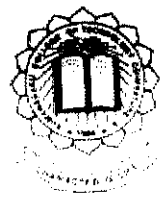


CAN INTERFACE WITH ARM7

A PROJECT REPORT

Submitted by

ANANTH.P	71203106301
PREMANAND.R	71203106037
SHARANG.M	71203106049
VINOTH.M	71203106059

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING IN

ELECTRONICS AND COMMUNICATION ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY, COIMBATORE

ANNA UNIVERSITY: CHENNAI 600 025

APRIL 2007

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"CAN INTERFACE WITH ARM7"** is the bonafide work of **"ANANTH. P, PREMANAND. R, SHARANG. M and VINOTH.M"** who carried out the project work under my supervision.

SIGNATURE

Dr. (Mrs). Rajeswari Mariappan

HEAD OF DEPARTMENT

Electronics & Communication Engineering,
Kumaraguru College of Technology,
Coimbatore - 641006

SIGNATURE

Mr. S. Govindaraju

SUPERVISOR

Professor
Electronics & Communication Engineering,
Kumaraguru College of Technology,
Coimbatore - 641006

The candidates with University Register Nos. 71203106301, 71203106037, 71203106049, 71203106059 were examined by us in the project viva-voce examination held on 23.04.07

INTERNAL EXAMINER

EXTERNAL EXAMINER

Date: 18/04/2007

To whomsoever it may concern

This is to certify that the students of Final Year ECE of Kumaraguru College of Technology, Coimbatore, TamilNadu, had undergone their final year project work in our esteemed organization successfully. Their project details are given below,

Name : **Mr.P.Ananth (71203106301)**
Mr.R.Premanand (71203106037)
Mr.M.Sharang (71203106049)
Mr.M.Vinoth (71203106059)

Project Title : **"CAN Interface with GUI Design for ARM7"**
Duration : **December - 2006 to April -2007**

Further to the company rules and regulations, source code is not permitted to be taken out of the premises.

During the project period, the students conduct and the performance were good.

For TriozTech,

Magesh Kumar.M,
(Project Co-Ordinator)

ACKNOWLEDGEMENT

We are greatly indebted to our beloved Principal **Dr.Joseph V. Thanikal, M.E**,Ph.D, who has been the backbone of all our deeds.

We profusely thank **Dr. (Mrs). Rajeswari Mariappan**, M.E, Ph.D, Head of Department, Department of Electronics and Communication Engineering, for lending a helping hand in this project.

Our sincere thanks to our External Guide Mr.M. Magesh Kumar, project co-ordinator, TRIOZ TECH.

We are highly grateful to our beloved Project Coordinator **Prof. K. Ramprakash**, Professor and our Project guide, Mr.**S. Govindaraju**, Professor, E.C.E. Department for their valuable guidance, constant encouragement and advice rendered throughout the project period for the successful completion of the project.

We owe much to all Lab Technicians of ECE department for their support, and motivation throughout the entire phase of the project in the development of the hardware and software.

We are grateful to all the faculty members of the Department of Electronics and Communication Engineering, who have helped us.

We also thank our parents without whom we couldn't have come so far and friends for their timely help that culminated as good in the end.

ABSTRACT

The study "CAN interface with ARM7" is aimed to provide real time automotive output. This project is done to reduce the size of hardware and increase the speed of the data transmission.

CAN (controller area network) is a broadcast, differential serial bus standard, for connecting electronic control units (ECU). The major portion of the project is the CAN. Two sensors are used in this project, from which the analog data are transmitted to the ARM7 processor, which has in-built CAN and ADC (analog to digital converter). The analog signal from the sensor is converted into digital signal in the ADC.

The digital signal thus converted is transmitted to the serial communication port with the help of the inbuilt CAN. CAN features an automatic 'arbitration free' transmission. A CAN message transmitted with highest priority will 'win' the arbitration, and the node transmitting the lower priority message will sense this and back off and wait. These signals received by the serial communication reaches the CAN transceiver and this is the second CAN in the project. The signal from this CAN is transmitted to the PIC microcontroller. The PIC microcontroller is used here instead of using two ARM boards as it is economical. The signal is never processed but passed to the GUI (Graphical User Interface) in the system. The GUI is programmed in the system for the installation of the project. This program is coded in 'Visual Basic' and the source for ARM7 processor is in 'C'.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

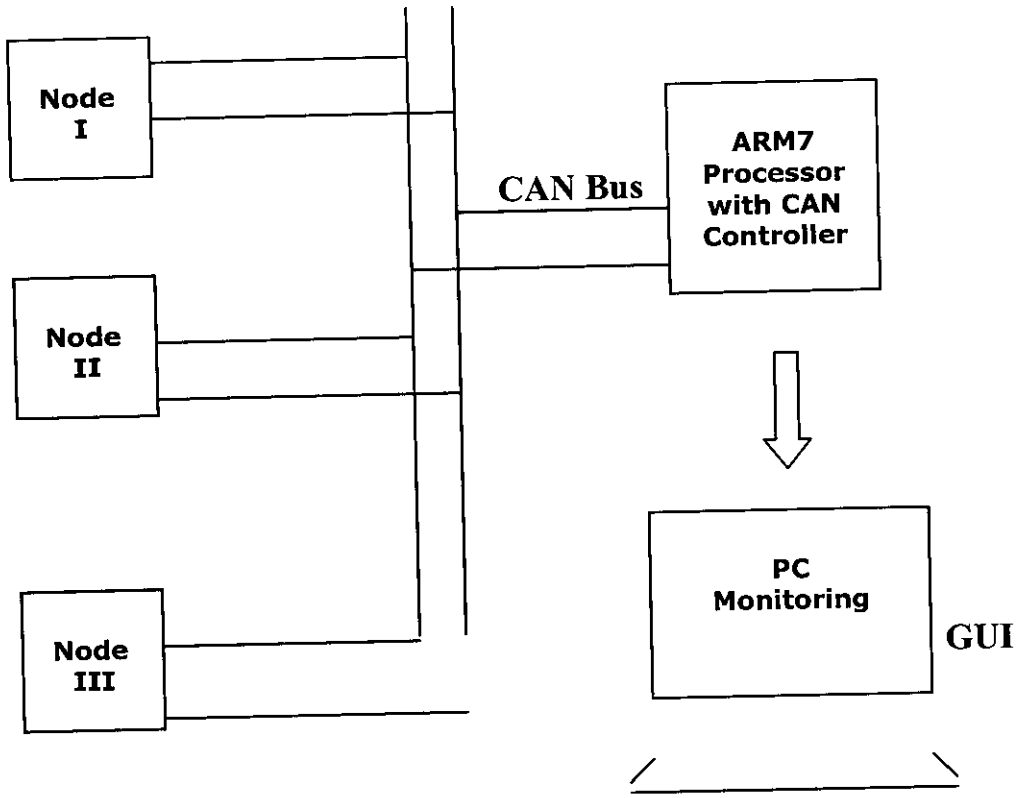
1.1 INTRODUCTION OF PROJECT

Electronics plays key part in almost all fields . In the automobile industry, we have got digital speedometers, electronic carburetors and so on. By introducing network, it will be easy and efficient to control the electronics in the automobile. Here we use a protocol called CAN Protocol to implement such a network.

The reason why we chose CAN is due to two main reasons. The basic reason is that we can have many number of nodes in the network. The other one is that all of these nodes can be connected using a two line bus.

We have decided to have two nodes as slaves and one master. The slaves being controlling the sensors and the master being a PC. The sensors are used to determine the speed, temperature of the engine and whether there has been a gas leakage or not. The master will monitor these parameters and will take appropriate measures whenever the limit is exceeded.

1.2 BLOCK DIAGRAM



Where Node I, II, III may be,

- Speedometer
- Fuel meter
- Fuel Pressure, etc…

Controller Area Network:

CAN, a serial bus network of microcontrollers that connects devices, sensors and actuators in a system or sub-system for real-time control applications. There is no addressing scheme used in controller area networks, as in the sense of conventional addressing in networks (such as Ethernet). Rather, messages are broadcast to all the nodes in the network using an identifier unique to the network. Based on the identifier, the individual nodes decide whether or not to process the message and also determine the priority of the message in terms of competition for bus access. This method allows for uninterrupted transmission when a collision is detected, unlike Ethernets that will stop transmission upon collision detection.

Controller area networks were first developed for use in automobiles. Equipped with an array of sensors, the network is able to monitor the systems that the automobile depends on to run properly and safely. Beyond automobiles, controller area networks can be used as an embedded communication system for microcontrollers as well as an open communication system for intelligent devices.

Features:

- Supports 1 Mb/s operation
- Implements ISO-11898 standard physical layer requirements
- Suitable for 12V and 24V systems
- Externally-controlled slope for reduced RFI emissions
- Detection of ground fault (permanent dominant) on TXD input
- Power-on reset and voltage brown-out protection
- An unpowered node or brown-out event will not disturb the CAN bus
- Low current standby operation
- Protection against damage due to short-circuit conditions (positive or negative battery voltage)
- Protection against high-voltage transients
- Automatic thermal shutdown protection
- Up to 112 nodes can be connected
- High noise immunity due to differential bus Implementation

Operation of CAN:

A CAN transceiver is used to Transmit and Receive data's. In CAN Communication, each node serve as a Master which is capable of Transmitting and Receiving the data's. Three Nodes (Speedometer, Fuel Pressure, and Fuel Meter) is connected to the CAN transceiver. These nodes will get the data's and Transmit to the CAN transceiver through CAN Bus. By using Inbuilt CAN in our ARM7 processor we can receive those data's. If necessary LCD display can be connected to view the received output.

Graphical User Interface (GUI):

Graphical User Interface Provides a user interface between the user and devices. In this Project GUI is used to view the output of the CAN data's in our PC (Personal Computer). For Example, if the data received from the Speedometer is 1500 RPM, the output in PC will indicate the corresponding reading.

1.3 SCOPE OF THE PROJECT

The project is for automation in automobiles. This project will find a good value if it is combined with car automation. The parameters that have to be monitored can be also controlled using this arrangement. It is possible to control one end or one controller with the help of the output of the other controller. This is like a feedback system but between many systems. This will help to maintain certain parameters that will be important for proper working and safety of the automobiles.

