Microcontroller Based Data Access Control System Using Vhdl

P-513

PROJECT REPORT

SUBMITTED BY

SUNDARESAN. V
SUDERSON. S
MOHAN SHARMA
FELIX VETHARAJ. S

GUIDED BY

Mr. R. K. PONGIANNAN, B.E, MISTE.
LECTURER, EEE

2000 – 2001

IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE OF
Bachelor of Engineering in
Electrical and Electronics Engineering
OF THE BHARATHIYAR UNIVERSITY, COIMBATORE.

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY
Coimbatore- 641006

KUMARAGURU COLLEGE of TECHNOLOGY

Coimbatore-641 006

Department of Electrical & Electronics Engineering

Certificate

This is to certify that the Project Report entitled

"Microcontroller Based Data Access Control System Using VHDL"

has been submitted by

Mr. Y. SUNDARESAN, S. SUDERSON, MOHAN SHARMA, S. FELIX VETHARAJ

in partial fulfillment for the award of the degree Bachelor of Engineering in Electrical and Electronics Engineering of Bharathiar University, Coimbatore – 641 046 during the academic year 2000-2001.

Guide

Date : 14-3-2001.

Dr. K. A. PALANISWAMY, B.E., M.Sc.(Engg.), Ph.D.,
M.I.S.T.E., C.Engg.(I), F.I.E.
Professor and Head
Department of Electrical and Electronics Engineering
Kumaraguru College of Technology
Coimbatore 641 006

Head of the Department

Date : 14.3.01

Certified that the candidate with the university Registration No._____ was examined in the project viva-voce on _____.

Internal Examiner

External Examiner

A.G. AUTOMATION

4 (New No.7), Padmanabha Nagar, 2nd Street, Adyar, Chennai - 600 020.
Tel. : 4466265 / 4423963 Fax : 4338117 E-mail : agautomat@eth.net

- Merlin Gerin
- Modicon
- Square D
- Telemecanique

TO WHOMSOEVER IT MAY CONCERN

This is certify that the following students of the Electrical and Electronic Engineering branch of Kumara Guru College of Technology, Coimbatore, have completed the project work titled "Micro Controller based data access control system", sponsored by us. During the course of their project work, we found them to be very much dedicated, hard working and technically sound.

1. S. Felix Vetharaj (97EEE16)
2. Mohan Sharma (97EEE31)
3. S. Suderson (97EEE53)
4. V. Sundaresan (97EEE54)

We wish them all success.

With Regards,

G. SUNDARESAN

Chief Executive
AG AUTOMATION

ACKNOWLEDGEMENT

We express our profound gratitude and sincere thanks to our guide Mr. R. K. Pongiannan, B.E., Lecturer, Department Of Electrical and Electronics Engineering, Kumaraguru College of Technology, for his able guidance, constant encouragement and kind co-operation. Without his valuable suggestions, constructive criticism and meticulous guidance at every stage, this work would not have attained fruition.

We are greatly indebted to our Professor and Head of the Dept. of Electrical and Electronics Engineering Dr. K. A. Palaniswamy, M.Sc(Engg), Ph.D., FIE, C.Eng.(I),MISTE for his encouragement and support.

We thank our beloved Principal Dr. K.K. Padmanabhan, B.Sc.(Engg), M.Tech., Ph.D., for giving the necessary facilities to carry out the project.

We are extremely grateful to Mr. G. Sundaresan, Managing Director, A.G. Automation Pvt. Ltd., Chennai for having sponsored the project.

We are thankful to other faculty members, non-teaching staff and student friends for their dynamic support.

SYNOPSIS

As the saying goes "Better safe than sorry", this project **"Microcontroller Based Data Access Control System using VHDL"** is aimed at providing high level data security in various applications like Credit Card, ATM card and in places where data security is of utmost importance. Conventional trends such as barcodes, magnetic cards, sim cards etc, are found less secure by the way that the modification or simulation of an entry into the master software is simple with the biasing of the programmers. Hence we go in for designing the smart card using VHDL wherein the modification in the card is not at all possible once if the design is converted to IC package. Modification of data designing inside the card needs higher end design engineer interruption and research. The main advantage of using VHDL card is that any design changes in future can be easily and inexpensively implemented.

The smart card that we have designed is a Programmable Logic Device GAL 22v10B, programmed using VHDL. The main features of VHDL include less production cost and ease of changing the design.

Our project incorporates the AT89c51 microcontroller that compares the user-entered data and the data that is read from the smart card. A relay is activated by the microcontroller to drive the load when the outcome of the comparison is positive. If the result of the comparison is negative then an alarm is indicated by means of a buzzer.

CONTENTS

CHAPTER 1

INTRODUCTION

Security of data is of primary importance to major industrial and government organizations. Any discrepancy in data will result in a great loss. Such a discrepancy would bring into prominence the dearth in securing data vital to the firm.

We have, therefore, taken a bold step in this regard to solve the numerous problems and inadequacies involved in security of data.

In practice, a Smart Card is given to each customer. The customer has to carry the same every time to make a transaction. He then inserts the card in the slot provided. The microcontroller then reads the data from the card and then places it in the external RAM. The user then has to enter the details via the keypad, which are placed in the external RAM also. The data are then compared by the microcontroller. Based on the result of the comparison, either the person is allowed to proceed further with the intended purpose or a buzzer is activated to indicate the entry of unauthorized data.

The system designed is apt for commercial applications like limiting access to a high security area only to legitimate personnel, and in the case of providing a more secure credit card thereby preventing the interruption of hackers. The advantages of the system outweigh those of the competitors.

Though still a prototype, its functions are comparable and possibly outclass other commercial products of the same type.

1.1 OVERVIEW OF GENERAL BLOCK DIAGRAM

The Fig1.1 shows the block diagram of the system. It can be categorized into three main parts. They are:

1. Smart Card
2. AT89c51 microcontroller
3. I/O Units

1.1.1 Smart Card

The Smart Card holds the CMOS PLD GAL22v10B, which is programmed in VHDL. This card holds the name, password and identification code for an employee. The microcontroller sends a clock pulse to the smart card and the data is transferred from the card to the RAM that is the microcontroller places the data on to the data bus and then it is stored in the RAM.

1.1.2 Microcontroller

The program written in assembly will compare the entered data and the smart card data. If both the data are equal then the MicroController sends a control pulse to the relay, which drives the required load. On the other hand, if the result of comparison is false the MicroController will send a control pulse to the buzzer indicating an alarm.

1.1.3 I/O Units

The input unit is the keypad, which consist of eight keys, six of which are alphanumeric keys, one reset key and the other reserved for future use. The six alphanumeric keys perform various operations such as increment, decrement, left shift, right shift, enter and next. A LCD acts as an output unit which displays the characters typed by the keypad.

1.1.4 Chip Reduction

Another feature that VHDL finds its application is the reduction of chip set used in this project. For memory and 8255A selection we need a 2x4 decoder. Further various logic gates such as AND and NOT gates that are required for the generation of control signals are combined on to a single chip using VHDL. Hence this feature combines the function of three ICs that is 7404,7408,74LS274. Hence this feature reduces the overall size of the system.

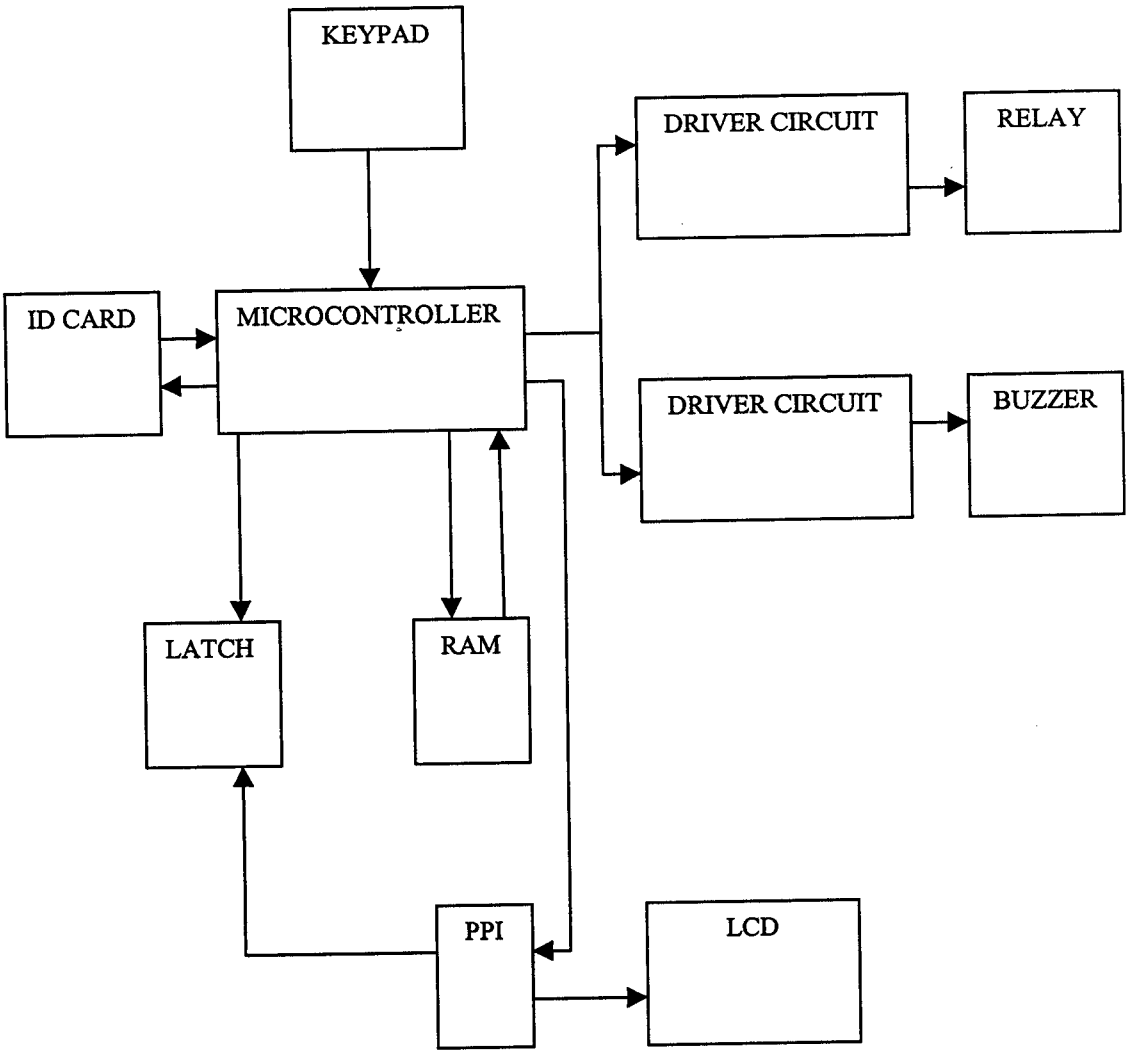


FIG 1.1 OVERALL BLOCK DIAGRAM

CHAPTER 2

FUNCTIONAL DESCRIPTION

This project features an efficient design of power supply, controller circuitry using the AT89c51 microcontroller and driver circuits for relay and buzzer.

2.1 POWER SUPPLY

The Fig. 2.1 shows the rectified power supply unit. This unit rectifies the input AC and also regulates it. The power supply circuit consists of the following units.

2.1.1 Transformer

In our project the step-down transformer has two secondaries. Across one secondary we get 15-0-15 volts and in the other we get 0-9 volts. Its current rating is 1A. The transformer is of the core type. The output of the secondaries is given to a rectifier.

2.1.2 Rectifier

These are two full wave rectifiers used for +12V and +5V respectively. The 12v supply is used to energize the relay and 5v is to drive the controller circuitry.

2.1.3 Full wave bridge rectifier

In a full wave bridge rectifier, four diodes are used. During the positive half cycle, diodes D1 and D3 conduct and during the negative half cycle, diodes D2 and D4 conduct. So we get the rectified DC. Voltage drop across the diode is 0.6V. The rectified DC voltage consists of AC ripple components that should be filtered.

2.1.4 Filter circuit

The output DC voltage from the rectifier may contain AC ripple components. We can filter or smooth out AC variation from the rectifier voltage shunt capacitor is normally used as a filter. A large

capacitor shorted with the load resistor causes only a small part of AC to pass through the load, producing a small ripple voltage.

2.1.5 Voltage regulator

The function of the voltage regulator is to provide a stable dc voltage for powering other electronic circuits, independent of the load current, temperature and ac line voltage variations. IC voltage regulators are used for their low cost, high reliability, reduction in size and excellent performance. Examples of monolithic regulators are 78xx series and 723 general-purpose regulators, 78xx are three terminal, positive, fixed voltage regulators. The last two numbers indicate the output voltage. The ICs used in the power supply circuit are 7812,7805.

2.2 SELECTION OF TRANSFORMER, DIODE AND CAPACITOR

2.2.1 Transformer selection

The transformer steps down the high voltage AC signal to a suitable value for rectification. The most important factor to be considered while selecting the step down transformer is the output load current.

Transformer secondary current

I_s=1.8 * output dc current, maximum output dc current = 750 ma.

So, transformer secondary current.

I_s =1.8 * 750 mA

Therefore, I_s = 1.5 A.

The other factors to be considered for transformer selection can be given by:

1. Load regulation
2. Temperature rise
3. Shielding.

So, We are choosing a step down transformer with a secondary current rating of 1.5A.

2.2.2 Diode selection

The power supply uses a full wave bridge rectifier circuit to rectify the Ac-input signal. The diode selection is based on diode rating.

Diode current, I_d = 2 to 3 times the load current

I_d =3 * 750 mA

I_d =2.5 A.

so ,we have chosen a diode of current rating of 2.5 A for the Full Wave Bridge Rectifier.

2.2.3 Capacitor selection

The input capacitor is used to maintain a stabilized dc supply to the input terminal of the regulator.

It can be selected by using the formula

C = IL * 6 * 10(-3) / (V)**

Where

IL = maximum load current (750 mA)

V = input voltage to the capacitor (5V)

Hence,

C = 750 mA * 6 * 10 **(-3)/ 5.

C = 1000 microfarads.

The output filter capacitor value is chosen as 10 microfarads.

2.3 RELAY,BUZZER DRIVER CIRCUIT (12V/2.5A)

The circuit shown in Fig.2.2 features a transistor switch that turns on when there is a signal of over 2.5V. To turn the transistor off the positive signal should be brought to zero. The diode provides discharge path for the back emf generated by the collapsing magnetic field of the relay coil when the power is switched off.

Initially when there is no 5v pulse from the P1.7 of the MicroController the transistor is in off position and the relay is not energized. When the MicroController turns on the relay, the transistor BC547 gets turned on and provides a closed path for the relay to operate.

The assembly is very easy and the only thing to make sure is that the direction of the diode must be correct. The bar on the diode should match the bar on the overlay.

2.4 CONTROLLER CIRCUIT

The diagram shown in Fig.2.3 the next page is the main functional circuit of this project. Before going into a detailed analysis about the functional flow, let us study the various interfaces done in various ports of the AT89c51 MicroController.

2.4.1 Port 0

Port 0 acts as an I/O port to output the lower order address/data lines. As an output port, each pin can sink eight TTL inputs. When 1's are written to port 0 pins, the pins can be used as high impedance inputs. Port 0 here acts as a multiplexed lower order address/data bus during accesses to external program and data memory.

2.4.2 Port 1

Port 1 is an 8-bit bi-directional I/O port. The port 1 output buffers can sink /source four TTL inputs. Here, we use port 1 to interface the keypad circuitry. Six alphanumeric keys are interfaced from P1.0 to P1.5. The remaining port 1 pins that is, P1.6 and P1.7 are used to interface the relay and the buzzer respectively.

2.4.3 Port 2

Port 2 acts as an eight-bit bi-directional I/O port with internal pull-ups. The port 2 output buffers can sink/source four TTL inputs. Here we use Port 2 to emit the high-order address byte during fetches from external program memory and during accesses to external data memory that use 16 bit addresses.

2.4.4 Port 3

Port 3 are generally confined to timer/counter operations but they can be used as an 8-bit bi-directional port with internal pull-ups. Here in this project we use Port 3 as a bi-directional I/O port. We interface the smart card consisting of four data lines, a clock and a reset to the P3.0 to P3.5 pins.

2.4.5 Clock Circuit

The clock used in this project is 12 MHz comprising of six machine cycles. This clock is provided by a crystal connected to XTAL1 and XTAL2 respectively.

2.4.6 8255A Programmable Peripheral Interface

A LCD is interfaced to the port C of the general programmable peripheral interface. Here in this project port A and C acts as outputs and port B as inputs.

2.4.7 Control Signals

RST -Reset Input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/PROG'-Address Latch Enable output pulse for latching the low byte of the address during access to external memory. This pin is also the program pulse input (PROG') during flash programming.

PSEN' –Program Store Enable is the read strobe to external program memory. When the At89c51 is executing code from external program memory, PSEN' is activated twice each machine cycle, except that two PSEN' activation's are skipped during each access to external data memory.

XTAL1 –Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

XTAL2 -Output from the inverting oscillator amplifier.

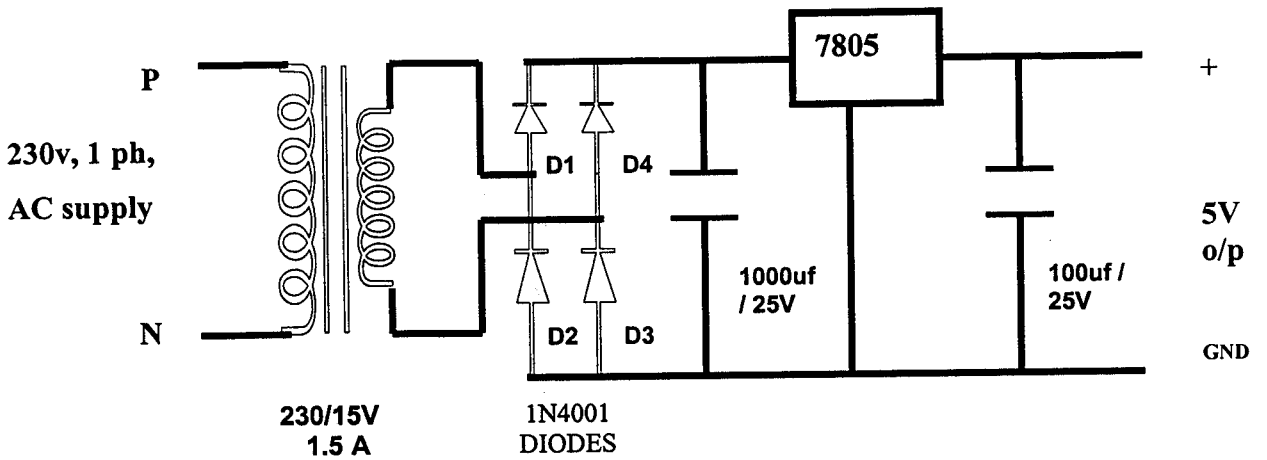
RD -This control signal reads data from the memory and this signal is generated by the logical AND operation on the operands PSEN and PRD.

WR -This control signal writes data into the memory and this signal is generated by the logical AND operation on the signals VCC and PWR.

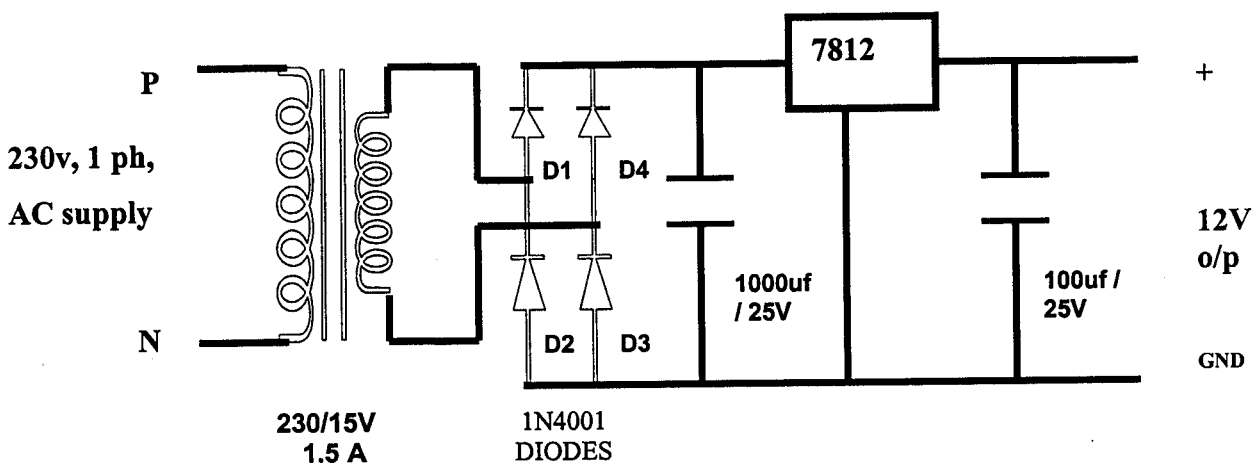
RAMS -This control signal to the memory is generated by the logical AND operation on the signals RAM and ERAM.

2.5 WORKING OPERATION

A brief overview of the overall functional flow of Fig.2.3 may be explained as follows. The relay connected at P1.7 is switched off on reset and during the initial startup. 8255A is selected by the 2x4 decoder that sends CS(Chip Select) signal to the 8255A Programmable peripheral interface. The control words required for the operation of the LCD such AS character display, display of cursor and right/left shift etc are sent from the RAM contents onto the controller RAM in the display. The character typed in from the keypad is stored sequentially in the internal RAM location. The microcontroller then sends a clock pulse to the smart card and the data read from the smart card is stored in the external RAM content. The data that are stored in the two locations are compared one by one. If the data in the first location itself proves to be incorrect then an alarm is indicated otherwise the RAM location is incremented and the next data is compared. If all the data in both the locations are correct then a relay is activated to drive the load. A reset key given to the pin 9 of the microcontroller resets all the register and RAM contents to zero.



RECTIFIED 5V SUPPLY



RECTIFIED 12V SUPPLY

FIG 2.1 POWER SUPPLY UNIT

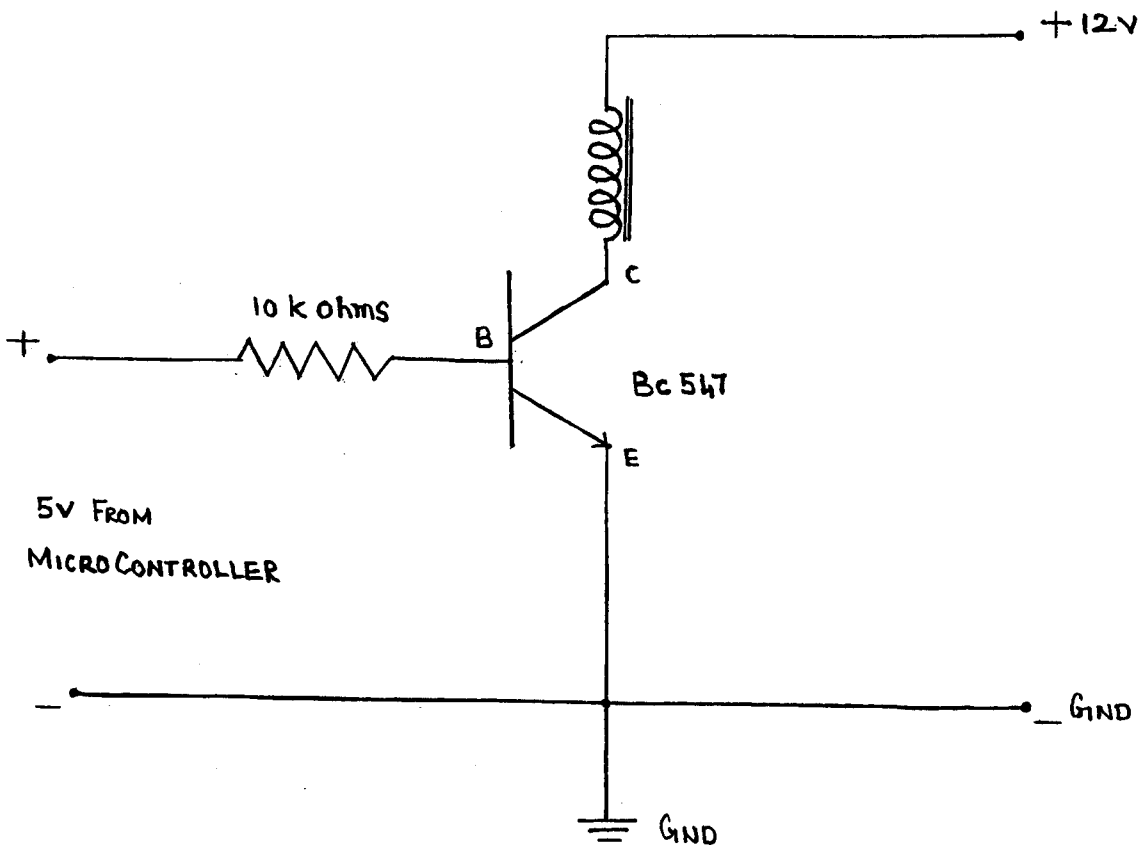


FIG 2.2 RELAY AND BUZZER DRIVER UNIT

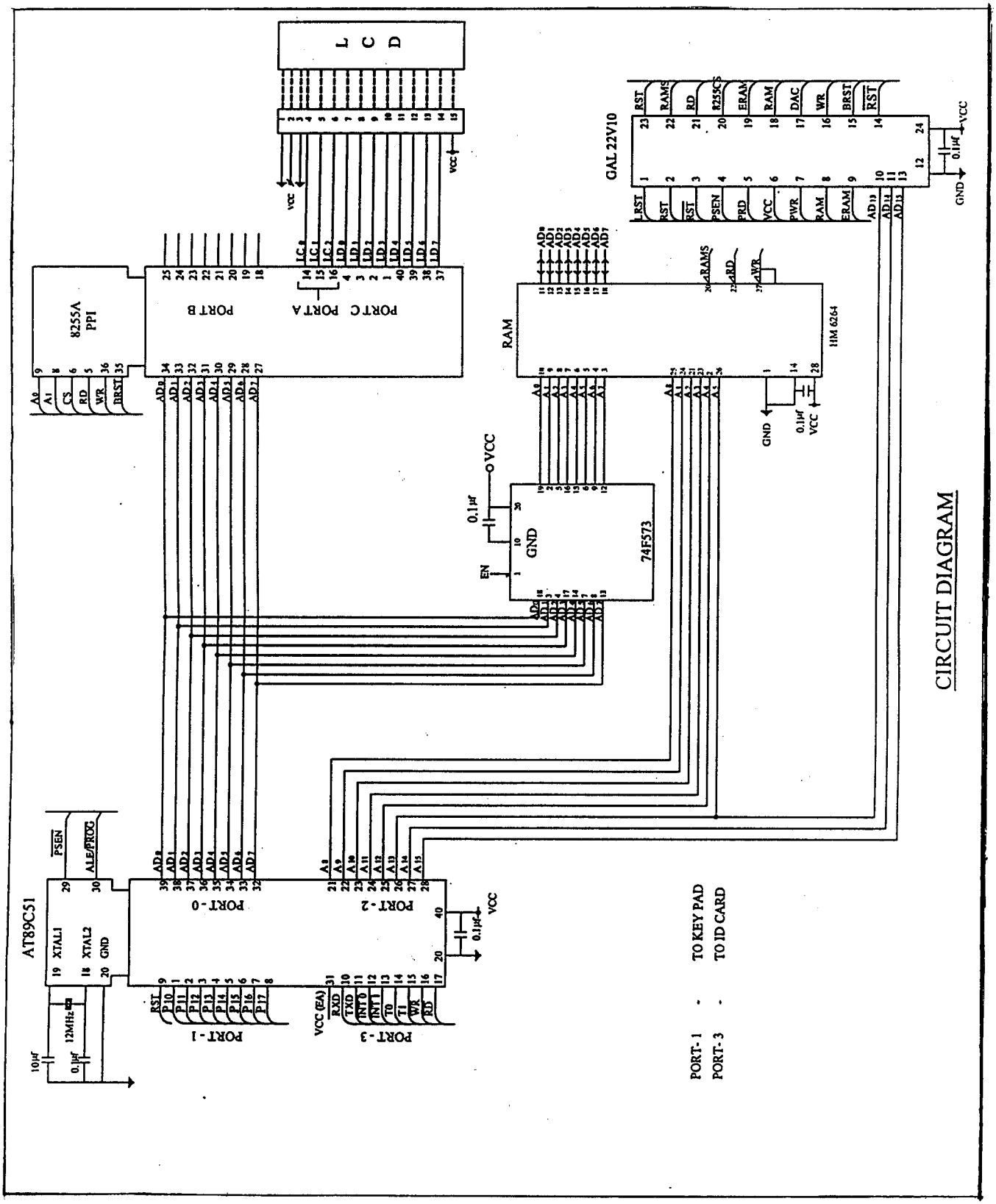


FIG 9.7 MAIN CIRCUIT DIAGRAM

CHAPTER 3

ARCHITECTURAL OVERVIEW

3.1 PROGRAMMABLE LOGIC DEVICES

Programmable Logic Devices are prefabricated IC's in which flexible interconnection layers are included. The interconnection layers are personalized by electronic means for a specific application by the end user. A PLD contains the equivalent of several thousand logic gates on a single IC package. A PLD circuit is developed by designing logic expressions, translating them into the format of the target PLD, and then installing them into the PLD using a PLD programmer. Thus a working device can be produced from a design in a few minutes and also, if needed, design changes can be quickly and inexpensively implemented within hours or minutes.

