ADAPTIVE TRANSPORTATION MANAGEMENT SYSTEM (ATMS)

A PROJECT REPORT

Submitted by

PRASAANTH.S.P	**Reg. No.:1110107068**
SABARINATHAN.B	**Reg. No.:1110107075**
SRINIVASAN.C	**Reg. No.:1110107102**
VENGATESH.B	**Reg. No.:1110107111**

In partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE-641049

(An Autonomous Institution Affiliated to Anna University, Chennai)

APRIL 2015

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE-641049

(An Autonomous Institution Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report titled **“ADAPTIVE TRANSPORTATION MANAGEMENT SYSTEM”** is the bonafide work of **“PRASAANTH.S.P, SABARINATHAN.B, SRINIVASAN.C, VENGATESH.B”** who carried out the project work under my supervision.

SIGNATURE
Dr.A.Vasuki M.E., Ph.D.,
Professor
Electronics & Communication Engineering
Kumaraguru College of Technology
Coimbatore.

SIGNATURE
Dr. Rajeswari Mariappan M.E., Ph.D.,
HEAD OF THE DEPARTMENT
Electronics & Communication Engineering
Kumaraguru College of Technology
Coimbatore.

The candidates with Register numbers 1110107068, 1110107075, 1110107102 and 1110107111 are examined by us in the project viva-voce examination held on …………………….

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

First we would like to express our praise and gratitude to the Lord, who has showered his grace and blessing enabling us to complete this project in an excellent manner. He has made all things in beautiful in his time.

We express our sincere thanks to our beloved Joint Correspondent,
Shri. Shankar Vanavarayar for his kind support and for providing necessary facilities to carry out the project work.

We would like to express our sincere thanks to our beloved Principal **Dr. R. S. Kumar M.E., Ph.D.,** who encouraged us with his valuable thoughts.

We would like to express our sincere thanks and deep sense of gratitude to our HOD, **Dr. Rajeswari Mariappan M.E., Ph.D.,** for her valuable suggestions and encouragement which paved way for the successful completion of the project.

We are greatly privileged to express our deep sense of gratitude to the Project Guide, **Dr.A.Vasuki M.E., Ph.D.**, Professor, for her continuous support throughout the project.

In particular, we wish to thank and express our everlasting gratitude to the Project Coordinator **Mrs.A.Kalaiselvi M.E., (Ph.D.,)** Assistant Professor for her expert counselling in each and every steps of project work and we wish to convey our deep sense of gratitude to all teaching and non-teaching staff members of ECE Department for their help and cooperation.

Finally, we thank our parents and our family members for giving us the moral support in all of our activities and our dear friends who helped us to endure our difficult times with their unfailing support and warm wishes

ABSTRACT

ADAPTIVE TRANSPORTATION MANAGEMENT SYSTEM (ATMS) provides automation on the roads which aims to regulate traffic in the best possible way. It is assisted by RFID technology which helps in tagging each vehicle in accordance to their vehicle registration number. The reader placed at the strategic points receives the information from the RFID tag and facilitates tracking, analysis of traffic data and adaptively regulating it. Moreover the project provides a sort of priority to the emergency vehicles such as VIP vehicles, ambulance, fire engines etc. i.e. when these vehicle approaches the junction, the reader senses it and regulates the traffic accordingly.

The project targets to create a vehicular database so that with the help of single RFID, the entire history of the vehicle and its owner can be obtained needless of other documents. This project thus eliminates the need for human intervention that arises due to the emergency situations by automating it and thereby increasing their efficiency.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

GPS	**-**	**G**lobal **P**ositioning **S**ystem
GSM	**-**	**G**lobal **S**ystem for **M**obiles
I2C	**-**	**I**nter **I**ntegrated **C**ircuits
IDE	**-**	**I**ntegrated **D**evelopment **E**nvironment
PIC	**-**	**P**eripheral Interface **C**ontroller
Rx	**-**	**R**eceiver
SPI	**-**	**S**erial **P**eripheral **I**nterface
Tx	**-**	**T**ransmitter
UART	**-**	**U**niversal **A**synchronous **R**eceiver **T**ransmitter

CHAPTER 1

INTRODUCTION

1.1 NEED FOR THE PROJECT

The need of the hour is a system to manage traffic in the urban areas. Increasing vehicle population is a major threat and also the reason for the traffic congestion in the urban areas. People cannot reach their destination at the correct time due to traffic. Though traffic cannot be fully avoided, it can be reduced. Our project aims to solve this crisis by making use of the traffic signals "intelligently" thereby sorting out the congestion problem.

