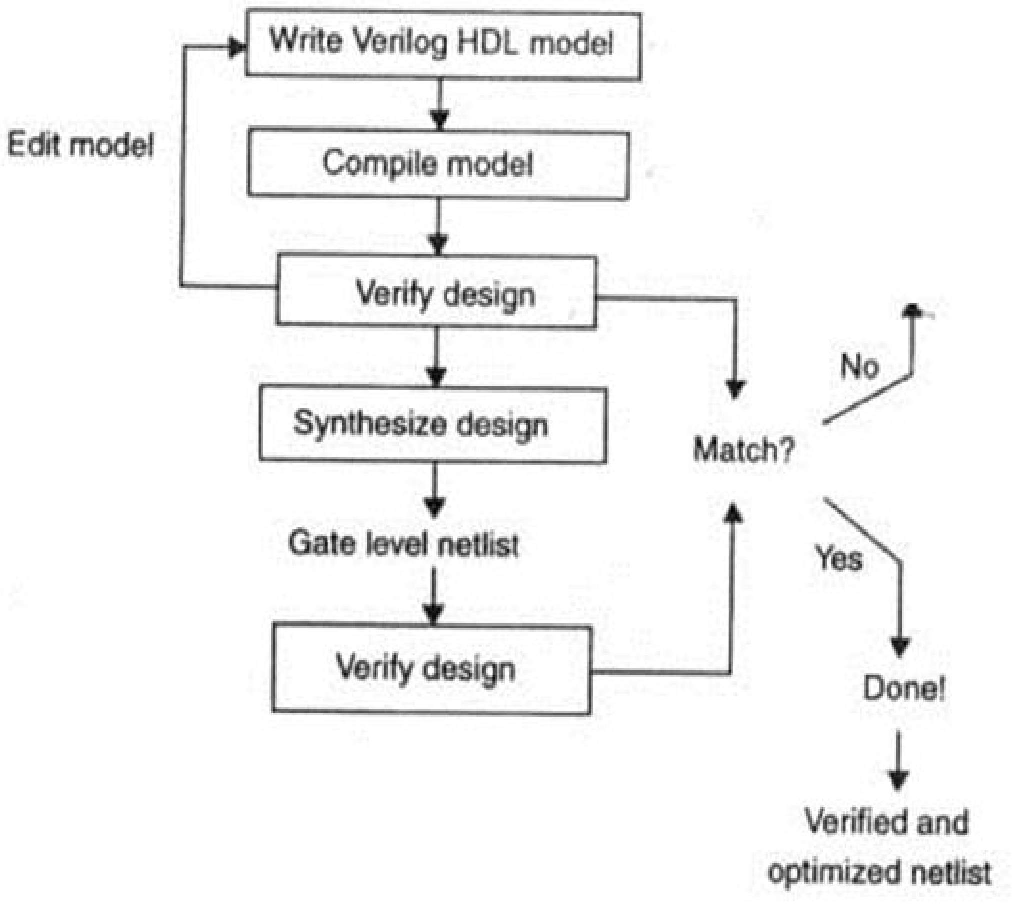


Fig.2.3 Typical design process

CHAPTER 3

INTRODUCTION AND LITERATURE SURVEY

3.1 Introduction

Automated Teller Machines (ATM) are banking kiosks available to consumers 24X7. These serve as data terminals for convenient money transactions. In these unattended machines, user has to use the ATM card (a card with the account information) and provide the Personal Identification Number (PIN) using a keypad. When the computer verifies the PIN, the customer gets a choice of transactions. Adhering to the instructions and entering proper choices, user can dispense cash, deposit cash, change PIN or just get an information on account balance.

Basically an ATM (Automated Teller Machine) is the combination of hardware and software. The hardware used is card swipe, keypad, display, etc. The software such as operating system controls this hardware's. Now a day's digital system is entered in replacing the software. After the invention of ATM, different groups design the different parts of the system. But in new generation there are different methods and tools are emerging, which can handle the design of system from system level specification. Security, flexibility and reliability of user's account get increased due to ATM and it becomes user friendly now. In this chapter architecture of an ATM is designed and the finite state diagram of the system is drawn. The system level analysis is made with the help of FSM of the system.

An **Automated Teller Machine** (**ATM**) is a computerized electronic device that helps the customers of a financial institution to access the financial transactions in a public place without any help from others. On most modern ATMs, the customer is identified by inserting a plastic ATM card with a magnetic stripe or a plastic smartcard with a chip that contains a unique card number and some security information, such as an expiration date. Security is provided by the customer entering a personal identification number (PIN).

In earlier days customers of one bank can able to access the ATM machines of that bank only, since banks have formed cooperative, nationwide networks so that a customer of one bank can use an ATM of another bank for cash access. Using an

ATM, customers can access their bank accounts in order to make cash withdrawals or check their account balances.

The types of banking transactions a customer can carryout are determined by capabilities of the particular banking machine and the programming of the institution operating the machine. Now a days ATMs are used for alternative uses like updating pass books, printing bank documents, paying bills etc.

3.2 ATM Controller Architecture Review

As the technology changes attains to its peak, the human needs also been increased to a more modernized level. Here we take up with a concept of ATM controller design. In general the ATM means Automated Teller Machine which makes us to leads the life much faster. We look into the technical aspect of ATM as our review.

The basic architecture of ATM is described as follows:

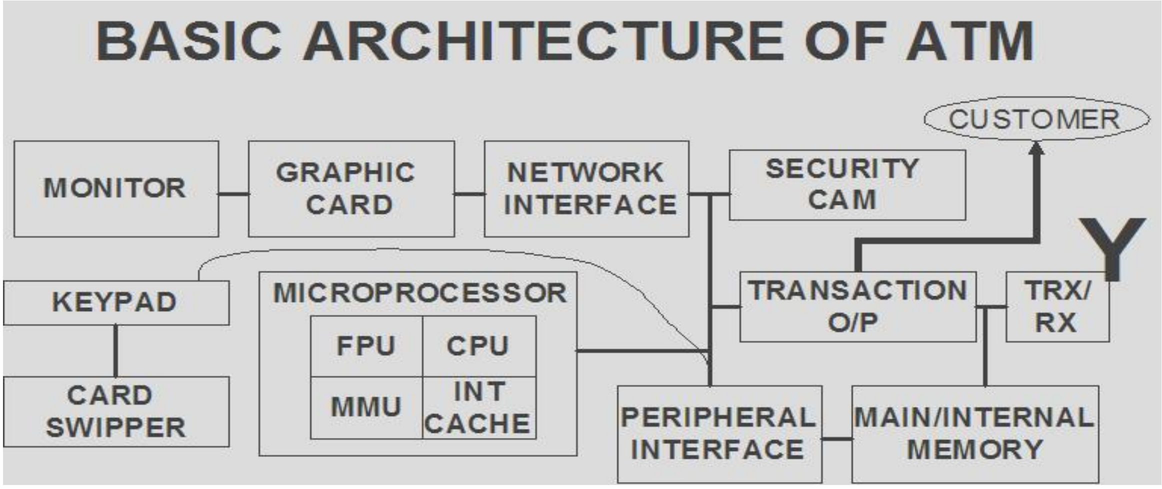


Fig.3.1 Basic Architecture of ATM

3.2.1 History

The ATM was invented by Scot John Shepherd-Barron. The world's first ATM was installed in a branch of Barclays in the northern London borough of Enfield, Middlesex, in 1967. Inspiration had struck Mr. Shepherd-Barron, now 82, while he was in the bath. A mechanical cash dispenser was developed and built by Luther

George Simian and installed in 1939 in New York City by the City Bank of New York, but removed after 6 months due to the lack of customer acceptance.

The ATM got smaller, faster and easier over the years. Thereafter, the history of ATMs paused for over 25 years, until De La Rue developed the first electronic ATM, which was installed first in Enfield Town in North London [2] on 27 June 1967 by Barclays Bank. The first ATMs accepted only a single-use token or voucher, which was retained by the machine. These worked on various principles including radiation and low-coercively magnetism that was wiped by the card reader to make fraud more difficult. The British engineer John Rose developed the idea of a PIN stored on the card in 1965.

However, the modern, networked ATM was invented in Dallas, Texas, by Don Wetzel in 1968. Wetzel was a department head at an automated baggage-handling company called Ductal. In 1995 the Smithsonian's National Museum of American History recognized Ductal and Wetzel as the inventors of the ATM. ATMs first came into wide UK use in 1973; the IBM 2984 was designed at the request of Lloyds Bank.

3.2.2 System Design Review

Hardware Description

An ATM is typically made up of the following devices:

- CPU (to control the user interface and transaction devices)
- Magnetic and/or Chip card reader (to identify the customer)
- PIN Pad (similar in layout to a Touch tone or Calculator keypad), often manufactured as part of a secure enclosure.
- Secure crypto processor, generally within a secure enclosure.
- Display (used by the customer for performing the transaction)
- Function key buttons (usually close to the display) or a Touch screen (used to select the various aspects of the transaction)
- Record Printer (to provide the customer with a record of their transaction)
- Vault (to store the parts of the machinery requiring restricted access)
- Housing (for aesthetics and to attach signage to)

Recently, due to heavier computing demands and the falling price of computer-like architectures, ATMs have moved away from custom hardware architectures using microcontrollers and/or application-specific integrated circuits to adopting a hardware architecture that is very similar to a personal computer.

Many ATMs are now able to use operating systems such as Microsoft Windows and Linux. Although it is undoubtedly cheaper to use commercial off-the-shelf hardware, it does make ATMs vulnerable to the same sort of problems exhibited by conventional computers.

3.3 Literature Review

ATMs are becoming increasingly popular because they make banking functions available to all the customers around the clock, and at variety of locations, in addition to banks. In olden days ATMs are manufactured mainly by using microprocessor or microcontroller. Due to heavier demands and the falling price of computer like architectures, ATMs have moved away from custom architectures using microcontroller and / or integrated circuits to adopting hardware architecture that is very similar to a personal computer. Although it is a cheaper to use commercial hardware, it does not mean that ATMs are free from some sort of problems exhibited by the personal computers.

With the migration to commodity pc hardware, standard commercial operating systems and programming environments can be used in ATMs. Typical platforms used in ATMs include Microsoft operating systems, Sun Microsystems, Solaris, Java, Linux and UNIX may also be used in these environments. In the current scenario Linux is also playing a leading role in the ATM market place. Most of the companies are trying to use Linux operating in there ATMs.

Most ATMs are connected to interbank networks, enabling people to withdraw and deposit money from machines not belonging to the bank where they have their account. This is a convenience; especially for people who are traveling it is possible to make withdrawals in places where one's bank has no branches, and even to withdraw local currency in a foreign country

ATMs mainly consists of a special type of secure crypto processor, which is a dedicated microprocessor mainly used for extracting the information from the card used by the customer. The security of ATM transactions relies mostly on the integrity of the secure crypto processor. Many ATMs usually print each transaction in a printout form that is rolled into a roll of paper stored inside the ATMs, which allows both the users of the ATMs and the financial institutions to keep records, in case there is a dispute. In some cases, transactions are posted directly to an electronic journal to reduce the need for paper work

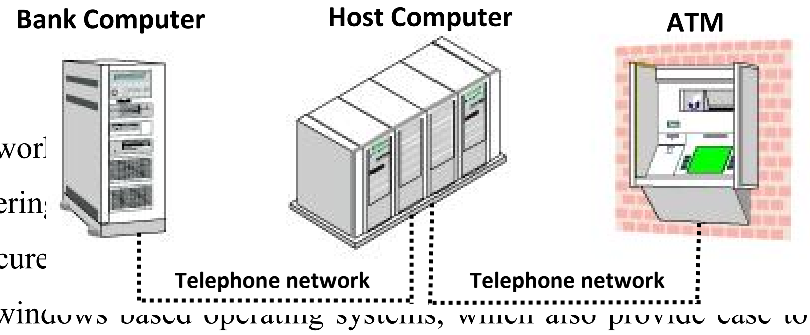
The physical security of the ATMs can be increased by using the techniques such as dye markers and smoke canisters. In order to give right person to access the account, and to increase the security level in the ATMs, security cameras and security guards are a common feature and the manufactures also deployed several different techniques on ATMs. The different techniques include biometric scanning, bar code scanning etc.

ATM machines are broken down into two different types of ATM classifications.

- Armored ATM: Armored ATM’s are used primarily as outdoor ATM’s. Such as outdoor entertainment districts, outdoor flea markets, outdoor concert and events, etc.
- Non-armored ATM's: Non-armored ATM’s are used primarily for indoor locations that are closed to the public when employees are not present. Such as indoor malls, movie theaters, night clubs, restaurants, bars, etc

3.4 Literature Survey on ATM Operations

ATM accesses the bank account through telephone networking, a host processor and a bank computer to verify the data. All the operations of ATM are in synchronization in reference to a common clock source. In case of any misconnection between the process steps, the whole transaction is rolled back to effect no change in the account status.



Banks wor akes care that no tamperin s to have complete secure llers are working on windows based operating systems, which also provide ease to designers for programming and modeling its operation. ATM controllers are sophisticated chip elements designed and programmed to make the transaction simple and user-friendly.

The ATM must be able to provide the following services to the customer:

- A customer must be able to make a cash withdrawal from any suitable account linked to the card, approval must be obtained from the bank before cash is dispensed.
- A customer must be able to make a deposit to any account linked to the card, consisting of cash and/or checks in an envelope. The customer will enter the amount of the deposit into the ATM, subject to manual verification when the envelope is removed from the machine by an operator. Approval must be obtained from the bank before physically accepting the envelope.
- A customer must be able to make a transfer of money between any two accounts linked to the card.
- A customer must be able to make a balance inquiry of any account linked to the card.