3.2 INTRODUCTION TO GAL 22V10B

GAL AS the name implies holds the acronym for " **GENERIC ARRAY LOGIC** ". GAL belong to the family of Low density PLDs using the latest E^2 CMOS technology.

The main features of this PLD include

- 10 OLMCs with variable Product Terms per OLMC
- Standard 24-pin DIP and 28-pin PLCC packages
- Industry Leading Performance
- Low power consumption
- Full reprogrammability

GAL 22v10B, at 4 ns max propagation delay time combines a high performance CMOS process with Electrically Erasable (E^2) floating gate technology to provide highest performance. CMOS circuitry allows the PLD to consume less power when compared to Bipolar 22v10 devices. E^2 Technology offers high speed (< 100ms) erase times providing the ability to reprogram the device quickly and offers data retention in excess of 20

years. The Fig 3.1 shows the functional block diagram of the GAL 22v10 PLD and its interconnection with the output macrocells.

3.2.1 GAL 22V10B – Functional overview

The GAL 22v10B contains a (44 x 132) programmable AND array that drives 10 output macro cells. The complete functional block diagram of GAL 22v10 shows that all the ten flip-flops are controlled by common clock, preset and reset signals. The clock signal comes from a dedicated input pin and the preset and the reset signals are supplied from product terms generated in the AND array.

3.2.2 Output logic macro cell (OLMC)

The GAL 22v10 has a variable number of product terms per OLMC, two OLMCs has access to eight product terms (Pins 14 and 23, DIP pinout, two have ten product terms (Pins 15 and 22), two have Twelve product terms (Pins 16 and 21), two have fourteen product terms (Pins 17 and 20) and two OLMCs have sixteen product terms (Pins 18 and 19). In addition to the product terms available for logic, each OLMC has an additional product term dedicated to output enable control. The output polarity of each OLMC can be individually programmed to be true or inverting, in either combinatorial mode or registered mode. This allows each output to be individually configured AS either active high or active low.

The Fig 3.2 shows the schematic diagram of the Output Logic Macro Cell (OLMC). The output of the macro cell is selected by a 4 – to – 1 Multiplexer, output options include the GAL combinational output and its complement and the Q and Q' flip-flop outputs. Each of the macro cells of the GAL 22v10B has two primary functional modes.

- Registered I/O
- Combinatorial I/O

The modes and the output polarity are set by two bits (S0 and S1), that are normally controlled by the Logic compiler. The various combinations

in which each of the four output options and the two feedback options are programmed are shown below.

S1	S0	OUTPUT	FEEDBACK
0	0	Combinatorial(High)	I/O pin
0	1	Combinatorial (Low)	I/O Pin
1	0	Registered (Q)	Flip-flop Output Q'
1	1	Registered (Q')	Flip-flop Output Q'

3.2.3 Registered I/O

The Fig.3.3 shows the GAL 22V10B in registered mode. The output pin associated with an individual OLMC is driven by the Q output of that OLMCs D – Type Flip-flop. Logic polarity of the output signal at the pin may be selected by specifying that the output buffer drive either true (active high) or inverted (active low). Output tri-state control is available as an individual product-term for each OLMC. The D – Flip-flop's / Q output is fed back into the AND array, with both the true and complement of the feedback available AS inputs to the AND array.

The GAL 22v10B has a product term for Asynchronous Reset (AR) and a product term for synchronous preset (SP). These two product terms are common to all registered OLMCs. The Asynchronous reset sets all registers to zero any time this product term is asserted. The synchronous preset sets all resisters to logic one on the rising edge of the next clock pulse after this product term is asserted. The AR and SP product terms will force the Q output of the flip-flop into the same state regardless of the polarity of the output. Therefore, a reset operation, which sets the register output to a zero, may result in either a high or low at the output pin, depending on the polarity of the pin Chosen.

3.2.4 Combinatorial I/O

The Fig.3.3 shows the GAL 22V10B in combinatorial mode. The pin associated with an individual OLMC is driven by the output of the sum term gate. Logic polarity of the output signal at the pin may be selected by specifying that the output buffer drive either true (active high) or inverted (active low). Output tri-state control is available as an individual product-term for each output and may be individually set by the compiler as either "ON" (dedicated Output), " OFF" (dedicated Input) or " product-term driven" (dynamic I/O). Feedback into the AND array is from the pin side of the output enable buffer. Both polarities (true and inverted) of the pin are fed back into the AND array.

3.2.5 Electronic signature

An Electronic signature (ES) is provided in every GAL 22v10B device. It contains 64 bits of reprogrammable memory that can contain user defined data. Some uses include user ID codes, revision number or inventory control. The signature data is available to the user independent of the state of the security cell.

The **JEDEC** map (Shown in Appendix) for the GAL 22v10B contains the 64 extra fuses for the electronic signature, for a total of 5892 fuses.

3.2.6 Security cell

A Security cell is present in every GAL 22v10B device to prevent the unauthorized copying of the array patterns. Once programmed this cell prevents further read access to the functional bits in the device. This cell can only be erased by re-programming the device, so the original configuration can never be examined once this cell is programmed.

3.3 ARCHITECTURE OF MICROCONTROLLER AT 89C51

A microcontroller consists of a powerful CPU tightly coupled with memory (RAM , ROM or EPROM), various I / O features like Serial ports , Parallel ports, Timer/Counters, Interrupt Controller, Data Acquisition interfaces-Analog to Digital Converter(ADC),Digital to Analog Converter (ADC),everything integrated onto a single Silicon Chip.

We generally prefer the use of a MicroController rather then the microprocessor based on the following advantages that a MicroController overweighs a microprocessor.

- ♦ Size of the board is decreased since the memory required such as ROM, RAM, EPROM etc., are embedded on a single chip
- ♦ The data transfer is in bits and not bytes as in real world applications. Hence they can be used for various switching operations

The AT89c51 microcontroller provides the following standard features.

- 8 Bit CPU optimized for control applications
- Extensive Boolean processing (Single - bit Logic) Capabilities.
- On - Chip Flash Program Memory
- On - Chip Data RAM
- Bi-directional and Individually Addressable I/O Lines
- Multiple 16-Bit Timer/Counters
- Full Duplex UART
- Multiple Source / Vector / Priority Interrupt Structure
- On - Chip Oscillator and Clock circuitry.
- On - Chip EEPROM
- SPI Serial Bus Interface
- Watch Dog Timer

The Fig 3.4 shows the architectural block diagram of AT89c51 microcontroller. The functions of the various SFRs are outlined below.

3.3.1 Accumulator

ACC is the accumulator register. The mnemonics for accumulator – specific instructions refer the register simply a A.

3.3.2 B Register

The B Register is used during multiply and divide operations. For other instructions it can be treated as another scratch pad register.

3.3.3 Program status word

The PSW register contains program status information and holds the information about various flags that are set as a result of certain instructions.

3.3.4 Stack pointer

The 8 – bit wide stack pointer is incremented before data is stored during PUSH and CALL executions. The stack may reside anywhere in the internal RAM area but the stack pointer (SP) is initialized to 07H after reset and the stack begins at location 08H.

3.3.5 Data pointer

The Data pointer consists of a high byte (DPH) and a low byte (DPL). Its intended function is to hold a 16 bit address. It may be manipulated as a 16 bit register or as two independent 8 – bit registers.

3.3.6 Serial data buffer

The serial data buffer is actually two separate registers, a transmit buffer and a receive buffer register. When data is moved to SBUF, it goes to the transmit buffer where it is held for serial transmission. When data is moved from SBUF, it comes from the receive buffer.

3.3.7 Power modes of Atmel 89C51 microcontroller

To exploit the power savings available in CMOS circuitry, Atmel 's Flash MicroController have two software-invited reduced power modes.

Idle mode: The CPU is turned off while the RAM and other on - chip peripherals continue operating . In this mode current draw is reduced to about 15 percent of the current drawn when the device is fully active.

Power down mode: All on-chip activities are suspended while the on - chip RAM continues to hold its data. In this mode, the device typically draws less than 15 MicroAmps and can be as low as 0.6 MicroAmps

3.3.8 Power on reset

When power is turned on, the circuit holds the RST pin high for an amount of time that depends on the capacitor value and the rate at which it charges.

To ensure a valid reset, the RST pin must be held high long enough to allow the oscillator to start up plus two machine cycles.

On-power-up , V_{cc} should rise within approximately 10ms. The oscillator start-up time depends on the oscillator frequency. For a 10 MHz crystal, the start-up time is typically 1ms.With the given circuit, reducing V_{cc} quickly to 0 causes the RST pin voltage to momentarily fall below 0V.However , this voltage is internally limited and will not harm the device.

3.3.9 Memory organization

All Atmel Flash MicroController have separate address spaces for Program and data memory. The logical separation of program and data memory allows the data memory to be accessed by 8 bit addresses which can be more quickly stored and manipulated by an 8 bit CPU. Nevertheless, 16 Bit data memory addresses can also be generated through the DPTR register.

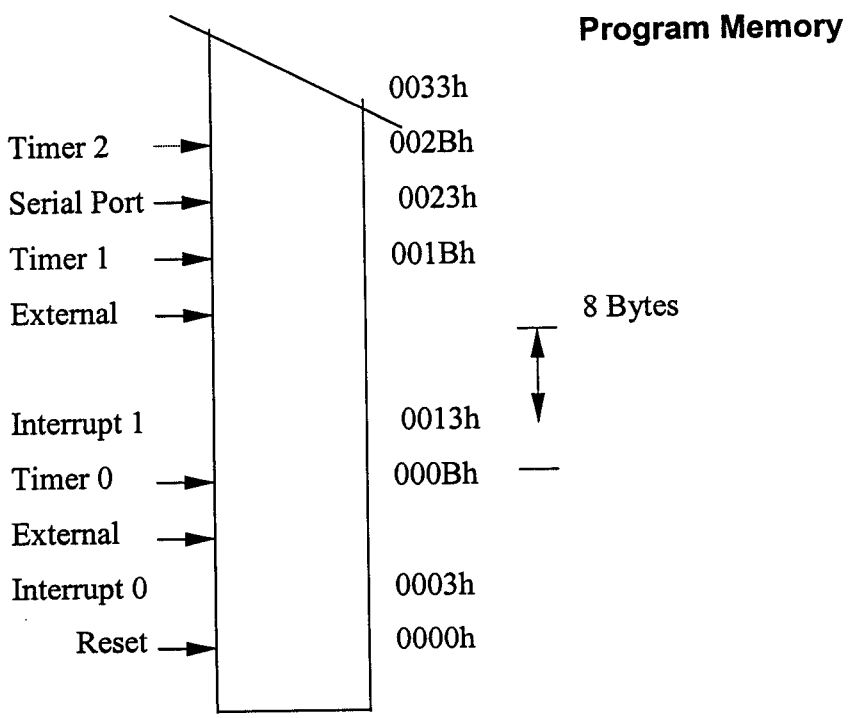
Program memory can only be read. There can be up to 64K bytes of directly addressable program memory. The read strobe for external program memory is the Program Store Enable Signal (PSEN)'.

Data memory occupies a separate address space from program memory. Up to 64K bytes of external memory can be directly addressed in the external data memory space. The CPU generates read and write signals , RD' and WR', during external data memory accesses.

External program memory and external data memory can be combined by applying the RD' and PSEN' signals to the inputs of an AND gate and using the output of the gate as the read strobe to the external program/data memory.

3.3.10 Program memory

The diagram below shows a map of the lower part of the program memory. After reset, the CPU begins execution from location 0000h. Each interrupt is assigned a fixed location in program memory. The interrupt causes the CPU to jump to that location, where it executes the service routine. External Interrupt 0 for example, is assigned to location 0003h. If external Interrupt 0 is used, its service routine must begin at location 0003h. If the interrupt in not used its service location is available as general purpose program memory.



The interrupt service locations are spaced at 8 byte intervals 0003h for external interrupt 0, 000Bh for Timer 0 ,0013h for External interrupt 1,001Bh for Timer1, and so on. If an Interrupt service routine is short enough (as is often the case in control applications). It can reside entirely within that 8-byte interval. Longer service routines can use a

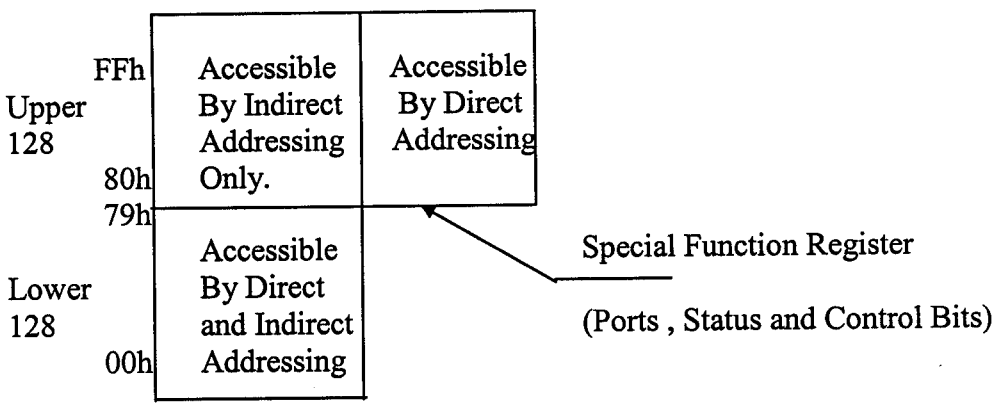
jump instruction to skip over subsequent interrupt locations, if other interrupts are in use.

The lowest addresses of program memory can be either in the on-chip Flash or in an external memory. To make this selection, strap the External Access (EA) pin to either Vcc or GND.

3.3.11 Data memory

The Internal Data memory is divided into three blocks namely,

1. The lower 128 Bytes of Internal RAM.
2. The Upper 128 Bytes of Internal RAM.
3. Special Function Register.



Internal Data memory Addresses are always 1 byte wide, which implies an address space of only 256 bytes. However, the addressing modes for internal RAM can in fact accommodate 384 bytes. Direct addresses higher than 7Fh access one memory space, and indirect addresses higher than 7Fh access a different Memory Space.

The lowest 32 bytes are grouped into 4 bank's of 8 registers. Program instructions call out these registers as R0 through R7. Two bits in the Program Status Word (PSW) Select, which register bank, is in use. This architecture allows more efficient use of code space, since register instructions are shorter than instructions that use direct addressing.

The next 16-bytes above the register bank's form a block of bit addressable memory space. The MicroController instruction set

includes a wide selection of single - bit instructions and these instructions can directly address the 128 bytes in this area. These bit addresses are 00h through 7Fh.

Either direct or indirect addressing can access all of the bytes in lower 128 bytes. The upper 128 can only be accessed by indirect addressing. The upper 128 bytes of RAM are only in the devices with 256 bytes of RAM.

The Special Function Register includes Port latches, timers, peripheral controls etc. Direct addressing can only access these registers. In general, all Atmel MicroController have the same SFRs at the same addresses in SFR space as the AT89C51 and other compatible microcontroller. However, upgrades to the AT89C51 have additional SFRs.

Sixteen addresses in SFR space are both byte and bit addressable. The bit addressable SFRs are those whose address ends in 000B. The bit addresses in this area are 80H through FFH

3.3.12 AT89C51 Addressing modes

The addressing modes of AT89c51 flash MicroController instruction set are as follows.

Direct addressing

In Direct addressing, an 8-bit address field in the instruction specifies the operand. Only internal data RAM and SFR's can be directly addressed.

Indirect addressing

In Indirect addressing, the instruction specifies a register that contains the address of the operand. Both internal and external RAM can indirectly addressed.

The address register for 8-bit addresses can be either the Stack Pointer or R0 or R1 of the selected register Bank. The address register for 16-bit addresses can be only the 16-bit data pointer register, DPTR.

Indexed addressing

Program memory can only be accessed via indexed addressing This addressing mode is intended for reading look-up tables in program memory . A 16 bit base register (Either DPTR or the Program Counter) points to the base of the table, and the accumulator is set up with the table entry number. Adding the Accumulator data to the base pointer forms the address of the table entry in program memory.

Another type of indexed addressing is used in the" case jump " instructions. In this case the destination address of a jump instruction is computed as the sum of the base pointer and the Accumulator data.

Register instruction

The register bank's, which contains registers R0 through R7 , can be accessed by instructions whose opcodes carry a 3-bit register specification. Instructions that access the registers this way make efficient use of code, since this mode eliminates an address byte. When the instruction is executed , one of four bank's is selected at execution time by the two bank select bits in PSW.

Register - specific instruction

Some Instructions are specific to a certain register. For example some instruction always operate on the Accumulator, so no address byte is needed to point to it. In these cases , the opcode itself points to the correct register. Instruction that refer to Accumulator as A assemble as Accumulator - specific Opcodes.

Immediate constants

The value of a constant can follow the opcode in program memory

For example,

MOV A,#100

Loads the accumulator with the decimal number 100. The same number could be specified in hex digit as 64h.

The Parity bit reflect the Number of 1s in the Accumulator. P=1 if the Accumulator contains an odd number of 1s, and P=0 if the Accumulator contains an even number of 1s. Thus, the number of 1s in the Accumulator plus P is always even .Two bits in the PSW are uncommitted and can be used as general purpose status flags.

3.3.14 Oscillator and clock circuit

All Atmel flash microcontroller have an on-chip oscillator that can be used as the clock source for the CPU. To use the on-chip oscillator, connect a crystal between XTAL1 and XTAL2 pins of the MicroController and connect capacitors to the ground. The frequency range is in the range from 1.2 MHz to 12 MHz.

To drive the chip with an internal oscillator, one would ground XTAL1 and XTAL2. Since the input to the clock generator is divide by two flip-flops there are no requirements on the duty cycle of the external oscillator signal.

3.3.15 CPU Timing

A machine cycle consists of 6 States numbering s1 to s6. Each state time lasts for two oscillator periods. Thus a machine cycle lasts 12 oscillator periods or one microsecond if the oscillator frequency is 12 MHz. Each state is divided into a phase 1 half and phases 2 half. Normally two program fetches are generated during each machine cycle even if the instruction does not require it. If the instruction being executed does not need more code bytes the CPU ignores the extra fetch and the PC is not incremented. Execution of one cycle instruction begins during state one of the machine cycle when the opcode is latched into the instruction register. A second fetch occurs during S4 of the same machine cycle. Execution is complete at the end of state 6 of the same machine cycle.

3.3.16 Interrupts

The AT89C51 provides 5 interrupts sources :Two External interrupts , two timer interrupts and a serial port interrupt.

The External Interrupts INT0 and INT1 can each either level activated or transition - activated, depending on bits IT0 and IT1 in

Register TCON. The Flags that actually generate these interrupts are the IE0 and IE1 bits in TCON. When the service routine is vectored to hardware clears the flag that generated an external interrupt only if the interrupt was transition - activated. If the interrupt was level - activated, then the external requesting source (rather than the on-chip hardware) controls the requested flag.

The Timer 0 and Timer 1 Interrupts are generated by Tf0 and Tf1, which are set by a rollover in their respective Timer/Counter Register (except for Timer 0 in Mode 3). When a timer interrupt is generated , the on-chip hardware clears the flag that generated it when the service routine is vectored to.

The Serial Port Interrupt is generated by the logical OR of RI and TI. Neither of these flag is cleared by hardware when the service routine is vectored to . In fact , the service routine normally must determine whether RI or TI generated the interrupt an the bit must be cleared in software.

In the Serial Port Interrupt is generated by the logical OR of RI and TI . Neither of these flags is cleared by hardware when the service routine is vectored to.

In fact, the service routine normally must determine whether RI to TI generated the interrupt and the bit must be cleared in software.

IE: Interrupt Register

EA	-	ET2	ES	ET1	EX1	ET0	EX0

Enable bit = 1 enabled the interrupt & 0 disables it

The table shows the various flags in the interrupt enable register.

SYMBOL	POSITION	FUNCTION
EA	IE.7	Disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1 then each interrupt source will be individually enabled or disabled by setting or clearing its enable bit.
-	IE.6	Reserved
-	IE.5	Reserved
E6	IE.4	Serial port interrupts enable bit.
ET1	IE.3	Timer 1 overflow interrupts enable bit.
EX1	IE.2	External interrupt 1 enable bit
ET0	IE.1	Timer 0 overflow interrupt enable bit.
EX0	IE.0	External interrupts 0 enables bit.

INTERRUPT PRIORITIES

Each interrupt source can also be individually programmed to one of two priority levels by setting or clearing the interrupt priority (IP) bit in the SFR.

A low priority interrupt can be interrupted by a high priority interrupt but not by another low priority interrupt. On the other hand a high priority interrupt cannot be interrupted by any other interrupt source. If interrupts of similar priorities are received an internal polling sequence determines which request is to be serviced.

3.4 PROGRAMMABLE PHERIPHERAL INTERFACE (8255A)

Fig 3.5 shows the internal block diagram of 8255A Programmable peripheral interface. It has 24 I/O pins that can be primarily grouped into two 8 bit parallel ports. A and B, with the remaining eight bits as ports C. The eight bits of ports C can be used as individual bits or be grouped in two four bits. C_{upper} and C_{lower} are defined by writing a control word in the control register.

The functions of 8255 are classified under two functions, the bit/set mode and the I/O mode. The BSR mode is used to set or reset the bits in port C.

The I/O mode is further divided into three modes: mode 0, mode 1, mode 2. In mode 0 operation, all ports function as simple I/O ports. Mode 1 is a handshake mode whereby ports A and B use bits from port C as handshake signals. In the handshake mode, two types of I/O data transfer can be implemented: status check and interrupt. In mode 2, Port A can be set up for bi-directional data transfer using handshake signals from port C.

3.4.1 Programming the 8255A

Constructing and sending 8255A control words

The MSB of the control word tells the 8255A which control word we are sending it. We use the mode definition control word format to tell the device what modes you want the ports to operate in. For the mode definition control word we usually put a 1 in the MSB. We use the bit set\reset control word format when we want to set or reset the output on a pin or port C or when we want to enable the interrupt output signals for handshake data transfers. The MSB is 0 for this control word.

Both control words are sent to the control register address for the 8255 A. As usual making up a control word consists of figuring out what to put in the 8 boxes one at a time.

3.4.2 Control logic

The control section has six lines. Their functions and connections are as follows.

RD' : This control signal enables the read operation. When the signal is low, the MicroController reads data from the selected I/O port of the 8255a.

WR' : This control signal enables the write operation. When the signal goes low, the MicroController writes into a selected I/O port or the control register.

RESET : This is an active high signal; it clears the control register and sets all port in the input mode.

CS', A_0 , A_1 : These are device select signals. CS' is connected to a decoded address and A_0 and A_1 are generally connected to the MicroController address lines A_0 and A_1 respectively. The CS' signal is the master chip select signal and A_0 and A_1 specify one of the I/O ports or the control register as given below.

CS'	A_1	A_0	Selected
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control Reg
1	X	X	Not Selected

3.4.3 MODE 0 : Simple input or output

In this mode, ports A and B are used as two simple 8-bit I/O ports and port c as two 4 bit ports. Each port (or half port as in the case of port C) can be programmed to function simply as an input or an output port. The I/O features in mode 0 are as follows:

- Outputs are latched
- Inputs are not latched
- Ports do not have handshake or interrupt capability.

3.4.4 MODE 1 : Input or output with handshake

In mode 1 handshake signals are exchanged between the MicroController and peripherals prior to data transfer. The feature of the mode include the following.

- Two Ports (A & B) function AS 8 bit I/O ports
- Each port uses three lines from port C AS handshake signals and the remaining two lines can be used for simple I/O functions.
- Input and Output are latched.
- Interrupt logic is supported.

3.4.5 MODE 2 : Bi-directional data transfer

This mode is primarily used in applications such as data transfer between two computers or floppy disc controller interface. In this mode, port A can be configured AS the bi-directional port and port B either in mode 0 or mode 1. PortA uses five signals from port C as handshake signals for data transfer. The remaining three signals from port C can be used as simple I/O or as handshake for port B.

3.5 RANDOM ACCESS MEMORY (6264 STATIC RAM)

The Fig.3.6 shows the HM6264 high-performance CMOS static RAM. It is organized as 8192 words by 8 bits(8k x 8). Easy memory expansion is enabled by an active low chip enable CE', an active HIGH chip enable CE_2 and active low O/P enable(OE') and three state drivers.

Both devices has an automatic power-down feature(CE'), reducing the power consumption by over 70% when de-selected. An active low write enable signal(WE') controls the writing/reading operation of the memory. When CE' and WE' inputs are both low and CE_2 is high, data on the eight data input/output pins (I/O 0 through I/O 7) is written into the memory location addressed by the address present on the address pins(A0 through A2). Reading the device is accomplished by selecting the device and enabling the outputs. CE''and OE' active low, CE_2 active high, while WE' remains inactive or high. Under these conditions, the contents of the location addressed by the information on address pins is present on the eight data input/output pins.

3.5.1 Truth table :

WE'	CS1'	CS2	OE'	Mode
X	H	X	X	Not Selected
X	X	L	X	
H	L	H	H	Output disabled
H	L	H	L	Read
L	L	H	H	Write
L	L	H	L	Write

The input/output pins remain in high impedance state unless the chip is selected, outputs are enabled and write enable (WE') is high. A die coat is used to insure alpha immunity.

The following are certain requirements of HM6264 when used with the MicroController.

1. HM6264 requires address lines to identify a memory register. The number of address lines required is determined from the no of registers. The address lines necessary for the memory chip should be connected with that of the MicroController.
2. It requires a chip select(CS') signal to enable the chip. The remaining address lines of the MicroController can be connected to the CS' signal through an interfacing logic.
3. The address lines connected to the CS' select the chip and the address lines connected to the address lines of the memory chip select the register.
4. The control signal RD' enables the output buffer and data from the selected register are made available on the output lines. Similarly, the control signal write (WR') enables the input buffer and data on the input lines are written to memory cells. The MicroController can use its control signals to enable the buffers and the databus to transport the contents of selected register between the MicroController and memory.