1.2 BLOCK DIAGRAM:

The overall block diagram of the system is shown in Fig 1.1

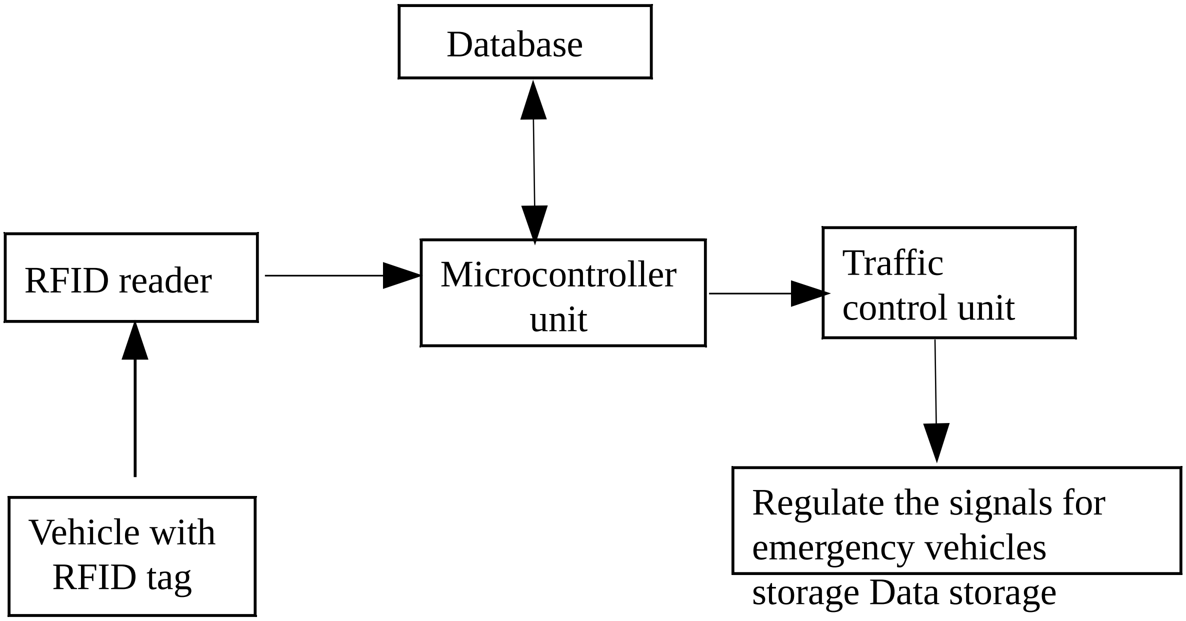


Fig 1.1 Simplified Block Diagram of ATMS

The blocks involved are the RFID reader, Microcontroller unit, Traffic control unit and database.

The RFID block consists of reader and tag where the tag is fixed in every vehicle. The reader is installed in the traffic signals. The tag present in vehicles are read at this juncture.

Microcontroller used here is Raspberry Pi. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It directs the entire process. It gets the details from the reader and compares it with the database to perform adequate actions.

Creation of file, which contains the entire database of vehicles. Several files can be created in which the details of the vehicles can be loaded in it. Each file contains the details of a particular vehicle and the history of the vehicle is updated periodically.

Signal unit which has the traffic signal was emulated by PIC. The micro controller receives the tag information from the reader and loads information into the file. Based on the pre-loaded program in the micro controller the necessary traffic signal changes takes place**.**

The traffic signal is emulated using PIC16F877A**.** PIC16F877A features 256 bytes of EEPROM data memory, self-programming, an ICD, 2 Comparators, 8 channels of 10-bit Analog-to-Digital (A/D) converter, 2 capture/compare/PWM functions, the synchronous serial port can be configured as either 3-wire Serial Peripheral Interface or the 2-wire Inter-Integrated Circuit bus and a Universal Asynchronous Receiver Transmitter (USART). Pins are assigned in PIC in such a way all the traffic signal changes are emulated.