The ATM will communicate each transaction to the bank and obtain verification that it was allowed by the bank. Ordinarily, a transaction will be considered complete by the bank once it has been approved. In the case of a deposit, a second message will be sent to the bank indicating that the customer has deposited the envelope. If the customer fails to deposit the envelope within the timeout period, or presses cancel instead, no second message will be sent to the bank and the deposit will not be credited to the customer.

If the bank determines that the customer's PIN is invalid, the customer will be required to re-enter the PIN before a transaction can proceed. If the customer is unable to successfully enter the PIN after three tries, the card will be permanently retained by the machine, and the customer will have to contact the bank to get it back. If a transaction fails for any reason other than an invalid PIN, the ATM will display an explanation of the problem, and will then ask the customer whether he/she wants to do another transaction.

The ATM will provide the customer with a printed receipt for each successful transaction, showing the date, time, machine location, type of transaction, account(s), amount, and ending and available balance(s) of the affected account.

The ATM will have a key-operated switch that will allow an operator to start and stop the servicing of customers. After turning the switch to the "on" position, the operator will be required to verify and enter the total cash on hand. The machine can only be turned off when it is not servicing a customer. When the switch is moved to the "off" position, the machine will shut down, so that the operator may remove deposit envelopes and reload the machine with cash, blank receipts, etc.

The ATM will also maintain an internal log of transactions to facilitate resolving ambiguities arising from a hardware failure in the middle of a transaction. Entries will be made in the log when the ATM is started up and shut down, for each message sent to the Bank (along with the response back, if one is expected), for the dispensing of cash, and for the receiving of an envelope. Log entries may contain card numbers and amounts, but for security will never contain a PIN.

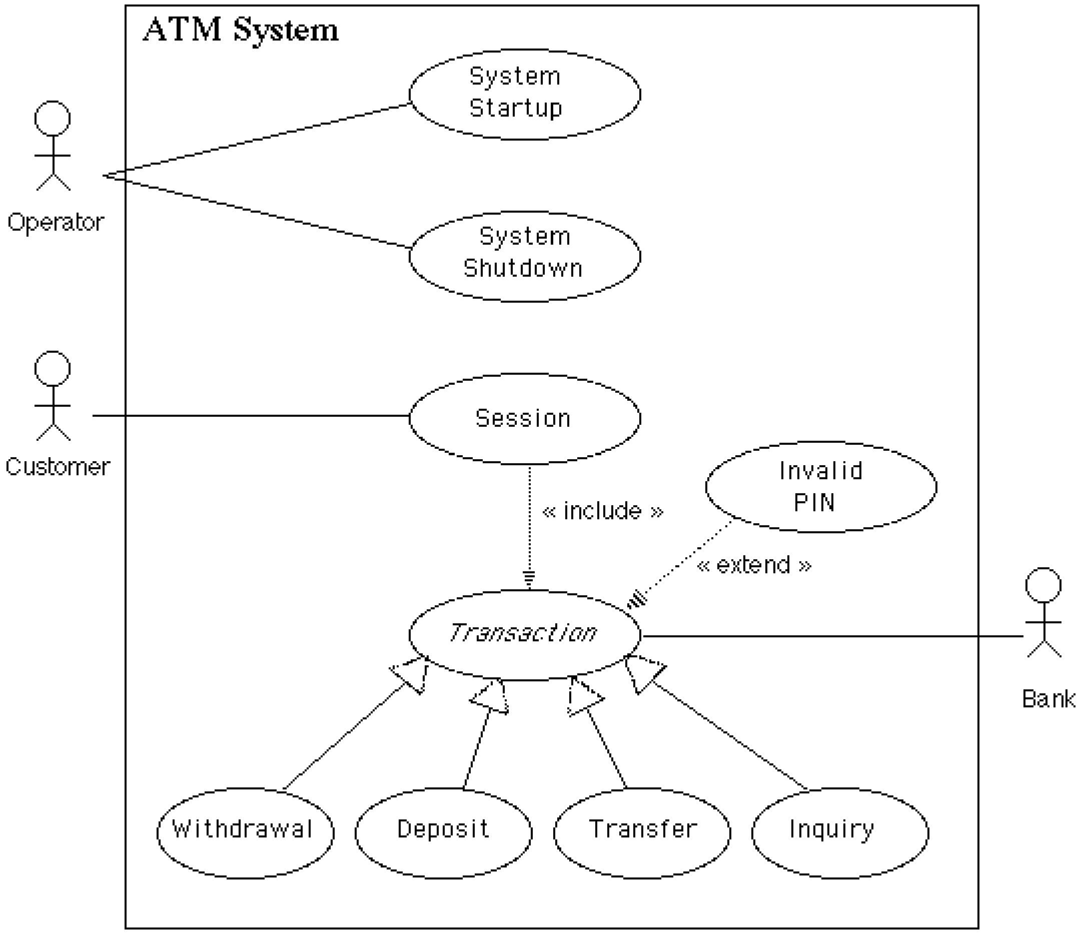


Fig.3.3 General Flow of ATM Process

- ***System Startup Case***

The system is started up when the operator turns the operator switch to the "on" position. The operator will be asked to enter the amount of money currently in the cash dispenser, and a connection to the bank will be established. Then the servicing of customers can begin.

- ***System Shutdown Case***

The system is shut down when the operator makes sure that no customer is using the machine, and then turns the operator switch to the "off" position. The connection to the bank will be shut down. Then the operator is free to remove deposited envelopes, replenish cash and paper, etc.

- ***Session Case***

A session is started when a customer inserts an ATM card into the card reader slot of the machine. The ATM pulls the card into the machine and reads it. If the reader cannot read the card due to improper insertion or a damaged stripe, the card is ejected, an error screen is displayed, and the session is aborted. The customer is asked to enter PIN, and is then allowed to perform one or more transactions, choosing from a menu of possible types of transaction in each case.

After each transaction, the customer is asked whether he/she would like to perform another. When the customer is through performing transactions, the card is ejected from the machine and the session ends. If a transaction is aborted due to too many invalid PIN entries, the session is also aborted, with the card being retained in the machine. The customer may abort the session by pressing the Cancel key when entering a PIN or choosing a transaction type.

- ***Transaction Case***

A transaction use case is started within a session when the customer chooses a transaction type from a menu of options. The customer will be asked to furnish appropriate details (e.g. account involved, amount). The transaction will then be sent to the bank, along with information from the customer's card and the PIN the customer entered.

If the bank approves the transaction, any steps needed to complete the transaction (e.g. dispensing cash or accepting an envelope) will be performed, and then a receipt will be printed. Then the customer will be asked whether he/she wishes to do another transaction.

If the bank reports that the customer's PIN is invalid, the Invalid PIN extension will be performed and then an attempt will be made to continue the transaction. If the customer's card is retained due to too many invalid PINs, the transaction will be aborted, and the customer will not be offered the option of doing another.

State-Chart for One Session

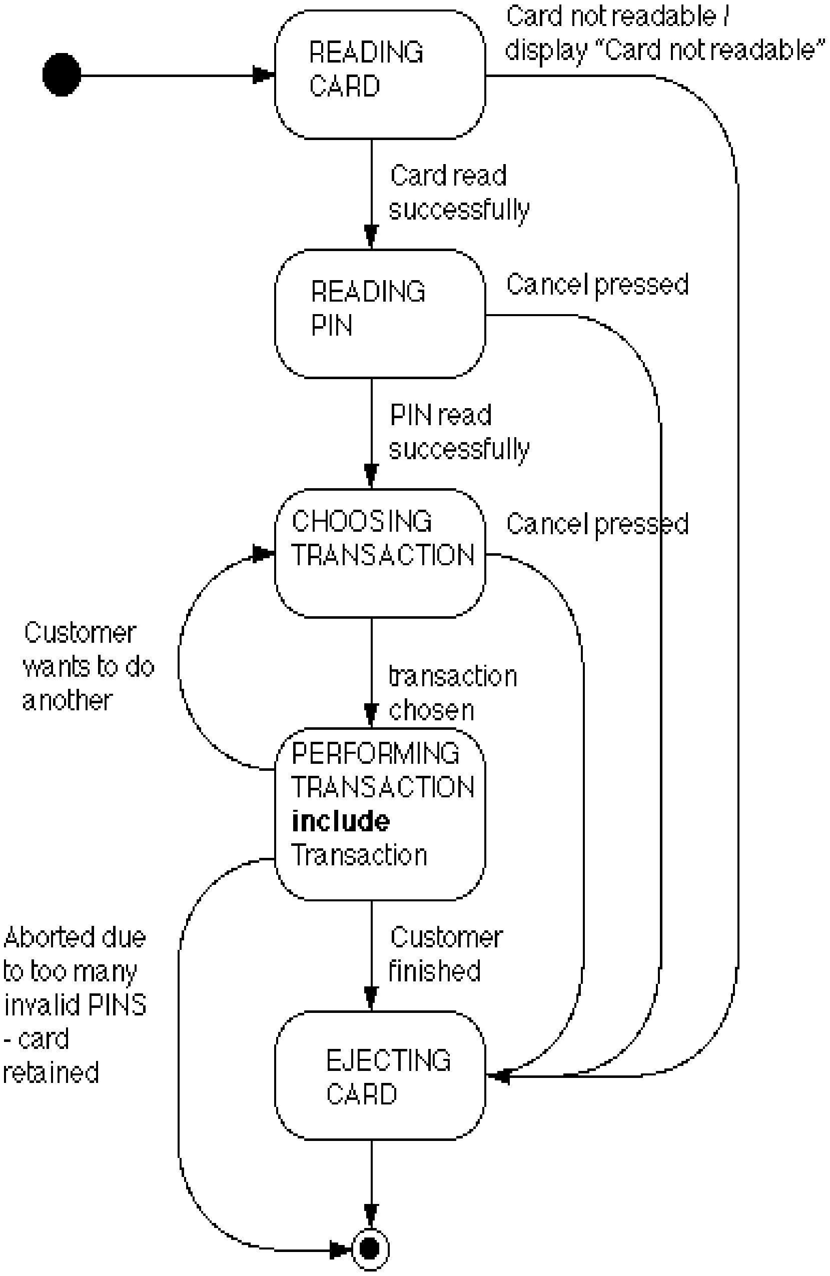
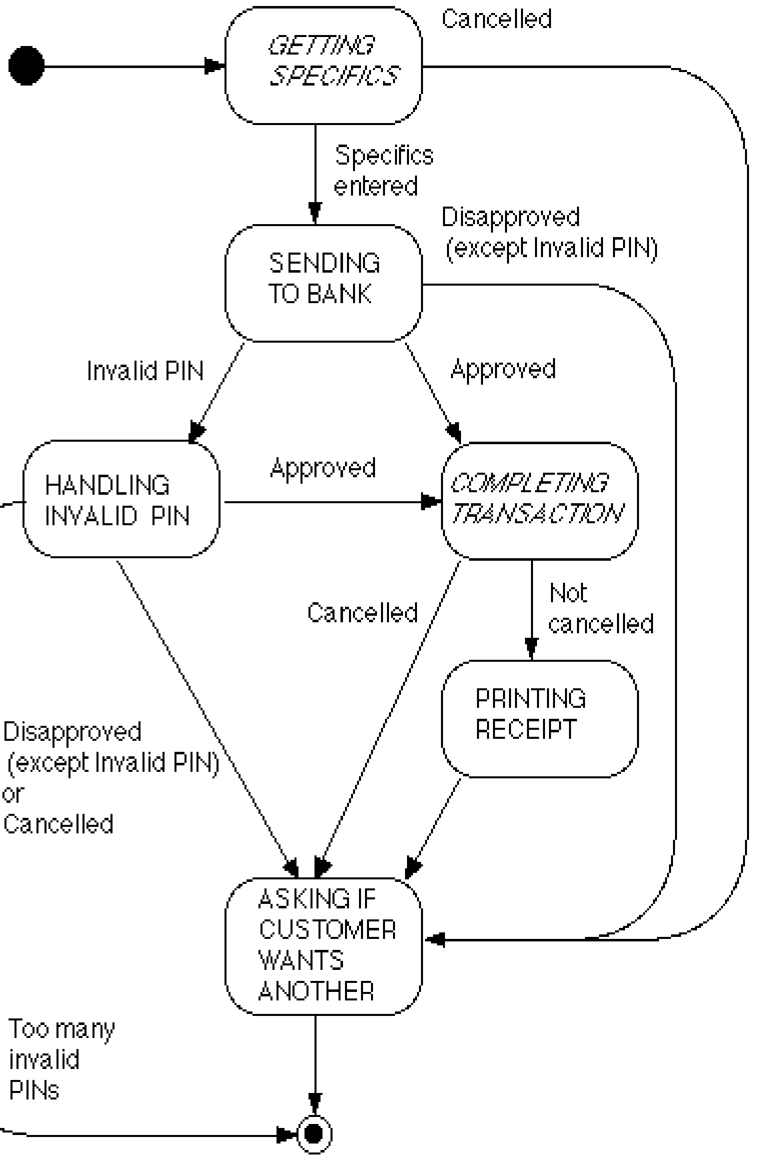

READING CARD
Card not readable / display "Card not readable"
Card read successfully
READING PIN
Cancel pressed
PIN read successfully
CHOOSING TRANSACTION
Cancel pressed
Customer wants to do another
transaction chosen
PERFORMING TRANSACTION include Transaction
Aborted due to too many invalid PINS - card retained
Customer finished
EJECTING CARD

Fig.3.4 Flow Chart for One Session

If a transaction is cancelled by the customer, or fails for any reason other than repeated entries of an invalid PIN, a screen will be displayed informing the customer of the reason for the failure of the transaction, and then the customer will be offered the opportunity to do another.