Functional Block Diagram

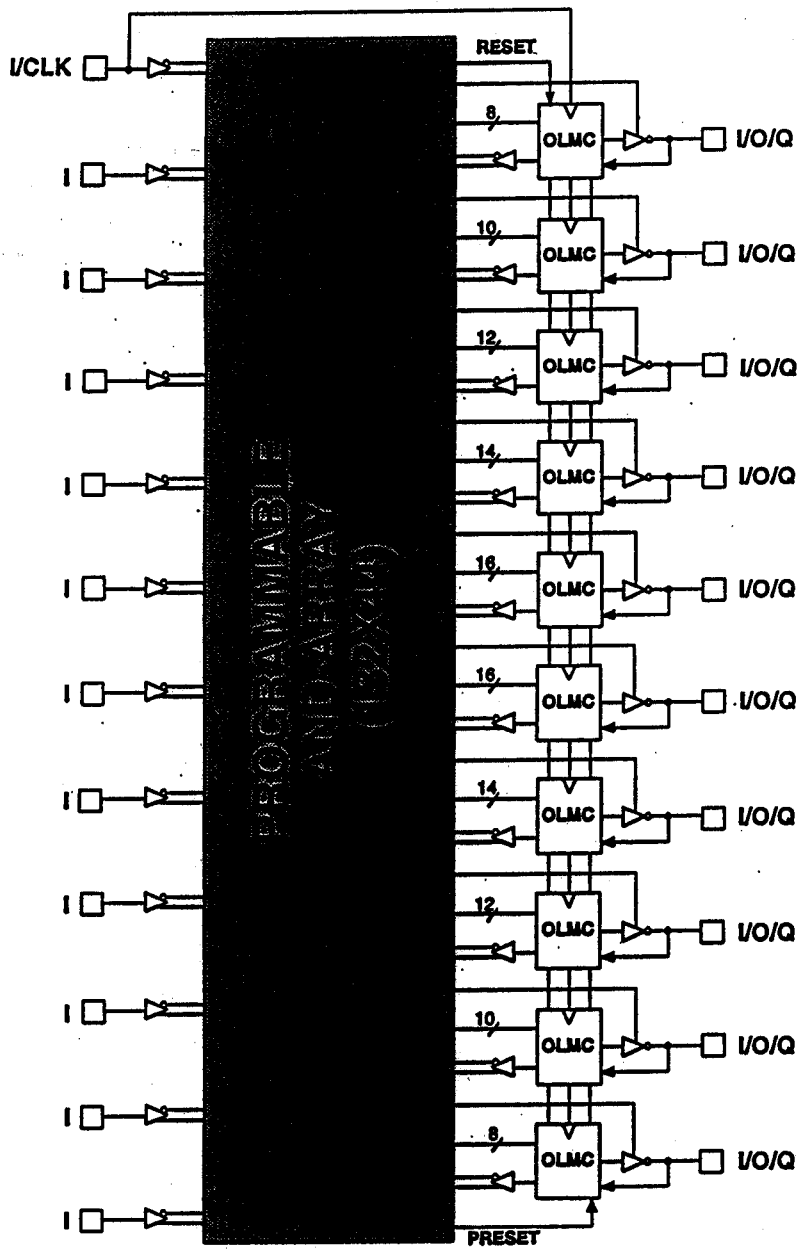


FIG 3.1 FUCTIONAL BLOCK DIAGRAM OF GAL

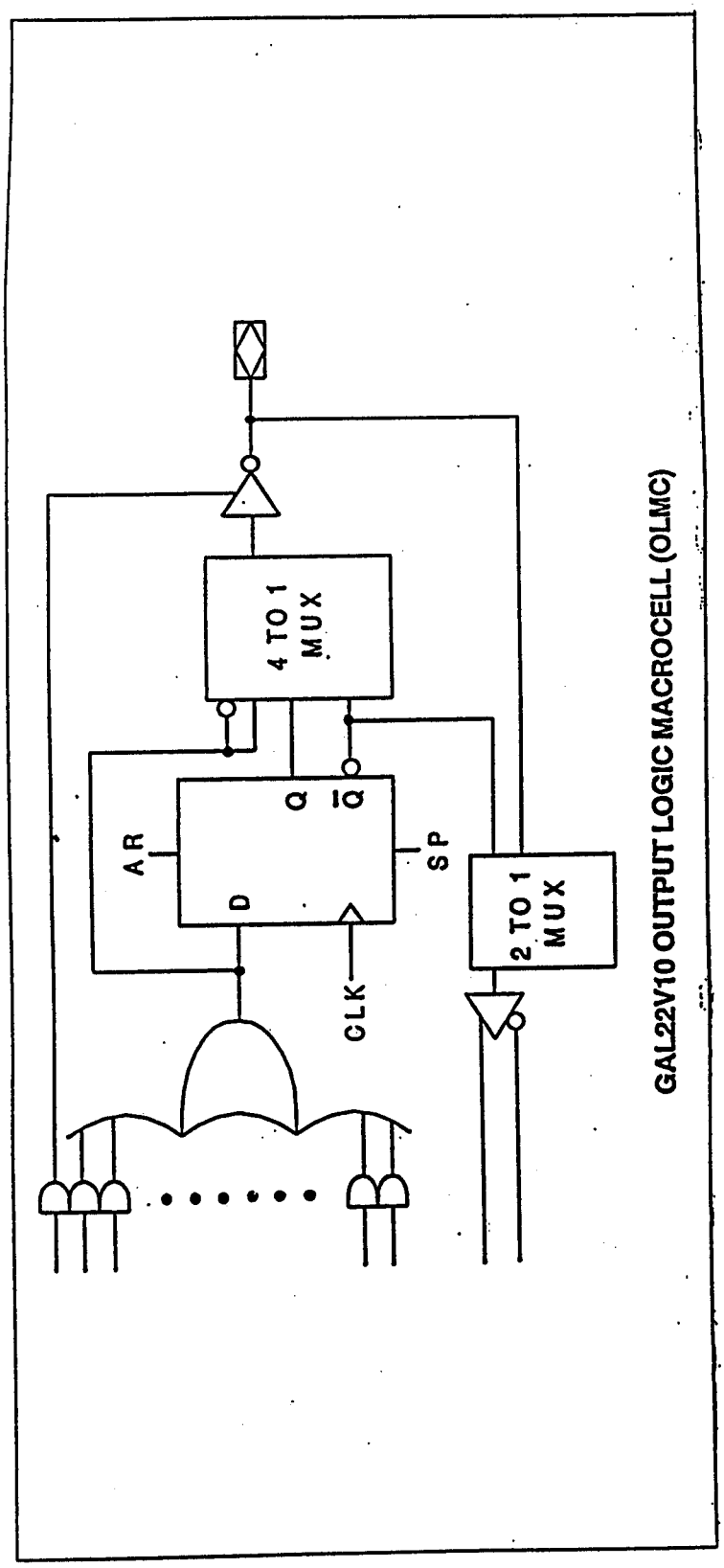
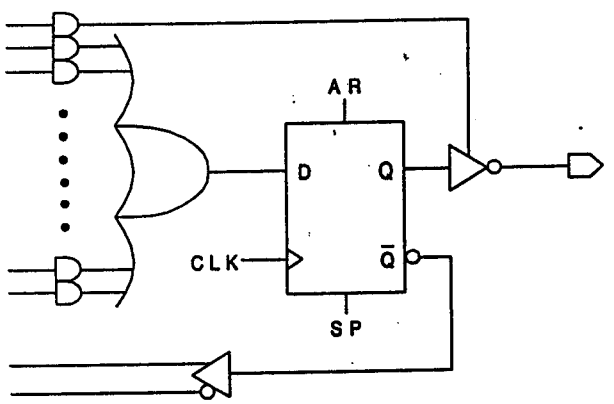


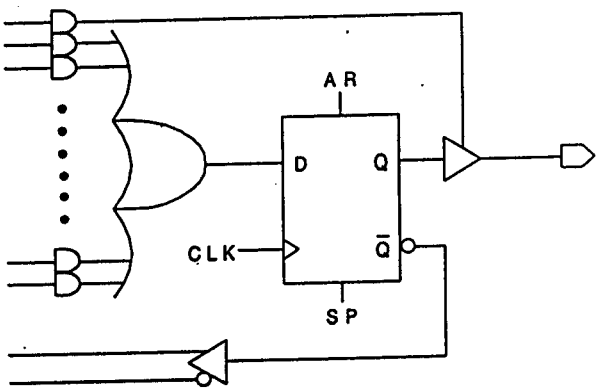
FIG 3.2 OUTPUT LOGIC MACROCELL

Registered Mode



ACTIVE LOW

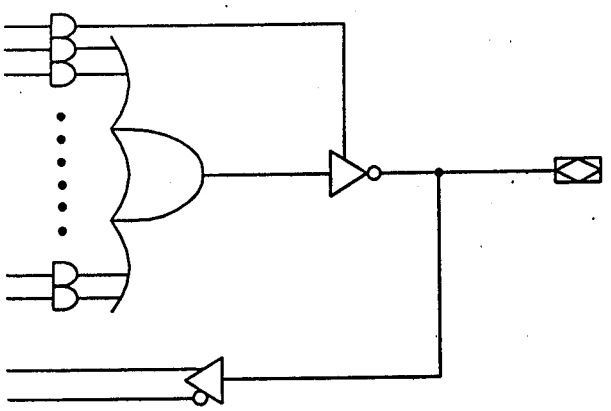
$S_0 = 0$
$S_1 = 0$



ACTIVE HIGH

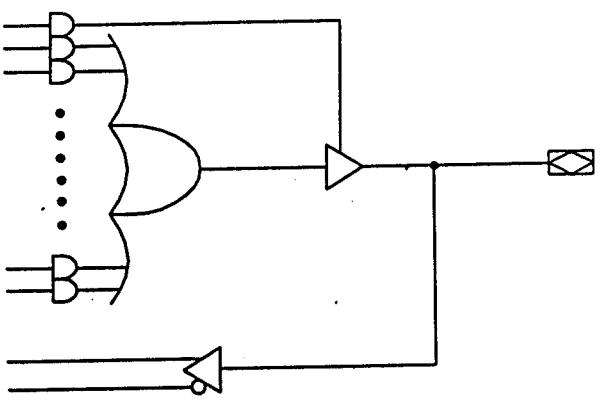
$S_0 = 1$
$S_1 = 0$

Combinatorial Mode



ACTIVE LOW

$S_0 = 0$
$S_1 = 1$



ACTIVE HIGH

$S_0 = 1$
$S_1 = 1$

FIG 3.3 - REGISTERED I/O & COMBINATIONAL I/O

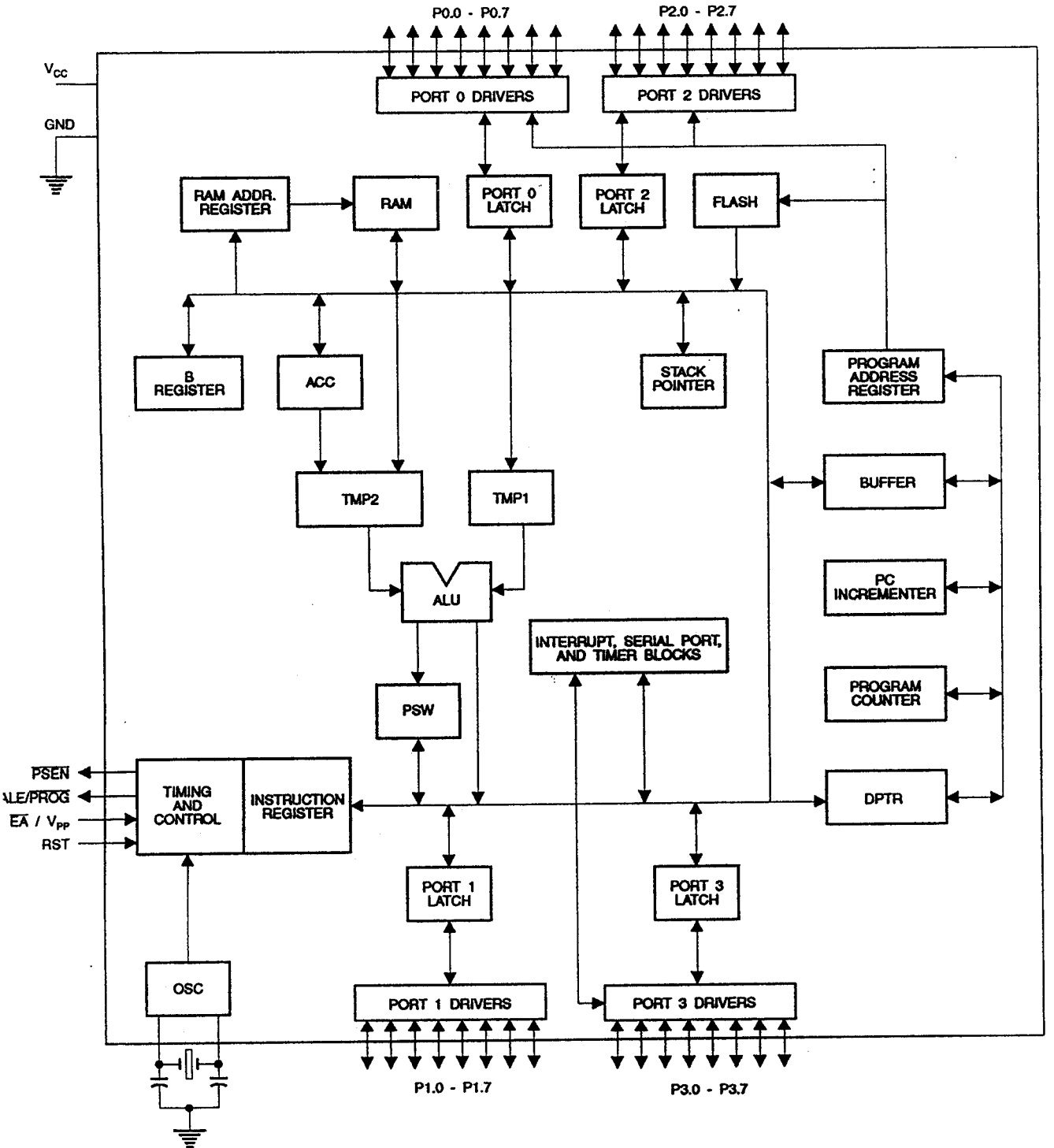


FIG 3.4 ARCHITECTURE OF AT89C51

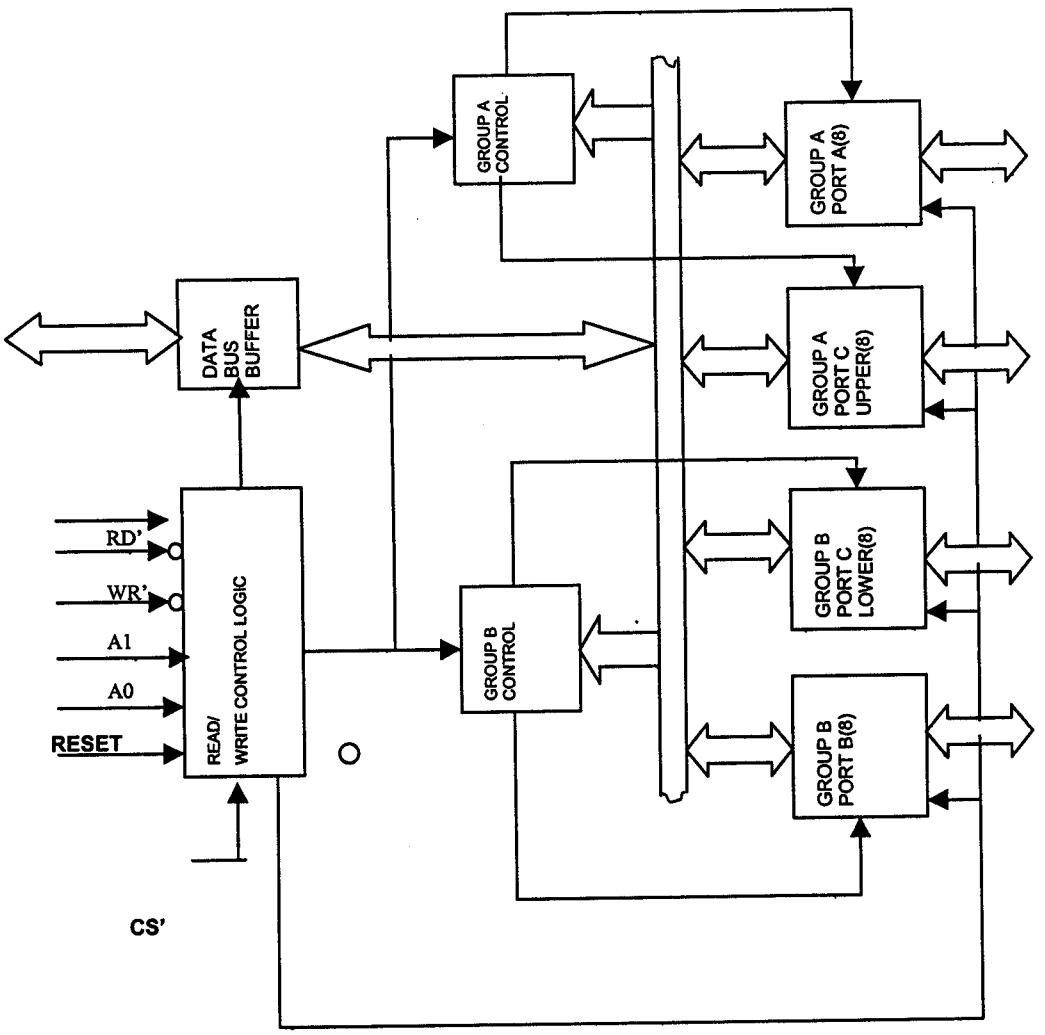


FIG 3.5 PROGRAMMABLE PHERIPHERAL INTERFACE

Logical Diagram

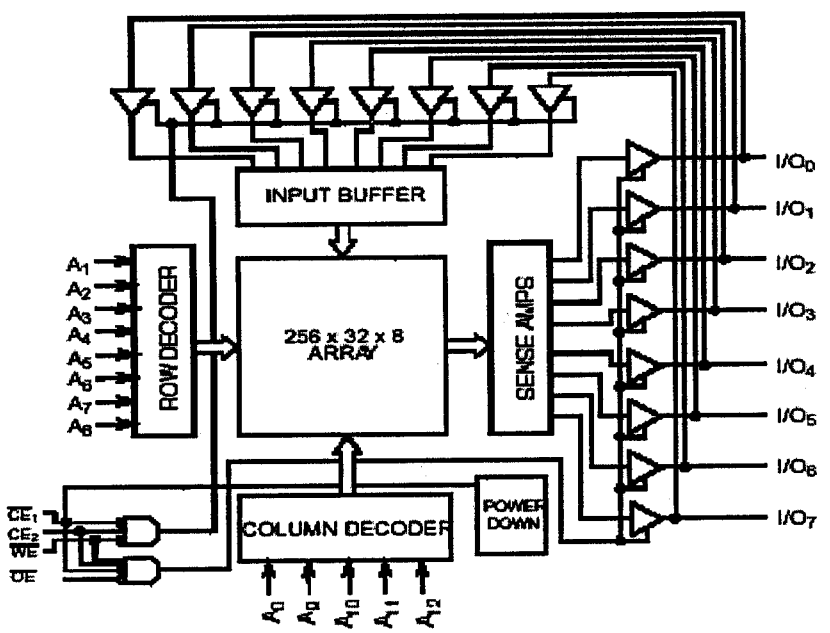


FIG 3.6 HM 6264 RAM

CHAPTER 4

INPUT / OUTPUT UNITS

CHAPTER 4

INPUT / OUTPUT UNITS

Serving the needs of input and output, we make use of an L.C.D as the display unit and keypad as the input unit.

4.1 DISPLAY UNIT

A Liquid Crystal Display (L.C.D – CDM 16216) is used for the displaying purpose. The L.C.D has the provision of displaying 16 characters/line for two lines. The L.C.D comprises of a controller I.C, DDRAM, CGRAM and CGROM.

The Fig. 4.1 shows the LCD unit. The controller I.C has two registers one is the Instruction Register (IR) and the other is the Data Register (DR). The IR stores the Instruction codes and the address information for DDRAM and CGRAM. The DR temporarily stores data to be written to/ read from the DDRAM or CGRAM. When a data is sent to the DR for writing, the display of the character takes place automatically as an internal operation. Similarly, when the address code is sent to the IR, the data from the specified address is automatically transferred from the DDRAM or CGRAM to the DR. The Register Selection table is as follows.

RS	R/W	Operation
0	0	IR write AS an internal operation.
0	1	Read Busy flag(DB_7) and Address
1	0	Counter (DB_0 to DB_7) DR write AS an internal operation
1	1	(DR to DD RAM or CG RAM) DR read AS an internal operation (DD RAM or CG RAM to DR)

The data of the Address Counter is output to DB_0 to DB_6 while R/W =1 and RS=0. The character codes are usually stored in the form of 8 bit character codes present in the character font table. With the pin configuration of the controller I.C, we will be able to acquire a detailed knowledge regarding the connections to be made:

PIN NO.	SYMBOL	DESCRIPTION
Pin 1	VSS	Ground Terminal, 0V
Pin 2	VDD	Supply Terminal, +5V
Pin 3	VL	Liquid Crystal Drive Voltage.
Pin 4	RS	Register Select: RS=0- Instruction Reg. RS=1- Data Reg.
Pin 5	R/W	Read/Write R/W=1 – Read R/W=0 – Write
Pin 6	E	Enables Read/Write
Pin 7 to Pin 14	DBo to DB7	Bi-directional Data bus
Pin 15 Pin 16	Backlight Supply	In case of 15 pin module, supply is given to 15th pin and in 16 pin module, supply is given to 16th pin while 15th pin is grounded.

The Module, interfaced to the system, can be treated as RAM (Memory Mapping), input/output, Expanded or Parallel I/O (input/output mapping). Since there is no conventional Chip select signal, developing a strobe signal for the Enable signal (E) and applying appropriate signals to the Register Select (RS) and Read/Write(R/W) signals are important. The module is selected by gating a decoded Module – address with the host processor's Read/Write strobe. The resultant signal, applied to the LCD's Enable (E) input, clocks in the data.

The 'E' signal must be a positive going digital strobe, which is active while data and control information are stable and true. The falling edge of the Enable signal enables the data/instruction register of the controller. The read and write strobes of the host, which provides the 'E' signal, should not be liked to the module's R/W line. An address bit, which sets up earlier in the host's machine cycle can be, used AS R/W.

When the controller is performing an internal operation, the busy flag (DB_7) will set and will not accept any instruction. The user should check the Busy flag or should provide a delay of approximately 2 ms after each instruction. The L.C.D can e interfaced either to 4 bit or 8 bit Micro processor/ MicroController. For 4-bit data interface, the bus lines DB_4 to DB_7 are used for data transfer, while DB_0 to DB_3 lines are disabled. The data transfer is complete when the 4-bit data has been transferred twice. The busy flag must be checked after the 4-bit data has been transferred twice. Two more 4-bit operations then transfer the busy flag and address counter data. For 8-bit data, all 8 bus lines (DB_0 to DB_7) are used.

4.1.1 Instruction codes

Here are a list of certain instruction codes that are used for various LCD operations carried out by the MicroController whenever necessary.

- Clear Display - 01H
- Return Home - 02H,03H

- ♦ Entry Mode Set - 04H to 07H
- ♦ Display on/off - 08H to 0fH
- ♦ Cursor shift - 10H to 1FH
- ♦ Function set - 20H to 3FH
- ♦ Set CG RAM address - 40H to 7FH

4.1.2 Initialization by instruction

1) Turn on the power and wait for more than 15ms after VDD rises to 4.5 V
2) Set interface AS 8 bits long using the function set.
3) Wait for more than 4.1ms
4) Set interface AS 8 bits long using the function set.
5) Wait for more than 100 micro secs.
6) Set interface AS 8 bits long using the function set.
7) Busy flag (DB_7) can be checked now. When Busy flag is not checked, the waiting time between instructions should be longer than the instruction execution time.
8) Set interface AS 4-bits / 8-bits, number of display lines and character font using function set. The number of display lines and character font cannot be changed after this point.
9) Check the busy flag or give a time delay of 2ms.
10) Set display off (Display on/off control) .
11) Check the busy flag or give a time delay of 2ms.
12) Set display clear (clear display)
13) Set cursors move direction and specify display shift (Entry mode)
14) Check the busy flag or give a time delay of 2ms.
15) Initialization completed.

4.2 INPUT UNIT - KEYPAD

The Fig.4.1 shows the Keypad Circuit. The keys from k1 to k8 are tied through 10k resistors and when a key is pressed, the corresponding line is grounded. When all keys are open and if the MicroController reads port1, the reading on the data bus will be FFH("11111111" Initially set to 1). When any key is pressed the reading will be less than FFH.

For Eg., If K7 is pressed, the output of port 1 will be 7FH(01111111). This readings will be encoded into the binary equivalent of the digit 7(0111). By using software routines, the flow diagram shown in Fig.4.3 can be implemented.

Thus the keys along with the pull-in resistors form the input setup for the device and these resistors are always tied up to the V_{cc} i.e. 5V supply.

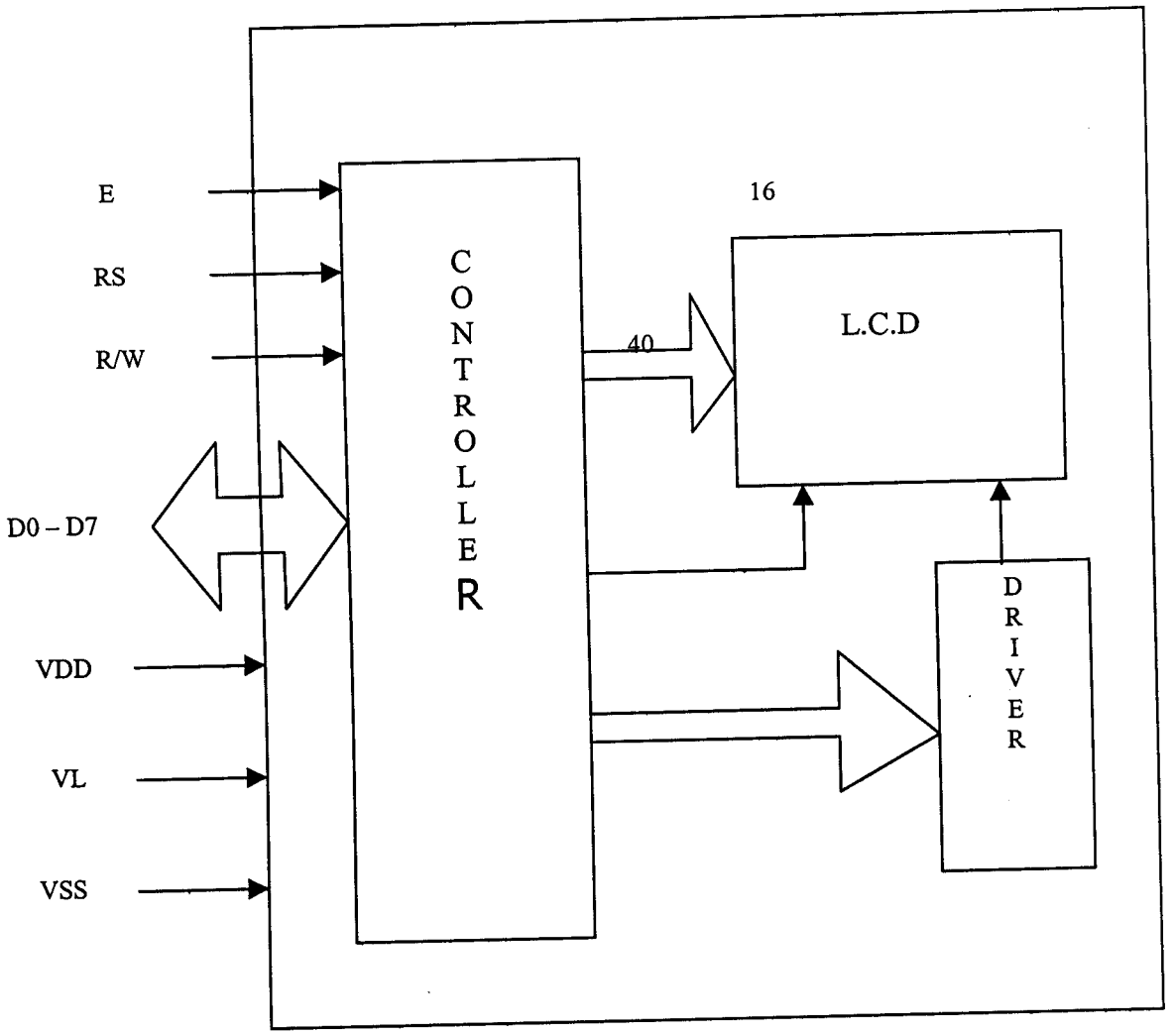


FIG 4.1 LCD BLOCK DIAGRAM

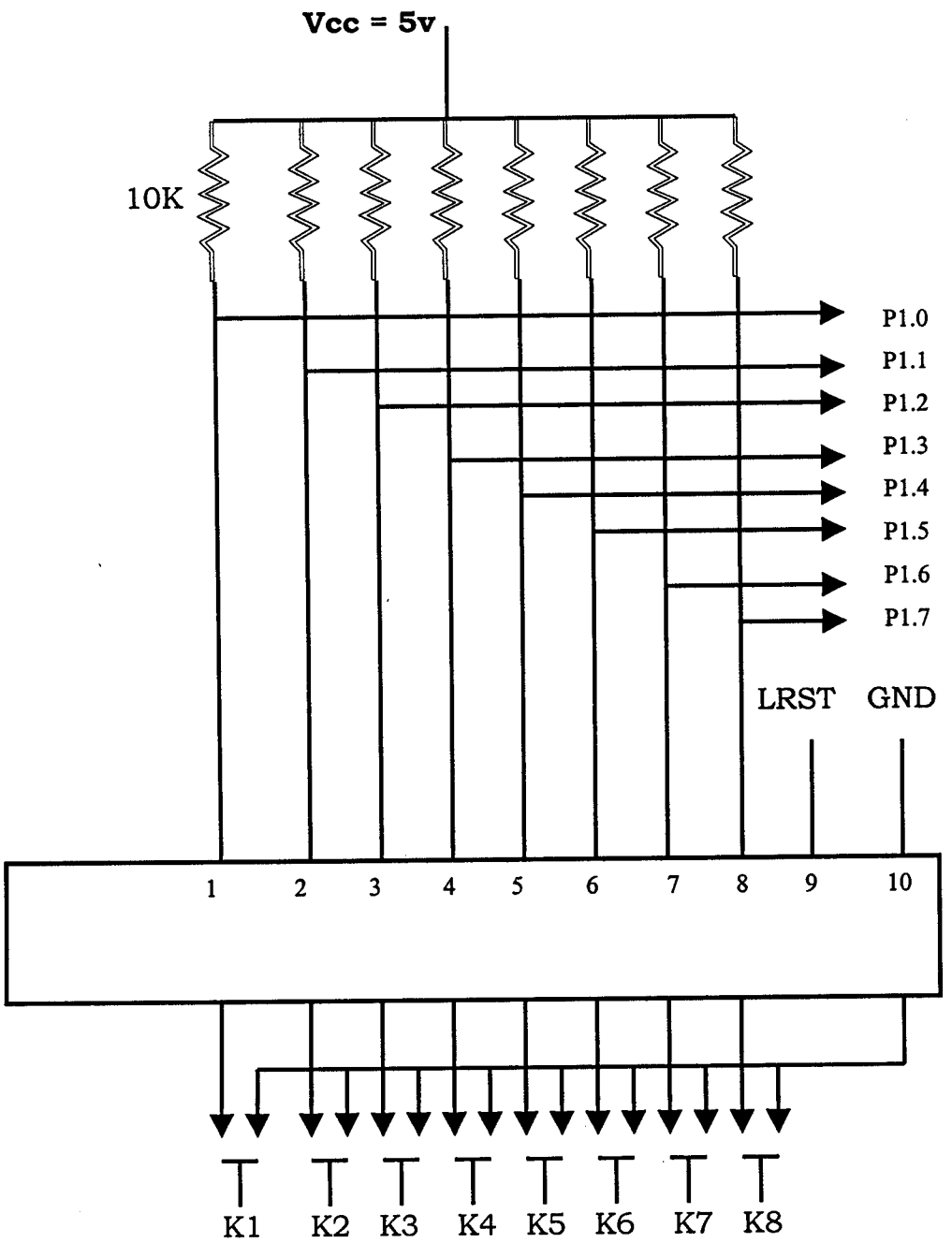


FIG 4.1 KEYPAD CIRCUITRY

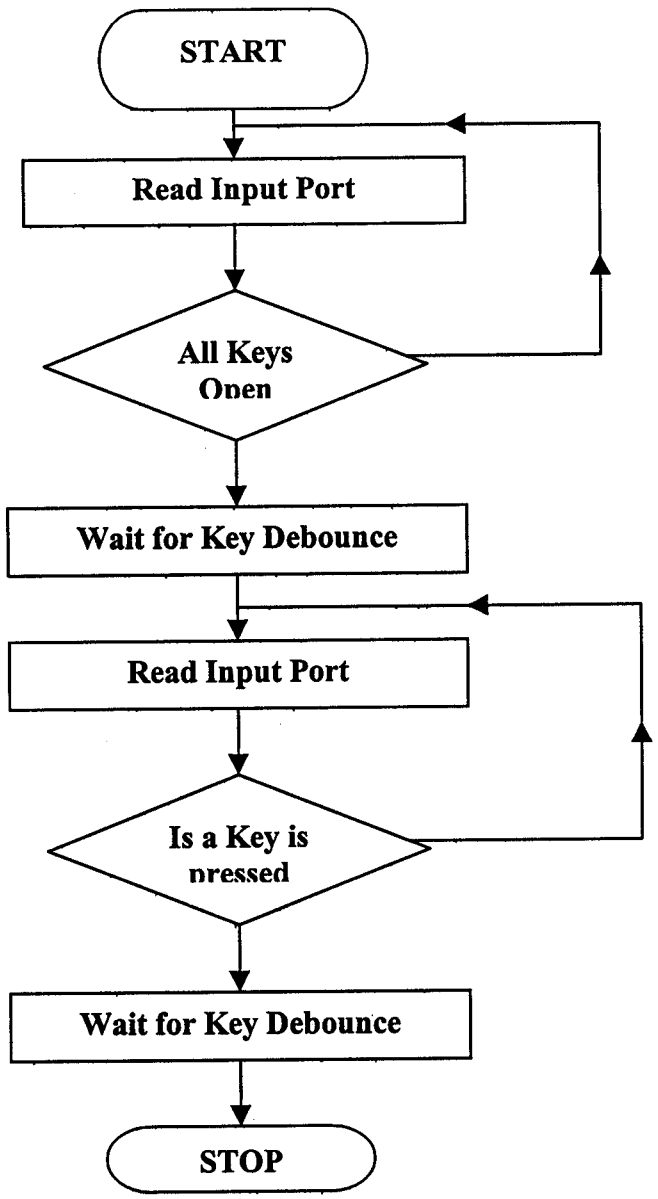


FIG 4.2 KEYPAD FLOW DIAGRAM

CHAPTER 5

VHSIC HDL

CHAPTER 5

VHSIC HDL

5.1 WHAT IS VHDL

VERY HIGH SPEED INTEGRATED CIRCUITS HARDWARE DESCRIPTION LANGUAGE (VHDL)

VHDL is a programming language that has been designed and optimized for describing the behavior of hardware digital circuits and systems. As such, VHDL combines features of a simulation modeling language, a design entry language, a test language, and a netlist language.