The tags used in our system are passive RFID tags which does not require an active power source but it can be read only within the small distance. However for practical implementation of the project active tags can be used which has advantage of covering larger area.

In Chapter 2 and 3, the hardware components required and the software tools required for the implementation of the project is discussed. Chapter 4 deals with the working of the entire system and Chapter 5 concludes the project.

CHAPTER 2

HARDWARE DESCRIPTION

2.1 INTRODUCTION

The hardware components present in the ATMS are microcontroller, a seven segment display and an RFID module. The heart of the system is the microcontroller which directs the entire system.

2.2 RASPBERRY PI

The Raspberry Pi is a series of credit card-sized single-board computers developed in the UK by the Raspberry Pi Foundation with the intention of promoting the teaching of basic computer science in schools. The image of Raspberry Pi is shown in the Fig 2.1

Fig.2.1 Raspberry Pi

2.2.1 PERFORMANCE OF PRE-PI 2 MODELS

On the CPU level the performance is similar to a 300 MHz Pentium II of 1997-1999. The GPU provides 1Gpixel/s or 1.5Gtexel/s of graphics processing or 24 GFLOPS of general purpose computing performance. The graphics capabilities of the Raspberry Pi are roughly equivalent to the level of performance of the Xbox of 2001.Raspberry Pi 2 is much more powerful, while the GPU is identic.

2.2.2 OVERCLOCKING

The first generation Raspberry Pi chip operated at 700 MHz by default and did not become hot enough to need a heat sink or special cooling, unless the chip was overclocked. The second generation runs at 900 MHz by default, and also does not become hot enough to need a heat sink or special cooling. Again overclocking may heat up the SoC more than usual.

Most Raspberry Pi chips could be overclocked to 800 MHz and some even higher to 1000 MHz There are reports the second generation can be similarly overclocked, in extreme cases, even to 1500 MHz (discarding all safety features and over voltage limitations). In the Raspbian Linux distro the overclocking options on boot can be done by a software command running “sudo raspi-config” without voiding the warranty. In those cases the Pi automatically shuts the overclocking down in case the chip reaches 85 °C (185 °F), but it is possible to overrule automatic over voltage and overclocking settings (voiding the warranty). In that case, one can try putting an appropriately sized heat sink on it to keep the chip from heating up far above 85 °C.

Newer versions of the firmware contain the option to choose between five overclock (“turbo”) presets that when turned on try to get the most performance out of the SoC without impairing the lifetime of the Pi. This is done by monitoring the core temperature of the chip, and the CPU load, and dynamically adjusting clock speeds and the core voltage. When the demand is low on the CPU, or it is running too hot, the performance is throttled, but if the CPU has much to do, and the chip's temperature is acceptable, performance is temporarily increased, with clock speeds of up to 1 GHz, depending on the individual board, and on which of the turbo settings is used.

2.2.3 RAM

On the older beta model B boards, 128 MB was allocated by default to the GPU, leaving 128 MB for the CPU. On the first 256 MB release model B (and model A), three different splits were possible.

2.2.4 NETWORKING

Though the model A and A+ do not have an 8P8C ("RJ45") Ethernet port, they can be connected to a network using an external user-supplied USB Ethernet or Wi-Fi adapter. On the model B and B+ the Ethernet port is provided by a built-in USB Ethernet adapter.

2.2.5 REAL-TIME CLOCK

The Raspberry Pi does not come with a real-time clock, which means it cannot keep track of the time of day while it is not powered on.

As alternatives, a program running on the Pi can get the time from a network time server or user input at boot time.

A real-time clock (such as the DS1307) with battery backup can be added (often via the I²C interface).

2.2.6 OPERATING SYSTEMS

The Raspberry Pi primarily uses Linux-kernel-based operating systems.

The ARM11 chip at the heart of the Pi (pre-Pi 2) is based on version 6 of the ARM. The current releases of several popular versions of Linux, including Ubuntu, will not run on the ARM11. It is not possible to run Windows on the original Raspberry Pi, though the new Raspberry Pi 2 will be able to run Windows 10. The Raspberry Pi 2 currently only supports Ubuntu Snappy Core, Raspbian, OpenELEC and RISC OS.