2.CONTROLLER AREA NETWORK

2.1 INTRODUCTION

The controller area network (CAN) is a serial communications protocol which efficiently supports distributed real-time control with a very high level of security. Its domain of application ranges from high speed networks to low cost multiplex wiring. In automotive electronics, engine control units, sensors, anti-skid-systems, etc. are connected using CAN with bit rates up to 1 Mbits/s. At the same time it is cost effective to build into vehicle body electronics, e.g. lamp clusters electric windows etc. to replace the wiring harness otherwise required.

The intension of this specification is to achieve compatibility between any two CAN implementations. Compatibility, however, has different aspects regarding e.g. electrical features and the interpretation of data to be transferred. To achieve design transparency and implementation flexibility CAN has been subdivided into different layers according to ISO/OSI Reference Model:

- The data link layer
  - The Logical Link Control (LLC) sublayer
  - The Medium Access Control (MAC) sublayer
- The physical Layer

Note that in previous versions of the CAN specifications the services and functions of the LLC and MAC sublayers of the Data Link Layer had been described in layers denoted as 'object layer' and 'transfer layer'.

The scope of LLC sublayer is

- To provide services for data transfer and for remote data request,
- To decide which messages received by the LLC sublayer are actually to be accepted,
- To provide means for recovery management and overload notifications. There is much freedom in defining object handling.

The scope of the MAC sublayer mainly is the transfer protocol, i.e. controlling the framing, performing arbitration, Error, Checking, Error signaling and fault confinement. Within the MAC sublayer it is decided whether the bus is free for starting a new transmission or whether a reception is just starting. Also some general features of the bit timing are regarded as part of the MAC sublayer. It is in the nature of the MAC sublayer that there is no freedom for modification. The scope of the physical layer is the actual transfer of the bits between the different nodes with respect to all electrical properties.

2.2 BASIC CONCEPTS

CAN have the following properties:

- Prioritization of messages
- Guarantee of latency times
- Configuration flexibility
- System wide data consistency
- Error detection and error signaling

- Automatic retransmission of corrupted messages as soon as the bus is idle again.
- Distinction between temporary errors and permanent failures of nodes and autonomous switching off of defect nodes layered architecture of CAN according to the OSI reference model.
- The physical layer defines how signals are actually transmitted and therefore deals with the description of bit timing, bit encoding and synchronization. Within this specification the driver/ receiver characteristics of the physical layer are not defined so as to allow transmission medium and signal level implementations to be optimized for their application.
- The MAC sublayer represents the kernel of the CAN protocol it presents messages received from the LLC sublayer and accepts messages to be transmitted to the LLC sublayer. The MAC sublayer is responsible for Message Framing, Arbitration, Acknowledgement, Error Detection and Signaling. The MAC sublayer are supervised by a management entity called Fault Confinement which is self-checking mechanism for distinguishing short disturbances from permanent failures.

2.3 DATA TRANSMISSION

CAN features an automatic 'arbitration free' transmission. A CAN message transmitted with highest priority will 'win' the arbitration, and the node transmitting the lower priority message will sense this and back off and wait.

This is achieved by CAN transmiting data through a binary model of "dominant" bits and "recessive" bits where dominant is a logical 0 and

recessive is a logical 1. This means open collector, or 'wired or' physical implementation of the bus (but since dominant is 0 this is sometimes referred to as wired-AND). If one node transmits a dominant bit and another node transmits a recessive bit then the dominant bit "wins" (a logical AND between the two).

Truth tables for dominant/recessive and logical AND

Bus state with two nodes transmitting

	dominant	recessive
dominant	*dominant*	*dominant*
recessive	dominant	*recessive*

Logical AND

	0	1
0	0	0
1	0	1

So, if you are transmitting a recessive bit, and someone sends a dominant bit, you see a dominant bit, and you know there was a collision. (All other collisions are invisible.) The way this works is that a dominant bit is asserted by creating a voltage across the wires while a recessive bit is simply not asserted on the bus. If anyone sets a voltage difference, everyone sees it, hence, dominant. Thus there is no delay to the higher priority

message, and the node transmitting the lower priority message automatically attempts to re-transmit 6 bit clocks after the end of the dominant message.

When used with a differential bus, a Carrier Sense Multiple Access/Bitwise Arbitration (CSMA/BA) scheme is often implemented: if two or more devices start transmitting at the same time, there is a priority based arbitration scheme to decide which one will be granted permission to continue transmitting. The CAN solution to this is prioritised arbitration (and for the dominant message delay free), making CAN very suitable for real time prioritised communications systems.

During arbitration, each transmitting node monitors the bus state and compares the received bit with the transmitted bit. If a dominant bit is received when a recessive bit is transmitted then the node stops transmitting (i.e., it lost arbitration). Arbitration is performed during the transmission of the identifier field. Each node starting to transmit at the same time sends an ID with dominant as binary 0, starting from the high bit. As soon as their ID is a larger number (lower priority) they'll be sending 1 (recessive) and see 0 (dominant), so they back off. At the end of ID transmission, all nodes but one have backed off, and the highest priority message gets through unimpeded.

Bit Timing

Each node in a CAN network has its own clock, and no clock is sent during data transmission. Synchronization is done by dividing each bit of the frame into a number of segments: Synchronisation, Propagation, Phase 1 and Phase 2. The Length of each segment can be adjusted.

2.4 LAYERS

Based on levels of abstraction, the structure of the CAN protocol can be described in terms of the following layers:

- Application Layer
- Object Layer
  - Message Filtering
  - Message and Status Handling

Layer

- Transfer

  The Transfer Layer represents the kernel of the CAN protocol. It presents messages received to the object layer and accepts messages to be transmitted from the object layer. The transfer layer is responsible for bit timing and synchronization, message framing, arbitration, acknowledgement, error detection and signaling, and fault confinement. It performs:
  - Fault Confinement
  - Error Detection
  - Message Validation
  - Acknowledgement
  - Arbitration
  - Message Framing
  - Transfer Rate and Timing
  - Information Routing

- Physicallayer

  The physical layer defines how the signals are actually transmitted.

  Tasks include:

  - Signal Level and Bit Representation
  - Transmission Medium

2.5 FRAMES

A CAN network can be configured to work with two different message (or "frame") formats: the standard or base frame format (or CAN 2.0 A), and the extended frame format (or CAN 2.0 B). The only difference between the two formats is that the "CAN base frame" supports a length of 11 bits for the identifier, and the "CAN extended frame" supports a length of 29 bits for the identifier, made up of the 11-bit identifier ("base identifier") and an 18-bit extension ("identifier extension"). The distinction between CAN base frame format and CAN extended frame format is made by using the IDE bit, which is transmitted as dominant in case of an 11-bit frame, and transmitted as recessive in case of a 29-bit frame. CAN controllers, which support extended frame format messages are also able to send and receive messages in CAN base frame format. All frames begin with a start-of-frame (SOF) bit that, obviously, denotes the start of the frame transmission.

2.6 FRAME TYPES

CAN has four frame types:

- Data frame: a frame containing node data for transmission
- Remote frame: a frame requesting the transmission of a specific identifier
- Error frame: a frame transmitted by any node detecting an error
- Overload frame: a frame to inject a delay between data and/or remote frames

Data frame

The data frame is the only frame for actual data transmission. There are two message formats:

- Base frame format: with 11 identifier bits
- Extended frame format: with 29 identifier bits

The CAN standard requires the implementation *must* accept the base frame format and *may* accept the extended frame format, but *must* tolerate the extended frame format.

Base frame format

Field name	Length (bits)	Purpose
Start-of-frame	1	Denotes the start of frame transmission
Identifier	11	A (unique) identifier for the data
Remote transmission request (RTR)	1	Must be dominant (0)Optional
Identifier extension bit (IDE)	1	Must be dominant (0)Optional
Reserved bit (r0)	1	Reserved bit (it must be set to dominant (0), but accepted as either dominant or recessive)
Data length code (DLC)	4	Number of bytes of data (0-8 bytes)
Data field	0-8 bytes	Data to be transmitted (length dictated by DLC field)
CRC	15	Cyclic redundancy check
CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1) and any receiver can assert a dominant (0)
ACK delimiter	1	Must be recessive (1)
End-of-frame (EOF)	7	Must be recessive (1)

Extended frame format

The frame format is as follows:

Field name	Length (bits)	Purpose
Start-of-frame	1	Denotes the start of frame transmission
Identifier A	11	First part of the (unique) identifier for the data
Substitute remote request (SRR)	1	Must be recessive (1)Optional
Identifier extension bit (IDE)	1	Must be recessive (1)Optional
Identifier B	18	Second part of the (unique) identifier for the data
Remote transmission request (RTR)	1	Must be dominant (0)
Reserved bits (r0, r1)	2	Reserved bits (it must be set dominant (0), but accepted as either dominant or recessive)
Data length code (DLC)	4	Number of bytes of data (0-8 bytes)
Data field	0-8 bytes	Data to be transmitted (length dictated by DLC field)
CRC	15	Cyclic redundancy check

CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1) and any receiver can assert a dominant (0)
ACK delimiter	1	Must be recessive (1)
End-of-frame (EOF)	7	Must be recessive (1)

The two identifier fields (A & B) combined form a 29-bit identifier.

Remote frame

The remote frame is identical to the data frame except:

- the RTR bit is set to recessive (1)
- data length contains the number of bytes that are required from the data frame

Error frame

Error frame consists of two different fields

The first field is given by the superposition of ERROR FLAGS contributed from different stations. The following second field is the ERROR DELIMITER.

There are two types of error flags

Active Error Flag

Transmitted by a node detecting an error on the network that is in error state "error active".

Passive Error Flag

Transmitted by a node detecting an active error frame on the network that is in error state "error passive".

Overload frame

The overload frame contains the two bit fields Overload Flag and Overload Delimiter. There are two kinds of overload conditions that can lead to the transmission of an overload flag:

The internal conditions of a receiver, which requires a delay of the next data frame or remote frame.

Detection of a dominant bit during intermission.