As a simulation modeling language, VHDL includes many features appropriate for describing the behavior of electronic components ranging from simple logic gates to complete microprocessors and custom chips. Features of VHDL allow electrical aspects of circuit behavior (such as rise time and fall times of signals, delays through gates, and functional operation) to be precisely described. The resulting VHDL simulation models can then be used as building blocks in larger circuits (using schematics, block diagrams or system-level VHDL descriptions) for the purpose of simulation. Just as high-level programming languages allow complex design concepts to be expressed as computer programs, VHDL allows the behavior of complex electronic circuits to be captured into a design system for automatic circuit synthesis or for system simulation. This process is called **design entry,** and is the first step taken when a circuit concept is to be realized using computer-aided design tools and the entire flow can observed in Fig.5.1.

Design entry using VHDL is very much like software design using a software programming language. Like Pascal, C and C++, VHDL includes features useful for structured design techniques, and offers a rich set of control and data representation features. Unlike these other programming languages, VHDL provides features allowing concurrent

events to be described. This is important because the hardware being described using VHDL is inherently concurrent in its operation. Users of PLD programming languages such as PALASM, ABEL, CUPL and others will find the concurrent features of VHDL quite familiar.

One area where hardware design differs from software design is in the area of testing. One of the most important (and under-utilized) aspects of VHDL is its use as a way to capture the performance specification for a circuit, in the form of what is commonly referred to as a **Test bench.** Test benches are VHDL descriptions of circuit stimulus and corresponding expected outputs that verify the behavior of a circuit over time. Test benches should be an integral part of any VHDL project and should be created in parallel with other descriptions of the circuit. And, while VHDL is a powerful language with which to enter new designs at a high level, it is also useful as a low-level form of communication between different tools in a computer based design environment.

5.2 HARDWARE

Thinking in hardware, it turns out that writing effective VHDL RTL is easy. The key is to realize that the code describes hardware, not software. The main idea of "thinking in hardware" is to always have a good idea of what the lines of code really mean in terms of hardware. After writing a sequence of RTL, we should be able to sketch out a quick block diagram of what the code physically represents.

5.2.1 Timing

This is the most important part of "Thinking in Hardware". Writing RTL that does not take a signal's timing into consideration is not useful. Signals may stabilize at any point in the clock cycle. Register-driven signals always have the earliest stability time, and different signals may have different maturity times depending on how much combinational logic precedes them in the timing path.

Writing an RTL process that uses four 32-bit addresses in series is usually impractical. The designer must recognize that a clock stage is needed to break up the timing path. A register should be called out to hold an intermediate term, and the final addition(s) should be done in the next clock cycle. The concept of relative signal timing is nothing new in the hardware design, but it is easy to have the perception that VHDL will somehow magically take care of the issue. VHDL takes care of gate-level complexity. It does not take care of gate-level timing.

To synthesize an entire chip, a certain methodology must be decided upon and consistently. Synopsis provides two fundamental means to achieve complete-chip synthesis: iterative entity compile and hierarchical compile. The project used the iterative entity compile approach.

The idea of this approach is as follows:

1. Synthesize each RTL entity independently.
2. "Link" the design hierarchy. This is not "synthesis"; rather, it is a means for the synthesizer to gather the chip's high level structural and timing information into one database. A complete-chip timing report is available here.
3. "Characterize" each entity. This synthesizer process calculates all information pertinent to the entity being characterized (output signal loads, input signal drive strengths and arrival times, and delay constraints.)
4. Go back to step 1 and use the characterized data as input into synthesizer.

The final VHDL code was very similar to the pre-synthesized code. However, we did have to pull many "excess" registers out of the design in order for the vendor to be able to route the chip.

5.2.2 Hardware simulation

Hardware description languages are modeling tools for creating a hardware model. One such application is simulation. Simulation is the act of exercising a model of an actual component for analyzing its conduct under a given set of conditions and/or stimuli. In

hardware design environment, a hardware description uses component models and definitions from a simulation library to form a simulatable hardware model. The hardware model and a set of stimuli are used as inputs of a hardware simulation engine. The simulator produces simulation results that are indicative of the hardware component for the given set of test data. Details of the results obtained from a simulation run depend on the level of details of the model.

Simulators can be used at any design stage. At the upper levels of the design process, simulation provides information regarding the functionality of the system under design. Low level or device simulation, runs much more slowly but provides more detailed information about the timing and functionality of the circuit. Regardless of the level of design to which a simulation program is applied, digital system simulators have generally been classified into oblivious and event-driven simulators. In oblivious simulation, each circuit component is evaluated at fixed time points, while in event driven simulation, a component is evaluated only when one of its input changes.

5.3 VHDL REQUIREMENTS

The "Department of Defense Requirements for Hardware Description Languages" specifies that the VHSIC HDL should be a language for design and description of hardware. It indicates that VHDL should be usable for design documentation, high level design, simulation, synthesis, and testing of hardware and as a driver for physical design tool. Since in an actual digital system, all small or large elements of the system are active simultaneously and perform their tasks concurrently, the concurrency aspect of VHDL is heavily emphasized. In HDL, concurrency means that transfer statements, descriptions of components, and instantiations of gates or logical units are all executed such that in the end they appear to have been executed simultaneously.

5.4 DESIGN UNITS

One concept unique to VHDL (when compared to software programming languages and to its main rival, Verilog HDL) is the concept of a design unit. Design units in VHDL (which may also be referred to as library units) are segments of VHDL code that can be compiled separately and stored in a library.

The Fig.5.2 illustrates the relationship between these five design Units:

5.4.1 Entity

All designs are expressed in terms of entities. An entity is the most basic building block in a design. The uppermost level of the design is the top-level entity. If the design is hierarchical, then the top-level description will be lower-level entities contained in the top-level entity description.

5.4.2 Architecture

All entities that can be simulated have an architecture description. The architecture describes the behavior of the entity. A single entity can have multiple architectures. One architecture might be behavioral, while another might be a structural description of the design.

5.4.3 Configuration

A configuration statement is used to bind a component instance to an entity-architecture pair. A configuration can be considered as a parts list for a design. It describes which behavior to use for each entity, much like parts list describes which part to use for each part in the design.

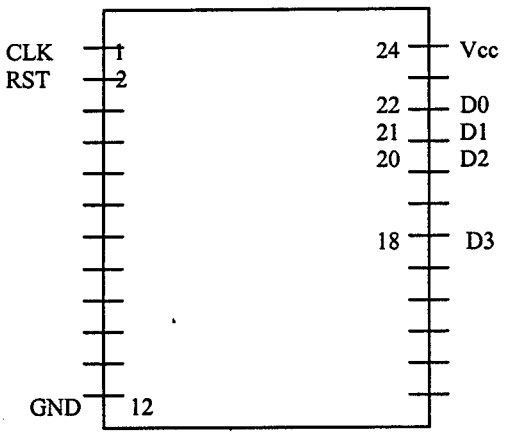
5.4.4 Package

A package is a collection of commonly used data types and subprograms used in a design. Think of a package as a toolbox that contains tools used to build designs

5.4.5 Package body

A Package body is used to store the definitions of functions and procedures that were declared in the corresponding package declaration.

5.5 DESIGN OF SMART CARD



GAL 22V10B

5.5.1 Details of GAL 22V10B Interface

1. The clock from the micro controller(Pin 3.4) is given to the Pin 1 of the smart card
2. The reset from the microcontroller(Pin 3.5) applied to Pin 2 is of asynchronous type which resets all register content to zero
3. For every clock pulse, 4 bits are available as outputs at the pins 18,20,21,and 22.

Hence, for every 2 clock pulses, a character of 8-bits is sent to the location in the RAM

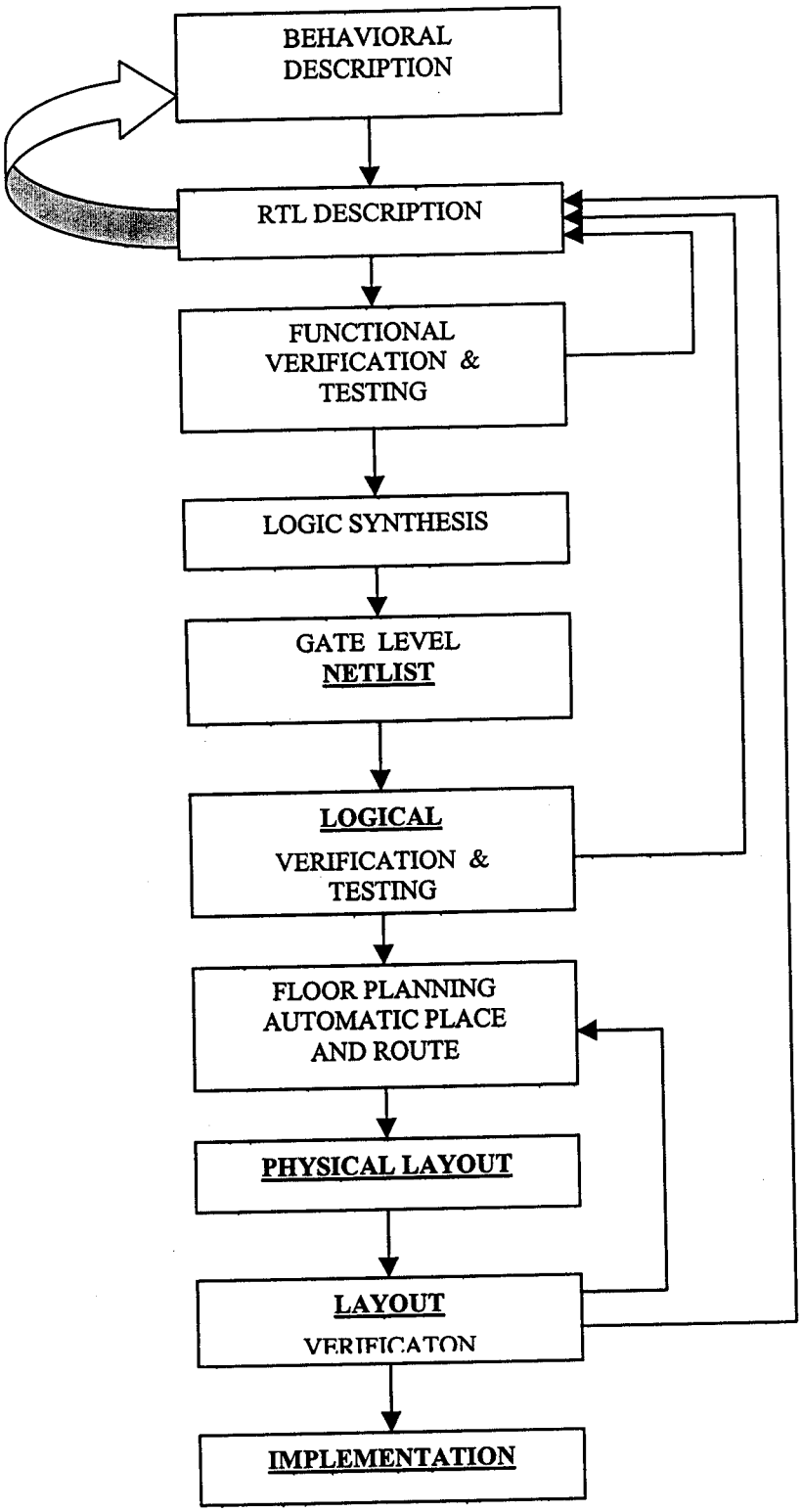


FIG 5.1 TYPICAL DESIGN FLOW

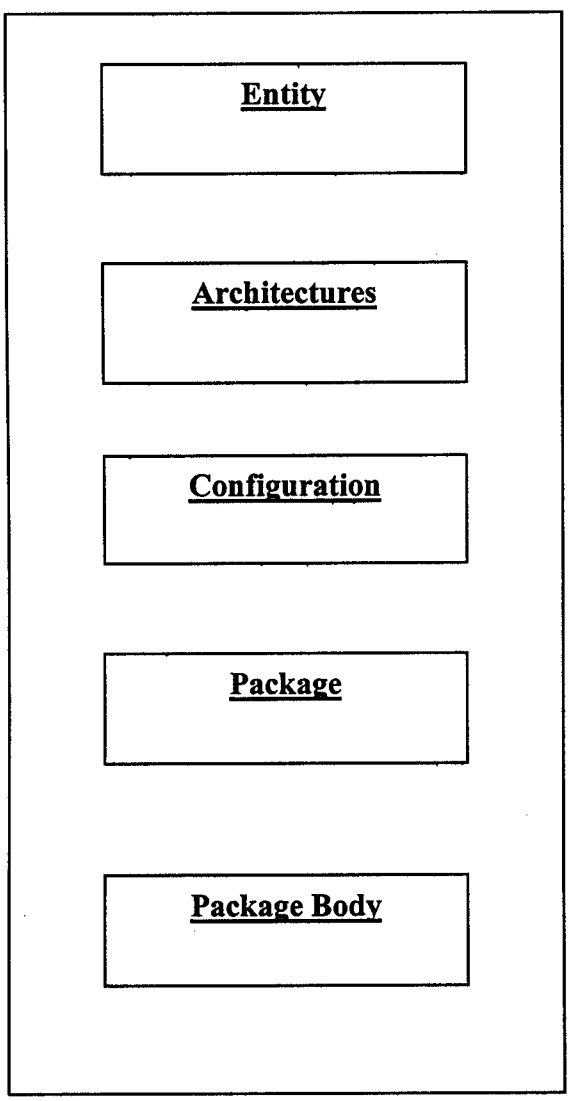


FIG 5.2 DESIGN ENTITIES

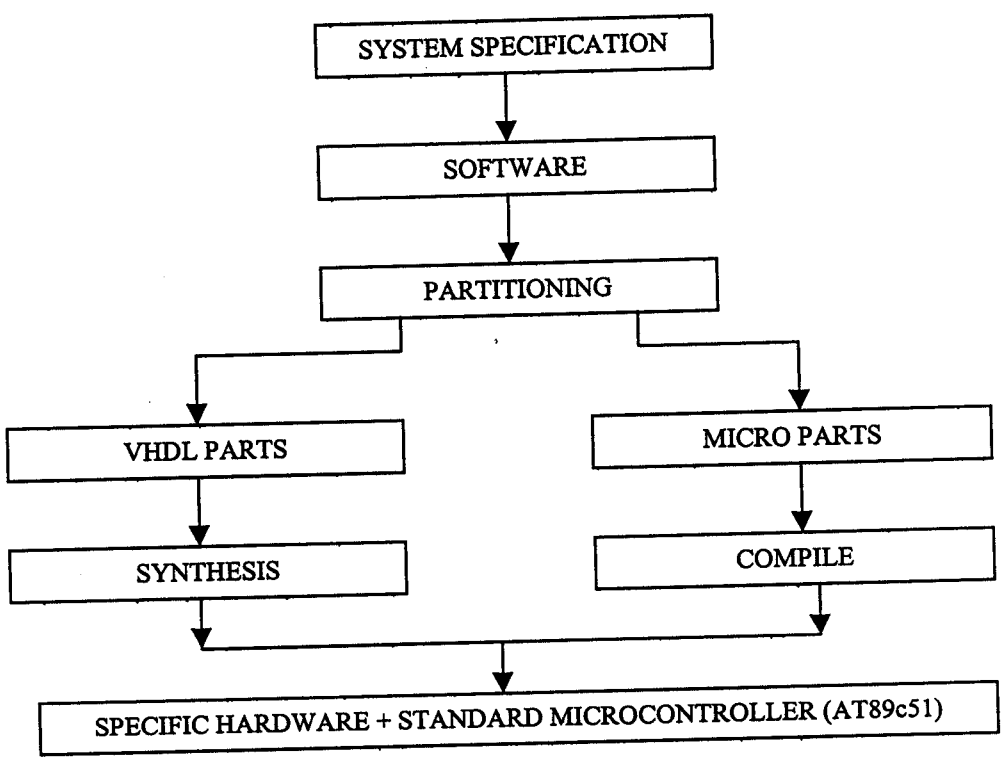
CHAPTER 6

SOFTWARE

CHAPTER 6
SOFTWARE

For a complete system, there should be the perfect integration of both hardware and software. So far we had been going through the hardware, now we shall look into the software part. The devices that need programming attention is that of the Programmable Logic Device GAL 22v10B and the other is that of the At89c51 MicroController. They are respectively programmed in VHDL and Assembly Language.

The trajectorial flow of the programming part is illustrated below



In order to program the PLD GAL 22v10B in VHDL, Warp (Ver. 4) was the major tool used. Warp (Ver. 4) is a registered trademark of Cypress Semiconductors. It is a very effective tool when it comes to PLD programming and supports all the latest IEEE standards (1164).

The heart of software programming lies in Assembly programming to program the AT89c51 MicroController. Both these devices are programmed via the Universal Programmer. It is a user-friendly device that can be interfaced with a PC.

6.1 PLD PROGRAMMING

Programming a PLD is very easy, flexible and inexpensive as compared to other programmable devices. Due to the inherent advantages of VHDL, it finds a wide application when it comes to programming a PLD. The various features include:

1. Less design time
2. Low production cost
3. Easy to change the design
4. Can adapt to fast changing environments

6.1.1 Programming algorithm

1. Include the library "ieee.std_logic_1164.all"
2. Include the library "work.STD_ARITH.all"

Counter

1. Create the structure counter
2. Declare clock, reset as inputs and an array counter_out as output of the entity counter
3. Wait for the clock pulse from the MicroController.
4. Check whether reset is high
5. If reset is high go to step 6 else go to step 7
6. Set counter_out to "000000" go to step 8
7. Set counter_out to counter_out + 1
8. Go to step 3

Multiplexer

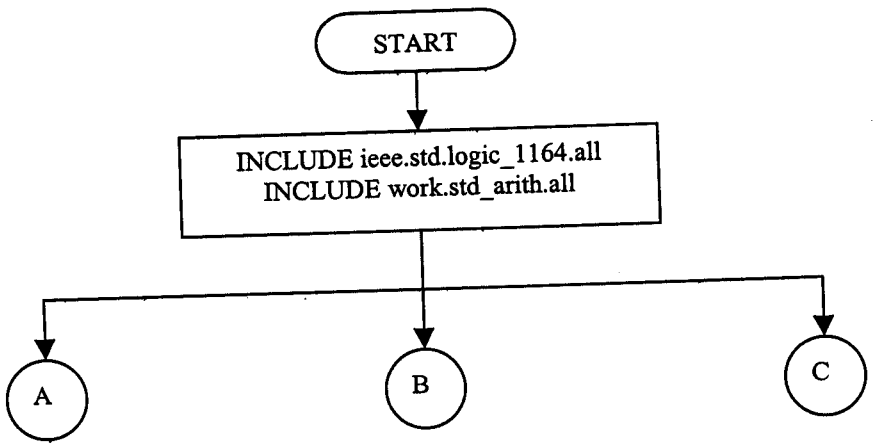
1. Create the structure multiplexer
2. Declare reset, an array counter_in of size 5, data_in of size 3 as inputs and Y as output
3. The output of the counter is the input of the multiplexer

4. Monitor output of the counter.
5. If there is any change in the output of the counter go to step 6 else go to step 4
6. Check if reset is zero. If 'yes' go to step 8 else set output as zero.
7. Go to step 4
8. For Every clock pulse send four bits as Multiplexer outputs
9. For every two clock pulses, eight bits from the Multiplexer are sent to the MicroController as one character.
10. From steps 8 and 9, the characters " EMPYREAL" are taken as name, "KCT" as password and "2001" as identification code
11. Go to step 4

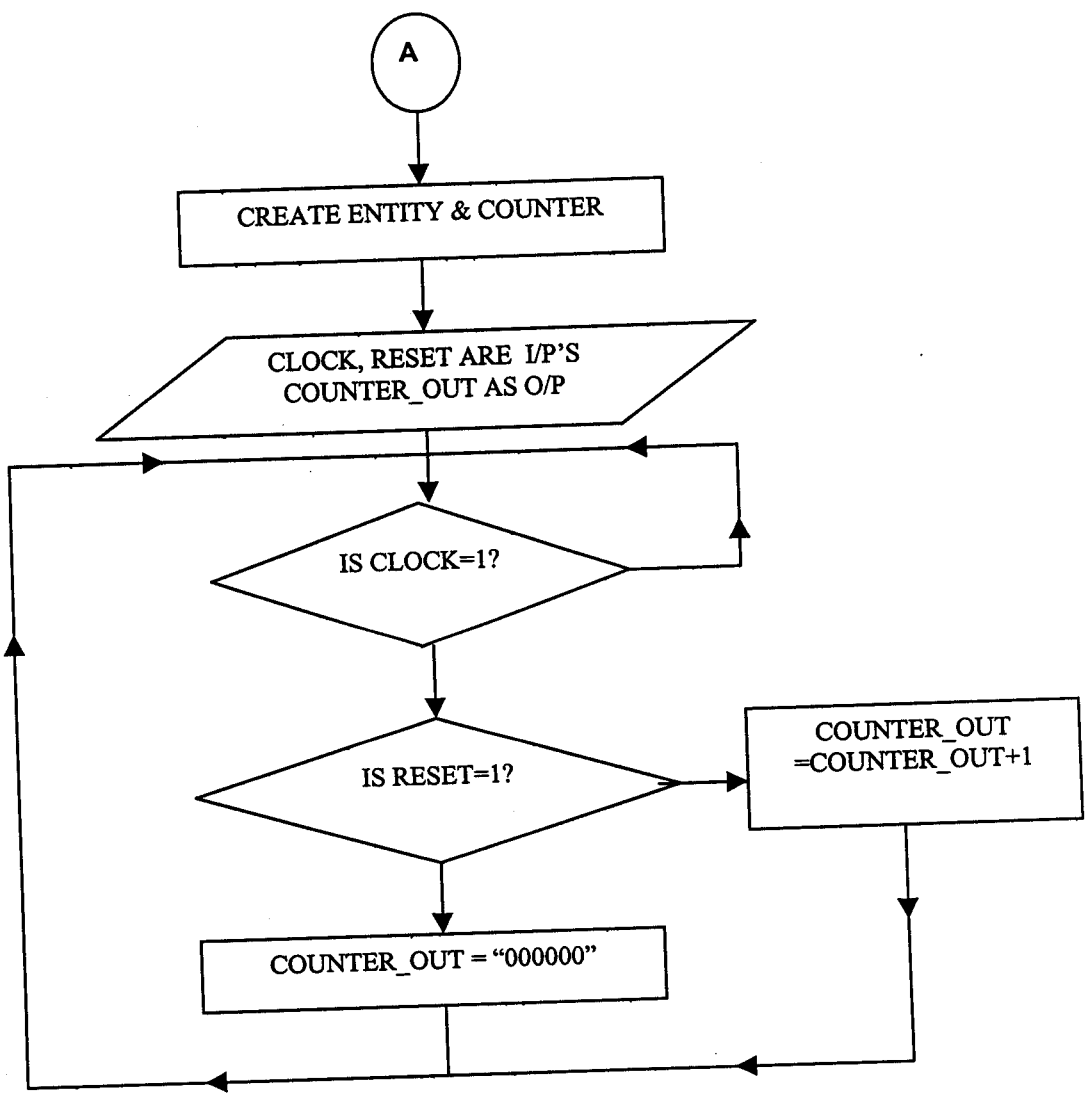
TOP

1. Create the structure TOP with clock clk, reset, an array data_in of size 3 as inputs and an array data_out of size 4 as output
2. Declare the component counter as in the algorithm for the counter.
3. Declare the component Multiplexer as in the algorithm for multiplexer.
4. Include the above two components in the structure TOP.
5. Declare an array of signals of size 6.
6. Map the ports clock, reset, of structure TOP with that of the component counter respectively.
7. Map the ports data_in, data_out of structure Top with that of the component Multiplexer
8. Stop the program.