The install manager for the Raspberry Pi is NOOBS. The operating systems included with NOOBS are:

- Archlinux ARM
- OpenELEC
- Pidora (Fedora Remix)
- Puppy Linux
- Raspbmc and the XBMC open source digital media centre
- RISC OS – The operating system of the first ARM-based computer
- Raspbian (recommended for Raspberry Pi 1) – Maintained independently of the Foundation; based on the ARM hard-float (armhf) Debian 7 'Wheezy' architecture port originally designed for ARMv7 and later processors (with Jazelle RCT/ThumbEE, VFPv3, and NEON SIMD extensions), compiled for the more limited ARMv6 instruction set of the Raspberry Pi. A minimum size of 4 GB SD card is required.

2.3 RFID MODULE

Radio-frequency identification (RFID) is the wireless use of electromagnetic fields to transfer data, for the purposes of automatically identifying and tracking tags attached to objects. The tags contain electronically stored information. Some tags are powered by electromagnetic induction from magnetic fields produced near the reader. Some types collect energy from the interrogating radio waves and act as a passive transponder. Other types have a local power source such as a battery and may operate at hundreds of meters from the reader. RFID is one method for Automatic Identification and Data Capture (AIDC). Fig 2.2 shows the image of the RFID reader and tag used in our project.

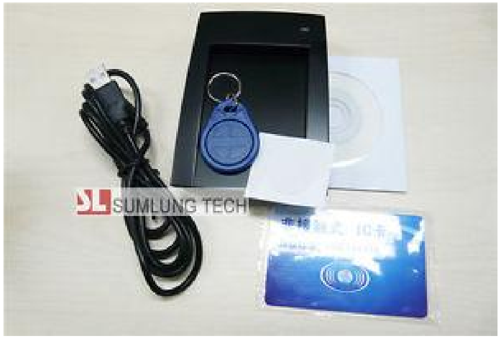


Fig.2.2 RFID reader and tag

RFID tags are used in many industries. For example, an RFID tag attached to an automobile during production can be used to track its progress through the assembly line; RFID-tagged pharmaceuticals can be tracked through warehouses; and implanting RFID microchips in livestock and pets allows positive identification of animals.

Since RFID tags can be attached to cash, clothing, and possessions, or implanted in animals and people, the possibility of reading personally-linked information without consent has raised serious privacy concerns.

A radio-frequency identification system uses tags, or labels attached to the objects to be identified. Two-way radio transmitter-receivers called interrogators or readers send a signal to the tag and read its response.

RFID tags can be either passive, active or battery-assisted passive. An active tag has an on-board battery and periodically transmits its ID signal. A battery-assisted passive (BAP) has a small battery on board and is activated when in the presence of an RFID reader. A passive tag is cheaper and smaller because it has no battery; instead, the tag uses the radio energy transmitted by the reader. However, to operate a passive tag, it must be illuminated with a power level roughly a thousand times stronger than for signal transmission. That makes a difference in interference and in exposure to radiation.

2.3.1 WORKING

RFID tags contain two parts: an integrated circuit for storing and processing information, modulating and demodulating a radio-frequency (RF) signal, collecting DC power from the incident reader signal, and other specialized functions; and an antenna for receiving and transmitting the signal. The tag information is stored in a non-volatile memory. The RFID tag includes either fixed or programmable logic for processing the transmission and sensor data, respectively.

An RFID reader transmits an encoded radio signal to interrogate the tag. The RFID tag receives the message and then responds with its identification and other information. This may be only a unique tag serial number, or may be product-related information such as a stock number, lot or batch number, production date, or other specific information. Since tags have individual serial numbers, the RFID system design can discriminate among several tags that might be within the range of the RFID reader and read them simultaneously.

RFID systems can be **classified** by the type of tag and reader.

A Passive Reader Active Tag (PRAT) system has a passive reader which only receives radio signals from active tags (battery operated, transmit only). The reception range of a PRAT system reader can be adjusted from 1–2,000 feet (0–600 m), allowing flexibility in applications such as asset protection and supervision.

An Active Reader Passive Tag (ARPT) system has an active reader, which transmits interrogator signals and also receives authentication replies from passive tags.