The start of an overload frame due to case 1 is only allowed to be started at the first bit time of an expected intermission, whereas overload frames due to case 2 start one bit after detecting the dominant bit. Overload Flag consists of six dominant bits. The overall form corresponds to that of the active error flag. The overload flag's form destroys the fixed form of the intermission field. As a consequence, all other stations also detect an overload condition and on their part start transmission of an overload flag. Overload Delimiter consists of eight recessive bits. The overload delimiter is of the same form as the error delimiter.

Inter frame spacing

Data frames and remote frames are separated from preceding frames by a bit field called interframe space. Overload frames and error frames are not preceded by an interframe space and multiple overload frames are not separated by an interframe space. Interframe space contains the bit fields

intermission and bus idle and, for error passive stations, which have been transmitter of the previous message, suspend transmission.

2.7 BIT STUFFING

In CAN frames, a bit of opposite polarity is inserted after five consecutive bits of the same polarity. This practice is called bit stuffing, and is due to the "Non Return to Zero" (NRZ) coding adopted. The "stuffed" data frames are destuffed by the receiver. Since bit stuffing is used, six consecutive bits of the same type (111111 or 000000) are considered an error. Bit stuffing implies that sent data frames could be larger than one would expect by simply enumerating the bits shown in the tables above.

3. ARM7 PROCESSOR

3.1 INTRODUCTION

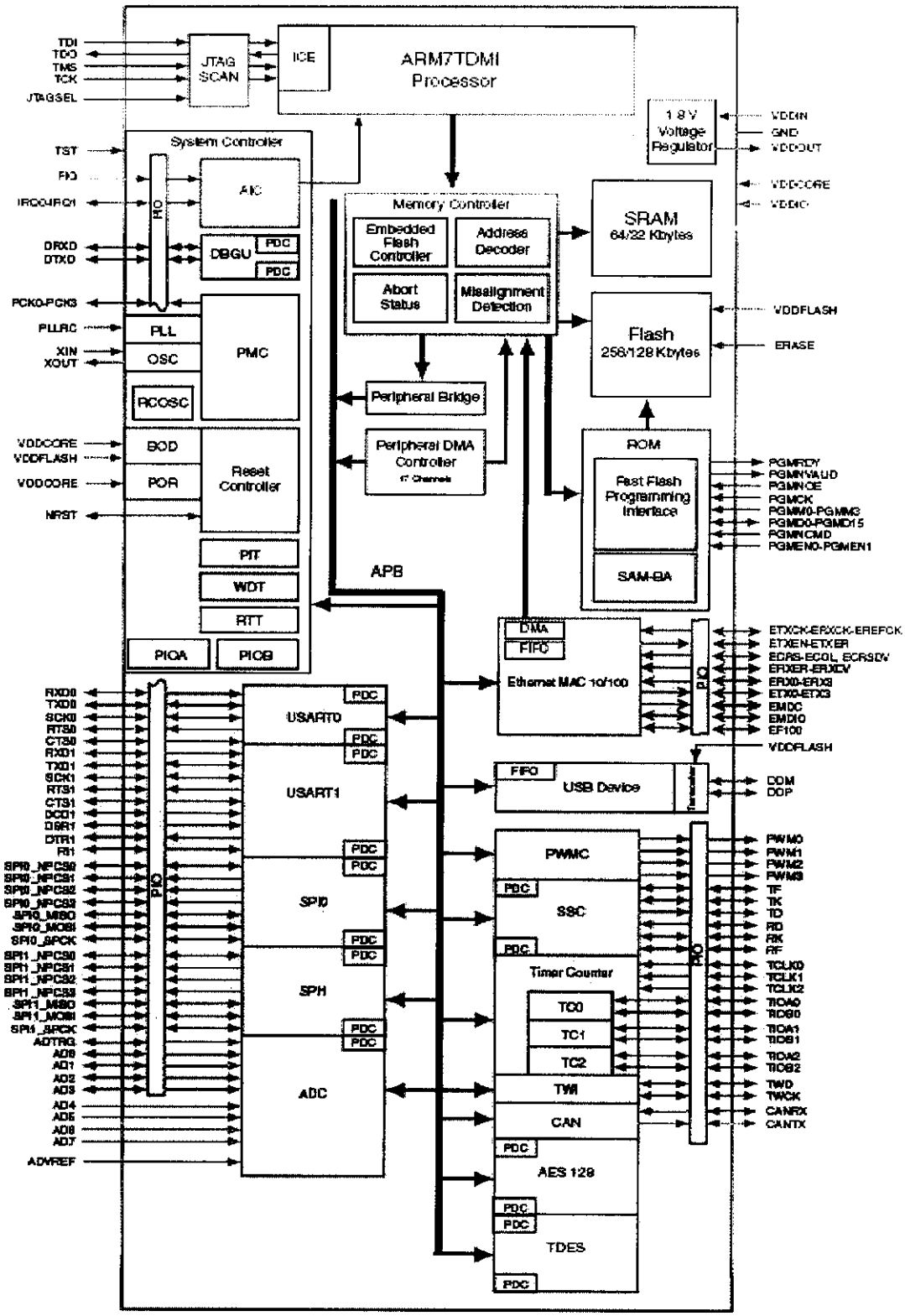
Atmel's AT91SAM7X256/128 is a member of a series of highly integrated Flash microcontrollers based on the 32-bit ARM RISC processor. It features 256/128 Kbyte high-speed Flash and 64/32 Kbyte SRAM, a large set of peripherals, including an 802.3 Ethernet MAC, a CAN controller, an AES 128 Encryption accelerator and a Triple Data Encryption System. A complete set of system functions minimizes the number of external components.

The embedded Flash memory can be programmed in-system via the JTAG-ICE interface or via a parallel interface on a production programmer prior to mounting. Built-in lock bits and a security bit protect the firmware from accidental overwrite and preserve its confidentiality.

The AT91SAM7X256/128 system controller includes a reset controller capable of managing the power-on sequence of the microcontroller and the complete system. Correct device operation can be monitored by a built-in brownout detector and a watchdog running off an integrated RC oscillator.

By combining the ARM7TDMI processor with on-chip Flash and SRAM, and a wide range of peripheral functions, including USART, SPI, CAN Controller, Ethernet MAC, AES 128 accelerator, TDES, Timer Counter, RTT and Analog-to-Digital Converters on a monolithic chip, the AT91SAM7X256/128 is a powerful device that provides a flexible, cost-effective solution to many embedded control applications requiring secure communication over, for example, Ethernet, CAN wired and Zigbee wireless networks.

Figure 3.1 Overview Of ARM7 Processor



3.3 SIGNAL DESCRIPTION:

Table 3.1 Signal Description List

Signal Name	Function	Type	Active Level	Comments
Power				
VDDIN	Voltage Regulator and ADC Power Supply Input	Power		3V to 3.6V
VDDOUT	Voltage Regulator Output	Power		1.85V
VDDFLASH	Flash and USB Power Supply	Power		3V to 3.6V
VDDIO	I/O Lines Power Supply	Power		3V to 3.6V
VDDCORE	Core Power Supply	Power		1.65V to 1.95V
VDDPLL	PLL	Power		1.65V to 1.95V
GND	Ground	Ground		
Clocks, Oscillators and PLLs				
XIN	Main Oscillator Input	Input		
XOUT	Main Oscillator Output	Output		
PLLRC	PLL Filter	Input		
PCK0 - PCK3	Programmable Clock Output	Output		
ICE and JTAG				
TCK	Test Clock	Input		No pull-up resistor
TDI	Test Data In	Input		No pull-up resistor.
TDO	Test Data Out	Output		
TMS	Test Mode Select	Input		No pull-up resistor.
JTAGSEL	JTAG Selection	Input		Pull-down resistor.
Flash Memory				
ERASE	Flash and NVM Configuration Bits Erase Command	Input	High	Pull-down resistor
Reset/Test				
NRST	Microcontroller Reset	I/O	Low	Pull-Up resistor, Open Drain Output
TST	Test Mode Select	Input	High	Pull-down resistor
Debug Unit				
DRXD	Debug Receive Data	Input		
DTXD	Debug Transmit Data	Output		
AIC				
IRQ0 - IRQ1	External Interrupt Inputs	Input		
FIQ	Fast Interrupt Input	Input		
PIO				
PA0 - PA30	Parallel IO Controller A	I/O		Pulled-up input at reset
PB0 - PB30	Parallel IO Controller B	I/O		Pulled-up input at reset

Signal Name	Function	Type	Active Level	Comments
USB Device Port				
DDM	USB Device Port Data -	Analog		
DDP	USB Device Port Data +	Analog		
USART				
SCK0 - SCK1	Serial Clock	I/O		
TXD0 - TXD1	Transmit Data	I/O		
RXD0 - RXD1	Receive Data	Input		
RTS0 - RTS1	Request To Send	Output		
CTS0 - CTS1	Clear To Send	Input		
DCD1	Data Carrier Detect	Input		
DTR1	Data Terminal Ready	Output		
DSR1	Data Set Ready	Input		
RI1	Ring Indicator	Input		
Synchronous Serial Controller				
TD	Transmit Data	Output		
RD	Receive Data	Input		
TK	Transmit Clock	I/O		
RK	Receive Clock	I/O		
TF	Transmit Frame Sync	I/O		
RF	Receive Frame Sync	I/O		
Timer/Counter				
TCLK0 - TCLK2	External Clock Inputs	Input		
TIOA0 - TIOA2	I/O Line A	I/O		
TIOB0 - TIOB2	I/O Line B	I/O		
PWM Controller				
PWM0 - PWM3	PWM Channels	Output		
Serial Peripheral Interface - SPIx				
SPIx_MISO	Master In Slave Out	I/O		
SPIx_MOSI	Master Out Slave In	I/O		
SPIx_SPCK	SPI Serial Clock	I/O		
SPIx_NPCS0	SPI Peripheral Chip Select 0	I/O	Low	
SPIx_NPCS1-NPCS3	SPI Peripheral Chip Select 1 to 3	Output	Low	
Two-wire Interface				
TWD	Two-wire Serial Data	I/O		
TWCK	Two-wire Serial Clock	I/O		

4. POWER SUPPLIES

4.1 INTRODUCTION:

The present chapter introduces the operation of power supply circuits built using filters, rectifiers, and then voltage regulators. Starting with an ac voltage, a steady dc voltage is obtained by rectifying the ac voltage, then filtering to a dc level, and finally, regulating to obtain a desired fixed dc voltage. The regulation is usually obtained from an IC voltage regulator unit, which takes a dc voltage and provides a somewhat lower dc voltage, which remains the same even if the input dc voltage varies, or the output load connected to the dc voltage changes.