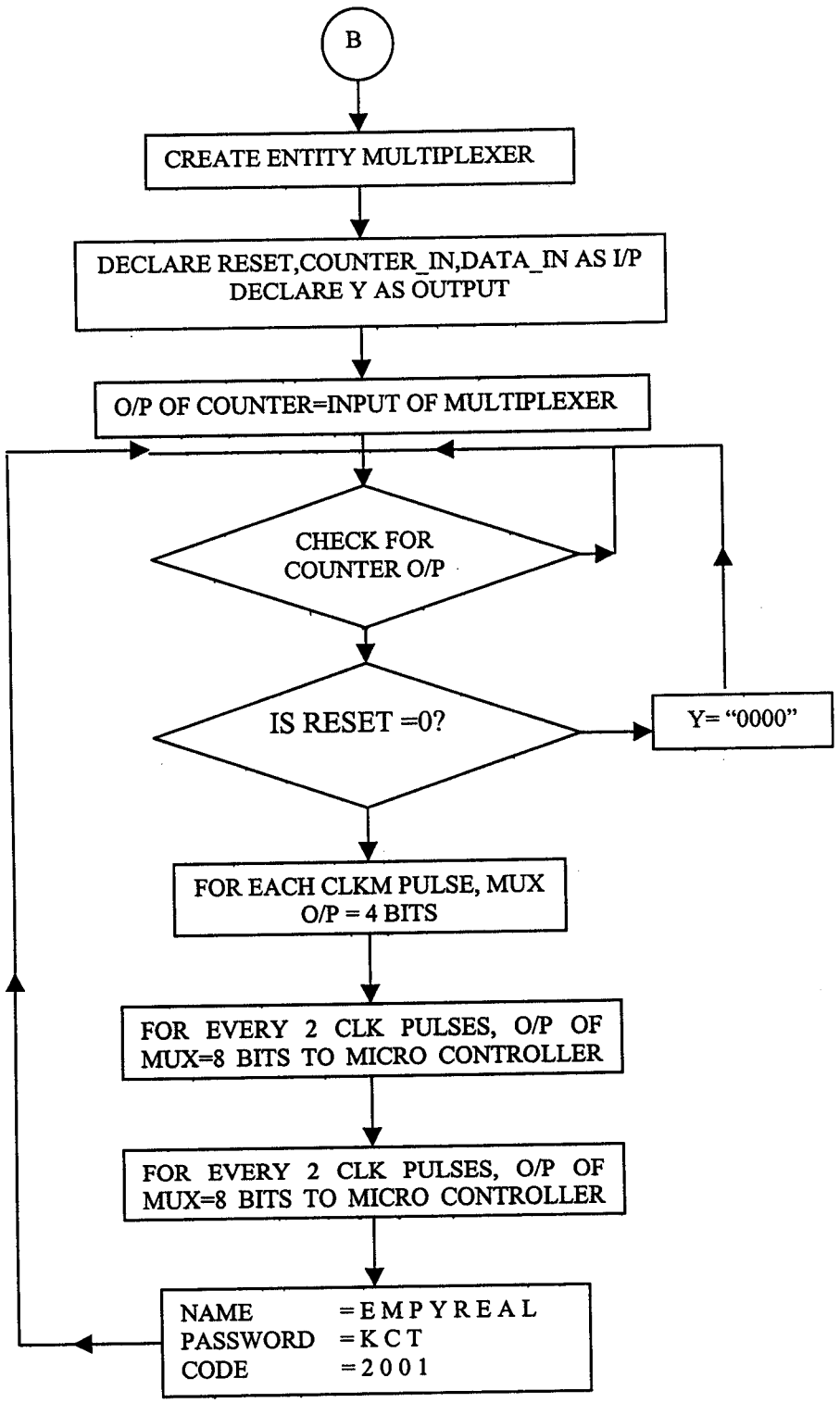
6.1.2 Program flow



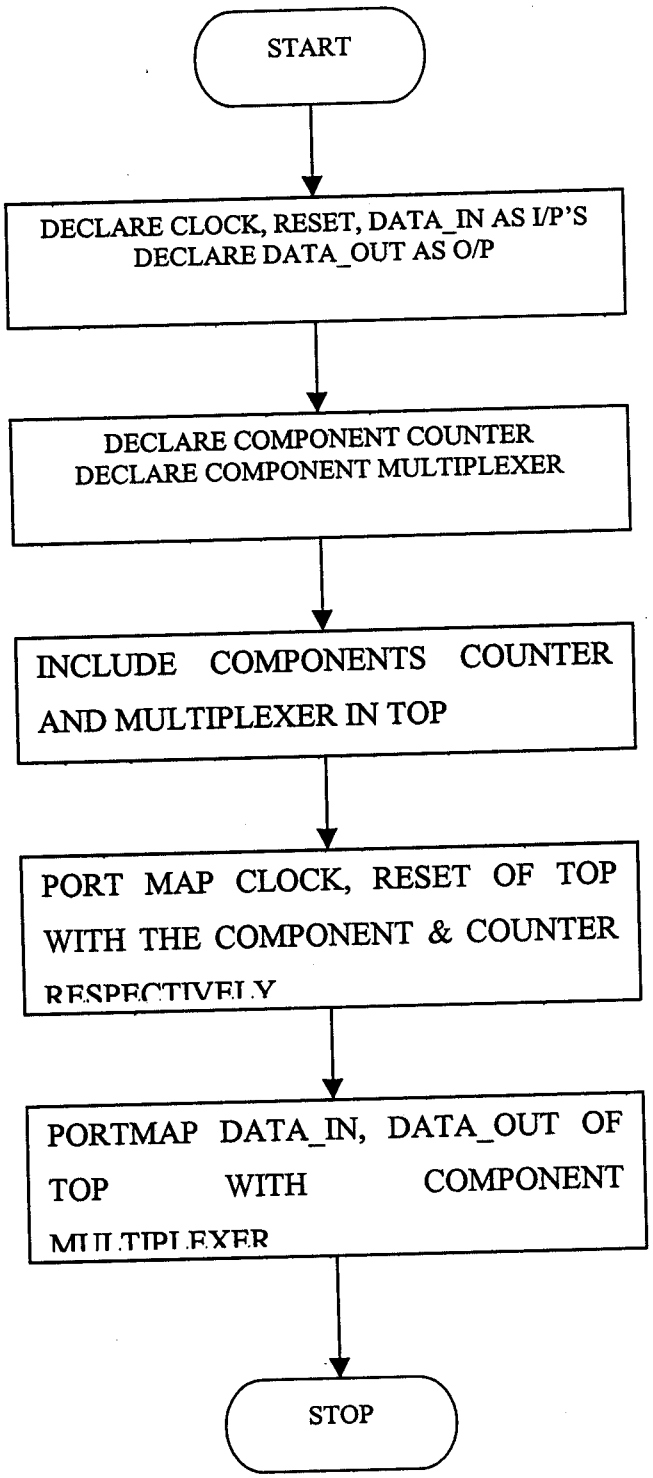
Counter:



Multiplexer



TOP


START
DECLARE CLOCK, RESET, DATA_IN AS I/P'S
DECLARE DATA_OUT AS O/P
DECLARE COMPONENT COUNTER
DECLARE COMPONENT MULTIPLEXER
INCLUDE COMPONENTS COUNTER AND MULTIPLEXER IN TOP
PORT MAP CLOCK, RESET OF TOP WITH THE COMPONENT & COUNTER RESPECTIVELY
PORTMAP DATA_IN, DATA_OUT OF TOP WITH COMPONENT MULTIPLEXER
STOP

6.1.3 Programming the GAL 22V10B using VHDL

```
library ieee;
use ieee.std_logic_1164.all;
USE work.STD_ARITH.all;

ENTITY counter IS
        PORT        (clk, reset: IN STD_LOGIC;
                        counter_out:    BUFFER    STD_LOGIC_VECTOR(5
downto 0));
END  counter;

ARCHITECTURE count OF counter IS

BEGIN

proc1:        PROCESS
        BEGIN
        WAIT UNTIL clk = '1';
                IF reset = '1' THEN
                        counter_out <= "000000";
                ELSE
                        counter_out <= counter_out + 1;
                END IF;
        END PROCESS;
END count;

library ieee;
use ieee.std_logic_1164.all;
use work.numeric_std.all;

entity multiplexor is port(
                        reset     : in std_logic;
                        counter_in :  in std_logic_vector(5 downto 0);
                        data_in    :  in std_logic_vector(3 downto 0);
                         y         :  buffer std_logic_vector(3 downto 0));
end multiplexor;

architecture mux of multiplexor is
begin
a : process(counter_in)
begin
if reset='0' then

if counter_in="000001" then          --dev 000
                y <= "0101"; --5
        elsif counter_in="000010" then
                y <= "0100"; --4,E
```

```
elsif counter_in="000011" then
        y <= "1101";  --d
elsif counter_in="000100" then
        y <= "0100";  -- 4,m
elsif counter_in="000101" then
 y <= "0000";--0
elsif counter_in="000110" then
        y <= "0101"; --5 p
elsif counter_in="000111" then
        y <= "1001"; --9
elsif counter_in="001000" then        --dev 001
         y <= "0101"; --4 y
elsif counter_in="001001" then
           y <= "0010";--2
elsif counter_in="001010" then
         y <= "0101";--5 R
elsif counter_in="001011" then
        y <= "0101";--5
elsif counter_in="001100" then
        y <= "0100"; --4 e
elsif counter_in="001101" then
        y <= "0001";--1
elsif counter_in="001110" then
        y <= "0100"; --4a
elsif counter_in="001111" then
        y <= "1100"; --c
elsif counter_in="010000" then        --dev 010
        y <= "0100";          --4 l
elsif counter_in="010001" then
        y <= "1011"; --b
elsif counter_in="010010" then
        y <= "0100";-- 4 K
elsif counter_in="010011" then
        y <= "0011"; --3
elsif counter_in="010100" then
        y <= "0100"; --4c
elsif counter_in="010101" then
        y <= "0100"; --4
elsif counter_in="010110" then
        y <= "0101"; --5 t
elsif counter_in="010111" then
        y <= "0000"; --1
elsif counter_in="011000" then        --dev 011
        y <= "0010";          --4 a
elsif counter_in="011001" then
        y <= "0000"; --8
elsif counter_in="011010" then
        y <= "0010";--4 h
elsif counter_in="011011" then
```

```
        y <= "0000";--1
   elsif counter_in="011100" then
        y <= "0010";--4  a
   elsif counter_in="011101" then
        y <= "0000";
   elsif counter_in="011110" then      --dev 100
        y <= "0010";
   elsif counter_in="011111" then
        y <= "0000";
   elsif counter_in="100000" then
        y <= "0010";
   elsif counter_in="100001" then
        y <= "0010"; --2
   elsif counter_in="100010" then
        y <= "0011";--3
   elsif counter_in="100011" then
        y <= "0000";--0
   elsif counter_in="100100" then
        y <= "0011";--3
   elsif counter_in="100101" then
        y <= "0000";--0
   elsif counter_in="100110" then
        y <= "0011";--3
   elsif counter_in="100111" then
        y <= "0001";--1
   elsif counter_in="101000" then
        y <= "0011";--3
   elsif counter_in="101001" then
        y <= "0000";
   elsif counter_in="101010" then
        y <= "0010";
   elsif counter_in="101011" then
        y <= "0000";
   elsif counter_in="101100" then
        y <= "0010";
   elsif counter_in="101101" then
         y <= "0000";
   elsif counter_in="101110" then
         y <= "0010";
   elsif counter_in="101111" then
         y <= "0000";
   elsif counter_in="110000" then
         y <= "0010";
   else
         y <= "0000";
   end if;
else
  y <= "0000";
end if;
```

```
end process a;
end;

library ieee;
use ieee.std_logic_1164.all;
use work.numeric_std.all;
use work.std_arith.all;

entity top is port(
     clk,reset  : in std_logic;
     data_in    : in std_logic_vector(3 downto 0);
     data_out   : buffer std_logic_vector(3 downto 0));
end top;

architecture top_level of top is

component  counter
   port(clk,reset  :       in std_logic;
        counter_out:       buffer std_logic_vector(5 downto 0));
end component;


component multiplexor
   port(reset      : in std_logic;
        counter_in : in std_logic_vector(5 downto 0);
        data_in    : in std_logic_vector(3 downto 0);
        y          : buffer std_logic_vector(3 downto 0));
end component;

for u2:multiplexor use entity work.multiplexor(mux);
for u3:counter use entity work.counter(count);

signal count1:std_logic_vector(5 downto 0);
begin

    u3 : counter     port map(clk,reset,count1);
    u4 : multiplexor port map(reset,count1,data_in,data_out);
end;
```

6.1.4. Program to combine the logic gates and decoder

```
Library ieee ;
   Use ieee.std_logic_1164.ALL;
   use work.numeric_std.ALL;
   use work.std_arith.ALL;

Entity decode is
port(a,b,c,a1,a2,a3,a4,a5,a6,ni1,ni2,ni3: in std_logic;
y0,y1,y2,y3,x,y,z,n1,n2,n3: out std_logic);
end decode;

Architecture dec of decode is
 begin
 process(a,b,c)
  begin
        y3<=not((not a) and b and c);
        y2<=not((not a) and b and (not c));
        y1<=not((not a) and (not b) and c);
        y0<=not((not a) and (not b) and (not c));
    x<=a1 and a2;
    y<=a3 and a4;
    z<=a5 and a6;
    n1<=not ni1;
    n2<=not ni2;
    n3<=not ni3;
   end process;
end dec;
```

6.2 PROGRAMMING THE AT89c51 IN ASSEMBLY LANGUAGE

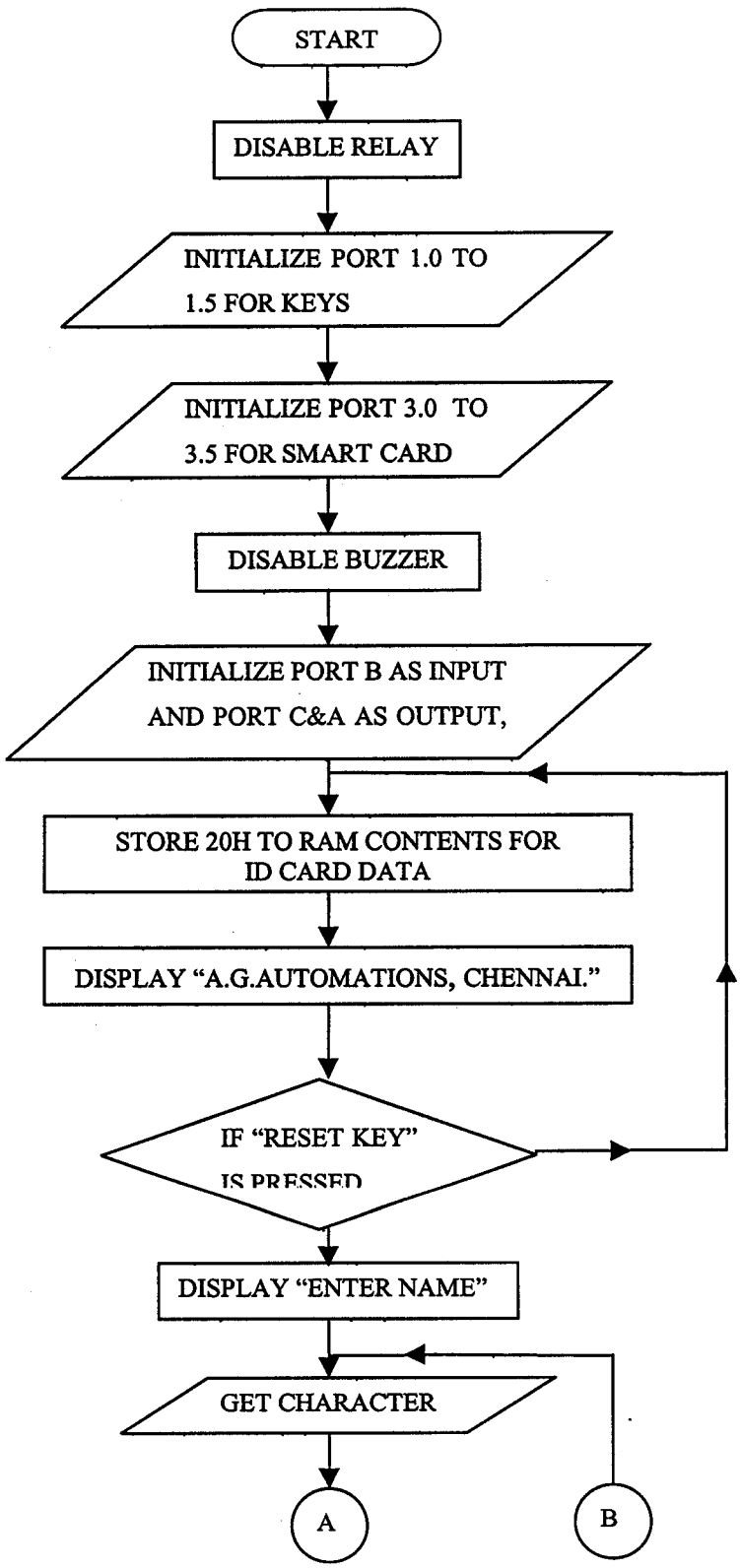
Programming a MicroController is very efficient when compared to a microprocessor or any high level language when it comes to hardware control. Further, a MicroController encapsulates internal RAM, on-chip clock oscillator, inbuilt EPROM etc., which makes programming much easier.

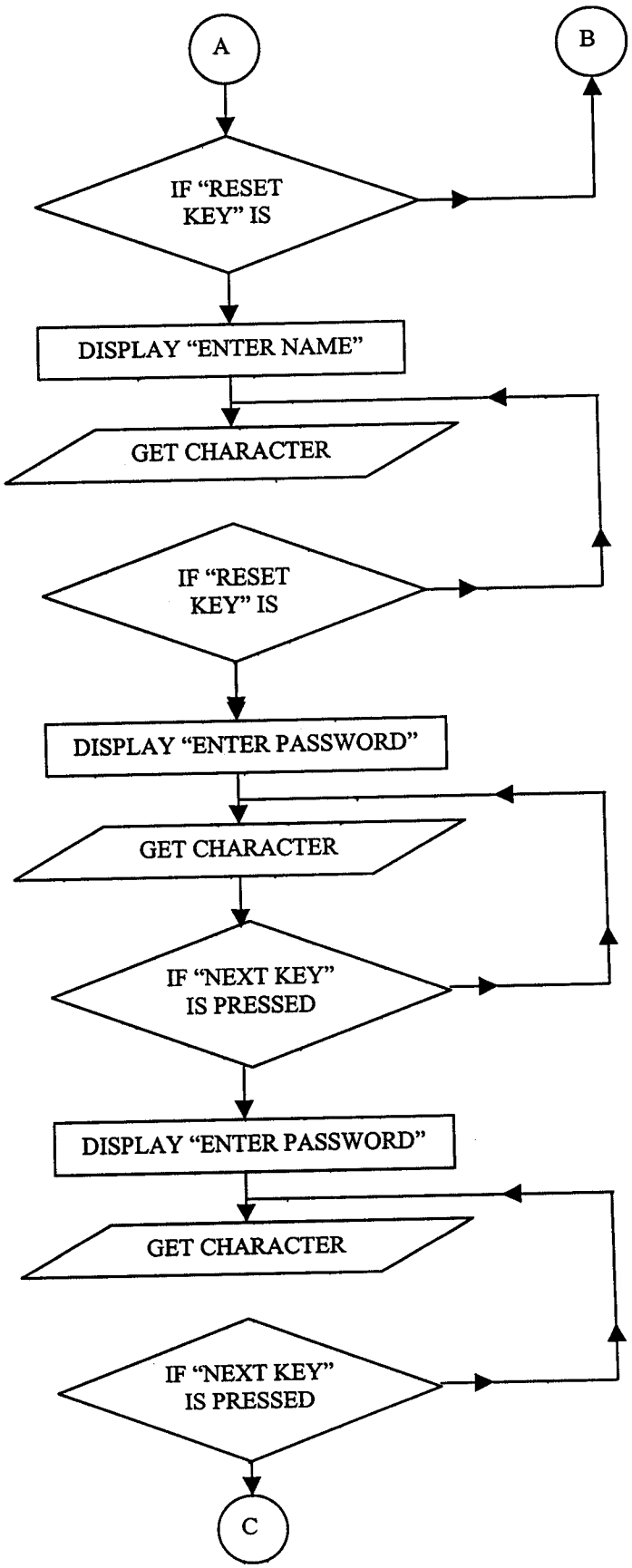
6.2.1 Programming algorithm

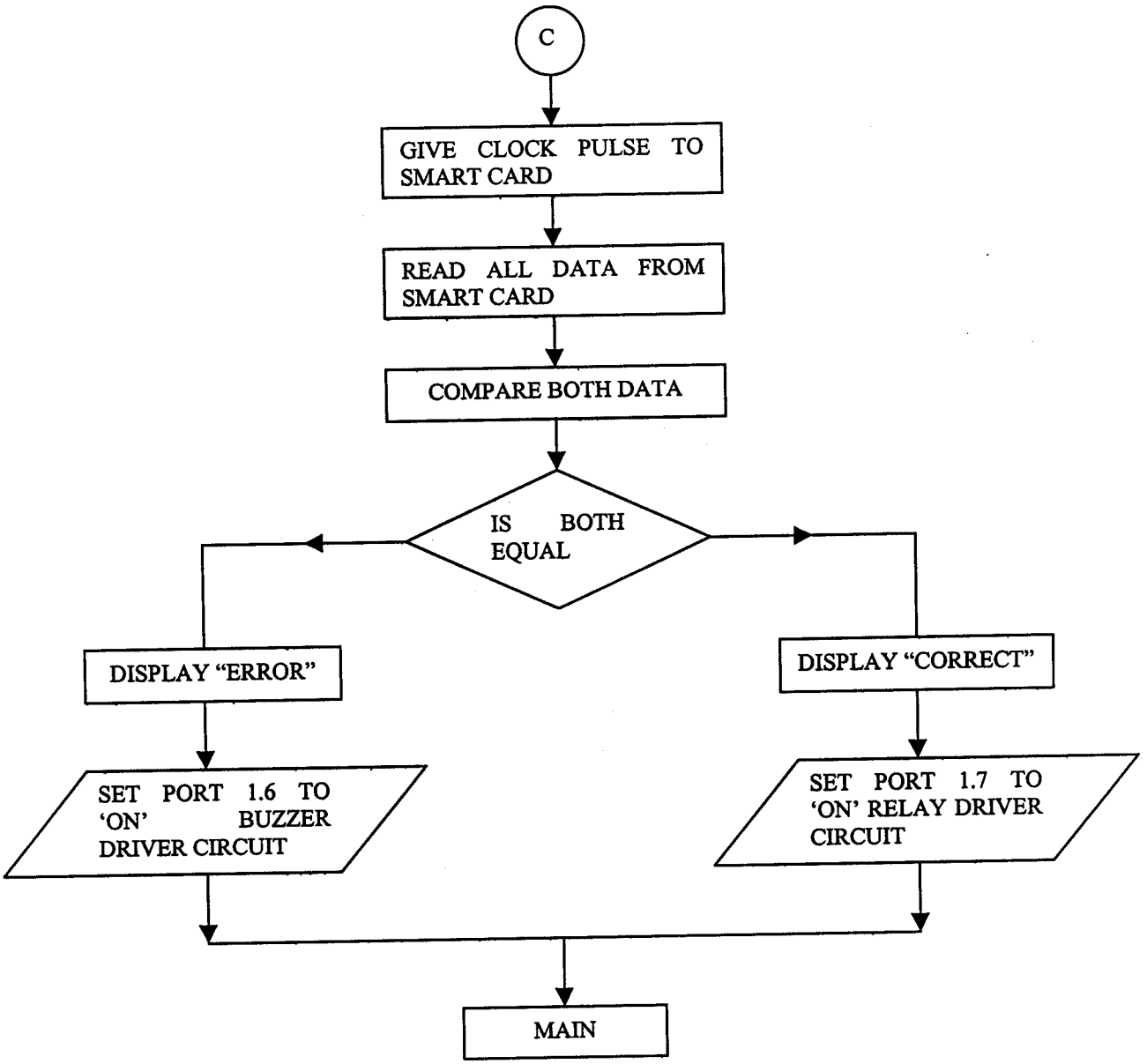
1. Initialize the program counter to the start of program memory
2. Clear P1.7 to disable the relay
3. Set P1.0 to 1.5 to initialize the keys in the keypad
4. A push-button reset key is connected to P1.8 to RESET the RAM contents and that of the registers
5. Set P3.0 to 3.3 for smart card data and P3.4 to 3.5 for clock and reset to the smart card
6. Clear P1.6 to disable buzzer
7. Initialize Port B as input and port C & A as output in mode 0 operation for the control word 82H
8. Store 20H(space) to RAM contents for smart card data
9. Display "A.G.Automations"
10. Check whether the reset key is pressed. If yes, go to step 11 else step 8
11. Check whether P1.5 (ENT) key is pressed. If yes, go to step 12 else step 8
12. Display "Enter Name"
13. Get character by sending suitable control words to the LCD CGRAM
14. Check whether NEXT key is pressed. If yes, go to step 15 else step 13
15. Display "Enter Password"
16. Get character by sending suitable control words to the LCD CGRAM
17. Check whether NEXT key is pressed. If yes, go to step 18 else step 16
18. Display "Enter Code"
19. Get character by sending suitable control words to the LCD CGRAM
20. Check whether NEXT key is pressed. If yes, go to step 21 else step 19
21. Give clock pulse to the smart card
22. Read the multiplexer outputs from the smart card

23. Compare the data entered and that read from the card
24. If found equal, display "CORRECT" and drive the relay and return to step 1
25. If found unequal, display "ERROR" and drive the buzzer and display the multiplexer outputs (Verification Purpose)

6.2.2 Program flow




A
B
IF "RESET KEY" IS
DISPLAY "ENTER NAME"
GET CHARACTER
IF "RESET KEY" IS
DISPLAY "ENTER PASSWORD"
GET CHARACTER
IF "NEXT KEY" IS PRESSED
DISPLAY "ENTER PASSWORD"
GET CHARACTER
IF "NEXT KEY" IS PRESSED
C


C
GIVE CLOCK PULSE TO SMART CARD
READ ALL DATA FROM SMART CARD
COMPARE BOTH DATA
IS BOTH EQUAL
DISPLAY "ERROR"
DISPLAY "CORRECT"
SET PORT 1.6 TO 'ON' BUZZER DRIVER CIRCUIT
SET PORT 1.7 TO 'ON' RELAY DRIVER CIRCUIT
MAIN