An Active Reader Active Tag (ARAT) system uses active tags awoken with an interrogator signal from the active reader. A variation of this system could also use a Battery-Assisted Passive (BAP) tag which acts like a passive tag but has a small battery to power the tag's return reporting signal.

Fixed readers are set up to create a specific interrogation zone which can be tightly controlled. This allows a highly defined reading area for when tags go in and out of the interrogation zone. Mobile readers may be hand-held or mounted on carts or vehicles.

2.3.2 SIGNALLING

Signalling between the reader and the tag is done in several different incompatible ways, depending on the frequency band used by the tag. Active tags may contain functionally separated transmitters and receivers, and the tag need not respond on a frequency related to the reader's interrogation signal.

Often more than one tag will respond to a tag reader, for example, many individual products with tags may be shipped in a common box or on a common pallet. Collision detection is important to allow reading of data. Two different types of protocols are used to "singulate" a particular tag, allowing its data to be read in the midst of many similar tags. In a slotted Aloha system, the reader broadcasts an initialization command and a parameter that the tags individually use to pseudo-randomly delay their responses.

An example of binary tree method of identifying an RFID tag is shown in Fig 2.3.

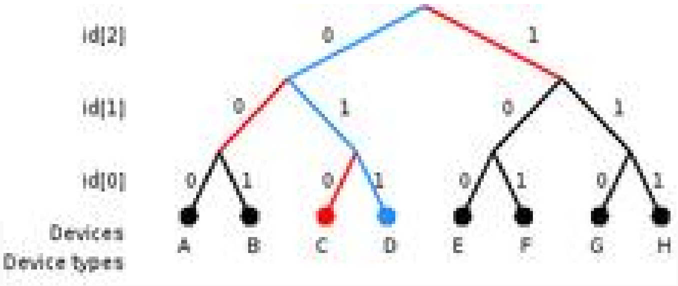


Fig.2.3 Binary tree method of identifying an RFID tag.

2.3.3 USES

The RFID tag can be affixed to an object and used to track and manage inventory, assets, people, etc. For example, it can be affixed to cars, computer equipment, books, mobile phones, etc. RFID offers advantages over manual systems or use of bar codes. The tag can be read inside a case, carton, box or other container, and unlike barcodes, RFID tags can be read hundreds at a time.

2.3.4 ACCESS CONTROL

RFID tags are widely used in identification badges, replacing earlier magnetic stripe cards. These badges need only be held within a certain distance of the reader to authenticate the holder. The image of the RFID antenna which is used for vehicular access control is shown in the Fig 2.4

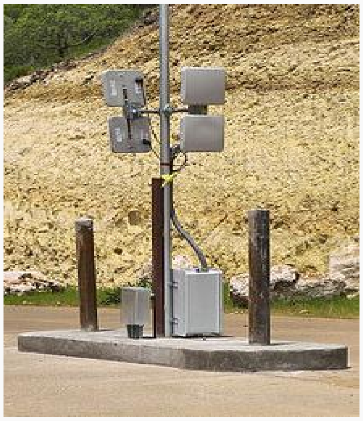


Fig.2.4 RFID antenna for vehicular access control

Tags can also be placed on vehicles, which can be read at a distance, to allow entrance to controlled areas without having to stop the vehicle and present a card or enter an access code.

2.3.5 INTELLIGENT TRANSPORTATION SYSTEM

RFID is used in intelligent transportation systems. In New York City, RFID readers are deployed at intersections to track E-Z Pass tags as a means for monitoring the traffic flow. The data is fed through the broadband wireless infrastructure to the traffic management centres to be used in adaptive traffic control of the traffic lights.

2.3.6 COMPLEMENT TO BARCODE

RFID tags are often a complement, but not a substitute, for UPC or EAN barcodes. They may never completely replace barcodes, due in part to their higher cost and the advantage of multiple data sources on the same object. Also, unlike RFID labels, barcodes can be generated and distributed electronically, e.g. via e-mail or mobile phone, for printing or display by the recipient. An example is airline boarding passes. The new EPC, along with several other schemes, is widely available at reasonable cost.

The storage of data associated with tracking items will require many terabytes. Filtering and categorizing RFID data is needed to create useful information. It is likely that goods will be tracked by the pallet using RFID tags, and at package level with Universal Product Code (UPC) or EAN from unique barcodes.