State-Chart for One Transaction
(italicized operations are unique to each
particular type of transaction)

Fig.3.5 Flow Chart for One Transaction

- ***Withdrawal Transaction Case***

A withdrawal transaction asks the customer to choose a type of account to withdraw from (e.g. checking) from a menu of possible accounts, and to choose a amount from a menu of possible amounts. The system verifies that it has sufficient money on hand to satisfy the request before sending the transaction to the bank. If not, the customer is informed and asked to enter a different amount. If the transaction is approved by the bank, the appropriate amount of cash is dispensed by the machine before it issues a receipt. The dispensing of cash is also recorded in the ATM's log. The customer pressing the Cancel key any time prior to choosing the dollar amount can cancel a withdrawal transaction.

- ***Deposit Transaction Case***

A deposit transaction asks the customer to choose a type of account to deposit to (e.g. checking) from a menu of possible accounts, and to type in a amount on the keyboard. The transaction is initially sent to the bank to verify that the ATM can accept a deposit from this customer to this account. If the transaction is approved, the machine accepts an envelope from the customer containing cash and/or checks before it issues a receipt. Once the envelope has been received, a second message is sent to the bank, to confirm that the bank can credit the customer's account - contingent on manual verification of the deposit envelope contents by an operator later.

- ***Invalid PIN Extension***

An invalid PIN extension is started from within a transaction when the bank reports that the customer's transaction is disapproved due to an invalid PIN. The customer is required to re-enter the PIN and the original request is sent to the bank again. If the bank now approves the transaction, or disapproves it for some other reason, the original use case is continued; otherwise the process of re-entering the PIN is repeated.

Once the PIN is successfully re-entered, it is used for both the current transaction and all subsequent transactions in the session. If the customer fails three times to enter the correct PIN, the card is permanently retained, a screen is displayed informing the customer of this and suggesting he/she contact the bank, and the entire customer session is aborted.

3.5 ATM Controller and its Working

ATM controller controls the entire operation of the ATM. Through the control state machine, it communicates with all other blocks and interacts with memory for all updates. It works in handshake mode for all memory operations.

The ATM controller logic is grouped into few sub-modules to organize and simplify the operation. Each module is defined for a particular task. Controller checks the interface to all modules and manages the correlation between them. The major modules of the controller are:

- Controller State Machine
- Card Information storage module
- PIN parser
- Fund Checker
- Entered Amount parser
- Transaction verifier

3.5.1 Controller State Machine

Controller State Machine (CSM) is the brain of the ATM, which monitors its working. It detects the status signals and generates proper control signals for further action. It needs to generate appropriate signals for the display section for every sub-task been performed. For secured access and proper updates, the CSM works in handshake mode with the memory. It is a programmed microcontroller to control all the activities of the controller.

3.5.2 Card Information Storage Module

When a card is swiped, the card swiper sends a high CardScanned signal to the ATM controller to represent availability of the 16-bit CardNumber and 4-bit PIN. The CardScanned signal enables 4-bit PIPO register at the card information storage module to store it. The CardNumber is fed to a 16-bit 2:1 multiplexer. CheckStatus and StoreAccNum signals from CSM control the select line of the multiplexer. Initially the CSM generated a low CheckStatus signal to read the CardNumber and provide it to the LUT for checking its LockStatus.

If the account is locked, the controller generates AccountStatus signal to the display and welcome signal to initialize the ATM. If unlocked, StoreAccNum signal is set high to enable the 16-bit PIPO register to store the AccountNumber. In addition, enterPin signal is generated for the display to ask user for PIN.

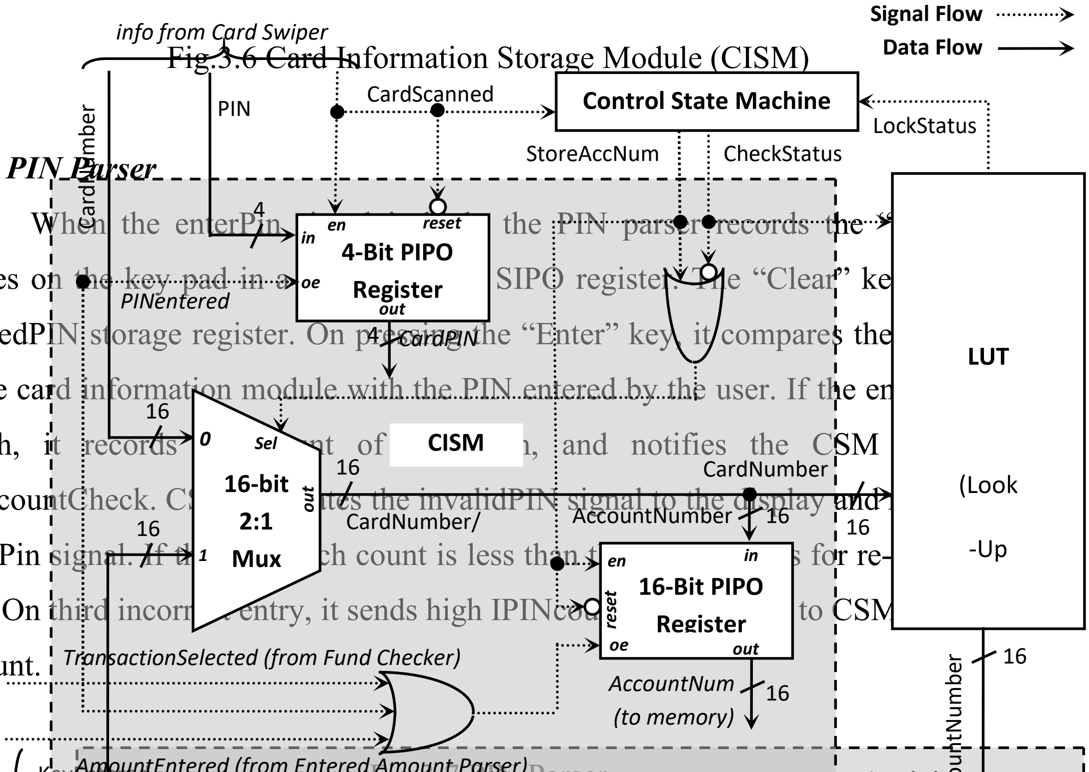


Fig.3.6 Card Information Storage Module (CISM)

3.5.3 PIN Parser

When the enterPin the PIN parser records the entries on the key pad in a SIPO register. The “Clear” ke enteredPIN storage register. On pressing the “Enter” key, it compares the in the card information module with the PIN entered by the user. If the en match, it records t of n, and notifies the CSM IPINcountCheck. CS tes the invalidPIN signal to the display and enterPin signal. If th ch count is less than t s for re- PIN. On third incorr entry, it sends high IPINco to CSM account.

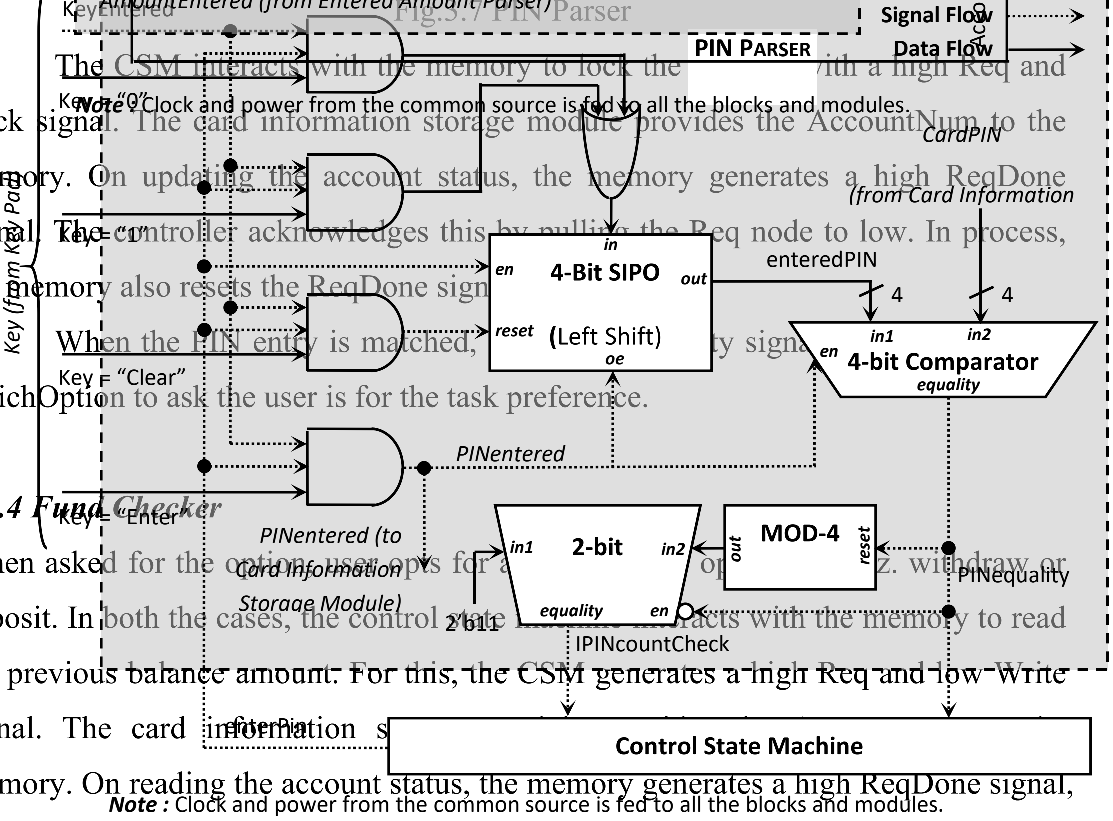


Fig.3.7 PIN Parser

The CSM interacts with the memory to lock the with a high Req and Lock signal. The card information storage module provides the AccountNum to the memory. On updating the account status, the memory generates a high ReqDone signal. The controller acknowledges this by pulling the Req node to low. In process, the memory also resets the ReqDone sign

When the PIN entry is matched, ty signa whichOption to ask the user is for the task preference.

3.5.4 Fund Checker

When asked for the option, user opts for a op z. withdraw or deposit. In both the cases, the control state machine interacts with the memory to read the previous balance amount. For this, the CSM generates a high Req and low Write signal. The card information s memory. On reading the account status, the memory generates a high ReqDone signal, and provides the oldBalance at AmountOut. The controller acknowledges this by pulling the Req node to low. In process, the memory also resets the ReqDone signal. The AmountOut from the memory is stored in a 16-bit PIPO register in the fund checker, and displayed. This amount is sent to the entered amount parser and display when TransactionSelected signal goes high.

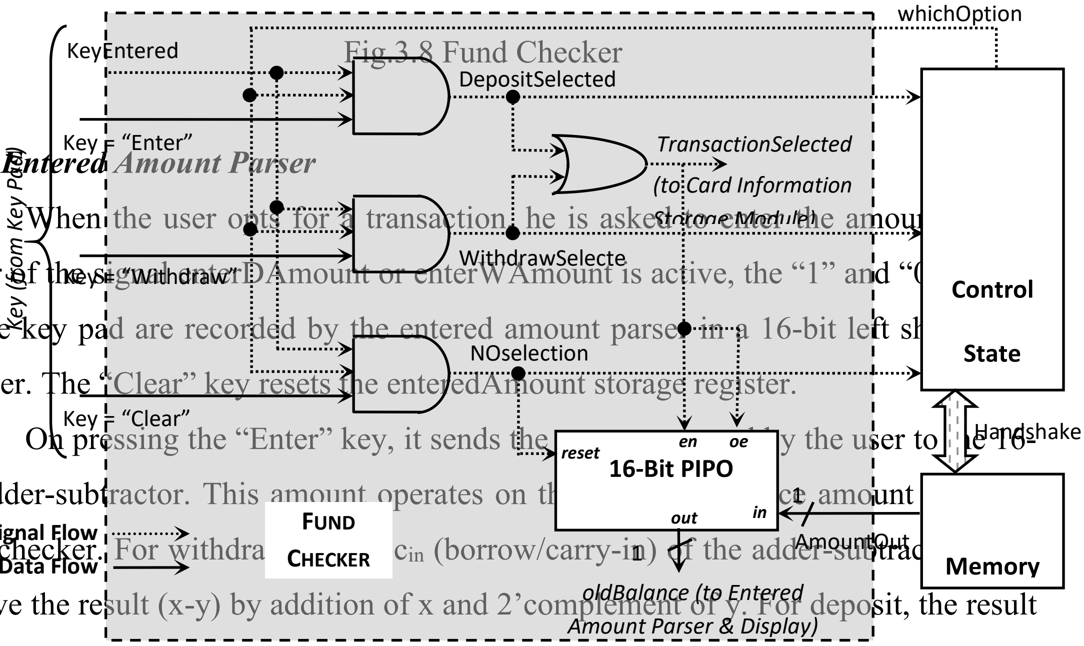


Fig.3.8 Fund Checker

Note : Clock and power from the common source is fed to all the blocks and modules.

3.5.5 Entered Amount Parser

When the user opts for a transaction, he is asked to enter the amou [illegible] either of the signal enterDAmount or enterWAmount is active, the "1" and "[illegible] on the key pad are recorded by the entered amount parser in a 16-bit left sh[illegible] register. The "Clear" key resets the enteredAmount storage register.