6.2.3 Program code for AT89C51

```
                ;---------------------------------------
                ;         ORIGIN OF THE PROGRAM
                ;---------------------------------------
0000                            org     0000h
0000    c2 97                   clr     p1.7   ; RELAY OFF
0002                    main:
0002    78 40           initial: mov    r0,#40h
0004    74 20                   mov     a,#20h
0006    f6               init:  mov     @r0,a
0007    08                      inc     r0
0008    b8 7f fb                cjne    r0,#7fh,init
000b    90 21 00                mov     dptr,#2100h
000e    f0               init1: movx    @dptr,a
000f    a3                      inc     dptr
0010    e5 82                   mov     a,dpl
0012    b4 2f f9                cjne    a,#2fh,init1

0015    d2 90                   setb    p1.0
0017    d2 91                   setb    p1.1
0019    d2 92                   setb    p1.2
001b    d2 93                   setb    p1.3
001d    d2 94                   setb    p1.4
001f    d2 95                   setb    p1.5

0021    d2 b0                   setb    p3.0
0023    d2 b1                   setb    p3.1
0025    d2 b2                   setb    p3.2
0027    d2 b3                   setb    p3.3
0029    c2 b4                   clr     p3.4    ;clock
002b    d2 b5                   setb    p3.5
002d    c2 96                   clr     p1.6

    ;---------------------------------------
    ;       PORT B is INPUT AND PORT C & A
    ;        IS OUTPUT IN MODE ZERO OPERATION
    ;             FOR CONTROL WORD 82H
    ;---------------------------------------
002f    74 82                   mov     a,#82h
0031    90 40 03                mov     dptr,#4003h
0034    f0                      movx    @dptr,a
0035    74 00                   mov     a,#00h
0037    90 40 02                mov     dptr,#4002h ;init port B
003a    f0                      movx    @dptr,a
003b    12 0a 6b                lcall   delay
003e    90 40 00                mov     dptr,#4000h

0041    74 38                   mov     a,#38h
```

```
                                        ;spec.function
                                        ;set for bit len. 8

   0043   f0                  movx    @dptr,a
   0044   12 0a 40            lcall   clk_inst
   0047   74 08               mov     a,#08h   ;display off
   0049   90 40 00            mov     dptr,#4000h
   004c   f0                  movx    @dptr,a
   004d   12 0a 40            lcall   clk_inst
   0050   74 01               mov     a,#01h    ;clear
display
   0052   90 40 00            mov     dptr,#4000h
   0055   f0                  movx    @dptr,a
   0056   12 0a 40            lcall   clk_inst
   0059   74 06               mov     a,#06h
;increment the cursor
   005b   90 40 00            mov     dptr,#4000h
   005e   f0                  movx    @dptr,a
   005f   12 0a 40            lcall   clk_inst
   0062   90 40 00            mov     dptr,#4000h
   0065   74 0c               mov     a,#0ch    ;display on
   0067   f0                  movx    @dptr,a
   0068   12 0a 40            lcall   clk_inst
   006b   90 40 00            mov     dptr,#4000h
   006e   74 45               mov     a,#45h    ;CG RAM
address
   0070   f0                  movx    @dptr,a
   0071   12 0a 40            lcall   clk_inst
   0074   74 01               mov     a,#01h    ;To write
    in data register
   0076   90 40 00            mov     dptr,#4000h
   0079   f0                  movx    @dptr,a
   007a   12 0a 40            lcall   clk_inst
   007d   74 80               mov     a,#80h      ;first
line of display
   007f   90 40 00            mov     dptr,#4000h
   0082   f0                  movx    @dptr,a
   0083   12 0a 40            lcall   clk_inst
   0086   90 40 00            mov     dptr,#4000h
   0089   74 0f               mov     a,#0fh      ;on for
display,cursor & blink
   008b   f0                  movx    @dptr,a
   008c   12 0a 40            lcall    clk_inst
   008f   75 34 00            mov     34h,#00h
   0092   7c 00               mov     r4,#00h
   0094   74 80         first: mov    a,#80h      ;first
line
   0096   90 40 00            mov     dptr,#4000h
   0099   f0                  movx    @dptr,a
   009a   12 0a 40            lcall   clk_inst
```

```
   009d   7e 41            mov     r6,#"A"     ;blank
space
   009f   12 0a 36         lcall   dat_writ
   00a2   7e 2e            mov     r6,#"."
   00a4   12 0a 36         lcall   dat_writ
   00a7   7e 47            mov     r6,#"G"
   00a9   12 0a 36         lcall   dat_writ
   00ac   7e 2e            mov     r6,#"."
   00ae   12 0a 36         lcall   dat_writ
   00b1   7e 41            mov     r6,#"A"
   00b3   12 0a 36         lcall   dat_writ
   00b6   7e 75            mov     r6,#"u"
   00b8   12 0a 36         lcall   dat_writ
   00bb   7e 74            mov     r6,#"t"
   00bd   12 0a 36         lcall   dat_writ
   00c0   7e 6f            mov     r6,#"o"
   00c2   12 0a 36         lcall   dat_writ
   00c5   7e 6d            mov     r6,#"m"
   00c7   12 0a 36         lcall   dat_writ
   00ca   7e 61            mov     r6,#"a"
   00cc   12 0a 36         lcall   dat_writ
   00cf   7e 74            mov     r6,#"t"
   00d1   12 0a 36         lcall   dat_writ
   00d4   7e 69            mov     r6,#"i"
   00d6   12 0a 36         lcall   dat_writ
   00d9   7e 6f            mov     r6,#"o"
   00db   12 0a 36         lcall   dat_writ
   00de   7e 6e            mov     r6,#"n"
   00e0   12 0a 36         lcall   dat_writ
   00e3   7e 73            mov     r6,#"s"
   00e5   12 0a 36         lcall   dat_writ
   00e8   7e 20            mov     r6,#" "
   00ea   12 0a 36         lcall   dat_writ
   00ed   74 c0            mov     a,#0c0h     ;first
line
   00ef   90 40 00         mov     dptr,#4000h
   00f2   f0               movx    @dptr,a
   00f3   12 0a 40         lcall   clk_inst
   00f6   7e 20            mov     r6,#" "     ;blank
space
   00f8   12 0a 36         lcall   dat_writ
   00fb   7e 20            mov     r6,#" "
   00fd   12 0a 36         lcall   dat_writ
   0100   7e 20            mov     r6,#" "
   0102   12 0a 36         lcall   dat_writ
   0105   7e 20            mov     r6,#" "
   0107   12 0a 36         lcall   dat_writ
   010a   7e 43            mov     r6,#"C"
   010c   12 0a 36         lcall   dat_writ
   010f   7e 68            mov     r6,#"h"
   0111   12 0a 36         lcall   dat_writ
```

```
   0114   7e 65              mov     r6,#"e"
   0116   12 0a 36           lcall   dat_writ
   0119   7e 6e              mov     r6,#"n"
   011b   12 0a 36           lcall   dat_writ
   011e   7e 6e              mov     r6,#"n"
   0120   12 0a 36           lcall   dat_writ
   0123   7e 61              mov     r6,#"a"
   0125   12 0a 36           lcall   dat_writ
   0128   7e 69              mov     r6,#"i"
   012a   12 0a 36           lcall   dat_writ
   012d   7e 20              mov     r6,#" "
   012f   12 0a 36           lcall   dat_writ
   0132   7e 20              mov     r6,#" "
   0134   12 0a 36           lcall   dat_writ
   0137   7e 20              mov     r6,#" "
   0139   12 0a 36           lcall   dat_writ
   013c   7e 20              mov     r6,#" "
   013e   12 0a 36           lcall   dat_writ
   0141   7e 20              mov     r6,#" "
   0143   12 0a 36           lcall   dat_writ
   0146   20 95 fd  trigger: jb      p1.5,trigger
;password key
   0149   30 95 fd  trig_1:  jnb     p1.5,trig_1
   014c   74 80              mov     a,#80h        ;first
line
   014e   90 40 00           mov     dptr,#4000h
   0151   f0                 movx    @dptr,a
   0152   12 0a 40           lcall   clk_inst
   0155   7e 45              mov     r6,#"E"       ;blank
space
   0157   12 0a 36           lcall   dat_writ
   015a   7e 6e              mov     r6,#"n"
   015c   12 0a 36           lcall   dat_writ
   015f   7e 74              mov     r6,#"t"
   0161   12 0a 36           lcall   dat_writ
   0164   7e 65              mov     r6,#"e"
   0166   12 0a 36           lcall   dat_writ
   0169   7e 72              mov     r6,#"r"
   016b   12 0a 36           lcall   dat_writ
   016e   7e 20              mov     r6,#" "
   0170   12 0a 36           lcall   dat_writ
   0173   7e 4e              mov     r6,#"N"
   0175   12 0a 36           lcall   dat_writ
   0178   7e 61              mov     r6,#"a"
   017a   12 0a 36           lcall   dat_writ
   017d   7e 6d              mov     r6,#"m"
   017f   12 0a 36           lcall   dat_writ
   0182   7e 65              mov     r6,#"e"
   0184   12 0a 36           lcall   dat_writ
   0187   7e 20              mov     r6,#" "
   0189   12 0a 36           lcall   dat_writ
```

```
   018c   7e 20              mov     r6,#" "
   018e   12 0a 36           lcall   dat_writ
   0191   7e 20              mov     r6,#" "
   0193   12 0a 36           lcall   dat_writ
   0196   7e 20              mov     r6,#" "
   0198   12 0a 36           lcall   dat_writ
   019b   7e 20              mov     r6,#" "
   019d   12 0a 36           lcall   dat_writ
   01a0   7e 20              mov     r6,#" "
   01a2   12 0a 36           lcall   dat_writ
   01a5   74 c0              mov     a,#0c0h         ;first
line
   01a7   90 40 00           mov     dptr,#4000h
   01aa   f0                 movx    @dptr,a
   01ab   12 0a 40           lcall   clk_inst
   01ae   7e 20              mov     r6,#" "         ;blank
space
   01b0   12 0a 36           lcall   dat_writ
   01b3   7e 20              mov     r6,#" "
   01b5   12 0a 36           lcall   dat_writ
   01b8   7e 20              mov     r6,#" "
   01ba   12 0a 36           lcall   dat_writ
   01bd   7e 20              mov     r6,#" "
   01bf   12 0a 36           lcall   dat_writ
   01c2   7e 20              mov     r6,#" "
   01c4   12 0a 36           lcall   dat_writ
   01c7   7e 20              mov     r6,#" "
   01c9   12 0a 36           lcall   dat_writ
   01cc   7e 20              mov     r6,#" "
   01ce   12 0a 36           lcall   dat_writ
   01d1   7e 20              mov     r6,#" "
   01d3   12 0a 36           lcall   dat_writ
   01d6   7e 20              mov     r6,#" "
   01d8   12 0a 36           lcall   dat_writ
   01db   7e 20              mov     r6,#" "
   01dd   12 0a 36           lcall   dat_writ
   01e0   7e 20              mov     r6,#" "
   01e2   12 0a 36           lcall   dat_writ
   01e5   7e 20              mov     r6,#" "
   01e7   12 0a 36           lcall   dat_writ
   01ea   7e 20              mov     r6,#" "
   01ec   12 0a 36           lcall   dat_writ
   01ef   7e 20              mov     r6,#" "
   01f1   12 0a 36           lcall   dat_writ
   01f4   7e 20              mov     r6,#" "
   01f6   12 0a 36           lcall   dat_writ
   01f9   7e 20              mov     r6,#" "
   01fb   12 0a 36           lcall   dat_writ
   01fe   20 95 fd  tri   :  jb      p1.5,tri      ;password
key
   0201   30 95 fd  tri_1:   jnb     p1.5,tri_1
```

```
   0204   74 80            mov     a,#80h      ;first
line of display
   0206   90 40 00          mov     dptr,#4000h
   0209   f0                movx    @dptr,a
   020a   12 0a 40          lcall   clk_inst
   020d   90 40 00          mov     dptr,#4000h
   0210   74 0f             mov     a,#0fh      ;on for
display,cursor  & blink
   0212   f0                movx    @dptr,a
   0213   12 0a 40          lcall    clk_inst
   0216   7e 20             mov     r6,#20h
;blank space
   0218   12 0a 36          lcall   dat_writ
   021b   7e 20             mov     r6,#20h
;blank space
   021d   12 0a 36          lcall   dat_writ
   0220   7e 20             mov     r6,#20h
;blank space
   0222   12 0a 36          lcall   dat_writ
   0225   7e 20             mov     r6,#20h
;blank space
   0227   12 0a 36          lcall   dat_writ
   022a   7e 20             mov     r6,#20h
;blank space
   022c   12 0a 36          lcall   dat_writ
   022f   7e 20             mov     r6,#20h
;blank space
   0231   12 0a 36          lcall   dat_writ
   0234   7e 20             mov     r6,#20h
;blank space
   0236   12 0a 36          lcall   dat_writ
   0239   7e 20             mov     r6,#20h
;blank space
   023b   12 0a 36          lcall   dat_writ
   023e   7e 20             mov     r6,#20h
;blank space
   0240   12 0a 36          lcall   dat_writ
   0243   7e 20             mov     r6,#20h
;blank space
   0245   12 0a 36          lcall   dat_writ
   0248   7e 20             mov     r6,#20h
;blank space
   024a   12 0a 36          lcall   dat_writ
   024d   7e 20             mov     r6,#20h
;blank space
   024f   12 0a 36          lcall   dat_writ
   0252   7e 20             mov     r6,#20h
;blank space
   0254   12 0a 36          lcall   dat_writ
```

```
   0257   7e 20              mov      r6,#20h
;blank space
   0259   12 0a 36           lcall    dat_writ
   025c   7e 20              mov      r6,#20h      ;blank
space
   025e   12 0a 36           lcall    dat_writ
   0261   7e 20              mov      r6,#20h      ;blank
space
   0263   12 0a 36           lcall    dat_writ
   0266   74 80              mov      a,#80h       ;first
line of display
   0268   90 40 00           mov      dptr,#4000h
   026b   f0                 movx     @dptr,a
   026c   12 0a 40           lcall    clk_inst
   026f   90 40 00           mov      dptr,#4000h
   0272   74 0f              mov      a,#0fh       ;on for
display,cursor & blink
   0274   f0                 movx     @dptr,a
   0275   12 0a 40           lcall     clk_inst
   0278   78 40              mov      r0,#40h
   027a   30 90 0f  start2:  jnb      p1.0,inc
;incrementing key
   027d   30 91 2b           jnb      p1.1,dec
;decrementing key
   0280   30 92 47           jnb      p1.2,rig_sif;right
shift key
   0283   30 93 5e           jnb      p1.3,lef_sif;left
shift key
   0286   30 94 75           jnb      p1.4,code
;comparing key
   0289   02 02 7a           ljmp     start2
   028c   0e         inc:    inc      r6
   028d   74 06               mov      a,#06h
;incrementing the cursor
   028f   90 40 00            mov      dptr,#4000h
   0292   f0                  movx     @dptr,a
   0293   12 0a 40            lcall     clk_inst
   0296   12 0a 36            lcall    dat_writ
   0299   74 10               mov      a,#10h       ;shift
cursor position to the left
   029b   90 40 00            mov      dptr,#4000h
   029e   f0                  movx     @dptr,a
   029f   12 0a 40            lcall     clk_inst
   02a2   12 0a 74            lcall    delay1
   02a5   12 0a 74            lcall    delay1
   02a8   02 02 7a            ljmp     start2
   02ab   1e         dec:     dec      r6
   02ac   74 06               mov      a,#06h
;incrementing the cursor
   02ae   90 40 00            mov      dptr,#4000h
   02b1   f0                  movx     @dptr,a
```

```
   02b2   12 0a 40            lcall    clk_inst
   02b5   12 0a 36            lcall    dat_writ
   02b8   74 10               mov      a,#10h       ;shift
cursor position to left
   02ba   90 40 00            mov      dptr,#4000h
   02bd   f0                  movx     @dptr,a
   02be   12 0a 40            lcall    clk_inst
   02c1   12 0a 74            lcall    delay1
   02c4   12 0a 74            lcall    delay1
   02c7   02 02 7a            ljmp     start2
   02ca   74 17      rig sif: mov      a,#17h
   02cc   90 40 00            mov      dptr,#4000h
   02cf   f0                  movx     @dptr,a
   02d0   12 0a 40            lcall    clk_inst     ;to
write in ins reg
   02d3   12 0a 74            lcall    delay1
   02d6   12 0a 74            lcall    delay1
   02d9   7e 40               mov      r6,#40h
   02db   85 30 31            mov      31h,30h
   02de   05 30               inc      30h
;increment 30h register
   02e0   08                  inc      r0
   02e1   02 02 7a            ljmp     start2
   02e4   74 13     lef_sif:  mov      a,#13h
   02e6   90 40 00            mov      dptr,#4000h
   02e9   f0                  movx     @dptr,a
   02ea   12 0a 40            lcall    clk_inst     ;to
write in ins reg
   02ed   12 0a 74            lcall    delay1
   02f0   12 0a 74            lcall    delay1
   02f3   7e 40               mov      r6,#40h
   02f5   85 30 31            mov      31h,30h
   02f8   15 30               dec      30h
;decrement 30h register
   02fa   18                  dec      r0
   02fb   02 02 7a            ljmp     start2
   02fe   74 80       code:   mov      a,#80h  ;first line
   0300   90 40 00            mov      dptr,#4000h
   0303   f0                  movx     @dptr,a
   0304   12 0a 40            lcall    clk_inst
   0307   7e 20               mov      r6,#" " ;blank space
   0309   12 0a 36            lcall    dat_writ
   030c   7e 20               mov      r6,#" "
   030e   12 0a 36            lcall    dat_writ
   0311   7e 45               mov      r6,#"E"
   0313   12 0a 36            lcall    dat_writ
   0316   7e 6e               mov      r6,#"n"
   0318   12 0a 36            lcall    dat_writ
   031b   7e 74               mov      r6,#"t"
   031d   12 0a 36            lcall    dat_writ
   0320   7e 65               mov      r6,#"e"
```

```
   0322   12 0a 36              lcall    dat_writ
   0325   7e 72                 mov      r6,#"r"
   0327   12 0a 36              lcall    dat_writ
   032a   7e 20                 mov      r6,#" "
   032c   12 0a 36              lcall    dat_writ
   032f   7e 50                 mov      r6,#"P"
   0331   12 0a 36              lcall    dat_writ
   0334   7e 61                 mov      r6,#"a"
   0336   12 0a 36              lcall    dat_writ
   0339   7e 73                 mov      r6,#"s"
   033b   12 0a 36              lcall    dat_writ
   033e   7e 73                 mov      r6,#"s"
   0340   12 0a 36              lcall    dat_writ
   0343   7e 77                 mov      r6,#"w"
   0345   12 0a 36              lcall    dat_writ
   0348   7e 6f                 mov      r6,#"o"
   034a   12 0a 36              lcall    dat_writ
   034d   7e 72                 mov      r6,#"r"
   034f   12 0a 36              lcall    dat_writ
   0352   7e 64                 mov      r6,#"d"
   0354   12 0a 36              lcall    dat_writ
   0357   20 95 fd  trigge:     jb       p1.5,trigge
;password key
   035a   74 80                 mov      a,#80h        ;first
line of display
   035c   90 40 00              mov      dptr,#4000h
   035f   f0                    movx     @dptr,a
   0360   12 0a 40              lcall    clk_inst
   0363   90 40 00              mov      dptr,#4000h
   0366   74 0f                 mov      a,#0fh        ;on for
display,cursor,& blink
   0368   f0                    movx     @dptr,a
   0369   12 0a 40              lcall     clk_inst
   036c   7e 20                 mov      r6,#20h ;blank space
   036e   12 0a 36              lcall    dat_writ
   0371   7e 20                 mov      r6,#20h ;blank space
   0373   12 0a 36              lcall    dat_writ
   0376   7e 20                 mov      r6,#20h ;blank space
   0378   12 0a 36              lcall    dat_writ
   037b   7e 20                 mov      r6,#20h ;blank space
   037d   12 0a 36              lcall    dat_writ
   0380   7e 20                 mov      r6,#20h ;blank space
   0382   12 0a 36              lcall    dat_writ
   0385   7e 20                 mov      r6,#20h ;blank space
   0387   12 0a 36              lcall    dat_writ
   038a   7e 20                 mov      r6,#20h ;blank space
   038c   12 0a 36              lcall    dat_writ
   038f   7e 20                 mov      r6,#20h ;blank space
   0391   12 0a 36              lcall    dat_writ
   0394   7e 20                 mov      r6,#20h ;blank space
   0396   12 0a 36              lcall    dat_writ
```

```
   0399   7e 20              mov     r6,#20h ;blank space
   039b   12 0a 36           lcall   dat_writ
   039e   7e 20              mov     r6,#20h ;blank space
   03a0   12 0a 36           lcall   dat_writ
   03a3   7e 20              mov     r6,#20h ;blank space
   03a5   12 0a 36           lcall   dat_writ
   03a8   7e 20              mov     r6,#20h ;blank space
   03aa   12 0a 36           lcall   dat_writ
   03ad   7e 20              mov     r6,#20h ;blank space
   03af   12 0a 36           lcall   dat_writ
   03b2   7e 20              mov     r6,#20h ;blank space
   03b4   12 0a 36           lcall   dat_writ
   03b7   7e 20              mov     r6,#20h ;blank space
   03b9   12 0a 36           lcall   dat_writ
   03bc   74 80              mov     a,#80h      ;first
line of display
   03be   90 40 00           mov     dptr,#4000h
   03c1   f0                 movx    @dptr,a
   03c2   12 0a 40           lcall   clk_inst
   03c5   90 40 00           mov     dptr,#4000h
   03c8   74 0f              mov     a,#0fh      ;on for
display,cursor & blink
   03ca   f0                 movx    @dptr,a
   03cb   12 0a 40           lcall    clk_inst
   03ce   78 48              mov     r0,#48h
   03d0   30 90 0f  code2:   jnb     p1.0,inc1
;incrementing key
   03d3   30 91 2b           jnb     p1.1,dec1
;decrementing key
   03d6   30 92 47           jnb     p1.2,rig_sif1
;right shift key
   03d9   30 93 5e           jnb     p1.3,lef_sif1
;left shift key
   03dc   30 94 75           jnb     p1.4,pass
;comparing key
   03df   02 03 d0           ljmp    code2
   03e2   0e         inc1:   inc     r6
   03e3   74 06              mov     a,#06h
;incrementing the cursor
   03e5   90 40 00           mov     dptr,#4000h
   03e8   f0                 movx    @dptr,a
   03e9   12 0a 40           lcall    clk_inst
   03ec   12 0a 36           lcall   dat_writ
   03ef   74 10              mov     a,#10h
;shift cursor
 position to the left
   03f1   90 40 00           mov     dptr,#4000h
   03f4   f0                 movx    @dptr,a
   03f5   12 0a 40           lcall    clk_inst
   03f8   12 0a 74           lcall   delay1
   03fb   12 0a 74           lcall   delay1
```

```
   03fe   02 03 d0              ljmp     code2
   0401   1e          dec1:    dec      r6
   0402   74 06                mov      a,#06h
;incrementing the cursor
   0404   90 40 00              mov      dptr,#4000h
   0407   f0                    movx     @dptr,a
   0408   12 0a 40              lcall    clk_inst
   040b   12 0a 36              lcall    dat_writ
   040e   74 10                 mov      a,#10h
;shift cursor position to left
   0410   90 40 00              mov      dptr,#4000h
   0413   f0                    movx     @dptr,a
   0414   12 0a 40              lcall    clk_inst
   0417   12 0a 74              lcall    delay1
   041a   12 0a 74              lcall    delay1
   041d   02 03 d0              ljmp     code2
   0420   74 17     rig_sif1:   mov      a,#17h
   0422   90 40 00              mov      dptr,#4000h
   0425   f0                    movx     @dptr,a
   0426   12 0a 40              lcall    clk_inst        ;to
write in ins reg
   0429   12 0a 74              lcall    delay1
   042c   12 0a 74              lcall    delay1
   042f   7e 40                 mov      r6,#40h
   0431   85 30 31              mov      31h,30h
   0434   05 30                 inc      30h
;increment 30h register
   0436   08                    inc      r0
   0437   02 03 d0              ljmp     code2
   043a   74 13     lef_sif1:   mov      a,#13h
   043c   90 40 00              mov      dptr,#4000h
   043f   f0                    movx     @dptr,a
   0440   12 0a 40              lcall    clk_inst        ;to
write in ins reg
   0443   12 0a 74              lcall    delay1
   0446   12 0a 74              lcall    delay1
   0449   7e 40                 mov      r6,#40h
   044b   85 30 31              mov      31h,30h
   044e   15 30                 dec      30h
;decrement 30h register
   0450   18                    dec      r0
   0451   02 03 d0              ljmp     code2
   0454   74 80       pass:     mov      a,#80h
;first line
   0456   90 40 00              mov      dptr,#4000h
   0459   f0                    movx     @dptr,a
   045a   12 0a 40              lcall    clk_inst
   045d   7e 20                 mov      r6,#" "
;blank space
   045f   12 0a 36              lcall    dat_writ
   0462   7e 45                 mov      r6,#"E"
```

```
   0464   12 0a 36            lcall    dat_writ
   0467   7e 6e               mov      r6,#"n"
   0469   12 0a 36            lcall    dat_writ
   046c   7e 74               mov      r6,#"t"
   046e   12 0a 36            lcall    dat_writ
   0471   7e 65               mov      r6,#"e"
   0473   12 0a 36            lcall    dat_writ
   0476   7e 72               mov      r6,#"r"
   0478   12 0a 36            lcall    dat_writ
   047b   7e 20               mov      r6,#" "
   047d   12 0a 36            lcall    dat_writ
   0480   7e 43               mov      r6,#"C"
   0482   12 0a 36            lcall    dat_writ
   0485   7e 6f               mov      r6,#"o"
   0487   12 0a 36            lcall    dat_writ
   048a   7e 64               mov      r6,#"d"
   048c   12 0a 36            lcall    dat_writ
   048f   7e 65               mov      r6,#"e"
   0491   12 0a 36            lcall    dat_writ
   0494   7e 20               mov      r6,#" "
   0496   12 0a 36            lcall    dat_writ
   0499   7e 20               mov      r6,#" "
   049b   12 0a 36            lcall    dat_writ
   049e   7e 20               mov      r6,#" "
   04a0   12 0a 36            lcall    dat_writ
   04a3   7e 20               mov      r6,#" "
   04a5   12 0a 36            lcall    dat_writ
   04a8   7e 20               mov      r6,#" "
   04aa   12 0a 36            lcall    dat_writ
   04ad   20 95 fd   trigg:   jb       p1.5,trigg
;password key
   04b0   74 80               mov      a,#80h          ;first
line of display
   04b2   90 40 00            mov      dptr,#4000h
   04b5   f0                  movx     @dptr,a
   04b6   12 0a 40            lcall    clk_inst
   04b9   90 40 00            mov      dptr,#4000h
   04bc   74 0f               mov      a,#0fh          ;on
for display,cursor & blink
   04be   f0                  movx     @dptr,a
   04bf   12 0a 40            lcall     clk_inst
   04c2   7e 20               mov      r6,#20h
;blank space
   04c4   12 0a 36            lcall    dat_writ
   04c7   7e 20               mov      r6,#20h
;blank space
   04c9   12 0a 36            lcall    dat_writ
   04cc   7e 20               mov      r6,#20h
;blank space
   04ce   12 0a 36            lcall    dat_writ
```

```
   04d1   7e 20              mov      r6,#20h
;blank space
   04d3   12 0a 36           lcall    dat_writ
   04d6   7e 20              mov      r6,#20h
;blank space
   04d8   12 0a 36           lcall    dat_writ
   04db   7e 20              mov      r6,#20h
;blank space
   04dd   12 0a 36           lcall    dat_writ
   04e0   7e 20              mov      r6,#20h
;blank space
   04e2   12 0a 36           lcall    dat_writ
   04e5   7e 20              mov      r6,#20h
;blank space
   04e7   12 0a 36           lcall    dat_writ
   04ea   7e 20              mov      r6,#20h
;blank space
   04ec   12 0a 36           lcall    dat_writ
   04ef   7e 20              mov      r6,#20h
;blank space
   04f1   12 0a 36           lcall    dat_writ
   04f4   7e 20              mov      r6,#20h
;blank space
   04f6   12 0a 36           lcall    dat_writ
   04f9   7e 20              mov      r6,#20h
;blank space
   04fb   12 0a 36           lcall    dat_writ
   04fe   7e 20              mov      r6,#20h
;blank space
   0500   12 0a 36           lcall    dat_writ
   0503   7e 20              mov      r6,#20h
;blank space
   0505   12 0a 36           lcall    dat_writ
   0508   7e 20              mov      r6,#20h
;blank space
   050a   12 0a 36           lcall    dat_writ
   050d   7e 20              mov      r6,#20h
;blank space
   050f   12 0a 36           lcall    dat_writ
   0512   74 80              mov      a,#80h
;first line of display
   0514   90 40 00           mov      dptr,#4000h
   0517   f0                 movx     @dptr,a
   0518   12 0a 40           lcall    clk_inst
   051b   90 40 00           mov      dptr,#4000h
   051e   74 0f              mov      a,#0fh        ;on for
display,cursor & blink
   0520   f0                 movx     @dptr,a
   0521   12 0a 40           lcall     clk_inst
```

```
   0524   78 50                    mov     r0,#50h
   0526   30 90 0f   pass2:       jnb     p1.0,inc2
;incrementing key
   0529   30 91 2b                jnb     p1.1,dec2
;decrementing key
   052c   30 92 47                jnb     p1.2,rig_sif2
;right hift key
   052f   30 93 5e                jnb     p1.3,lef_sif2
;left shift key
   0532   30 94 75                jnb     p1.4,load
;comparing key
   0535   02 05 26                ljmp    pass2
   0538   0e            inc2:    inc     r6
   0539   74 06                   mov     a,#06h
;incrementing the cursor
   053b   90 40 00                mov     dptr,#4000h
   053e   f0                      movx    @dptr,a
   053f   12 0a 40                lcall    clk_inst
   0542   12 0a 36                lcall   dat_writ
   0545   74 10                   mov     a,#10h      ;shift
cursor position to the left
   0547   90 40 00                mov     dptr,#4000h
   054a   f0                      movx    @dptr,a
   054b   12 0a 40                lcall    clk_inst
   054e   12 0a 74                lcall   delay1
   0551   12 0a 74                lcall   delay1
   0554   02 05 26                ljmp    pass2
   0557   1e            dec2:    dec     r6
   0558   74 06                   mov     a,#06h
;incrementing the cursor
   055a   90 40 00                mov     dptr,#4000h
   055d   f0                      movx    @dptr,a
   055e   12 0a 40                lcall   clk_inst
   0561   12 0a 36                lcall   dat_writ
   0564   74 10                   mov     a,#10h      ;shift
cursor position to left
   0566   90 40 00                mov     dptr,#4000h
   0569   f0                      movx    @dptr,a
   056a   12 0a 40                lcall   clk_inst
   056d   12 0a 74                lcall   delay1
   0570   12 0a 74                lcall   delay1
   0573   02 05 26                ljmp    pass2
   0576   74 17      rig_sif2:    mov     a,#17h
   0578   90 40 00                mov     dptr,#4000h
   057b   f0                      movx    @dptr,a
   057c   12 0a 40                lcall   clk_inst    ;to
write in ins reg
   057f   12 0a 74                lcall   delay1
   0582   12 0a 74                lcall   delay1
```

```
   0585   7e 40
   0587   85 30 31
   058a   05 30
;increment 30h register
   058c   08
   058d   02 05 26
   0590   74 13     lef_sif2:
   0592   90 40 00
   0595   f0
   0596   12 0a 40
write in ins reg
   0599   12 0a 74
   059c   12 0a 74
   059f   7e 40
   05a1   85 30 31
   05a4   15 30
;decrement 30h register
   05a6   18
   05a7   02 05 26
   05aa   74 80      load :
line
   05ac   90 40 00
   05af   f0
   05b0   12 0a 40
   05b3   ae 40
;blank space
   05b5   12 0a 36
   05b8   ae 41
   05ba   12 0a 36
   05bd   ae 42
   05bf   12 0a 36
   05c2   ae 43
   05c4   12 0a 36
   05c7   ae 44
   05c9   12 0a 36
   05cc   ae 45
   05ce   12 0a 36
   05d1   ae 46
   05d3   12 0a 36
   05d6   ae 47
   05d8   12 0a 36
   05db   7e 20
   05dd   12 0a 36
   05e0   7e 20
   05e2   12 0a 36
   05e5   7e 20
   05e7   12 0a 36
   05ea   7e 20
   05ec   12 0a 36
   05ef   7e 20
   05f1   12 0a 36
```

```
mov     r6,#40h
mov     31h,30h
inc     30h

inc     r0
ljmp    pass2
mov     a,#13h
mov     dptr,#4000h
movx    @dptr,a
lcall   clk_inst     ;to

lcall   delay1
lcall   delay1
mov     r6,#40h
mov     31h,30h
dec     30h

dec     r0
ljmp    pass2
mov     a,#80h       ;first

mov     dptr,#4000h
movx    @dptr,a
lcall   clk_inst
mov     r6,40h

lcall   dat_writ
mov     r6,41h
lcall   dat_writ
mov     r6,42h
lcall   dat_writ
mov     r6,43h
lcall   dat_writ
mov     r6,44h
lcall   dat_writ
mov     r6,45h
lcall   dat_writ
mov     r6,46h
lcall   dat_writ
mov     r6,47h
lcall   dat_writ
mov     r6,#" "
lcall   dat_writ
mov     r6,#" "
lcall   dat_writ
mov     r6,#" "
lcall   dat_writ
mov     r6,#" "
lcall   dat_writ
mov     r6,#" "
lcall   dat_writ
```

```
   05f4   7e 20          mov     r6,#" "
   05f6   12 0a 36       lcall   dat_writ
   05f9   7e 20          mov     r6,#" "
   05fb   12 0a 36       lcall   dat_writ
   05fe   7e 20          mov     r6,#" "
   0600   12 0a 36       lcall   dat_writ
   0603   12 0a 74       lcall   delay1
   0606   12 0a 74       lcall   delay1
   0609   12 0a 74       lcall   delay1
   060c   12 0a 74       lcall   delay1
   060f   12 0a 74       lcall   delay1
   0612   12 0a 74       lcall   delay1
   0615   12 0a 74       lcall   delay1
   0618   12 0a 74       lcall   delay1
   061b   12 0a 74       lcall   delay1
   061e   12 0a 74       lcall   delay1
   0621   12 0a 74       lcall   delay1
   0624   12 0a 74       lcall   delay1
   0627   74 80          mov     a,#80h
;first line
   0629   90 40 00       mov     dptr,#4000h
   062c   f0             movx    @dptr,a
   062d   12 0a 40       lcall   clk_inst
   0630   ae 48          mov     r6,48h
;blank space
   0632   12 0a 36       lcall   dat_writ
   0635   ae 49          mov     r6,49h
   0637   12 0a 36       lcall   dat_writ
   063a   ae 4a          mov     r6,4ah
   063c   12 0a 36       lcall   dat_writ
   063f   ae 4b          mov     r6,4bh
   0641   12 0a 36       lcall   dat_writ
   0644   ae 4c          mov     r6,4ch
   0646   12 0a 36       lcall   dat_writ
   0649   ae 4d          mov     r6,4dh
   064b   12 0a 36       lcall   dat_writ
   064e   ae 4e          mov     r6,4eh
   0650   12 0a 36       lcall   dat_writ
   0653   ae 4f          mov     r6,4fh
   0655   12 0a 36       lcall   dat_writ
   0658   7e 20          mov     r6,#" "
   065a   12 0a 36       lcall   dat_writ
   065d   7e 20          mov     r6,#" "
   065f   12 0a 36       lcall   dat_writ
   0662   7e 20          mov     r6,#" "
   0664   12 0a 36       lcall   dat_writ
   0667   7e 20          mov     r6,#" "
   0669   12 0a 36       lcall   dat_writ
   066c   7e 20          mov     r6,#" "
   066e   12 0a 36       lcall   dat_writ
   0671   7e 20          mov     r6,#" "
```

```
   0673   12 0a 36          lcall    dat_writ
   0676   7e 20             mov      r6,#" "
   0678   12 0a 36          lcall    dat_writ
   067b   7e 20             mov      r6,#" "
   067d   12 0a 36          lcall    dat_writ
   0680   12 0a 74          lcall    delay1
   0683   12 0a 74          lcall    delay1
   0686   12 0a 74          lcall    delay1
   0689   12 0a 74          lcall    delay1
   068c   12 0a 74          lcall    delay1
   068f   12 0a 74          lcall    delay1
   0692   12 0a 74          lcall    delay1
   0695   12 0a 74          lcall    delay1
   0698   12 0a 74          lcall    delay1
   069b   12 0a 74          lcall    delay1
   069e   12 0a 74          lcall    delay1
   06a1   12 0a 74          lcall    delay1
   06a4   74 80             mov      a,#80h
;first line
   06a6   90 40 00          mov      dptr,#4000h
   06a9   f0                movx     @dptr,a
   06aa   12 0a 40          lcall    clk_inst
   06ad   ae 50             mov      r6,50h
;blank space
   06af   12 0a 36          lcall    dat_writ
   06b2   ae 51             mov      r6,51h
   06b4   12 0a 36          lcall    dat_writ
   06b7   ae 52             mov      r6,52h
   06b9   12 0a 36          lcall    dat_writ
   06bc   ae 53             mov      r6,53h
   06be   12 0a 36          lcall    dat_writ
   06c1   ae 54             mov      r6,54h
   06c3   12 0a 36          lcall    dat_writ
   06c6   ae 55             mov      r6,55h
   06c8   12 0a 36          lcall    dat_writ
   06cb   ae 56             mov      r6,56h
   06cd   12 0a 36          lcall    dat_writ
   06d0   ae 57             mov      r6,57h
   06d2   12 0a 36          lcall    dat_writ
   06d5   7e 20             mov      r6,#" "
   06d7   12 0a 36          lcall    dat_writ
   06da   7e 20             mov      r6,#" "
   06dc   12 0a 36          lcall    dat_writ
   06df   7e 20             mov      r6,#" "
   06e1   12 0a 36          lcall    dat_writ
   06e4   7e 20             mov      r6,#" "
   06e6   12 0a 36          lcall    dat_writ
   06e9   7e 20             mov      r6,#" "
   06eb   12 0a 36          lcall    dat_writ
   06ee   7e 20             mov      r6,#" "
   06f0   12 0a 36          lcall    dat_writ
```

```
06f3    7e 20
06f5    12 0a 36
06f8    7e 20
06fa    12 0a 36
06fd    12 0a 74
0700    12 0a 74
0703    12 0a 74
0706    12 0a 74
0709    12 0a 74
070c    12 0a 74
070f    12 0a 74
0712    12 0a 74
0715    12 0a 74
0718    12 0a 74
071b    12 0a 74
071e    12 0a 74
0721    d2 b5
0723    12 0a 74
0726    c2 b5
0728    7d 14
072a    90 21 00
072d    12 0a 60    loop3:
0730    e5 b0
0732    54 0f
;swap     a
0734    f5 36
0736    12 0a 60
0739    e5 b0
073b    54 0f
073d    c4
073e    45 36
0740    f0
0741    a3
0742    dd e9
0744    90 21 00
0747    78 40
0749    75 f0 14
074c    86 33       loop4:
074e    e0
074f    b5 33 03
0752    02 07 58
0755    02 08 32    loopxx:
0758    a3          xxxx:
0759    08
075a    d5 f0 ef
075d    74 80
;first line
075f    90 40 00
0762    f0
0763    12 0a 40
```

```
mov     r6,#" "
lcall   dat_writ
mov     r6,#" "
lcall   dat_writ
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
setb    p3.5    ;reset
lcall   delay1
clr     p3.5
mov     r5,#14h
mov     dptr,#2100h
lcall   cl_k
mov     a,p3
anl     a,#0fh

mov     36h,a
lcall   cl_k
mov     a,p3
anl     a,#0fh
swap    a
orl     a,36h
movx    @dptr,a
inc     dptr
djnz    r5,loop3
mov     dptr,#2100h
mov     r0,#40h
mov     b,#14h
mov     33h,@r0
movx    a,@dptr
cjne    a,33h,loopxx
ljmp    xxxx
ljmp    loop6
inc     dptr
inc     r0
djnz    b,loop4
mov     a,#80h

mov     dptr,#4000h
movx    @dptr,a
lcall   clk_inst
```

```
   0766   7e 20          mov     r6,#20h
;blank space
   0768   12 0a 36       lcall   dat_writ
   076b   7e 20          mov     r6,#20h
;blank space
   076d   12 0a 36       lcall   dat_writ
   0770   7e 20          mov     r6,#20h
;blank space
   0772   12 0a 36       lcall   dat_writ
   0775   7e 20          mov     r6,#20h
;blank space
   0777   12 0a 36       lcall   dat_writ
   077a   7e 20          mov     r6,#20h
;blank space
   077c   12 0a 36       lcall   dat_writ
   077f   7e 43          mov     r6,#43h   ;C
   0781   12 0a 36       lcall   dat_writ
   0784   7e 4f          mov     r6,#4fh   ;O
   0786   12 0a 36       lcall   dat_writ
   0789   7e 52          mov     r6,#52h   ;R
   078b   12 0a 36       lcall   dat_writ
   078e   7e 52          mov     r6,#52h   ;R
   0790   12 0a 36       lcall   dat_writ
   0793   7e 45          mov     r6,#45h   ;E
   0795   12 0a 36       lcall   dat_writ
   0798   7e 43          mov     r6,#43h   ;C
   079a   12 0a 36       lcall   dat_writ
   079d   7e 54          mov     r6,#54h   ;T
   079f   12 0a 36       lcall   dat_writ
   07a2   7e 20          mov     r6,#20h
;blank space
   07a4   12 0a 36       lcall   dat_writ
   07a7   7e 20          mov     r6,#20h
;blank space
   07a9   12 0a 36       lcall   dat_writ
   07ac   7e 20          mov     r6,#20h
;blank space
   07ae   12 0a 36       lcall   dat_writ
   07b1   7e 20          mov     r6,#20h
;blank space
   07b3   12 0a 36       lcall   dat_writ
   07b6   74 c0          mov     a,#0c0h
;second line
   07b8   90 40 00       mov     dptr,#4000h
   07bb   f0             movx    @dptr,a
   07bc   12 0a 40       lcall   clk_inst
   07bf   7e 20          mov     r6,#20h
;blank space
   07c1   12 0a 36       lcall   dat_writ
   07c4   7e 20          mov     r6,#20h
;blank space
```

```
   07c6   12 0a 36          lcall   dat_writ
   07c9   7e 20             mov     r6,#20h
;blank space
   07cb   12 0a 36          lcall   dat_writ
   07ce   7e 20             mov     r6,#20h
;blank space
   07d0   12 0a 36          lcall   dat_writ
   07d3   7e 20             mov     r6,#20h
;blank space
   07d5   12 0a 36          lcall   dat_writ
   07d8   7e 20             mov     r6,#20h
;blank space
   07da   12 0a 36          lcall   dat_writ
   07dd   7e 20             mov     r6,#20h
;blank space
   07df   12 0a 36          lcall   dat_writ
   07e2   7e 20             mov     r6,#20h
;blank space
   07e4   12 0a 36          lcall   dat_writ
   07e7   7e 20             mov     r6,#20h
;blank space
   07e9   12 0a 36          lcall   dat_writ
   07ec   7e 20             mov     r6,#20h
;blank space
   07ee   12 0a 36          lcall   dat_writ
   07f1   7e 20             mov     r6,#20h
;blank space
   07f3   12 0a 36          lcall   dat_writ
   07f6   7e 20             mov     r6,#20h
;blank space
   07f8   12 0a 36          lcall   dat_writ
   07fb   7e 20             mov     r6,#20h
;blank space
   07fd   12 0a 36          lcall   dat_writ
   0800   7e 20             mov     r6,#20h
;blank space
   0802   12 0a 36          lcall   dat_writ
   0805   7e 20             mov     r6,#20h
;blank space
   0807   12 0a 36          lcall   dat_writ
   080a   7e 20             mov     r6,#20h
;blank space
   080c   12 0a 36          lcall   dat_writ
   080f   d2 97             setb    p1.7
;setting port 1.7 to on relay ckt
   0811   12 0a 74          lcall   delay1
   0814   12 0a 74          lcall   delay1
   0817   12 0a 74          lcall   delay1
   081a   12 0a 74          lcall   delay1
   081d   12 0a 74          lcall   delay1
   0820   12 0a 74          lcall   delay1
```

```
   0823   12 0a 74                    lcall    delay1
   0826   12 0a 74                    lcall    delay1
   0829   12 0a 74                    lcall    delay1
   082c   12 0a 74                    lcall    delay1
   082f   02 00 02                    ljmp     main
   0832                       loop6:
   0832   74 80                       mov      a,#80h
;first line
   0834   90 40 00                 mov      dptr,#4000h
   0837   f0                       movx     @dptr,a
   0838   12 0a 40                    lcall    clk_inst
   083b   7e 20                       mov      r6,#20h
;blank space
   083d   12 0a 36                    lcall    dat_writ
   0840   7e 20                       mov      r6,#20h
;blank space
   0842   12 0a 36                    lcall    dat_writ
   0845   7e 20                       mov      r6,#20h
;blank space
   0847   12 0a 36                    lcall    dat_writ
   084a   7e 20                       mov      r6,#20h
;blank space
   084c   12 0a 36                    lcall    dat_writ
   084f   7e 20                       mov      r6,#20h
;blank space
   0851   12 0a 36                    lcall    dat_writ
   0854   7e 45                       mov      r6,#45h  ;E
   0856   12 0a 36                    lcall    dat_writ
   0859   7e 52                       mov      r6,#52h  ;R
   085b   12 0a 36                    lcall    dat_writ
   085e   7e 52                       mov      r6,#52h  ;R
   0860   12 0a 36                    lcall    dat_writ
   0863   7e 4f                       mov      r6,#4fh  ;O
   0865   12 0a 36                    lcall    dat_writ
   0868   7e 52                       mov      r6,#52h  ;R
   086a   12 0a 36                    lcall    dat_writ
   086d   7e 20                       mov      r6,#20h
;blank space
   086f   12 0a 36                    lcall    dat_writ
   0872   7e 20                       mov      r6,#20h
;blank space
   0874   12 0a 36                    lcall    dat_writ
   0877   7e 20                       mov      r6,#20h
;blank space
   0879   12 0a 36                    lcall    dat_writ
   087c   7e 20                       mov      r6,#20h
;blank space
   087e   12 0a 36                    lcall    dat_writ
   0881   7e 20                       mov      r6,#20h
;blank space
   0883   12 0a 36                    lcall    dat_writ
```

```
   0886   7e 20
;blank space
   0888   12 0a 36
   088b   74 c0
;second line
   088d   90 40 00
   0890   f0
   0891   12 0a 40
   0894   7e 20
;blank space
   0896   12 0a 36
   0899   7e 20
;blank space
   089b   12 0a 36
   089e   7e 20
;blank space
   08a0   12 0a 36
   08a3   7e 20
;blank space
   08a5   12 0a 36
   08a8   7e 20
;blank space
   08aa   12 0a 36
   08ad   7e 20
;blank space
   08af   12 0a 36
   08b2   7e 20
;blank space
   08b4   12 0a 36
   08b7   7e 20
;blank space
   08b9   12 0a 36
   08bc   7e 20
;blank space
   08be   12 0a 36
   08c1   7e 20
;blank space
   08c3   12 0a 36
   08c6   7e 20
;blank space
   08c8   12 0a 36
   08cb   7e 20
;blank space
   08cd   12 0a 36
   08d0   7e 20
;blank space
   08d2   12 0a 36
   08d5   7e 20
;blank space
   08d7   12 0a 36
```

```
   mov      r6,#20h

   lcall    dat_writ
   mov      a,#0c0h

mov      dptr,#4000h
movx     @dptr,a
   lcall    clk_inst
   mov      r6,#20h

   lcall    dat_writ
   mov      r6,#20h

   lcall    dat_writ
   mov      r6,#20h

   lcall    dat_writ
   mov      r6,#20h

   lcall    dat_writ
   mov      r6,#20h

   lcall    dat_writ
   mov      r6,#20h

   lcall    dat_writ
   mov      r6,#20h

   lcall    dat_writ
   mov      r6,#20h

   lcall    dat_writ
   mov      r6,#20h

   lcall    dat_writ
   mov      r6,#20h

   lcall    dat_writ
   mov      r6,#20h

   lcall    dat_writ
   mov      r6,#20h

   lcall    dat_writ
```

```
   08da   7e 20
;blank space
   08dc   12 0a 36
   08df   7e 20
;blank space
   08e1   12 0a 36
   08e4   d2 96
;clearing port 1.7 to off relay
   08e6   12 0a 74
   08e9   12 0a 74
   08ec   12 0a 74
   08ef   12 0a 74
   08f2   12 0a 74
   08f5   12 0a 74
   08f8   12 0a 74
   08fb   c2 96
   08fd   12 0a 74
   0900   12 0a 74
   0903   12 0a 74
   0906   12 0a 74
   0909   12 0a 74
   090c   12 0a 74
   090f   12 0a 74
   0912   12 0a 74
   0915   12 0a 74
   0918   12 0a 74
   091b   12 0a 74
   091e   12 0a 74
   0921   74 80
;first line
   0923   90 40 00
   0926   f0
   0927   12 0a 40
   092a   90 21 00
   092d   e0
   092e   fe
;blank space
   092f   12 0a 36
   0932   90 21 01
   0935   e0
   0936   fe
   0937   12 0a 36
   093a   90 21 02
   093d   e0
   093e   fe
   093f   12 0a 36
   0942   90 21 03
   0945   e0
   0946   fe
   0947   12 0a 36
   094a   90 21 04
```

```
mov     r6,#20h

lcall   dat_writ
mov     r6,#20h

lcall   dat_writ
setb    p1.6

lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
clr     p1.6
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
lcall   delay1
mov     a,#80h

mov     dptr,#4000h
movx    @dptr,a
lcall   clk_inst
mov     dptr,#2100h
movx    a,@dptr
mov     r6,a

lcall   dat_writ
mov     dptr,#2101h
movx    a,@dptr
mov     r6,a
lcall   dat_writ
mov     dptr,#2102h
movx    a,@dptr
mov     r6,a
lcall   dat_writ
mov     dptr,#2103h
movx    a,@dptr
mov     r6,a
lcall   dat_writ
mov     dptr,#2104h
```

```
094d    e0
094e    fe
094f    12 0a 36
0952    90 21 05
0955    e0
0956    fe
0957    12 0a 36
095a    90 21 06
095d    e0
095e    fe
095f    12 0a 36
0962    90 21 07
0965    e0
0966    fe
0967    12 0a 36
096a    90 21 08
096d    e0
096e    fe
096f    12 0a 36
0972    90 21 09
0975    e0
0976    fe
0977    12 0a 36
097a    90 21 0a
097d    e0
097e    fe
097f    12 0a 36
0982    90 21 0b
0985    e0
0986    fe
0987    12 0a 36
098a    90 21 0c
098d    e0
098e    fe
098f    12 0a 36
0992    90 21 0d
0995    e0
0996    fe
0997    12 0a 36
099a    90 21 0e
099d    e0
099e    fe
099f    12 0a 36
09a2    90 21 0f
09a5    e0
09a6    fe
09a7    12 0a 36
09aa    74 c0
;first line
   09ac    90 40 00
   09af    f0
```

```
movx    a,@dptr
mov     r6,a
lcall   dat_writ
mov     dptr,#2105h
movx    a,@dptr
mov     r6,a
lcall   dat_writ
mov     dptr,#2106h
movx    a,@dptr
mov     r6,a
lcall   dat_writ
mov     dptr,#2107h
movx    a,@dptr
mov     r6,a
lcall   dat_writ
mov     dptr,#2108h
movx    a,@dptr
mov     r6,a
lcall   dat_writ
mov     dptr,#2109h
movx    a,@dptr
mov     r6,a
lcall   dat_writ
mov     dptr,#210ah
movx    a,@dptr
mov     r6,a
lcall   dat_writ
mov     dptr,#210bh
movx    a,@dptr
mov     r6,a
lcall   dat_writ
mov     dptr,#210ch
movx    a,@dptr
mov     r6,a
lcall   dat_writ
mov     dptr,#210dh
movx    a,@dptr
mov     r6,a
lcall   dat_writ
mov     dptr,#210eh
movx    a,@dptr
mov     r6,a
lcall   dat_writ
mov     dptr,#210fh
movx    a,@dptr
mov     r6,a
lcall   dat_writ
mov     a,#0C0h

mov     dptr,#4000h
movx    @dptr,a
```

```
   09b0   12 0a 40        lcall    clk_inst
   09b3   90 21 10        mov      dptr,#2110h
   09b6   e0              movx     a,@dptr
   09b7   fe              mov      r6,a
;blank space
   09b8   12 0a 36        lcall    dat_writ
   09bb   90 21 11        mov      dptr,#2111h
   09be   e0              movx     a,@dptr
   09bf   fe              mov      r6,a
   09c0   12 0a 36        lcall    dat_writ
   09c3   90 21 12        mov      dptr,#2112h
   09c6   e0              movx     a,@dptr
   09c7   fe              mov      r6,a
   09c8   12 0a 36        lcall    dat_writ
   09cb   90 21 13        mov      dptr,#2113h
   09ce   e0              movx     a,@dptr
   09cf   fe              mov      r6,a
   09d0   12 0a 36        lcall    dat_writ
   09d3   7e 20           mov      r6,#20h
   09d5   12 0a 36        lcall    dat_writ
   09d8   7e 20           mov      r6,#20h
   09da   12 0a 36        lcall    dat_writ
   09dd   7e 20           mov      r6,#20h
   09df   12 0a 36        lcall    dat_writ
   09e2   7e 20           mov      r6,#20h
   09e4   12 0a 36        lcall    dat_writ
   09e7   7e 20           mov      r6,#20h
   09e9   12 0a 36        lcall    dat_writ
   09ec   7e 20           mov      r6,#20h
   09ee   12 0a 36        lcall    dat_writ
   09f1   7e 20           mov      r6,#20h
   09f3   12 0a 36        lcall    dat_writ
   09f6   7e 20           mov      r6,#20h
   09f8   12 0a 36        lcall    dat_writ
   09fb   7e 20           mov      r6,#20h
   09fd   12 0a 36        lcall    dat_writ
   0a00   7e 20           mov      r6,#20h
   0a02   12 0a 36        lcall    dat_writ
   0a05   7e 20           mov      r6,#20h
   0a07   12 0a 36        lcall    dat_writ
   0a0a   7e 20           mov      r6,#20h
   0a0c   12 0a 36        lcall    dat_writ
   0a0f   12 0a 74        lcall    delay1
   0a12   12 0a 74        lcall    delay1
   0a15   12 0a 74        lcall    delay1
   0a18   12 0a 74        lcall    delay1
   0a1b   12 0a 74        lcall    delay1
   0a1e   12 0a 74        lcall    delay1
   0a21   12 0a 74        lcall    delay1
   0a24   12 0a 74        lcall    delay1
   0a27   12 0a 74        lcall    delay1
```

```
0a2a    12 0a 74                        lcall    delay1
0a2d    12 0a 74                        lcall    delay1
0a30    12 0a 74                        lcall    delay1
0a33    02 00 02                        ljmp     main

                ;stop of this program

      ;-------------------------------------
      ;       SUBROUTINE PROGRAMS
      ;-------------------------------------

0a36    ee              dat_writ:       mov      a,r6
0a37    90 40 00                        mov      dptr,#4000h
0a3a    f0                              movx     @dptr,a
0a3b    f6                              mov      @r0,a
0a3c    12 0a 50                        lcall    clk_data
0a3f    22                              ret

    ;-----------------------------------------------------
    ;       SUROUTINE TO WRITE IN INSTRUCTION REGISTER
    ;-----------------------------------------------------

0a40    90 40 02        clk_inst:       mov      dptr,#4002h
0a43    74 04                           mov      a,#04h
0a45    f0                              movx     @dptr,a
0a46    12 0a 6b                        lcall    delay
0a49    74 00                           mov      a,#00h
0a4b    f0                              movx     @dptr,a
0a4c    12 0a 6b                        lcall    delay
0a4f    22                              ret

    ;-----------------------------------------------------
    ;       SUROUTINE TO WRITE IN INSTRUCTION REGISTER
    ;-----------------------------------------------------

0a50    90 40 02        clk_data:       mov      dptr,#4002h
0a53    74 05                           mov      a,#05h
0a55    f0                              movx     @dptr,a
0a56    12 0a 6b                        lcall     delay
0a59    74 01                           mov      a,#01h
0a5b    f0                              movx     @dptr,a
0a5c    12 0a 6b                        lcall     delay
0a5f    22                              ret

    ;-----------------------------------------------------
    ;     SUROUTINE TO PROVIDE clk_inst PULSE TO GAL IC
    ;               CONNECTED IN PORT P3.4
    ;-----------------------------------------------------

0a60    d2 b4           cl_k:           setb     p3.4
0a62    12 0a 6b                        lcall    delay
```

```
0a65    c2 b4                           clr     p3.4
0a67    12 0a 6b                        lcall   delay
0a6a    22                              ret

                ;-----------------------------------
                ;          DELAY  SUB ROUTINES
                ;-----------------------------------

0a6b    79 10       delay:      mov     r1,#10h
0a6d    7a 10       loops:      mov     r2,#10h
0a6f    da fe       loopd:      djnz    r2,loopd
0a71    d9 fa                   djnz    r1,loops
0a73    22                      ret
0a74    79 ff       delay1:     mov     r1,#0ffh
0a76    7a ff       loops1:     mov     r2,#0ffh
0a78    da fe       loopd1:     djnz    r2,loopd1
0a7a    d9 fa                   djnz    r1,loops1
0a7c    22                      ret
0a7d    79 b0       delay2:     mov     r1,#0b0h
0a7f    7a b0       loops2:     mov     r2,#0b0h
0a81    da fe       loopd2:     djnz    r2,loopd2
0a83    d9 fa                   djnz    r1,loops2
0a85    22                      ret

    ;-----------------------------------------------
    ;          COMPARING AND CHECKING THE VHDL
    ;             DATA AND DISPLAYED LCD DATA
    ;-----------------------------------------------

 0a86   ec          loop5:      mov     a,r4
 0a87   b4 00 19                cjne    a,#00,err
 0a8a   12 0a 60                lcall   cl_k
 0a8d   12 0a 60                lcall   cl_k
 0a90   12 0a 60                lcall   cl_k
 0a93   12 0a 60                lcall   cl_k
 0a96   12 0a 60                lcall   cl_k
 0a99   12 0a 60                lcall   cl_k
 0a9c   05 34                   inc     34h
 0a9e   e5 34                   mov     a,34h
 0aa0   b4 03 07                cjne    a,#03h,fi
 0aa3   02 08 32      err:      ljmp    loop6
 0aa6   0c            fir:      inc     r4
 0aa7   02 00 94                ljmp    first
 0aaa                fi:       ;ljmp    first1
 0758   xxxx      0149  trig_1    0146  trigger   0002
initial   0454  pass      0401  dec1
027a  start2    0aaa  fi        0a78  loopd1    0a74
delay1    0755  loopxx    0557  dec2
03d0  code2     0a81  loopd2    0a7d  delay2    0a60  cl_k
0a76  loops1    04ad  trigg
```

```
01fe  tri        0a7f  loops2     05aa  load       03e2  incl
0538  inc2       0a40  clk_inst
02e4  lef_sif    0aa6  fir        0a6d  loops      02ca
rig_sif   028c   inc       0357   trigge
0201  tri_1      0094  first      0002  main       0a6f  loopd
043a  lef_sif1   0a6b  delay
0590  lef_sif2   072d  loop3      02fe  code       02ab  dec
0a50  clk_data   0a36  dat_writ
074c  loop4      0526  pass2      0a86  loop5      0420
rig_sif1  0aa3   err       0832   loop6
0576  rig_sif2   000e  init1      0006  init
```