2.3.7 UNIQUENESS OF RFID

The unique identity is a mandatory requirement for RFID tags, despite special choice of the numbering scheme. RFID tag data capacity is large enough that each individual tag will have a unique code, while current bar codes are limited to a single type code for a particular product. The uniqueness of RFID tags means that a product may be tracked as it moves from location to location, finally ending up in the consumer's hands. This may help to combat theft and other forms of product loss.

2.3.8 DATA FLOODING

Not every successful reading of a tag (an observation) is useful for business purposes. A large amount of data may be generated that is not useful for managing inventory or other applications. For example, a customer moving a product from one shelf to another, or a pallet load of articles that passes several readers while being moved in a warehouse, are events that do not produce data that is meaningful to an inventory control system.

2.3.9 GLOBAL STANDARDIZATION

The frequencies used for UHF RFID in India are currently incompatible with those of Europe or Japan. Furthermore, no emerging standard has yet become as universal as the barcode. To address international trade concerns, it is necessary to use a tag that is operational within all of the international frequency domains.

2.4 SSD DISPLAY

A seven-segment display (SSD), or seven-segment indicator, is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot matrix displays.

2.4.1 CONSTRUCTION

The seven elements of the display can be lit in different combinations to represent the Arabic numerals. Often the seven segments are arranged in an oblique (slanted) arrangement, which aids readability. In most applications, the seven segments are of nearly uniform shape and size (usually elongated hexagons, though trapezoids and rectangles can also be used), though in the case of adding machines, the vertical segments are longer and more oddly shaped at the ends in an effort to further enhance readability. The numerals 6, 7 and 9 may be represented by two or more different glyphs on seven-segment displays, with or without a 'tail'.

The seven segments are arranged as a rectangle of two vertical segments on each side with one horizontal segment on the top, middle, and bottom. Additionally, the seventh segment bisects the rectangle horizontally. There are also fourteen-segment displays and sixteen-segment displays ; however, these have mostly been replaced by dot matrix displays.

2.4.2 PIN CONFIGURATION

The segments of a 7-segment display are referred to by the letters A to G, where the optional DP decimal point (an "eighth segment") is used for the display of non-integer numbers. In a simple LED package, typically all of the cathodes (negative terminals) or all of the anodes (positive terminals) of the segment LEDs are connected and brought out to a common pin; this is referred to as a "common cathode" or "common anode" device. Hence a 7 segment plus decimal point package will only require nine pins, though commercial products typically contain more pins, and/or spaces where pins would go, in order to match standard IC sockets. Integrated displays also exist, with single or multiple digits. Some of these integrated displays incorporate their own internal decoder, though most do not: each individual LED is brought out to a connecting pin as described. The seven segment display with only 12 pins is shown in the Fig 2.5

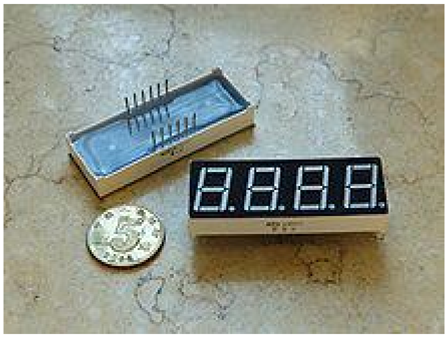


Fig.2.5 Multiplexed 4-digit, seven-segment display

2.4.3 WORKING

Multiple-digit LED displays as used in pocket calculators and similar devices used multiplexed displays to reduce the number of I/O pins required to control the display. To operate any particular segment of any digit, the controlling integrated circuit would turn on the cathode driver for the selected digit, and the anode drivers for the desired segments; then after a short blanking interval the next digit would be selected and new segments lit, in a sequential fashion. In this manner an eight digit display with seven segments and a decimal point would require only 8 cathode drivers and 8 anode drivers, instead of sixty-four drivers and IC pins.

Thus the hardware components which are used for developing the project is discussed in this chapter and chapter 3 gives the brief description of the software used in the project.