On pressing the "Enter" key, it sends the [illegible] y the user to [illegible]e 16-bit adder-subtractor. This amount operates on th[illegible]ce amount [illegible] fund checker. For withdra[illegible] c_{in} (borrow/carry-in) of the adder-subtrac[illegible] to have the result (x-y) by addition of x and 2'complement of y. For deposit, the result is (x+y). In any case, if the bc_{out} line goes high, it represents underflow for withdrawal and overflow for deposit. This line serves as status signal for the viability of the transaction. If high, CSM invalidates the transaction, else asks for verification from user to proceed.

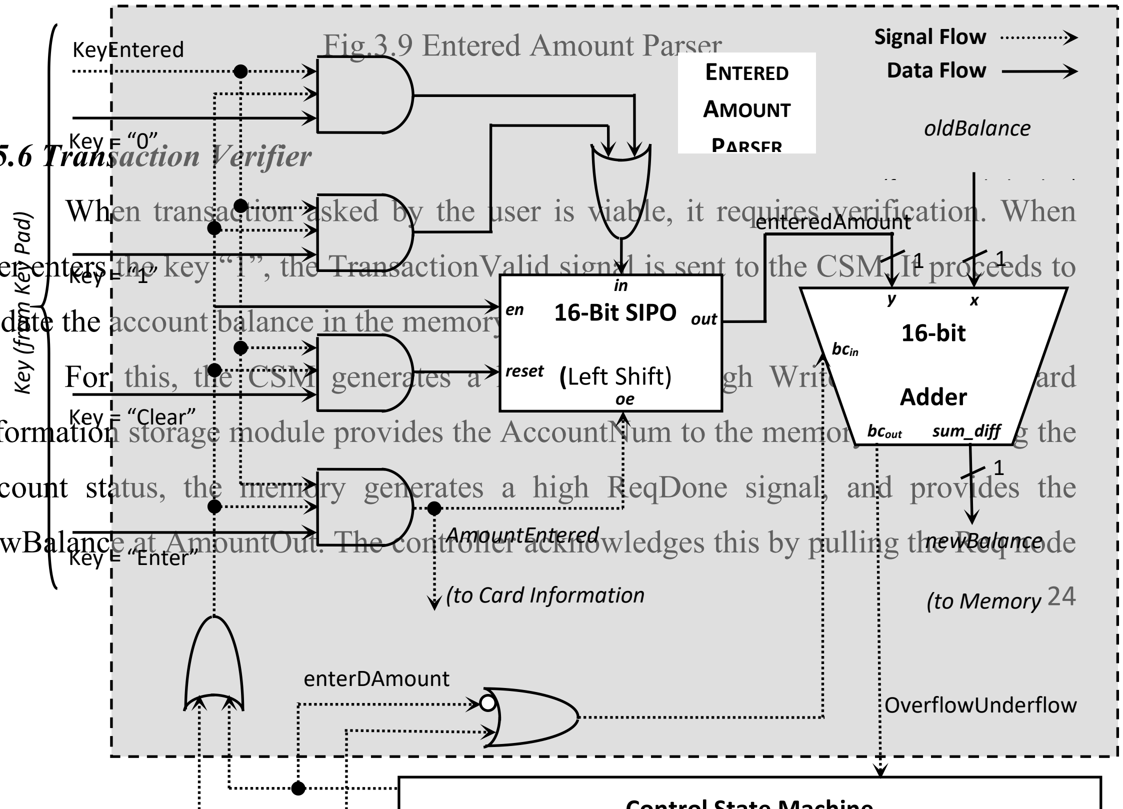


Fig.3.9 Entered Amount Parser

3.5.6 Transaction Verifier

When transaction asked by the user is viable, it requires verification. When user enters the key "1", the TransactionValid signal is sent to the CSM. It proceeds to update the account balance in the memory[illegible]

For this, the CSM generates a [illegible] gh Writ[illegible] ard information storage module provides the AccountNum to the memor[illegible] g the account status, the memory generates a high ReqDone signal, and provides the newBalance at AmountOut. The controller acknowledges this by pulling the Req node to low. In process, the memory also resets the ReqDone signal. The updated account balance is displayed to the user.

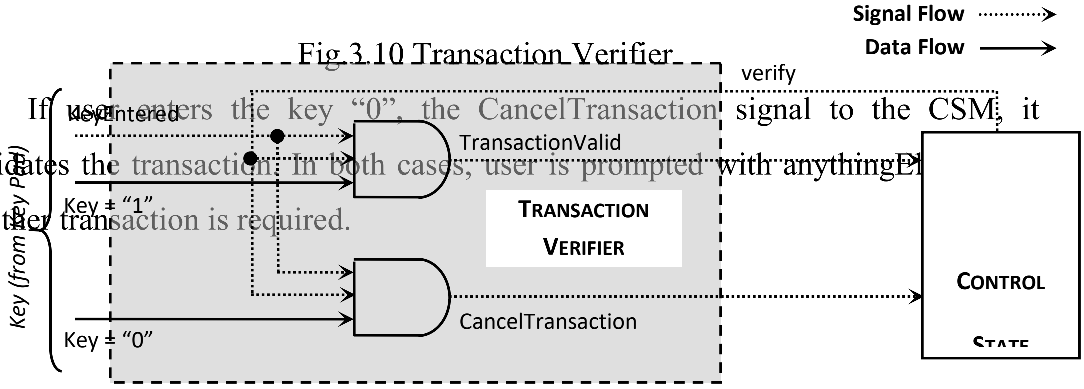


Fig 3.10 Transaction Verifier

If user enters the key “0”, the CancelTransaction signal to the CSM, it invalidates the transaction. In both cases, user is prompted with anythingEl any other transaction is required.

Note : Clock and power from the common source is fed to all the blocks and modules.

CHAPTER 4

DESIGN OF AN ATM CONTROLLER

4.1 Introduction

Basically in this work controller for an ATM is created. There are many different parts of the controller that is designed. First ATM controller should store the important information from the Card Swiper. When the CardScanned line goes high, a new card has been scanned and the data is valid on both the CardNumber line and the PIN line. Next ATM controller must be able to parse an entered PIN. When the Key Entered line goes high, at most one of the bits of Key will be high. A valid PIN is one that matches the one stored from the card, so it is 4 bits long and contains 1s and 0s. Once the PIN has been entered, the Enter key should be pressed. (If the Clear button is pressed, the inputted PIN should be reset). 3 invalid PINs results in a Locking of the account. When a PIN is invalid, the state machine should continue to try to get a new PIN until 3 invalid ones have been entered.

Once a valid PIN is entered the user has an option to either withdraw or deposit money. A deposit consists of getting the amount from the keypad and then checking with memory to see if the current balance would overflow (16 bits) based on the deposit amount. If the amount causes an overflow, the resulting balance is not written back memory the "anythingElse" signal is asserted as well as the "InvalidTransaction" signal. In the event that the transaction is valid, the user is taken to a screen where they can verify the transaction and new balance ("verify") from which an "Enter" input sends a request to update the balance in memory and upon completion prompts them with "anythingElse"

A withdraw works very similarly. A withdraw amount is entered from the keypad and then the controller must get the current balance from memory and determine if you can remove that much from the account. Again, if it is possible, the new balance is written to memory, if not then nothing is written to memory and the InvalidTransaction line goes high to the display, and then the controller should ask if there is another transaction to be completed. When the "is there another transaction?" ("anythingElse") screen is up, the "0" is no and "1" is yes. "0" will return the user to the initial state of waiting for a card to be swiped, and "1" returns them to the "which transaction?" state.

The handshaking with memory works as follows:

- ➢ The controller sets Req to high with a valid account number
- ➢ If Lock is HIGH then,

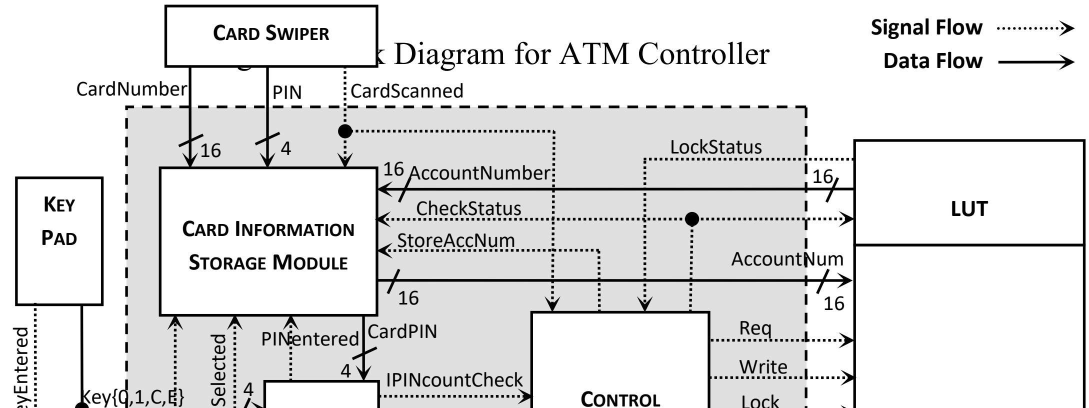
    - Memory will set ReqDone to high when it is done locking the account
    - When Req is set to low, ReqDone will go low
- Else if Write is LOW then…
    - Memory will set ReqDone to high and have the current balance on AmmountOut
    - when Req is set to low, ReqDone will go low
- Else if Write is HIGH then…
    - Memory will set ReqDone to high when done writing the new value that is valid on AmmountIn
    - When Req is set to low, ReqDone will go low

4.2 ATM Controller Block Diagram

All the above blocks integrate together to form the controller of ATM. The ATM controller as a whole interacts with all the other blocks of ATM – Card swiper, Key pad, LUT, memory and Display unit. The block diagram in the Fig.4.1 shows the detailed interaction of the ATM controller with other blocks as well within itself.

To summarize the operation, the ATM controller is activated on swipe of a card at the card swiper. It then checks its lock status and asks to enter the PIN only when unlocked.

User has to enter the correct PIN within three trials, after which ATM corresponds with the memory to lock the account and no further transaction on the account is possible. On getting the PIN matched, user is asked for the option for the operation he wants to do. For both withdrawal and deposit, he is been provided with the current balance of the account and asked for the transaction amount. A viable transaction amounts prompts user to verify the transaction, else flags an invalid transaction to the user. Similarly, if user, opts to neglect the verify option, the transaction goes to be invalid. If verified, the account details are updated and displayed.



k Diagram for ATM Controller

In all cases, and asks if he wants another transaction. If yes, which option is asked for else, the ATM resets with a welcome signal, waiting for the next card to be swiped. At every step, proper signals are generated to display the next instructions to the user. The display section has a LUT defined for each signal w.r.t. which the display screen changes. The signals that trigger the change of display are also used as the internal control signal to activate various modules of the controller. This results successful transactions every time, with elimination of any possibility of miscalculation.

4.3 Moore Machine State Diagram for ATM:

Basically FSM is a representation of the different transition taken place in a system. A state machine is an effective way to implement the control functions. The Moore Machine for the overall ATM controller is represented in fig.4.2. The description of state code and state input signal are tabulated in table 4.1 and table 4.2.

Operation:

Initially the system is in idle state and when it is ready to operate, S0 state is selected which represents the scan card. When the user gets the card inserted, scan line goes high which represents the next state S1. If the card line goes low, control signal will return back to the scan card state. In state S2, card number and pin number is stored in the registers. If any error occurs in storing the card information, the signal will go the state S0.

After storing the card information, control signal will leads to the screen in which the user allowed to enter the pin which is represented by the state S3. When the invalid pin is entered, signal will again goes to state S3 and allow the user to enter pin again. The counter will be installed which is represented as S4 state, to count the number of trials. If it exceeds three times, the account gets locked and control signal return to scan card (S0) state.

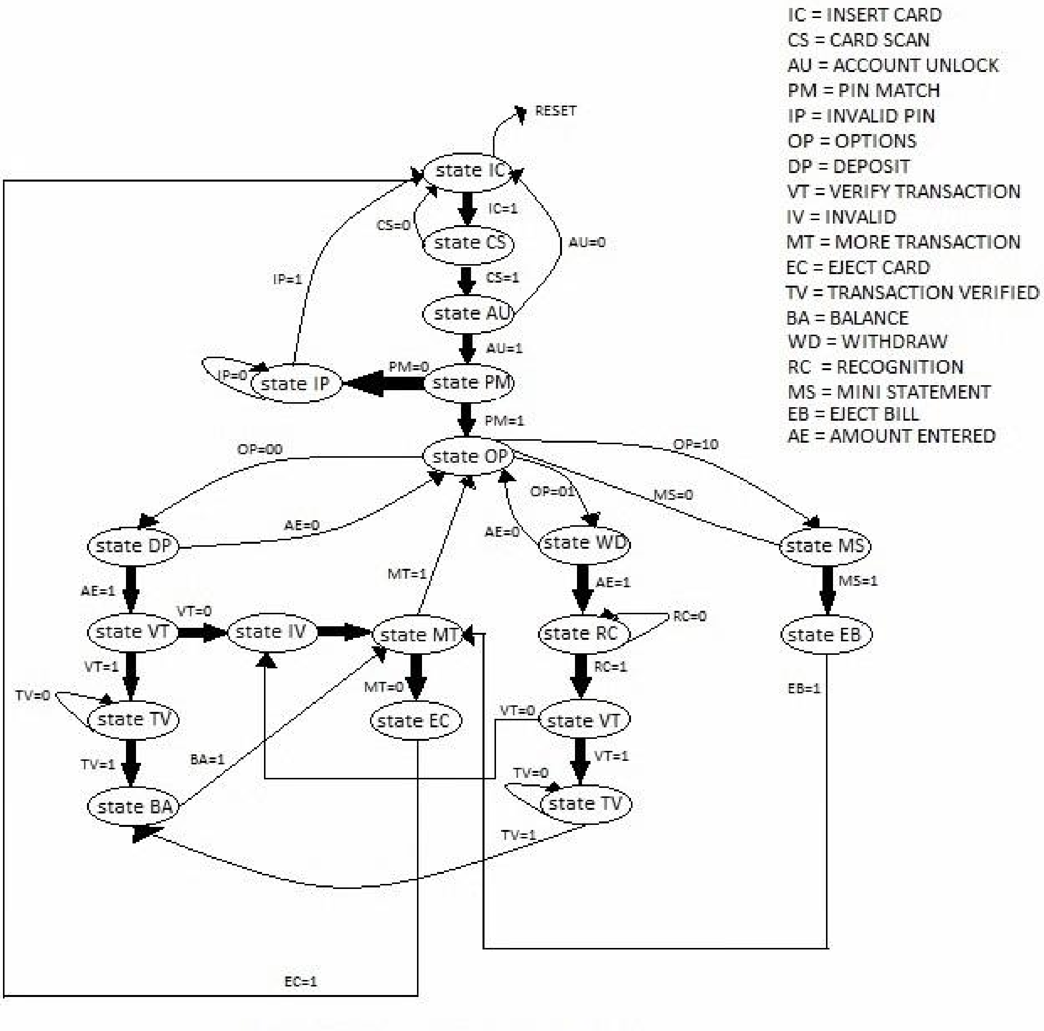
When the valid pin is entered i.e. pin get matched, control signal will lead to the state S5 in which user can select the transaction type.