CHAPTER 7

ADVANTAGES AND APPLIACTIONS

CHAPTER 7

ADVANTAGES & APPLICATIONS

7.1 ADVANTAGES

7.1.1 Better Security

The Security cell is provided in every GAL 22v10B device to prevent unauthorized copying of array patterns.

7.1.2 Ease of Programming

Simple array pattern in PLDs makes programming of the device easier and entire programming of the device takes only a few seconds

7.1.3 Versatility

This project is aimed at providing security but a small change in the overall design makes it applicable to various other systems such as credit cards, ATM cards and security identification cards.

7.1.4 Low production cost

The smart card designed in this project is very economical when compared to various conventional security identification cards like SIM, Barcode etc.,

7.1.5 Reduction in Hardware

This is achieved by the use of PLD to perform the action of logic gates (IC 7404 – NOT Gate, IC 7408 – AND Gate) and decoder (74LS378 – 3x8 Decoder) and hence there is an overall reduction in the total hardware used.

7.1.6 Data Retention

The GAL 22v10B offers data retention over an excess of 20 years.

7.1.7 Compatibility with Emerging Technologies

Our project is compatible with all higher versions of VHDL in case of system upgradation.

7.2 APPLICATIONS

- Credit Card Systems
- ATM systems
- Secret Buffer Systems for Military Applications
- Use in general security systems such as Bank lockers
- Voter Identification system
- Restricting unauthorized entry

CHAPTER 8

CONCLUSION

A system for the security of data has been designed and implemented. The complete unit is more reliable than existing ones. Moreover the developed circuit is quite simple and the chance of the system to fail is small. Our project has been working satisfactorily under various test conditions. The testing was successful and provides an efficient integration of both hardware and software. The results obtained have been upto industrial standards.