A block diagram containing the parts of a typical power supply and the voltage at various points in the unit is shown in fig 4.1. The ac voltage, typically 120 V rms, is connected to a transformer, which steps that ac voltage down to the level for the desired dc output. A diode rectifier then provides a full-wave rectified voltage that is initially filtered by a simple capacitor filter to produce a dc voltage. This resulting dc voltage usually has some ripple or ac voltage variation. A regulator circuit can use this dc input to provide a dc voltage that not only has much less ripple voltage but also remains the same dc value even if the input dc voltage varies somewhat, or the load connected to the output dc voltage changes. This voltage regulation is usually obtained using one of a number of popular voltage regulator IC units.

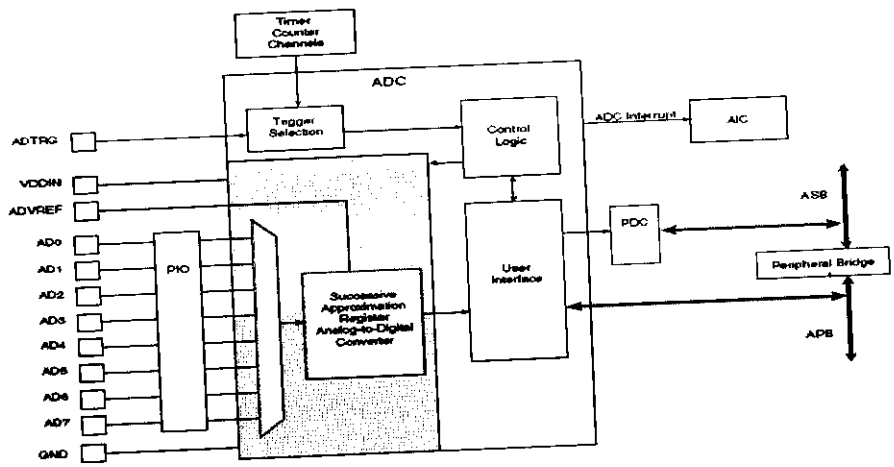
4.2 IC VOLTAGE REGULATORS:

Voltage regulators comprise a class of widely used ICs. Regulator IC units contain the circuitry for reference source, comparator amplifier, control

Pin Name	Description
VDDIN	Analog power supply
ADVREF	Reference voltage
AD0 - AD7	Analog input channels
ADTRG	External trigger

Table 3.5 ADC Pin Description

Figure 3.5 Analog-to-Digital Converter Block Diagram



3.7 Analog-to-digital Converter (ADC)

3.7.1 Overview

The ADC is based on a Successive Approximation Register (SAR) 10-bit Analog-to-Digital Converter (ADC). It also integrates an 8-to-1 analog multiplexer, making possible the analog-todigital conversions of up to eight analog lines. The conversions extend from 0V to ADVREF.

The ADC supports an 8-bit or 10-bit resolution mode, and conversion results are reported in a common register for all channels, as well as in a channel-dedicated register. Software trigger, external trigger on rising edge of the ADTRG pin or internal triggers from Timer Counter output(s) are configurable.

The ADC also integrates a Sleep Mode and a conversion sequencer and connects with a PDC channel. These features reduce both power consumption and processor intervention. Finally, the user can configure ADC timings, such as Startup Time and Sample & Hold Time

Block/Channel	Signal Name	Description
Channel Signal	XC0, XC1, XC2	External Clock Inputs
	TIOA	Capture Mode: Timer/Counter Input Waveform Mode: Timer/Counter Output
	TIOB	Capture Mode: Timer/Counter Input Waveform Mode: Timer/Counter Input/output
	INT	Interrupt Signal Output
	SYNC	Synchronization Input Signal

Table 3.3 Signal Name Description

Table 3.4 TC Pin List

Pin Name	Description	Type
TCLK0-TCLK2	External Clock Input	Input
TIOA0-TIOA2	I/O Line A	I/O
TIOB0-TIOB2	I/O Line B	I/O

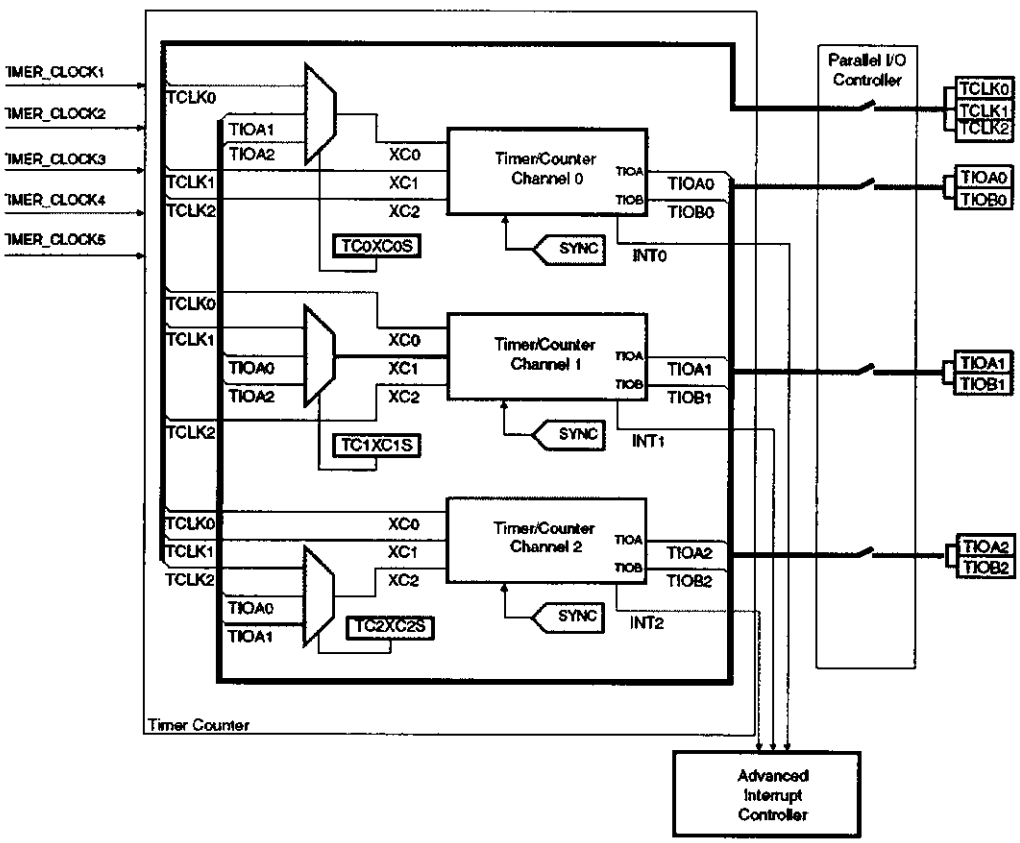
3.6 Timer/Counter (TC)

3.6.1 Overview

The Timer/Counter (TC) includes three identical 16-bit Timer/Counter channels. Each channel can be independently programmed to perform a wide range of functions including frequency measurement, event counting, interval measurement, pulse generation, delay timing and pulse width modulation. Each channel has three external clock inputs, five internal clock inputs and two multi-purpose input/output signals which can be configured by the user.

3.6.2 Block Diagram

Figure 3.4 Timer/Counter Block Diagram



3.5 Memory Controller (MC)

Overview:

The Memory Controller (MC) manages the ASB bus and controls accesses requested by the masters, typically the ARM7TDMI processor and the Peripheral DMA Controller. It features a bus arbiter, an address decoder, an abort status, a misalignment detector and an Embedded Flash Controller.

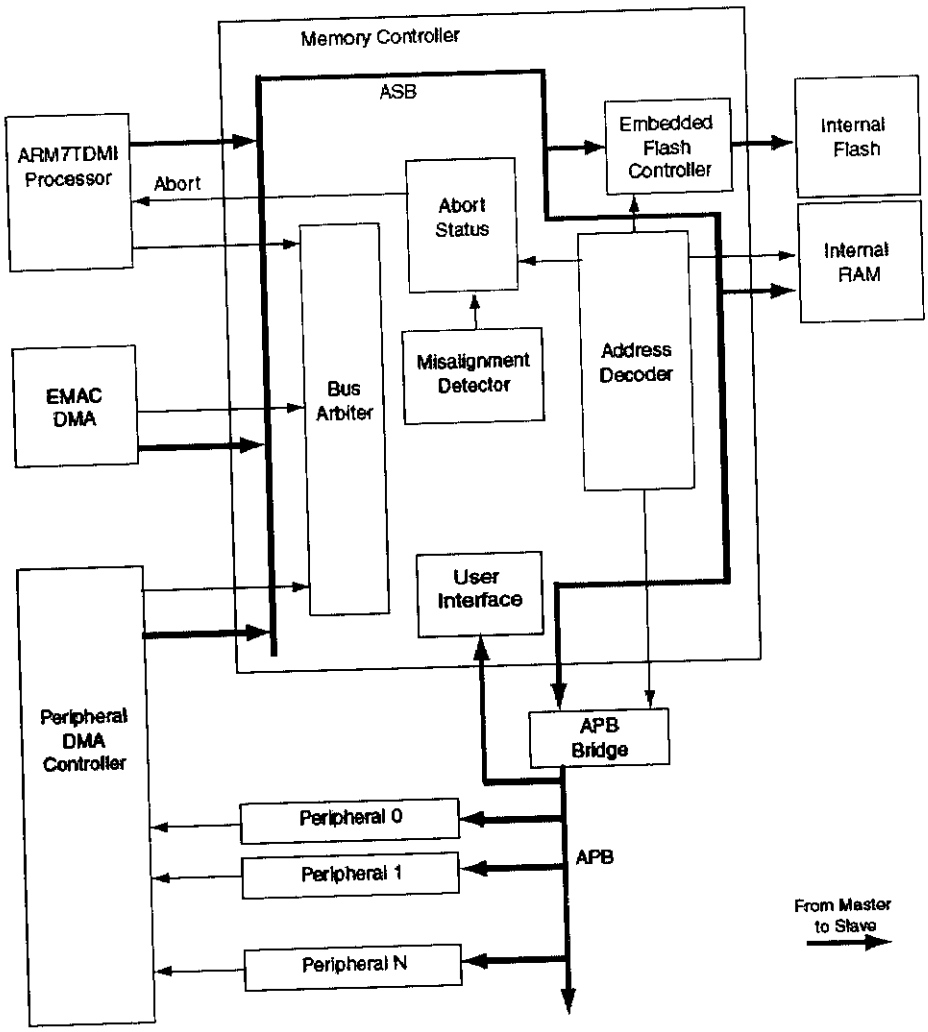


Figure 3.3 Memory Controller Block Diagram

Table 3.1 Signal Description List (Continued)