There are two types of transaction can be made as follows.

1) Deposit
2) Withdraw

Deposit state is represented by S6 in which the user allowed to deposit the amount through amount entering. User will be taken again to transaction type screen by making cancel deposit to high.

The amount entered will be added with the old balance that stored in memory and checked whether it overflow 16bit, represented by state S7. If balance get overflowed, control signal will go to invalid state S8 and then to anything else state S9. The state S10 represents entered amount is valid and is verified. Once verification is over, the updating of balance is made which is represented by the state S11. After updating, the signal will goes to the anything else state S9 from which user will be allowed to make more transaction by taking to state S5 else to the state S0.



MOORE STATE MACHINE DIAGRAM FOT ATM

Fig.4.2 Moore Machine Diagram for ATM Controller

Withdraw state is represented by S12 in which the user allowed to enter the withdraw amount from his account. If the withdraw process is cancelled, signal will again goes to the transaction type state S5 else the amount entered will be checked through the state S13.

Table 4.1 State Code Descriptions

State Code	State Description
S0	Scan Card
S1	Scan Line
S2	Storing information
S3	Enter PIN
S4	Count
S5	Transaction type
S6	Enter Deposit Amount
S7	Deposit Check
S8	Invalid
S9	Anything Else
S10	Deposit Verify
S11	Updating Balance
S12	Enter Withdraw Amount
S13	Withdraw Check
S14	Withdraw Verify

Table 4.2 Table showing input signals

Signal Code	Signal Description
IC	Insert Card
CS	Card Scan
AU	Account Unlocked
PM	PIN Matched
IPin	Invalid Pin Count
OP	Option Select
AE	Amount Entered
VT	Valid Transaction
MT	More Transaction
TV	Transaction Verified

After checking process will be carried out, if amount entered is invalid, control signal will go to invalid state S8 and from which signal goes to another transaction state S9. If the entered amount is valid, then state S14 will be processed which will verify the amount and leads to the updating balance state S11. In this state, the amount will get updated in memory. After updating, user is taken to another transaction state S9 from which he can make more transaction else taken to scan card state.

4.4 General Implementation Flow

The generalized implementation flow diagram of the project is represented as follows.

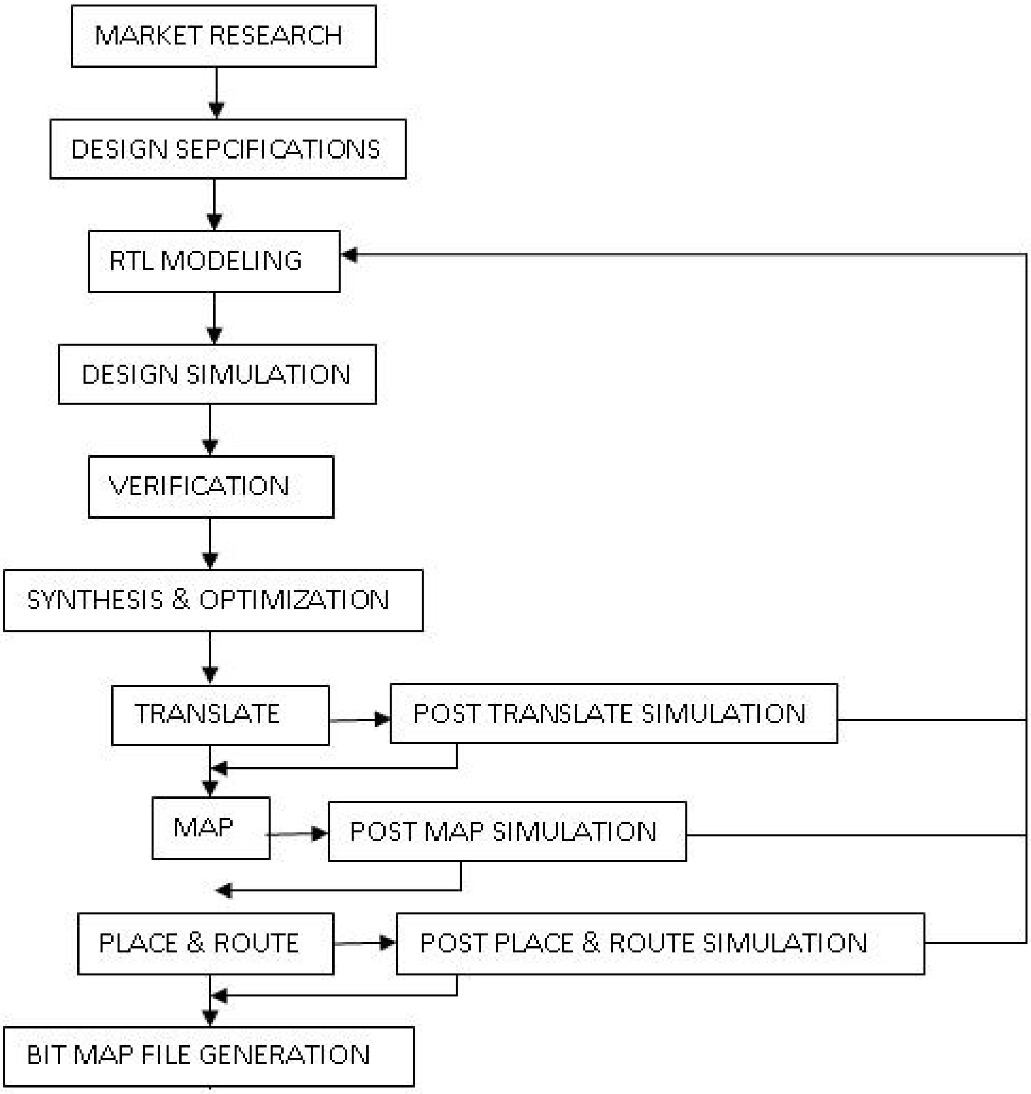


Fig.4.3 General Implementation Flow Diagram

Initially the market research should be carried out which covers the previous version of the design and the current requirements on the design. Based on this survey, the specification and the architecture must be identified. Then the RTL modeling should be carried out in Verilog HDL with respect to the identified architecture. Once the RTL modeling is done, it should be simulated and verified for all the cases. The functional verification should meet the intended architecture and should pass all the test cases.

Once the functional verification is clear, the RTL model will be taken to the synthesis process. Three operations will be carried out in the synthesis process such as

- Translate
- Map
- Place and Route

The developed RTL model will be translated to the mathematical equation format which will be in the understandable format of the tool. These translated equations will be then mapped to the library that is, mapped to the hardware. Once the mapping is done, the gates were placed and routed. Before these processes, the constraints can be given in order to optimize the design. Finally the BIT MAP file will be generated that has the design information in the binary format which will be dumped in the FPGA board.

4.5 Implementation Using Verilog HDL:

Using the Verilog, which is the third generation of Verilog IEEE 1800-2005 standard, does the implementation of the above Moore machine state diagram. Basically, data type is one that will not consider the actual state encoding that is used in the design and it will allow implementing the FSM of that design. The user-defined variables are get updated automatically.

The inputs of the FSM are taken as input to the code that is tabulated in table 4.1 and the output is taken as 'z'. In the input, pin entered is of 4-bits and the amount entered for both deposit and withdraw is of 16-bits which are declared as 'Logic'. The output is declared as the 'Logic' which is of four states 0, 1, x and z. This data type is identical to reg data type and is a user defined vector size. The user-defined variables, shown in table 4.2, are declared in the data type.

Totally three always blocks are used in the code in which two are sequential and one is combinational. Always block can be used to infer the combinational and sequential logic respectively. In first always block, if reset signal is set the signal will goes to 'scan card' state else goes to next state. Second always block represents the state machine operation and hence if the input signal is high, control signal will goes to next state else will goes to the previous state i.e. corresponding state respectively.

The sensitivity list has ***'negedge clk'*** which represents that the change takes place at every negative edge of the clock signal. Here the counter is installed which will be counting the entering of invalid pin. If it exceeds three times, the account gets locked and the signal will return to initial state and resets the counter. The (ipin_count) ‘Invalid pin count’ is declared as a ‘logic’ data type is used to increment the count. An addition operation is used in the verify state after which the entered amount gets added with previous balance. Similarly, a subtraction operation is performed for withdraw operation in order to subtract the entered amount with the previous balance. If the net balance is larger than 16-bits, then the transaction will become invalid. In third always block, the output are declared with respect to the state. Once the balance gets updated, the output signal is made to high.

4.6 Summary

- The implementation requirement which includes the primary input and primary output of the design and the proper notation and conventions were discussed.
- General implementation flow of the design were represented and explained in order to understand the proper flow.
- Implementation details have been discussed which includes implementation style of each process.

CHAPTER - 5

INTRODUCTION & INSTALLATION OF XILINX ISE 12.3 ON WIN 7

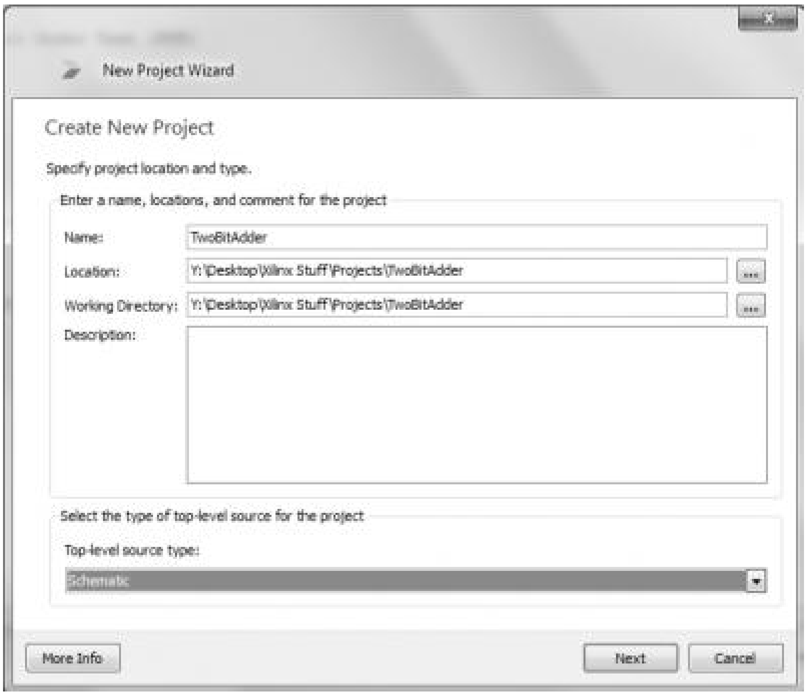
5.1 Introduction

Xilinx's ISE is "Xilinx ISE is a software tool produced by Xilinx for synthesis and analysis of HDL designs, which enables the developer to synthesize ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer."

The goal of this lab/tutorial is to get the reader familiar with the process of designing a simple digital electronic circuit, *compiling* it, and verifying it correct behavior with a simulator. Knowledge of digital logic (basic gates, flip-flops, *Moore* machines) is assumed. This lab is based on the excellent series of labs created for the CoolRunner CPLD by Tiffany Liu in her Independent Study in the CS. Dept. at Smith College.

5.2 Example: Creating a 2-bit Adder in Verilog

- ***New Project:***



New Project Wizard

Create New Project

Specify project location and type.