Considering the design, fabrication and installation cost of conventional data security systems, our project is offered at affordable price when produced in mass. The design is highly user friendly such that even a layman finds it very simple to use.

8.1 FUTURE ENHANCEMENT

'Change' is the only changeless phenomenon the world over.

We held this saying right from the inception of this project. New technologies emerge and subside as time goes on. But our project stands against winds of time. Our project can be enhanced in the following ways.

- This apparatus can be made PC compatible by providing an RS-232 serial I/O communication.
- This project can be extended to support on-line transactions through the telephone and satellite links.
- This system can be extended to industrial attendance systems which use punched cards today. But through our systems a single user smart card would combine all the needs such as attendance, salary transactions and bonus etc.

With the suggested future enhancement of the project, the level of security will definitely increase for a better tomorrow.

REFERENCES

1. Ramesh S. Gaonkar, "Microprocessor Architecture Programming and Applications with the 8085", Penram International Publishing, NewDelhi (India), 1997.
2. John B. Peatman, "Design with Microcontrollers", Mc Graw Hill Book Company, Singapore, 1988.
3. "Atmel Microcontroller Hand Book", Atmel corporation.
4. J. Bhasker, "VHDL Primer", Prentice Hall PTR, USA, 2000.
5. David Pellerin and Douglas Taylor, "VHDL Made Easy", Pearson Education Asia Pte. Ltd., Singapore, 2000.
6. Peter J. Ashenden, "The designers Guide to VHDL", Morgan kaufmann publishers inc., Singapore, 1996.
7. "Crystalonics Display-Users Manual", Crystalonics corporation, Hosur (India).

APPENDIX

Features

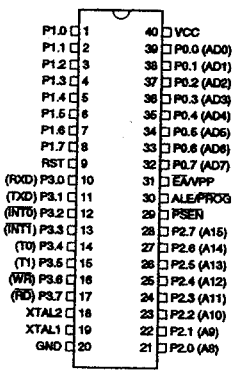
- Compatible with MCS-51™ Products
- 4K Bytes of In-System Reprogrammable Flash Memory
  - Endurance: 1,000 Write/Erase Cycles
- Fully Static Operation: 0 Hz to 24 MHz
- Three-level Program Memory Lock
- 128 x 8-bit Internal RAM
- 32 Programmable I/O Lines
- Two 16-bit Timer/Counters
- Six Interrupt Sources
- Programmable Serial Channel
- Low-power Idle and Power-down Modes

Description

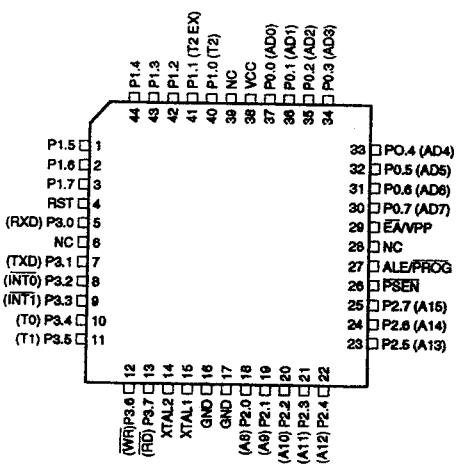
The AT89C51 is a low-power, high-performance CMOS 8-bit microcomputer with 4K bytes of Flash programmable and erasable read only memory (PEROM). The device is manufactured using Atmel's high-density nonvolatile memory technology and is compatible with the industry-standard MCS-51 instruction set and pinout. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional nonvolatile memory programmer. By combining a versatile 8-bit CPU with Flash on a monolithic chip, the Atmel AT89C51 is a powerful microcomputer which provides a highly-flexible and cost-effective solution to many embedded control applications.

Pin Configurations

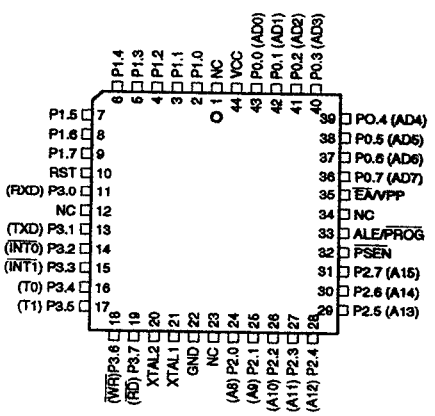
PDIP



PQFP/TQFP



PLCC



8-bit Microcontroller with 4K Bytes Flash

AT89C51

Rev. 0265G–02/00

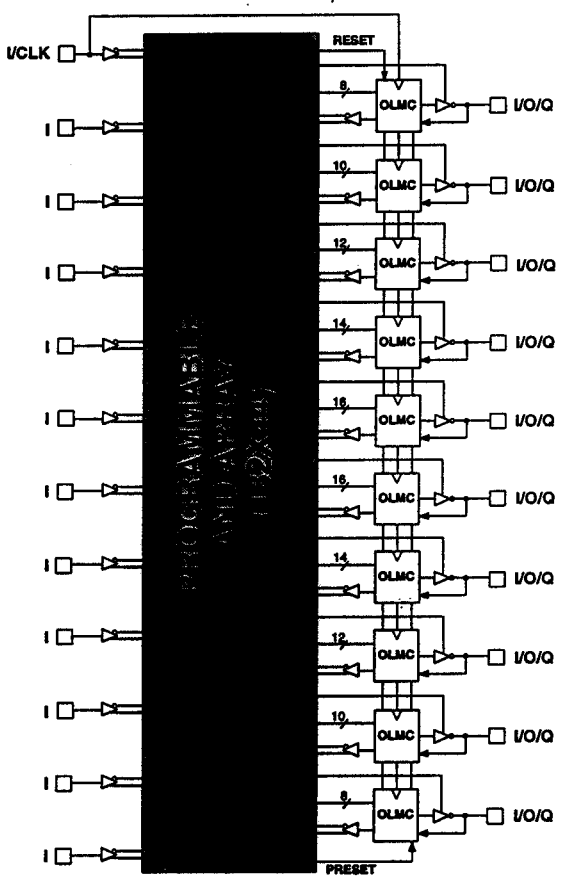
GAL22V10

High Performance E²CMOS PLD
Generic Array Logic™

Features

- HIGH PERFORMANCE E²CMOS® TECHNOLOGY
  - 4 ns Maximum Propagation Delay
  - Fmax = 250 MHz
  - 3.5 ns Maximum from Clock Input to Data Output
  - UltraMOS® Advanced CMOS Technology
- ACTIVE PULL-UPS ON ALL PINS
- COMPATIBLE WITH STANDARD 22V10 DEVICES
  - Fully Function/Fuse-Map/Parametric Compatible with Bipolar and UVCMOS 22V10 Devices
- 50% to 75% REDUCTION IN POWER VERSUS BIPOLAR
  - 90mA Typical Icc on Low Power Device
  - 45mA Typical Icc on Quarter Power Device
- E² CELL TECHNOLOGY
  - Reconfigurable Logic
  - Reprogrammable Cells
  - 100% Tested/100% Yields
  - High Speed Electrical Erasure (<100ms)
  - 20 Year Data Retention
- TEN OUTPUT LOGIC MACROCELLS
  - Maximum Flexibility for Complex Logic Designs
- PRELOAD AND POWER-ON RESET OF REGISTERS
  - 100% Functional Testability
- APPLICATIONS INCLUDE:
  - DMA Control
  - State Machine Control
  - High Speed Graphics Processing
  - Standard Logic Speed Upgrade
- ELECTRONIC SIGNATURE FOR IDENTIFICATION

Functional Block Diagram



Description

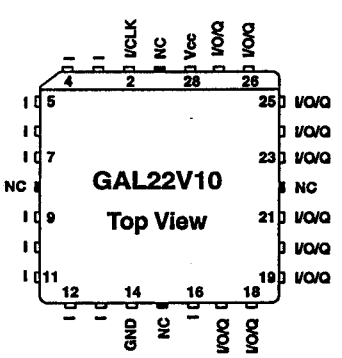
The GAL22V10, at 4ns maximum propagation delay time, combines a high performance CMOS process with Electrically Erasable (E²) floating gate technology to provide the highest performance available of any 22V10 device on the market. CMOS circuitry allows the GAL22V10 to consume much less power when compared to bipolar 22V10 devices. E² technology offers high speed (<100ms) erase times, providing the ability to reprogram or reconfigure the device quickly and efficiently.

The generic architecture provides maximum design flexibility by allowing the Output Logic Macrocell (OLMC) to be configured by the user. The GAL22V10 is fully function/fuse map/parametric compatible with standard bipolar and CMOS 22V10 devices.

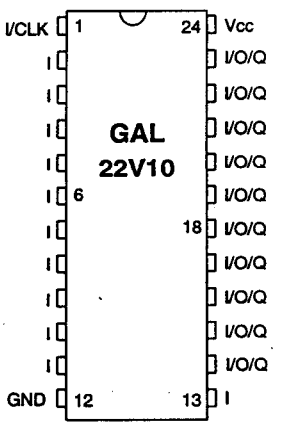
Unique test circuitry and reprogrammable cells allow complete AC, DC, and functional testing during manufacture. As a result, Lattice Semiconductor delivers 100% field programmability and functionality of all GAL products. In addition, 100 erase/write cycles and data retention in excess of 20 years are specified.

Pin Configuration

PLCC



DIP

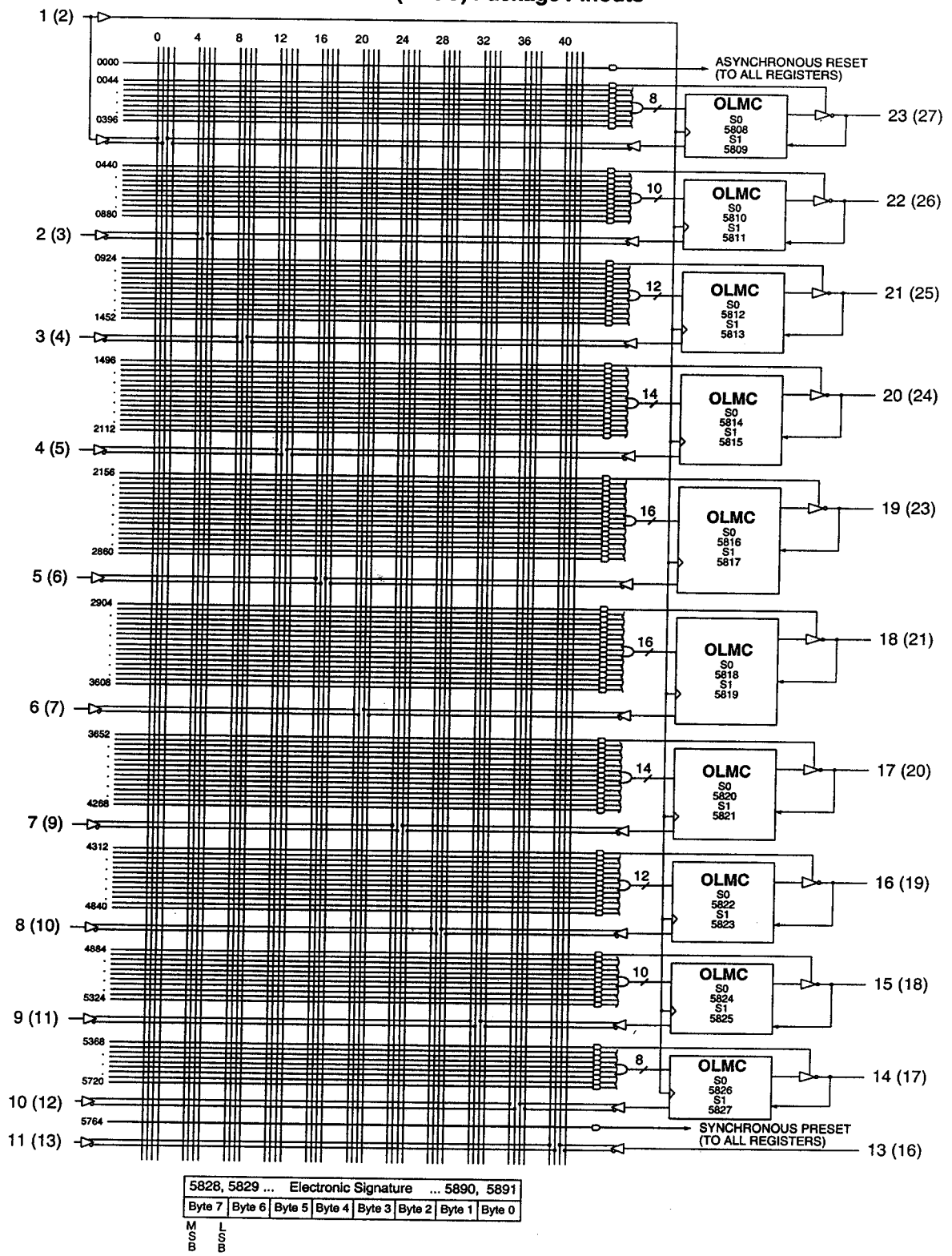


LATTICE SEMICONDUCTOR CORP., 5555 Northeast Moore Ct., Hillsboro, Oregon 97124, U.S.A.
Tel. (503) 268-8000; 1-800-LATTICE; FAX (503) 268-8556; http://www.latticesemi.com

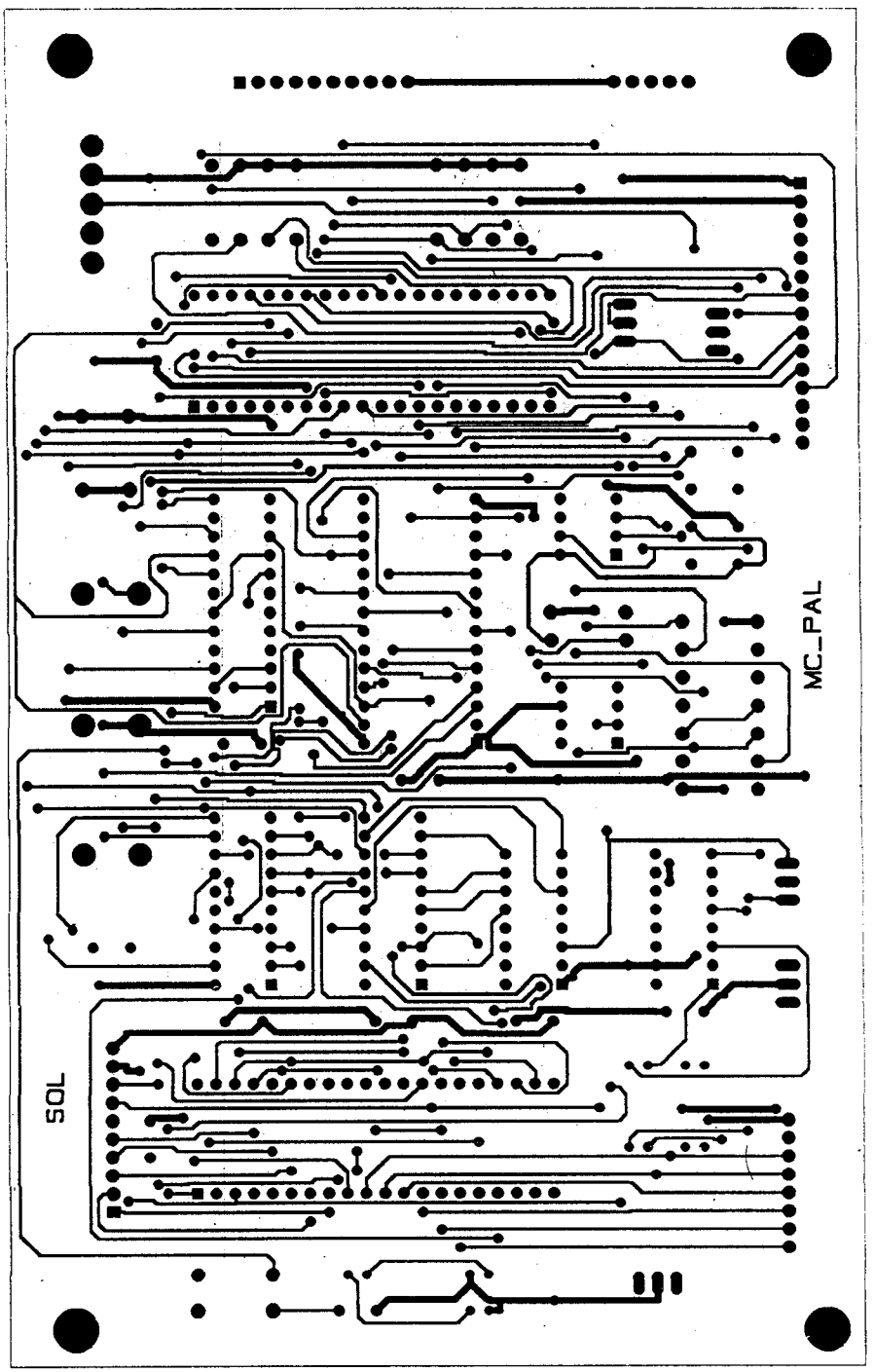
August 2000

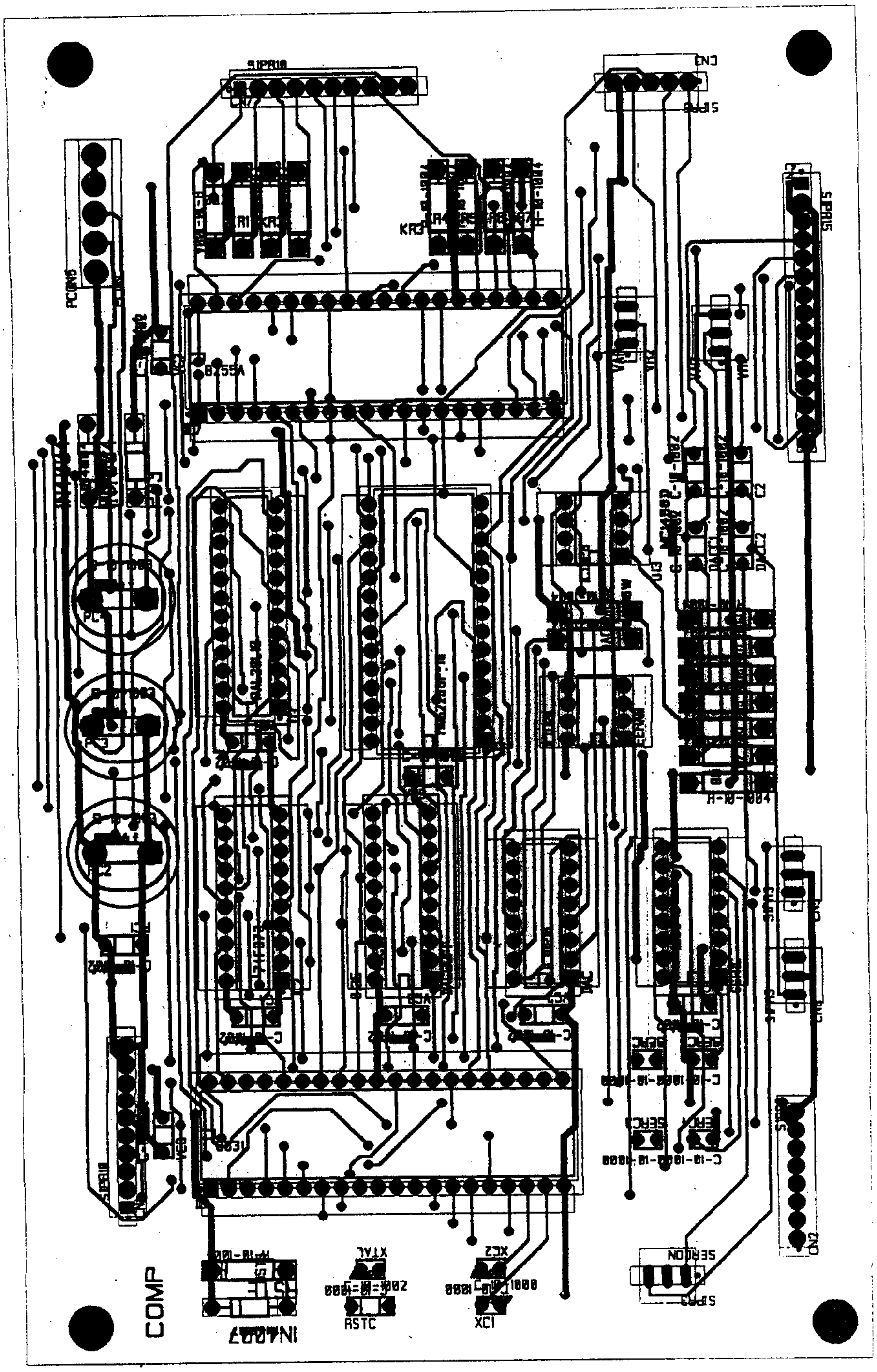
GAL22V10 Logic Diagram / JEDEC Fuse Map

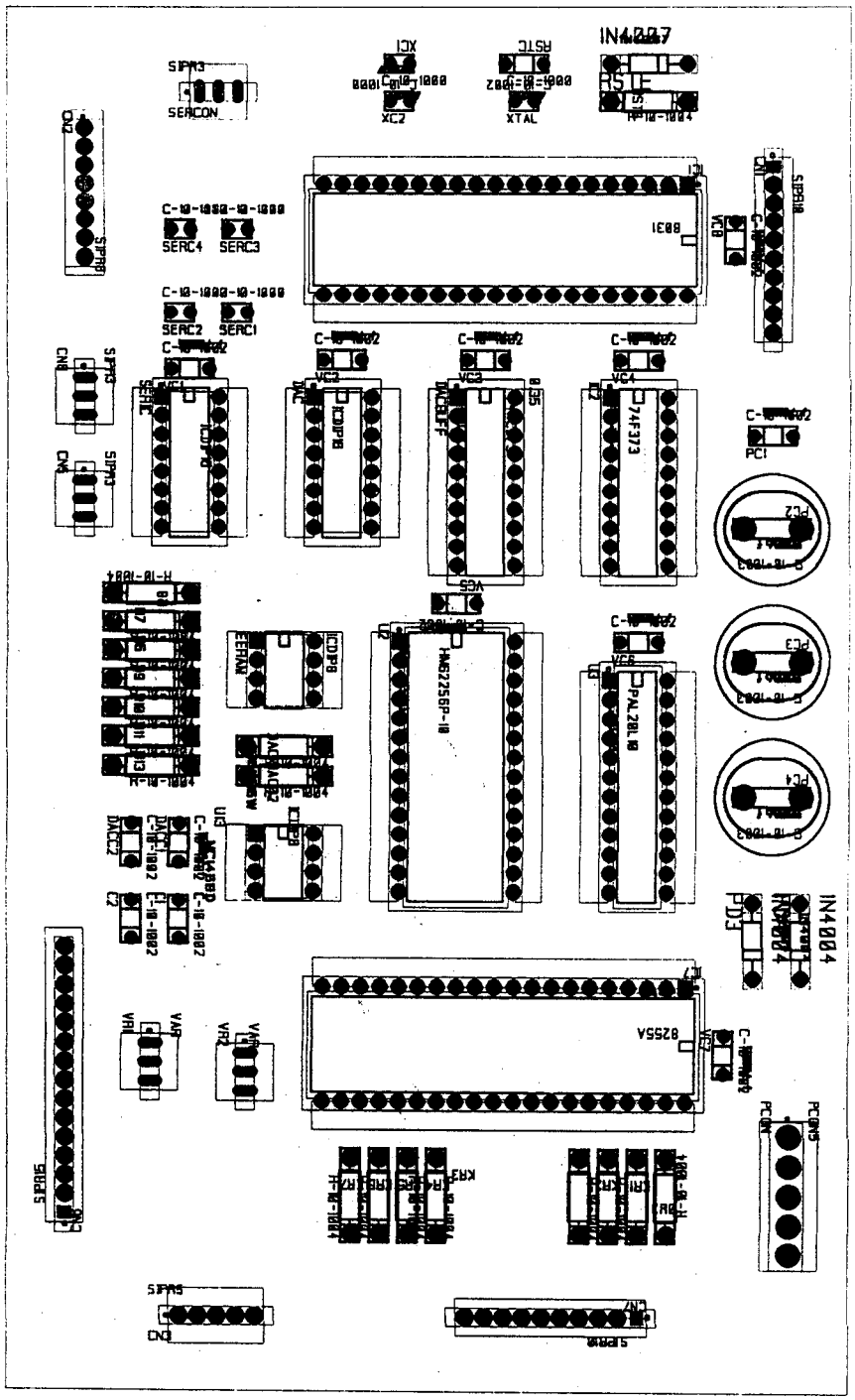
DIP (PLCC) Package Pinouts

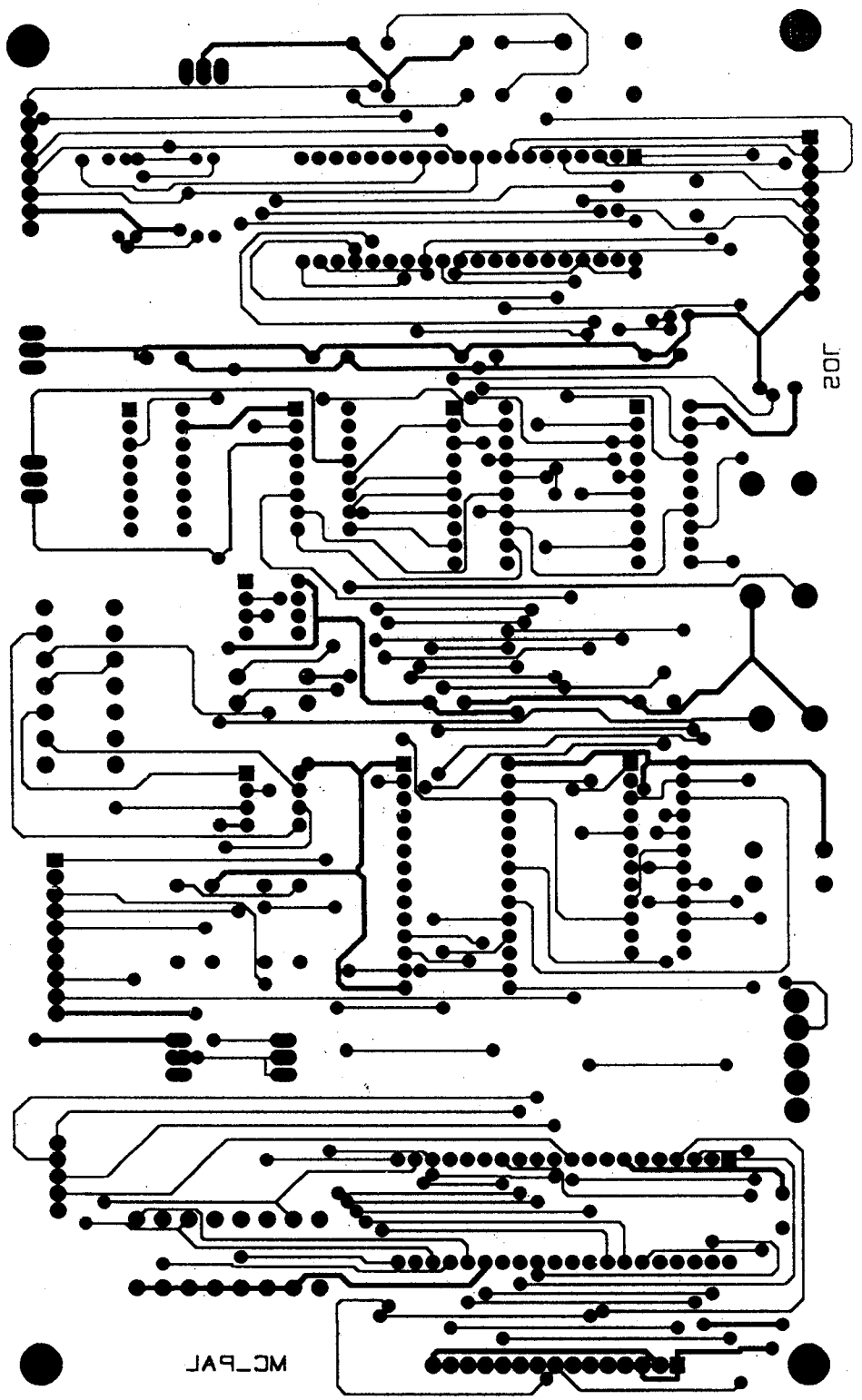


5828, 5829 ... Electronic Signature ... 5890, 5891							
Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
MSB							LSB


MC_PAL
SOL


SIPR10
CN3
PCON5
8255A
8031
XTAL
RSTC
XC1
XC2
SERCON
COMP
IN4007


IN4007
SERCON
XC1
XC2
RSTC
XTAL
RST
SERC4
SERC3
SERC2
SERC1
8031
74F373
PAL20L10
82C55A
IN4004


MC_PAL