Signal Name	Function	Type	Active Level	Comments
Analog-to-Digital Converter				
AD0-AD3	Analog Inputs	Analog		Digital pulled-up inputs at reset
AD4-AD7	Analog Inputs	Analog		Analog Inputs
ADTRG	ADC Trigger	Input		
ADVREF	ADC Reference	Analog		
Fast Flash Programming Interface				
PGMEN0-PGMEN1	Programming Enabling	Input		
PGMM0-PGMM3	Programming Mode	Input		
PGMD0-PGMD15	Programming Data	I/O		
PGMRDY	Programming Ready	Output	High	
PGMNVALID	Data Direction	Output	Low	
PGMNOE	Programming Read	Input	Low	
PGMCK	Programming Clock	Input		
PGMNCMD	Programming Command	Input	Low	
CAN Controller				
CANRX	CAN Input	Input		
CANTX	CAN Output	Output		
Ethernet MAC 10/100				
EREFCK	Reference Clock	Input		RMII only
ETXCK	Transmit Clock	Input		MII only
ERXCK	Receive Clock	Input		MII only
ETXEN	Transmit Enable	Output		
ETX0 - ETX3	Transmit Data	Output		ETX0 - ETX1 only in RMII
ETXER	Transmit Coding Error	Output		MII only
ERXDV	Receive Data Valid	Input		MII only
ECRSDV	Carrier Sense and Data Valid	Input		RMII only
ERX0 - ERX3	Receive Data	Input		ERX0 - ERX1 only in RMII
ERXER	Receive Error	Input		
ECRS	Carrier Sense	Input		MII only
ECOL	Collision Detected	Input		MII only
EMDC	Management Data Clock	Output		
EMDIO	Management Data Input/Output	I/O		
EF100	Force 100 Mbits/sec.	Output	High	RMII only

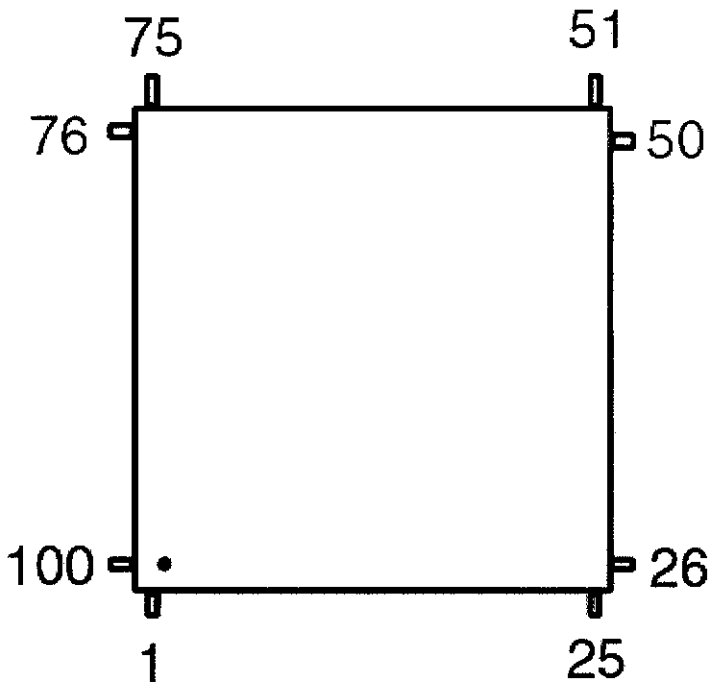
3.4 PACKAGE

The AT91SAM7X256/128 is available in 100-lead LQFP package.

100-lead LQFP Mechanical Overview

Figure 3.2 shows the orientation of the 100-lead LQFP package. A detailed mechanical description is given in the Mechanical Characteristics section of the full datasheet

Figure 3.2 100 - lead LQFP Mechanical Overview



1	ADVREF
2	GND
3	AD4
4	AD5
5	AD6
6	AD7
7	VDDOUT
8	VDDIN
9	PB27/AD0
10	PB28/AD1
11	PB29/AD2
12	PB30/AD3
13	PA8/PGMM0
14	PA9/PGMM1
15	VDDCORE
16	GND
17	VDDIO
18	PA10/PGMM2
19	PA11/PGMM3
20	PA12/PGMD0
21	PA13/PGMD1
22	PA14/PGMD2
23	PA15/PGMD3
24	PA16/PGMD4
25	PA17/PGMD5

26	PA18/PGMD6
27	PB9
28	PB8
29	PB14
30	PB13
31	PB6
32	GND
33	VDDIO
34	PB5
35	PB15
36	PB17
37	VDDCORE
38	PB7
39	PB12
40	PB0
41	PB1
42	PB2
43	PB3
44	PB10
45	PB11
46	PA19/PGMD7
47	PA20/PGMD8
48	VDDIO
49	PA21/PGMD9
50	PA22/PGMD10

Table 3.2 Pinout in 100-lead LQFP Package

51	TDI
52	GND
53	PB16
54	PB4
55	PA23/PGMD11
56	PA24/PGMD12
57	NRST
58	TST
59	PA25/PGMD13
60	PA26/PGMD14
61	VDDIO
62	VDDCORE
63	PB18
64	PB19
65	PB20
66	PB21
67	PB22
68	GND
69	PB23
70	PB24
71	PB25
72	PB26
73	PA27/PGMD15
74	PA28
75	PA29

76	TDO
77	JTAGSEL
78	TMS
79	TCK
80	PA30
81	PA0/PGMEN0
82	PA1/PGMEN1
83	GND
84	VDDIO
85	PA3
86	PA2
87	VDDCORE
88	PA4/PGMNCMD
89	PA5/PGMRDY
90	PA6/PGMNOE
91	PA7/PGMNVALID
92	ERASE
93	DDM
94	DDP
95	VDDFLASH
96	GND
97	XIN/PGMCK
98	XOUT
99	PLLRC
100	VDDPLL

device, and overload protection all in a single IC. Although the internal construction of the IC is somewhat different from that described for discrete voltage regulator circuits, the external operation is much the same. IC units provide regulation of either a fixed positive voltage, a fixed negative voltage, or an adjustably set voltage.

4.3 THREE-TERMINAL VOLTAGE REGULATORS:

Fig shows the basic connection of a three-terminal voltage regulator IC to a load. The fixed voltage regulator has an unregulated dc input voltage, Vi, applied to one input terminal, a regulated output dc voltage, Vo, from a second terminal, with the third terminal connected to ground. For a selected regulator, IC device specifications list a voltage range over which the input voltage can vary to maintain a regulated output voltage over a range of load current. The specifications also list the amount of output voltage change resulting from a change in load current (load regulation) or in input voltage (line regulation)

Fixed Positive Voltage Regulators:

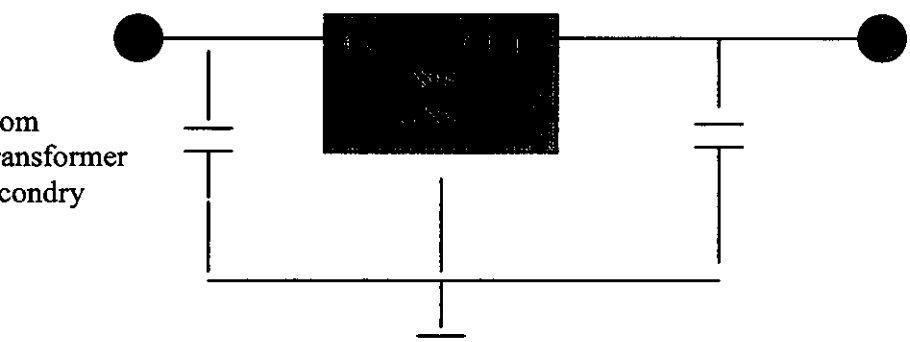


Figure 4.1

The series 78 regulators provide fixed regulated voltages from 5 to 24 V. Figure 19.26 shows how one such IC, a 7812, is connected to provide voltage regulation with output from this unit of +12V dc. An unregulated input voltage Vi is filtered by capacitor C1 and connected to the IC's IN terminal. The IC's OUT terminal provides a regulated + 12V which is filtered by capacitor C2 (mostly for any high-frequency noise). The third IC terminal is connected to ground (GND). While the input voltage may vary over some permissible voltage range, and the output load may vary over some acceptable range, the output voltage remains constant within specified voltage variation limits. These limitations are spelled out in the manufacturer's specification sheets. A table of positive voltage regulated ICs is provided in table 4.1.

TABLE 4.1 Positive Voltage Regulators in 7800 series

IC Part	**Output Voltage (V)**	**Minimum Vi (V)**
7805	+5	7.3
7806	+6	8.3
7808	+8	10.5
7810	+10	12.5
7812	+12	14.6
7815	+15	17.7

7818	+18	21.0
7824	+24	27.1

4.4 Block diagram

The ac voltage, typically 220V rms, is connected to a transformer, which steps that ac voltage down to the level of the desired dc output. A diode rectifier then provides a full-wave rectified voltage that is initially filtered by a simple capacitor filter to produce a dc voltage. This resulting dc voltage usually has some ripple or ac voltage variation.

A regulator circuit removes the ripples and also remains the same dc value even if the input dc voltage varies, or the load connected to the output dc voltage changes. This voltage regulation is usually obtained using one of the popular voltage regulator IC units.