Enter a name, locations, and comment for the project

Name: TwoBitAdder

Location: Y:\Desktop\Xilinx Stuff\Projects\TwoBitAdder

Working Directory: Y:\Desktop\Xilinx Stuff\Projects\TwoBitAdder

Description:

Select the type of top-level source for the project

Top-level source type: Schematic

Fig.5.1 Project window

- ❖ Open the ISE
- ❖ File/New Project
- ❖ Pick a name: **TwoBitAdderV**
- ❖ Accept the default location
- ❖ Top-Level source: **HDL**
- ❖ Project Settings:
    - Family: **CoolRunner2 CPLDs**
    - Device: **XC2C257** (this is the marking on the CPLD on the actual kit)
    - Package: **TQ144** (also marked on the CPLD on the actual kit)
    - Speed: -7
    - Keep all others unchanged.
- ❖ ***New Source***

- Click on top left icon (see image to the right) to add a new source to the project.
- Pick **Verilog Module** as the type
- Name it with a name that makes sense, e.g. **circuit2**.
- in the **Define Module** window, add **A** and **B** as inputs, and **Sum** and **Carry** as outputs.
- **Finish**
- You should then see the following *template* ready for you to complete:

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    16:36:46 04/16/2012
// Design Name:
// Module Name:    circuit2
// Project Name:
// Target Devices:
// Tool versions:
```

```
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module circuit2(
    input A,
    input B,
    output Carry,
    output Sum
    );


endmodule
```

- Edit the *Verilog* file, and add just to lines:

```
module circuit2(
    input A,
    input B,
    output Carry,
    output Sum
    );

        and( Carry, A, B );
        xor( Sum, A, B );

endmodule
```

- Type **Control-S** to save the Verilog code to file.

5.3. Implementation Constraints File

Because we are not downloading the programming file to a device, we can skip the **Implementation Constraints File** creation step.

5.4. Implement Design

- Select the **circuit2.sch** file in the **Hierarchy** window
- In the **Process** window, double click on **Implement Desgin**. This will automatically call all the actions listed above. The result is a programming file that will appear in the *TwoBitAdderV* project directory.

5.5. Testing the design with the simulator

This step will allow you to create a module that will make A and B take all the possible values ranging from 00, 01, 10, to 11, and see how the two-bit adder circuit reacts to it.

- First create a new simulation module: From the main menu, pick **Project** then **New Source**.
- Choose **Verilog Test Fixture** as the type of the module. Give it a meaningful name, for example *test.*
- Click **Next** and make sure that your original schematic module is selected.
- **Next** then **Finish**.
- the ISE will have generated a test module for us. It's almost what we need. We just need to modify it a tad, as shown below:

5.6. Program for two bit adder

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////
```

```
// Company:
// Engineer:
//
// Create Date:   16:48:53 04/16/2012
// Design Name:   circuit2
// Module Name:   Y:[...]
// Project Name:  TwoBittAdderV
// Target Device:
// Tool versions:
// Description:
//
// Verilog Test Fixture created by ISE for module: circuit2
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////////////////

module test;

	// Inputs
	reg A;
	reg B;

	// Outputs
	wire Carry;
	wire Sum;

	// Instantiate the Unit Under Test (UUT)
	circuit2 uut (
        .A(A),
        .B(B),
        .Carry(Carry),
        .Sum(Sum)
    );

    initial begin
        // Initialize Inputs
        A = 0;
        B = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here
        #10  A = 1;
        #10 A = 0; B = 1;
        #10   A = 1;

    end

endmodule
```

5.7. Steps for Simulation

- Click on the **Simulation** button on top of the **Hierarchy** pane.
- The **ISim Simulator** should appear in the **Process** pane, below.
- Double click on **Behavioral Check Syntax**
- Then, assuming the process completed successfully, double click on **Simulate Behavioral Model**
- A new window should open up, presenting a timing diagram. Use slider and the magnifying glass + and - icons to zoom in on the marker at Time 100ns, and see how **Sum** and **Carry** react to the changing **A** and **B** signals.
- Make sure you verify that the adder works correctly.

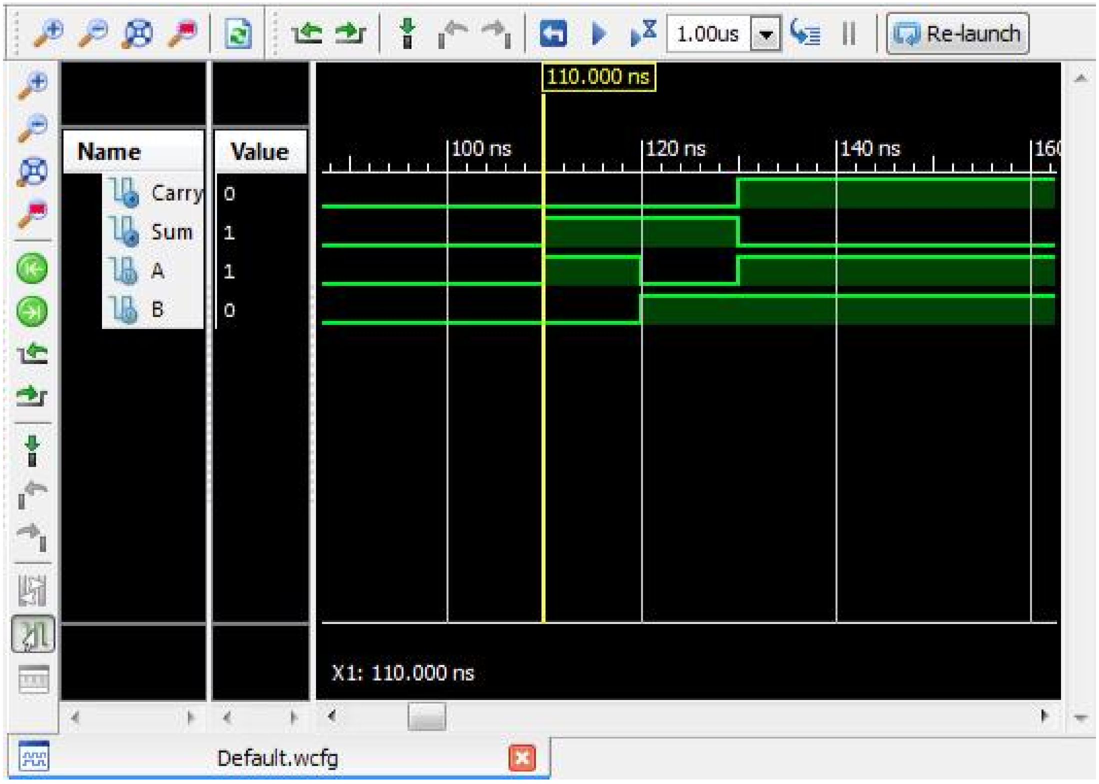


Fig.5.2 Output of example program

CHAPTER – 6

SOURSE CODE IN VERILOG

```
// Verilog Code for ATM

module ATM1 ( input clk, rst, insertcard, cardscan, acc_unlocked, deposit,withdraw,ministate,
amt_entered, valid_transaction, transaction_verified, more_transaction,
input [3:0] PIN,
input [15:0] Amount,
output reg z );

reg [1:0] ipin_count; // invalid PIN counter

parameter [4:0] scan_card = 5'b00000,
scan_line = 5'b00001,
storing_info = 5'b00010,
enter_pin = 5'b00011,
count = 5'b00100,
transaction_type = 5'b00110,
enter_deposit_amount = 5'b00111,
dep_check = 5'b01000,
invalid = 5'b01001,
anything_else = 5'b01010,
dep_verify = 5'b01011,
updating_balance = 5'b01100,
enter_wd_amount = 5'b1101,
wd_check = 5'b01110,
wd_verify = 5'b01111,
mini_state = 5'b10000;

reg [3:0] state,next;
reg [3:0] cardPIN=4'b1111;
reg [15:0] balance=16'b0001111111111000;
```

```
always @(posedge clk)
begin
    if(rst) begin
        state<=scan_card;
    end
    else
        state<=next;
end

always @(negedge clk)
begin
    case(state)
        scan_card:if(insertcard) //S0
                        next = scan_line;
                    else begin
                        next = scan_card;
                        ipin_count = 2'b0;
                    end
        scan_line:if(cardscan) //S1
                        next = storing_info;
                     else begin
                            next = scan_card;
                            ipin_count = 2'b0;
                        end
        storing_info:if(acc_unlocked) //S2
                            next = enter_pin;
                         else begin
                                next = scan_card;
                                ipin_count = 2'b0;
                         end
        enter_pin:if(PIN==cardPIN) //S3
                        next = transaction_type;
                    else begin
                            ipin_count = ipin_count+1;
                            next = count; end
        count:
```

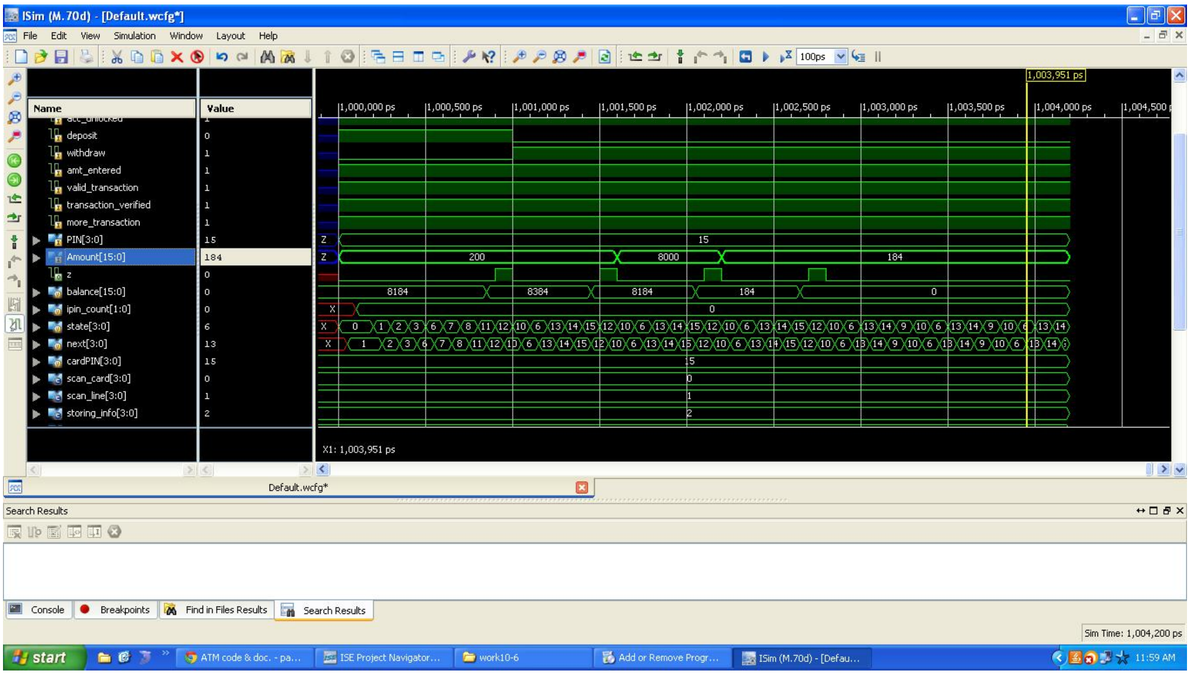
```
                                             begin
                                             if (PIN==cardPIN) //S4
                                     next = transaction_type;
                         else if(ipin_count != 3 ) begin
             ipin_count = ipin_count+1;
              next = count;
                                                 end
      else begin
          next = scan_card;
          ipin_count = 2'b0;
         end
      end
transaction_type:if(deposit) //S5
                  next = enter_deposit_amount;
                else if (withdraw)
                       next = enter_wd_amount;
                  else if (ministate)
                         next = scan_card;
enter_deposit_amount:if(amt_entered)//S6
                      next = dep_check;
                    else
                      next = transaction_type;
dep_check:if(Amount<=(~balance)) //S7
           next = dep_verify;
         else
           next = invalid;
invalid : next = anything_else; //S8
anything_else:if(more_transaction) //S9
               next = transaction_type;
             else
              begin
              next = scan_card;
              ipin_count = 2'b0;
              end
 dep_verify:if(transaction_verified)
                                  begin //S10
                  balance=balance+Amount;
                  next = updating_balance;
                                                          end
                else
                  next = dep_verify;
        updating_balance: next = anything_else; //S11
        enter_wd_amount:if(amt_entered ) //S12
                    next = wd_check;
                   else
                    next = transaction_type;
        wd_check:if(Amount<=balance && balance>=1'd0) //S13
                 next = wd_verify;
               else
                next = invalid;
        wd_verify:if(transaction_verified)
                              begin //S14
                  balance=balance-Amount;
                  next = updating_balance;
                                                         end
                else
                  next = wd_verify;
        mini_state:if(valid_transaction)
                               begin
                  next =  anything_else;
                  end
                else
                 next = scan_card;

  default : begin
          next = scan_card;
          ipin_count = 2'b0;
          end
 endcase
 end
 always @ (state)
  begin
  case(state)
      updating_balance : z=1;
      default : z=0;
  endcase
  end



endmodule
```

Output:



Name	Value
acc_unlocked	1
deposit	0
withdraw	1
amt_entered	1
valid_transaction	1
transaction_verified	1
more_transaction	1
PIN[3:0]	15
Amount[15:0]	184
z	0
balance[15:0]	0
ipin_count[1:0]	0
state[3:0]	6
next[3:0]	13
cardPIN[3:0]	15
scan_card[3:0]	0
scan_line[3:0]	1
storing_info[3:0]	2

X1: 1,003,951 ps

Sim Time: 1,004,200 ps

Fig.6.1 Output

CHAPTER - 7

RESULTS AND DISCUSSION

7.1 Introduction

The ATM controller and its functionality were discussed in the previous chapters. Now this chapter deals with the simulation and synthesis results of the ATM controller design. Here Modelsim tool is used in order to simulate the design and checks the functionality of the design. Once the functional verification is done, the design will be taken to the Xilinx tool for Synthesis process and the netlist generation.