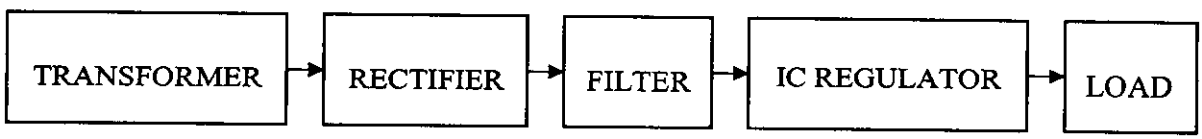


Figure 4.2 Block diagram (Power supply)

4.5 Working principle

Transformer

The potential transformer will step down the power supply voltage (0-230V) to (0-6V) level. Then the secondary of the potential transformer will be connected to the precision rectifier, which is constructed with the help of op–amp. The advantages of using precision rectifier are it will give peak voltage output as DC, rest of the circuits will give only RMS output.

Bridge rectifier

When four diodes are connected as shown in figure, the circuit is called as bridge rectifier. The input to the circuit is applied to the diagonally opposite corners of the network, and the output is taken from the remaining two corners.

Let us assume that the transformer is working properly and there is a positive potential, at point A and a negative potential at point B. the positive potential at point A will forward bias D3 and reverse bias D4.

The negative potential at point B will forward bias D1 and reverse D2. At this time D3 and D1 are forward biased and will allow current flow to pass through them; D4 and D2 are reverse biased and will block current flow.

The path for current flow is from point B through D1, up through RL, through D3, through the secondary of the transformer back to point B. this path is indicated by the solid arrows. Waveforms (1) and (2) can be observed across D1 and D3.

One-half cycle later the polarity across the secondary of the transformer reverse, forward biasing D2 and D4 and reverse biasing D1 and D3. Current flow will now be from point A through D4, up through RL,

through D2, through the secondary of T1, and back to point A. This path is indicated by the broken arrows. Waveforms (3) and (4) can be observed across D2 and D4.

The current flow through RL is always in the same direction. In flowing through RL this current develops a voltage corresponding to that shown waveform (5). Since current flows through the load (RL) during both half cycles of the applied voltage, this bridge rectifier is a full-wave rectifier.

One advantage of a bridge rectifier over a conventional full-wave rectifier is that with a given transformer the bridge rectifier produces a voltage output that is nearly twice that of the conventional full-wave circuit.

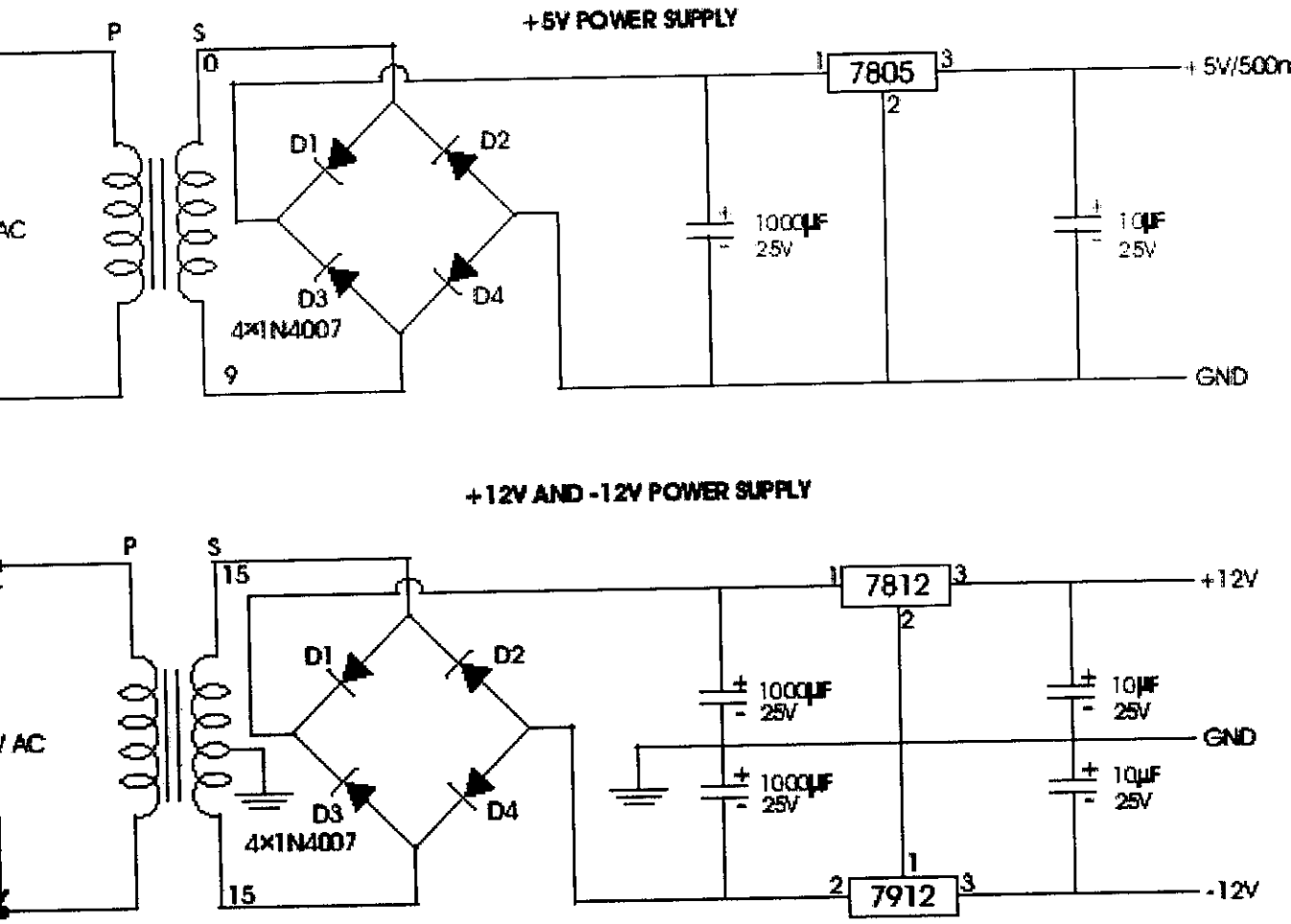
This may be shown by assigning values to some of the components shown in views A and B. assume that the same transformer is used in both circuits. The peak voltage developed between points X and y is 1000 volts in both circuits.

In the conventional full-wave circuit shown—in view A, the peak voltage from the center tap to either X or Y is 500 volts. Since only one diode can conduct at any instant, the maximum voltage that can be rectified at any instant is 500 volts.

The maximum voltage that appears across the load resistor is nearly-but never exceeds-500 v0lts, as result of the small voltage drop across the diode. In the bridge rectifier shown in view B, the maximum voltage that can be rectified is the full secondary voltage, which is 1000 volts.

Therefore, the peak output voltage across the load resistor is nearly 1000 volts. With both circuits using the same transformer, the bridge rectifier circuit produces a higher output voltage than the conventional full-wave rectifier circuit.

Figure 4.3 Circuit diagram (Power supply)



5 . RS-232 SERIAL COMMUNICATION

5.1 OVERVIEW:

In the 1960s as the use of time-share computer terminals became more widespread, modems were developed so that terminals could use phone lines to communicate with distant computers. As we stated earlier, modems and other devices used to send serial data are often referred to as data communication equipment or DCE. The terminals or computers that are sending or receiving the data are referred to as data terminal equipment or DTE. In response to the need for signal and handshake standards between DTE and DCE, the Electronic Industries Association (EIA) developed EIA standard RS-232C. This standard describes the functions of 25 signal and handshake pins for serial data transfer. It also describes the voltage levels. Impedance levels, rise and fall times, maximum bit rate, and maximum capacitance for these signal lines. Before we work our way through the 25 pin functions, we will take a brief look at some of the other hardware aspects of RS-232C.

RS-232C specifies 25 signal pins and it specifies that the DTE connector should be a male, and the DCE connector should be a female. A specific connector is not given, but the most commonly used connectors are the DB-25P male and the DB-25S female is used. When you are wiring up these connectors. It is important to note the order in which the pins are numbered.

The voltage levels for all RS-232C signals are as follows. A logic high, or mark, is a voltage between –3V and –15 V under load (-25 V no load). A logic low or space is a voltage between +3 V and +15 under load (+25 V no load). Voltage such as ±12 V are commonly used.

RS-422A, RS-423A, and RS-449

A major problem with RS-232C is that it can only transmit data reliable about 50 ft [16.4 m] at its maximum rate of 20,000 Bd. If longer lines are used, the transmission rate has to be drastically reduced. This limitation is caused by the open signal lines with a single common ground that are used for rs-232c.

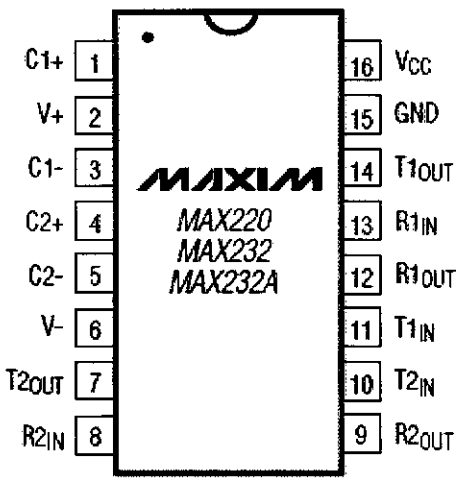
A newer standard, RS –422A specifies that each signal will be sent differentially over two adjacent wires in a ribbon cable or a twisted pair of wires as shown in Figure 13-11 a. Differential signals are produced by differential line drivers such as the MC 3487 and translated back to TTL levels by differential line receivers such as the MC3486. Data rates for this standard are 10 MBd for a distance of 50 ft [1220 m]. The higher transmission rate of RS-422A is possible because the differential lines are terminated by resistors so they act as transmission lines instead of simply open wires. A further advantage of differential signals is that any common-mode electrical noise induced in the two lines will be rejected by the differential line receiver. For RS-422A a logic high or mark is indicated by the B signal line being more positive than the A signal line. A logic low or space is indicated by the A signal line being more positive than the B signal line. The voltage difference between the two lines must be greater than 0.4 V, but not greater than 12 V. The common-mode voltage on the signal lines must be in the range of –7 V to +7 V.

Another EIA standard intended to solve the speed and distance problems of RS-232C is RS-423A. This standard specifies a low impedance signal-ended signal instead of the differential signal of RS-422A. The low-impedance signal can be sent over 50-Ω coaxial cable and terminated at the receiving end to prevent reflections. Figure 13-11b shows how an MC3487 driver and MC3486 receiver can be connected to produce the required signals. A logic high in this standard is represented by the A line being between 4 and 6 V negative with respect to the B line (ground), and a logic low is represented by the A line being 4-6 V positive with respect to ground.

The RS-422A and RS-423A standards do not specify connector pin numbers or handshake signals the way that RS-232C does. An additional EIA standard called RS-449 does this or the two. RS-449 specifies 37 signal pins on a main connector and 9 additional pins on an optional connector. The signals on these connectors are a superset of the RS-232C signals so adapters can be used to interface RS-232C equipment with RS-449 equipment. Still another EIA standard, RS-366, incorporates signals for automatic telephone dialing with modems.

5.1 PIN CONFIGURATION

Figure 5.1 MAX232 pinout



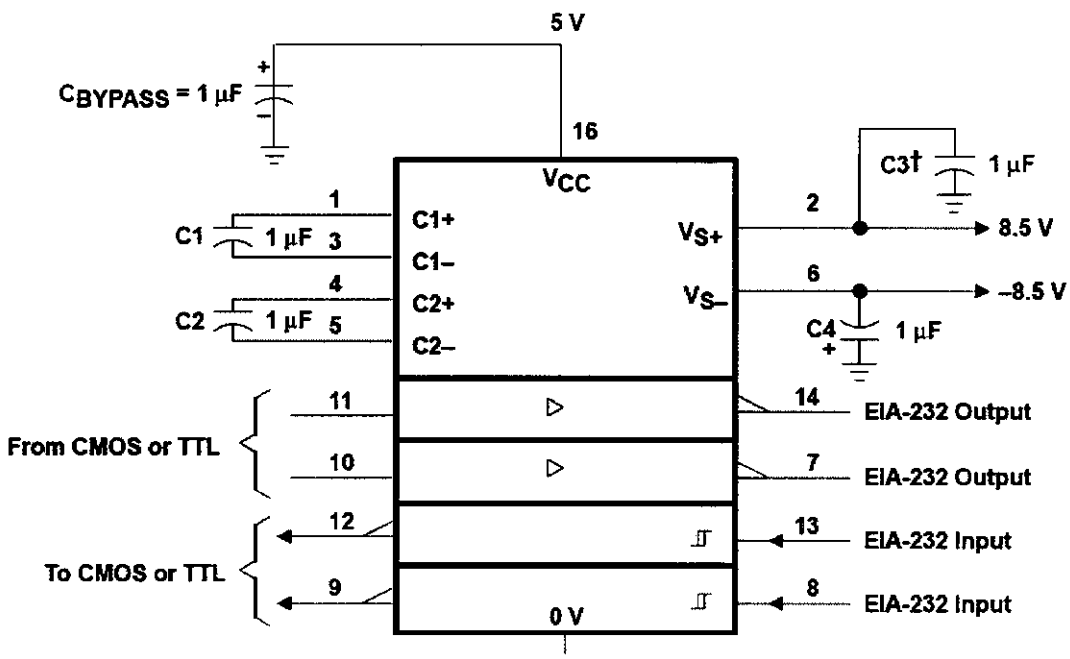


Figure 5.2 Typical Operating Circuit

6. SYSTEM DESIGN

The entire project has been subdivided into five main modules.

1.Input (nodes it is fuel meter,speedometer,etc)

2.ARM7 processor module

3.CAN Tansceiver module

4.Serialcommunicationmodule

5.GUI

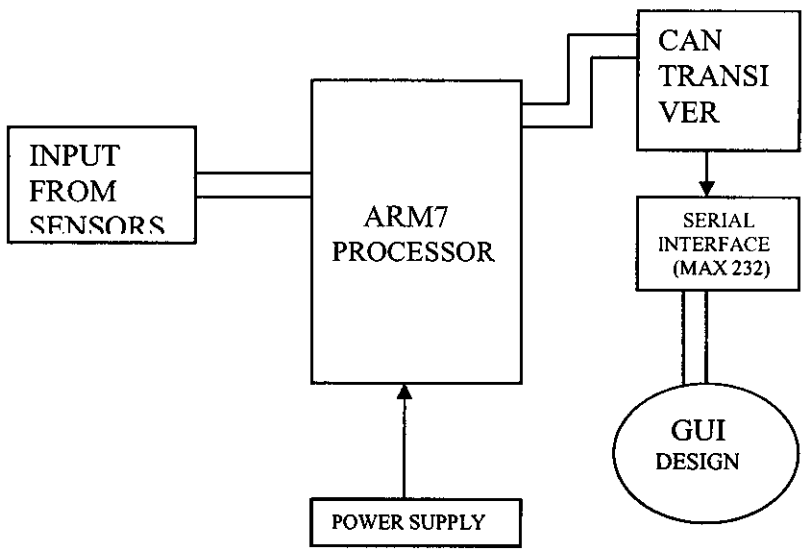


Figure 6.1 system Design

7. SOFTWARE IMPLEMENTATION

ALGORITHM:

Step1: Get the input from sensor(potentiometer).

Step2: Another input from the sensor is given to the timer interrupt.

Step3: First input is given to ADC to convert the analog data into the digital data.

Step4: Digital value is stored in temporary variable.

Step5: Temporary variable is moved to CAN data bytes.

Step6: CAN transceiver transmits the data from ARM7 processor.

Step7: PIC receives the (81258) CAN data frame.

Step8: CAN data frames are moved to another temporary variable using PIC processor.

Step9: The temporary variables are transmit to pc by using serial communication.

Step10: MAX232 driver IC is used to transmit/receive data for serial communication.

Step11: Serial communication is used as an interface between pc and hardware.

Step12: The data received can be viewed as an output in GUI.

8.FUTURE ENHANCEMENT

This project can be enhanced and can be made very efficient by adding more sensors and finding more parameters. The network can be extended to nearly hundred processors. This will be very efficient when used in unmanned aircrafts. When applying in automobiles we can use sensors for fuel flow in engine, to find coolant level, to find obstacles around the automobile and so on. In future there may be unmanned automatic cars or cars may be controlled using PCs through internet, in that case these parameters can be found and sent to master and can also be sent to PC through some means.

9.CONCLUSION

Thus the proposed project " CAN INTERFACE WITH ARM7" was successfully implemented in the lab using ARM processor, programming for ARM was done using 'C' language. We used only two sensors but number of nodes can be introduced in this there will not be any interruption due to the priority assignment for each node. Thus the study can be further explored by utilizing the features of the CAN and ARM completely.