The Appropriate test cases have been identified in order to test this modelled ATM controller design. Based on the identified values, the simulation results which describes the operation of the ATM controller has been achieved. This proves that the modelled design works properly as per the process.

7.2 Simulation Results

The test bench is developed in order to test the modeled design. This developed test bench will automatically force the inputs and will make the operations of ATM controller to perform.

7.2.1 *Verification Test Cases for the Design*

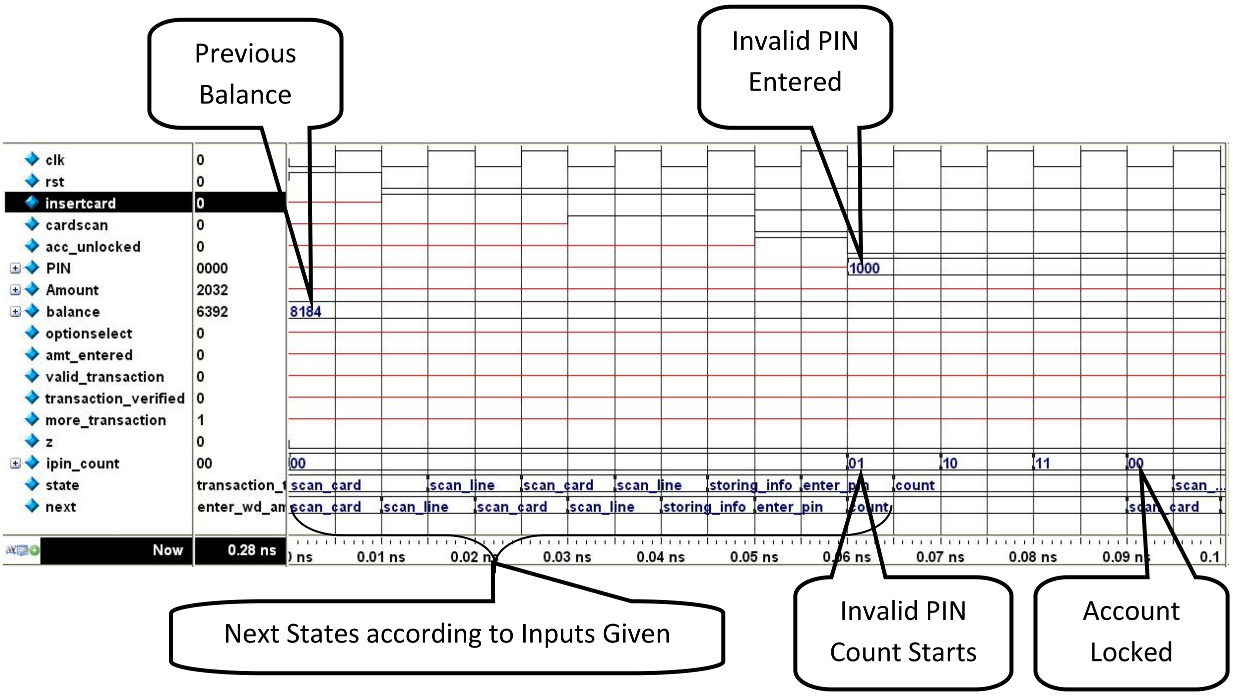
Above code description explains the implementation process and the different modes of operation. Now the implemented Verilog code is ready for the verification process that can be performed by passing the suitable test cases sequentially. So test passing of test case can be done through the test bench.

These test cases will clearly explain the operation of an ATM controller and that are can be verified. Here two initial blocks are available in which the first block is for declaring the clock and other block is for providing the test cases in order to verify the operation of ATM controller, at incremental time period.

The identified test cases are simulated through the simulation tool and the main module is synthesized using the synthesis tool. The simulation results are obtained by using the identified test cases, which shows the different modes of operation.

Case-1:

In this case, the account gets locked due to the entry of invalid pin for three times is shown in fig 3.1.



Previous Balance

Invalid PIN Entered

Next States according to Inputs Given

Invalid PIN Count Starts

Account Locked

Fig.7.1 Account Lock State

The reset signal is set to low and hence process starts from scan card state. By proper inputs given, the enter pin state can be reached. At this state, if invalid pin is entered, the counter starts to work and when count exceeds three, account gets locked which is clearly shown in Fig.7.1.

Case-2:

This case shows the overflow condition during deposit operation i.e. the summation of entered amount and previous balance exceeds the 16- bits representation.

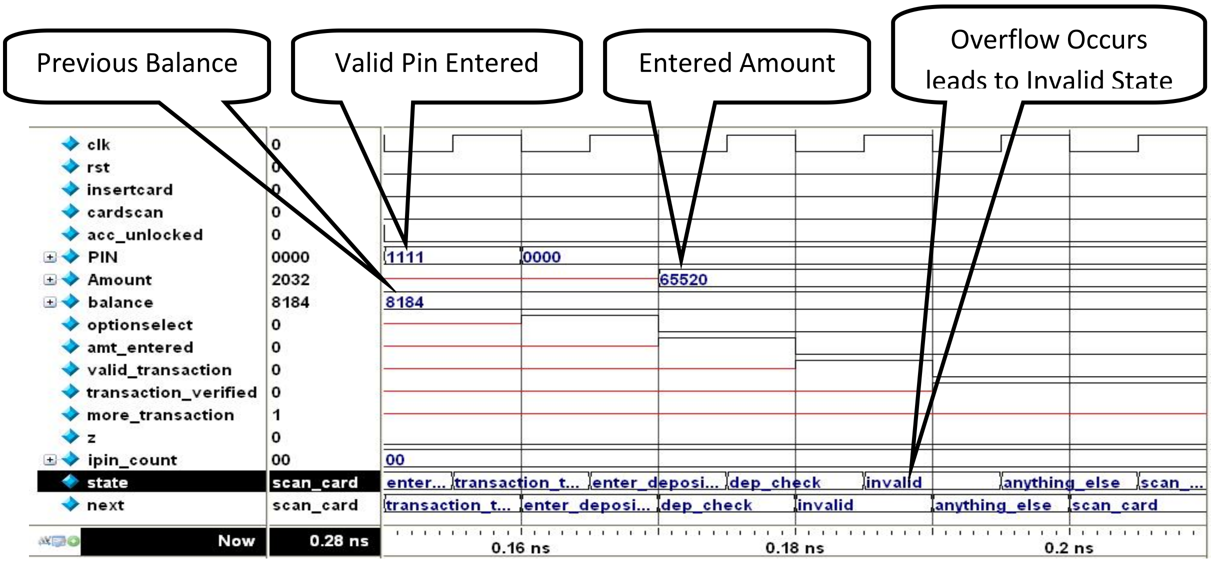


Fig.7.2 Occurrence of Deposit Overflow Condition

The valid pin (1111) is entered that leads to transaction type from which deposit is chosen. Once the amount is entered (65520) for deposit, the deposit check process id carried out. During checking process, if it exceeds 16-bits, overflow condition occurs and hence invalid state is obtained.

Case-3:

Here the amount cannot be withdrawn which is greater than the current balance available in the account shown in Fig 7.3.

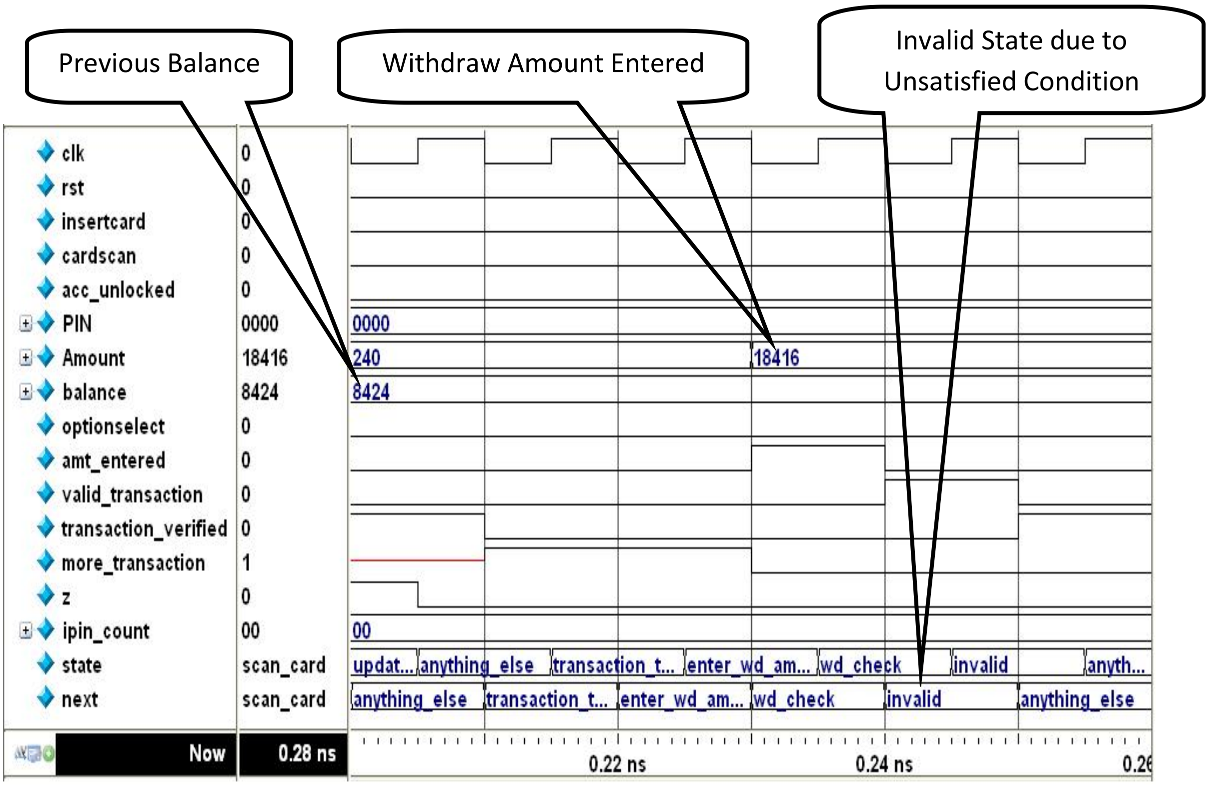


Fig.7.3 Occurrence of Invalid Condition in Withdraw

The withdraw process is chosen in which the withdraw amount (18416) should satisfy the condition mentioned. Once it is not satisfied, during the withdraw amount checking state it will get caught and hence the invalid state is obtained.

Case-4:

Once the valid pin is entered, the deposit can be chosen from the transaction type that is shown in Fig.7.4.

Here the valid pin (1111) is entered and the transaction type is obtained. From which the deposit is chosen in which the user will be allowed to enter the amount (240) that should be of 16-bits. Then the balance gets updated with the previous balance (8184) and is verified (8424). Then more transaction can be performed through anything else state.

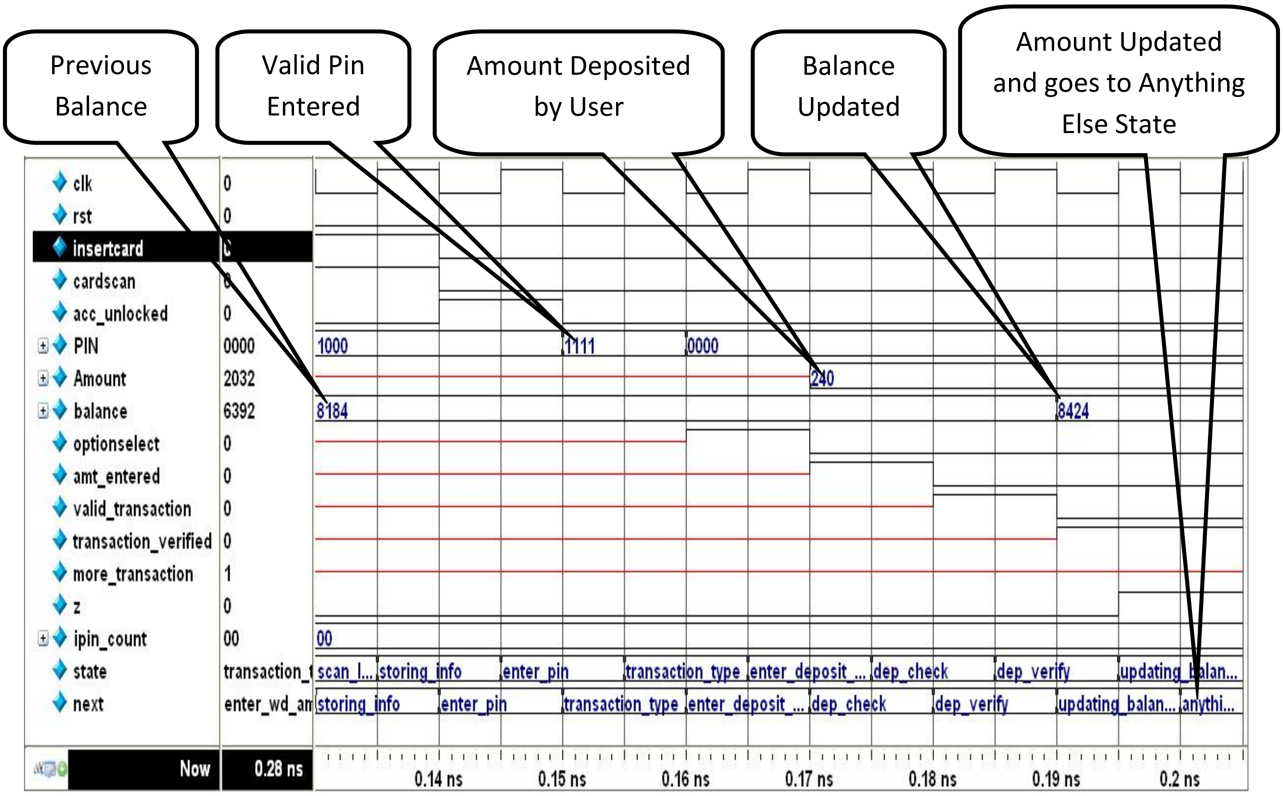


Fig.7.4. Amount Deposition State

Case-5:

In this case, the amount can be withdraw from the current balance in the account by user is shown in Fig.7.5.