APPENDIX

```
#include <p18f258.h>
#include "j1939.h"
#include <delays.h>
#include <timers.h>

#define KEY_OK          PORTAbits.RA0
#define KEY_UP      PORTAbits.RA1
#define KEY_LEFT        PORTAbits.RA2
#define KEY_DOWN    PORTAbits.RA3
#define KEY_RIGHT    PORTAbits.RA4

#define OK          1
#define UP      2
#define LEFT        3
#define DOWN    4
#define RIGHT   5

#define TACHO    0x01
#define SPEED    0x10
#define FUEL     0x30
#define GEAR     0x40

#define CHANNEL1      128
#define CHANNEL2      129
#define CHANNEL3      130
#define CHANNEL4      131

#define SCL     PORTCbits.RC3       // I2C clock
#define SDA     PORTCbits.RC4       // I2C data

#define OTHER_NODE 201
#define NODE 255

#define LED_D2   PORTBbits.RB0
#define LED_D3   PORTBbits.RB1
#define LED_D4   PORTBbits.RB4
```

```
#define LED_D5   PORTBbits.RB5

#define ON  1
#define OFF 0

unsigned char para_string[] = " Km/h      RPM     Ltrs  Gear:";
unsigned char km_string[] = "Km/h";
unsigned char rpm_string[] = "RPM";
unsigned char ltr_string[] = "Ltrs";
unsigned char gear_string[] ="Gear";

char title_string[] = "CAN Transmitter";
char press_ok[] = "Press OK to Start...";
char next_string[] = "  Press ->";
char error1[] = "Can not find CA...";
char demo_string[] = "Demo mode";
char practical_string[] = "Practical mode";
char pressOK_string[] = "To Select Press OK...";

void cursorset(unsigned char posx,unsigned char posy);
void Displaykey(unsigned char value);
void clearscreen(void);
void scroll_on(void);
void scroll_off(void);
void lcd(unsigned char *ptr);
int Transmit(unsigned char data);
int Start(void);
void Stop(void);
void const_disp(unsigned int value);
void display_char(unsigned char value);
void Init_MCU( void );
char keyscan(void);
void Demo_Mode(void);


unsigned int data_msb = 0;
unsigned char data_lsb = 0;
float data_rpm = 0.0;
float data_speed = 0.0;
float data_ltrs = 0.0;
```

```
unsigned char data_gear = 0;

unsigned char parameter = 0;
unsigned char para_id[5] = {0};
unsigned char msg_cnt = 0;
unsigned char Txdata1 = 0,Txdata2 = 0,Txdata3 = 0,Txdata4 = 0,Txdata5 =
0;
unsigned char ms_cnt = 0;
unsigned int data_buf[10] = {0};

unsigned char SOURCE_ADDRESS = 0;
unsigned char dest_add = 0;
unsigned char t[10] = {0};
unsigned char a,b,c,d;
unsigned char i=0,j=0,k=0,a,b,c;
unsigned char value1;
unsigned int cnt = 0;
unsigned char clear_flag = 0;
unsigned char mainscreen = 0;
unsigned char demo = 0,practical = 0;

J1939_MESSAGE MsgR, MsgT;

void LED_Toggle(void);
//void can_transmit (unsigned char data1,unsigned char data2,unsigned char
data3);
void can_transmit (unsigned char data1,unsigned char data2,unsigned char
data3,unsigned char data4,unsigned char data5);
unsigned char msg_update(void);

void InterruptHandlerLow (void);

// Low priority interrupt vector
#pragma code InterruptVectorLow = 0x0018
void InterruptVectorLow( void )
{
  _asm
    goto InterruptHandlerLow
  _endasm
}
```

```
// Low priority interrupt routine
#pragma code
#pragma interruptlow InterruptHandlerLow
void InterruptHandlerLow( void )
{
	ms_cnt++;
	if (PIR3 != 0x00)
		J1939_ISR();
	if (PIR1bits.TMR1IF)
	{
		PIR1bits.TMR1IF = 0;
	}
}

void can_transmit (unsigned char data1,unsigned char data2,unsigned char
data3,unsigned char data4,unsigned char data5)
{
	MsgT.Priority			= J1939_CONTROL_PRIORITY;
	MsgT.DataPage			= 0;
	MsgT.PDUFormat				= 254;
	MsgT.DestinationAddress = 200;
	MsgT.SourceAddress		= NODE;
	MsgT.DataLength		= 8;

	MsgT.Data[0]			= data1;		//	Tacho
MSB
	MsgT.Data[1]			= data2;		//	Tacho
LSB
	MsgT.Data[2]			= data3;		// Speed
	MsgT.Data[3]			= data4;		// Fuel
	MsgT.Data[4]			= data5;		// Gear


	cursorset(11,1);
	for(i=0;i<4;i++)
		display_char(' ');

	cursorset(1,1);
	for(i=0;i<4;i++)
```

```
        display_char(' ');

    cursorset(1,2);
    for(i=0;i<4;i++)
        display_char(' ');


    data_msb = MsgT.Data[0];
    data_lsb = MsgT.Data[1];

    data_msb = data_msb << 8;
    data_msb = data_msb | data_lsb;
    data_rpm = data_msb * 0.122;

    cursorset(11,1);
    const_disp(data_rpm);                          // display RPM

    data_speed = MsgT.Data[2] * 0.627;
    cursorset(1,1);
    const_disp(data_speed);

    data_ltrs = MsgT.Data[3] * 0.158;
    cursorset(1,2);
    const_disp(data_ltrs);

    data_gear = MsgT.Data[4];
    cursorset(16,2);

    if(data_gear == 0)
        display_char('N');

    if(data_gear == 6)
        display_char('R');

    else if(0 > data_gear < 6)
        const_disp(data_gear);

    if( J1939_EnqueueMessage ( &MsgT ) != RC_SUCCESS );
        LED_Toggle();
}
```

```
void Init_MCU( void )
{
	TRISAbits.TRISA2=0;
	TRISAbits.TRISA1=0;
	TRISC = 0x00;
	TRISB = 0x04;
	PORTB = 0x33;
	ADCON1 = 0x07;
	TRISA = 0x1F;
}

void main( void )
{
	unsigned char i;
	INTCON2bits.RBPU = 0;
	TRISB = 0;
	PORTB = 0;
	TRISC = 0x00;
	TRISAbits.TRISA2=0;
	TRISAbits.TRISA1=0;

	ADCON1 = 0x07;
	TRISA = 0x1F;

	J1939_Initialization( TRUE );
	INTCONbits.PEIE = 1;		// Enable perip heral interrupts
	INTCONbits.GIE = 1;			// Enable global interrupts
	while (J1939_Flags.WaitingForAddressClaimContention)
		J1939_Poll(5);

	para_id[1] = TACHO;
	para_id[2] = SPEED;
	para_id[3] = FUEL;
	para_id[4] = GEAR;

	clearscreen();
	lcd(&title_string[0]);

	Delay10KTCYx(100);
```

```
	WriteTimer1(15535);

	OpenTimer1 ( TIMER_INT_ON & T1_16BIT_RW & T1_SOURCE_INT & T1_PS_1_1 & T1_OSC1EN_OFF & T1_SYNC_EXT_OFF);

	LED_D2 = OFF;
	LED_D3 = OFF;
	LED_D4 = OFF;
	LED_D5 = OFF;
	mainscreen =1;
	clearscreen();
	lcd(&pressOK_string[0]);
	cursorset(1,2);
	lcd(&demo_string[0]);

	while((keyscan() != OK) && (keyscan() != DOWN));
	while(((keyscan() == OK) || (keyscan() == DOWN)) || (mainscreen == 1))
	{
		if(keyscan() == OK)
		{
			clearscreen();
			for(i=0;i<5;i++)
				display_char(' ');
			lcd(&demo_string[0]);
			mainscreen = 0;
			demo = 1;
			practical = 0;
		}

		if(keyscan() == DOWN)
		{
			cursorset(1,2);
			lcd(&practical_string[0]);
			while((keyscan() != OK) && (keyscan() != UP));
			if(keyscan() == OK)
			{
				clearscreen();
```

```
                        for(i =0 ;i < 3; i++)
                                display_char(' ');
                        lcd(&practical_string[0]);
                        mainscreen = 0;
                        demo = 0;
                        practical = 1;
                }

                if(keyscan() == UP)
                {
                        cursorset(1,2);
                        lcd(&demo_string[0]);
                        for(i = 0; i < 5; i++)
                                display_char(' ');
                }
        }
    }
    clearscreen();
    cursorset(4,1);
    lcd(&para_string[0]);
    data_buf[2] = 0xFF;
    while(1)
    {
        J1939_Initialization( TRUE );
        while (J1939_Flags.WaitingForAddressClaimContention)
            J1939_Poll(5);
        if(demo == 1)
            Demo_Mode();
    }
}

unsigned char msg_update(void)
{
    msg_cnt++;
    if(msg_cnt >= 5)
    {
        msg_cnt = 1;
    }
    return msg_cnt;
}
```

```
void Demo_Mode(void)
{
	unsigned int canvalue[10] = {0};

	if(data_buf[0] <= 0xFFFF)
		data_buf[0] = data_buf[0] + 2050;		// Tacho

	else
		data_buf[0] = 0;

	canvalue[1] = data_buf[0];

	Txdata2 = canvalue[1] & 0xFF;
	Txdata1 = canvalue[1] >> 8;

	if(data_buf[1] <= 0xFF)
		data_buf[1] = data_buf[1] + 8;		// Speed

	else
		data_buf[1] = 0;

	canvalue[2] = data_buf[1];

	Txdata3 = canvalue[2] & 0xFF;

	if(data_buf[2] >= 0x00)
		data_buf[2] = data_buf[2] - 32;		// Fuel
	else
		data_buf[2] = 0xFF;
	canvalue[3] = data_buf[2];
	Txdata4 = canvalue[3] & 0xFF;


	if(data_buf[3] <= 0x05)
		data_buf[3] = data_buf[3] + 1;		// Gear

	else
		data_buf[3] = 0;

		canvalue[4] = data_buf[3];
```

```
        Txdata5 = canvalue[4] & 0xFF;

        can_transmit (Txdata1,Txdata2,Txdata3,Txdata4,Txdata5);
}

void LED_Toggle(void)
{
        LED_D2 = ~LED_D2;
        LED_D3 = ~LED_D3;
        LED_D4 = ~LED_D4;
        LED_D5 = ~LED_D5;
}

void clearscreen(void)
{
    Start();
    Transmit(0x50);
    Transmit(0xFE);
    Transmit(0x58);
    Stop();
}

void display_char(unsigned char value)
{
   Start();
   Transmit(0x50);
   Transmit(value);
   Stop();
}

void lcd(unsigned char *ptr)
{

        while(*ptr != '\0')
        {
           Start();
           Transmit(0x50);
           Transmit(*ptr);
           Stop();
```

```
                ptr++;
        }
}

void cursorset(unsigned char posx,unsigned char posy)
{
    Start();

        Transmit(0x50);
        Transmit(0xFE);
        Transmit(0x47);
        Transmit(posx);
        Transmit(posy);

        Stop();

}

void scroll_on(void)
{
                Start();
        Transmit(0x50);
        Transmit(0xFE);
        Transmit(0x51);
        Stop();
}

void scroll_off(void)
{
                Start();
        Transmit(0x50);
        Transmit(0xFE);
        Transmit(0x52);
        Stop();
}
```

```
int Transmit(unsigned char data)
{


        a = 0x80;
        for(b=0;b<8;b++)
        {   c = 0;
                c = data & a;
                if(c==0)
                SDA=0;
                else
                SDA=1;

                for(d=0;d<50;d++);
                SCL=1;
                for(d=0;d<50;d++);
                SCL=0;
                a = a>>1;
        }
                SDA=1;
                for(d=0;d<100;d++);
                SCL=1;
                for(d=0;d<100;d++);
                SCL=0;

                return 0;
}

int Start(void)
{
        SDA=1;
        SCL=1;
        for(a=0; a<100;a++);
        SDA=0;
        for(a=0; a<100;a++);
        SCL=0;
        return 0;
}
```

```
void Stop(void)
{
        SDA=0;
        for(a=0; a<100;a++);
        SCL=1;
        for(a=0; a<100;a++);
        SDA=1;
}

void const_disp(unsigned int value)
{
   cnt=1;
   while(value != 0)
        {
                value1 = value % 10;
                t[cnt] = value1 + 48;
                value = value/10;
                cnt++;
        }
        cnt--;
        while(cnt>0)
        {
           Start();
           Transmit(0x50);
           Transmit(t[cnt]);
                Stop();
       cnt--;
        }
}

 char keyscan(void)
 {
        unsigned char scancode;

        if(KEY_OK == 0)
                scancode=1;

        else if(KEY_UP == 0)
                scancode=2;
```

```
    else if(KEY_LEFT == 0)
        scancode=3;

    else if(KEY_DOWN == 0)
        scancode=4;

    else if(KEY_RIGHT == 0)
        scancode=5;

    else
        scancode=0;

    return scancode;
}
```

REFERENCES

1. Bryon Gottfried(2001) Programming with 'C',Tata McGraw Hill .

2. Design with microcontrollers, Peatman

3. Douglas V. Hall (PIC microcontroller)

4. Visual basic 6 by kruglinski

5. Technical reference manual (Eectronics for you)

6. www.Discover circuit .com

7. www.Google . com (T91SAM7X256/AT91SAM7X128)

8. www.trioztech.com