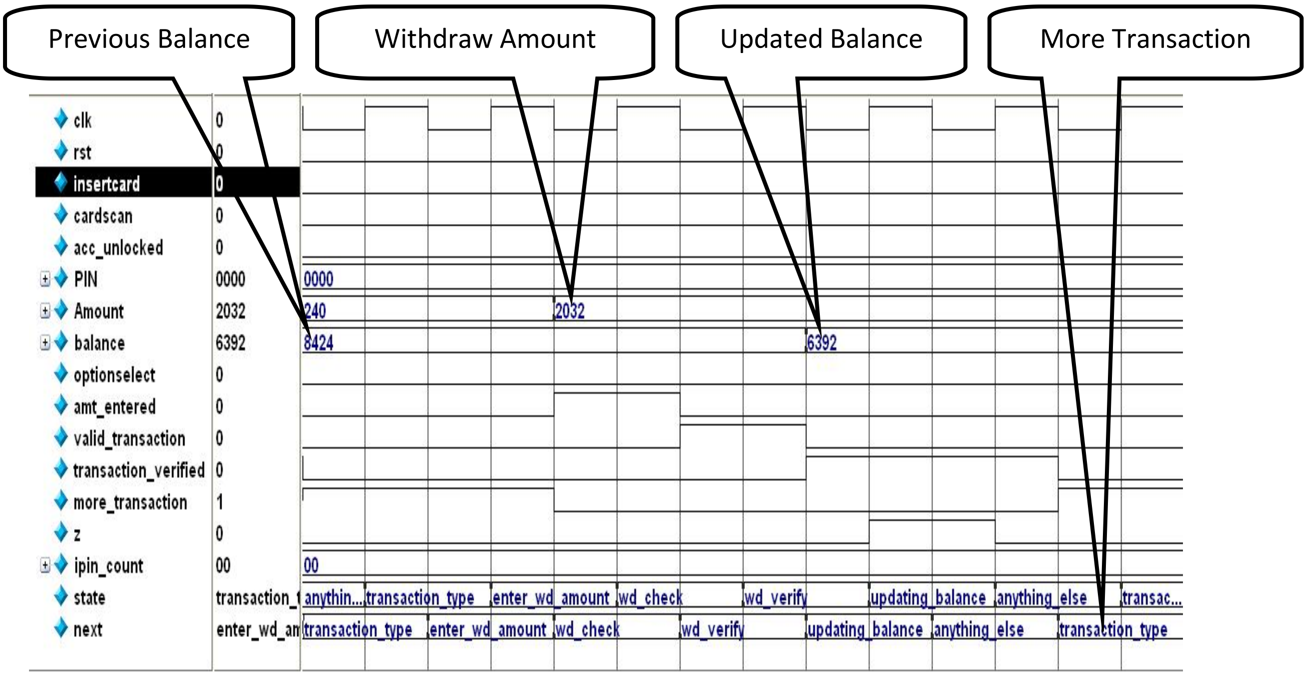


Fig.7.5 Amount Withdraw State

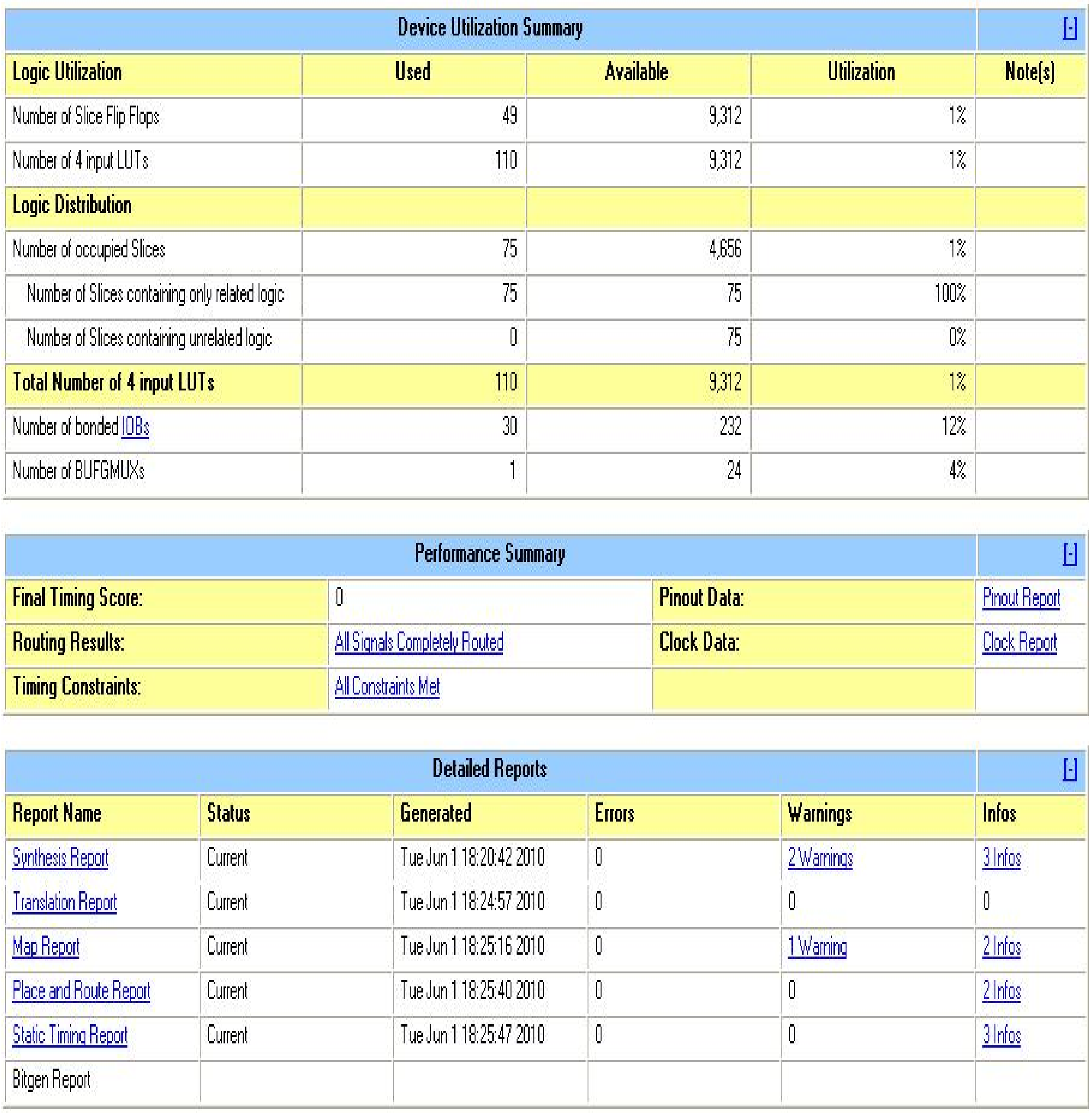
The Withdraw is chosen from the transaction type and amount is entered (2032). The entered amount is checked and is verified. Once the verification is over, the previous balance (8424) gets updated (6392). Then signal goes to anything else state that leads to more transaction.

Thus providing the proper test cases can carry the various modes of operation and those results are discussed.

7.3 Synthesis Result

The developed ATM Controller design is simulated and verified their functionality. Once the functional verification is done, the RTL model is taken to the synthesis process using the Xilinx ISE tool. In synthesis process, the RTL model will be converted to the gate level netlist mapped to a specific technology library. This ATM Controller design can be synthesized on the family of Spartan 3E.

Here in this Spartan 3E family, many different devices were available in the Xilinx ISE tool. In order to synthesis this design the device named as “XC3S500E” has been chosen and the package as "FG320" with the device speed such as "-4". The design of ATM Controller is synthesized and its results were analyzed as follows.



Device Utilization Summary [-]

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops	49	9,312	1%	
Number of 4 input LUTs	110	9,312	1%	
Logic Distribution				
Number of occupied Slices	75	4,656	1%	
Number of Slices containing only related logic	75	75	100%	
Number of Slices containing unrelated logic	0	75	0%	
Total Number of 4 input LUTs	110	9,312	1%	
Number of bonded IOBs	30	232	12%	
Number of BUFGMUXs	1	24	4%	

Performance Summary [-]

Final Timing Score:	0	**Pinout Data:**	Pinout Report
Routing Results:	All Signals Completely Routed	**Clock Data:**	Clock Report
Timing Constraints:	All Constraints Met		

Detailed Reports [-]

Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report	Current	Tue Jun 1 18:20:42 2010	0	2 Warnings	3 Infos
Translation Report	Current	Tue Jun 1 18:24:57 2010	0	0	0
Map Report	Current	Tue Jun 1 18:25:16 2010	0	1 Warning	2 Infos
Place and Route Report	Current	Tue Jun 1 18:25:40 2010	0	0	2 Infos
Static Timing Report	Current	Tue Jun 1 18:25:47 2010	0	0	3 Infos
Bitgen Report					

Fig.7.6. Device utilization summary

Device utilization summary:

This device utilization includes the following.

- Logic Utilization
- Logic Distribution
- Total Gate count for the Design

The device utilization summery is shown above in which its gives the details of number of devices used from the available devices and also represented in %. Hence as the result of the synthesis process, the device utilization in the used device and package is shown above.

Timing Summary:

- Speed Grade: -4
- Minimum period: 11.496ns (Maximum Frequency: 86.987MHz)
- Minimum input arrival time before clock: 6.892ns
- Maximum output required time after clock: 4.394ns
- Maximum combinational path delay: No path found

In timing summery, details regarding time period and frequency are shown are approximate while synthesize. After place and routing is over, we get the exact timing summery. Hence the maximum operating frequency of this synthesized design is given as 86.987 MHz and the minimum period as 11.496 ns. Here, OFFSET IN is the minimum input arrival time before clock and OFFSET OUT is maximum output required time after clock.

RTL Schematic:

The RTL (Register Transfer Logic) can be viewed as black box after synthesize of design is made. It shows the inputs and outputs of the system. By double-clicking on the diagram we can see gates, flip-flops and MUX.

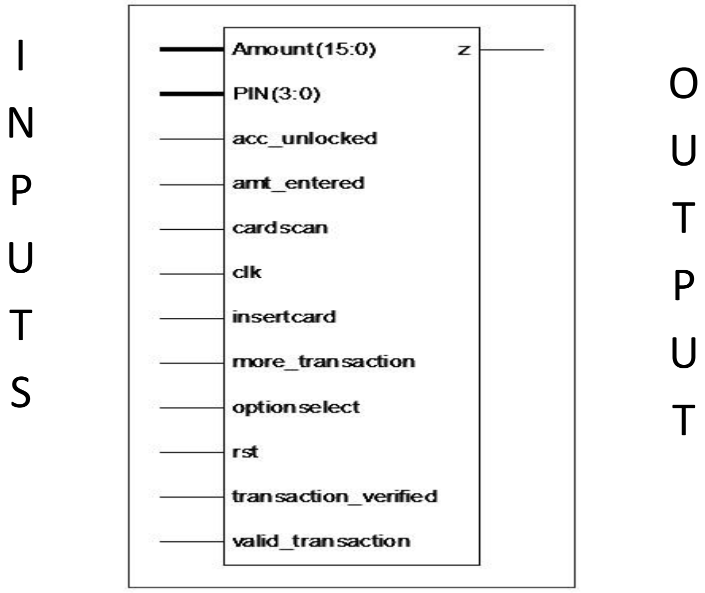


Fig.7.7 Schematic with Basic Inputs and Output

7.4 Summary

- The developed ATM Controller design is modelled and is simulated using the Modelsim tool.
- The simulation results are discussed by considering different cases.
- The RTL model is synthesized using the Xilinx tool in Spartan 3E and their synthesis results were discussed with the help of generated reports.

CHAPTER 8

CONCLUSION

8.1 Conclusion

Basically, ATM controller allows the user to interact with the memory and hence the security level is increased. This also makes the transaction in account gets easier. This work helps to understand the concept of Finite State Machine and designing architecture of a system. The familiarization of the procedure to develop the State Machine diagram of a system through system level analysis is improved. The ATM block diagram is studied and the corresponding Moore Machine State diagram is also analyzed. The FSM is modeled in Verilog HDL and verified for all the appropriate scenarios.

The simulation results have been verified for the different appropriate test cases. Hence the maximum operating frequency of this synthesized design is given as 86.987 MHz and the minimum time period as 11.496 ns.

8.2 Recommendation for Future Work

As future work,

- More accessories can be added to ATM such as coordination of ATM with mobile phone, shopping, online purchasing and some more.
- The optimization of system can also be done to have an effective transaction.

BIBLIOGRAPHY

- Information provided by MTS Technologies

WEBSITES:

- From Wikipedia
- Google search
- Howstuffworks.com
- Engineersgarage.com

TEXTBOOKS:

- Digital Logic and Computer Design by Morris Mano
- Advanced Microprocessors and Peripherals by K M Bhurchandi and A K Ray
- Microprocessors and Interfacing by Douglas V Hall