CHAPTER 3

SOFTWARE DESCRIPTION

3.1 INTRODUCTION

The software used in the system are Python and MPLAB. The programming language used is Python and development environment used is MPLAB. These two tools are discussed in this chapter.

3.2 PYTHON 2.7

Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale.

3.2.1 Features of Python

Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.

Python interpreters are available for installation on many operating systems, allowing Python code execution on a wide variety of systems. Using third-party tools, such as Py2exe or Py installer, Python code can be packaged into stand-alone executable programs for some of the most popular operating systems, allowing for the distribution of Python-based software for use on those environments without requiring the installation of a Python interpreter.

Python is a multi-paradigm programming language: object-oriented programming and structured programming are fully supported, and there are a number of language features which support functional programming and aspect-oriented programming (including by meta programming and by magic methods). Many other paradigms are supported using extensions, including design by contract and logic programming.

Python uses dynamic typing and a combination of reference counting and a cycle-detecting garbage collector for memory management. An important feature of Python is dynamic name resolution (late binding), which binds method and variable names during program execution. The design of Python offers only limited support for functional programming in the Lisp tradition.

Python's developers strive to avoid premature optimization, and moreover, reject patches to non-critical parts of CPython which would offer a marginal increase in speed at the cost of clarity. When speed is important, Python programmers use PyPy, a just-in-time compiler, or move time-critical functions to extension modules written in languages such as C. CPython is also available which translates a Python script into C and makes direct C level API calls into the Python interpreter.

Users and admirers of Python—most especially those considered knowledgeable or experienced—are often referred to as Pythonists, Pythonistas, and Pythoneers.

Python's statements include (among others):

- The *if* statement, which conditionally executes a block of code, along with **else** and *elif* (a contraction of else-if).
- The *for* statement, which iterates over an iterable object, capturing each element to a local variable for use by the attached block.
- The *while* statement, which executes a block of code as long as its condition is true.
- The *try* statement, which allows exceptions raised in its attached code block to be caught and handled by except clauses; it also ensures that clean-up code in afinally block will always be run regardless of how the block exits.
- The *class* statement, which executes a block of code and attaches its local namespace to a class, for use in object-oriented programming.
- The *def* statement, which defines a function or method.
- The *with* statement (from Python 2.5), which encloses a code block within a context manager (for example, acquiring a lock before the block of code is run and releasing the lock afterwards, or opening a file and then closing it), allowing RAII-like behaviour.
- The *pass* statement, which serves as a NOP. It is syntactically needed to create an empty code block.
- The *assert* statement, used during debugging to check for conditions that ought to apply.
- The *yield* statement, which returns a value from a generator function. From Python 2.5, yield is also an operator. This form is used to implement coroutines.

- The *import* statement, which is used to import modules whose functions or variables can be used in the current program.
- *Print* () was changed to a function in Python 3.

Python does not support tail-call optimization or first-class continuations, and, according to Guido van Rossum, it never will. However, better support for coroutine-like functionality is provided in 2.5, by extending Python's generators. Prior to 2.5, generators were lazy iterators; information was passed unidirectionally out of the generator.

3.2.2. Expressions

Python expressions are similar to languages such as C and Java:

- Addition, subtraction, and multiplication are the same, but the behaviour of division differs (see Mathematics for details). Python also added the ** operator for exponentiation.
- In Python, == compares by value, in contrast to Java, where it compares by reference. Python's is operator may be used to compare object identities (comparison by reference).
- Python uses the words and, or, not for its boolean operators rather than the symbolic &&, ||, ! Used in Java and C.
- Python has a type of expression termed a list comprehension. Python 2.4 extended list comprehensions into a more general expression termed a generator expression.
- Anonymous functions are implemented using lambda expressions; however, these are limited in that the body can only be a single expression.

Statements cannot be a part of an expression, so list and other comprehensions or lambda expressions, all being expressions, cannot contain statements. A particular case of this is that an assignment statement such as a = 1 cannot form part of the conditional expression of a conditional statement.

3.3 MPLAB IDE

MPLAB is a free integrated development environment for the development of embedded applications on PIC and dsPIC microcontrollers, and is developed by Microchip Technology.

MPLAB is a free integrated development environment for the development of embedded applications on PIC and dsPIC microcontrollers, and is developed by Microchip Technology.

3.3.1. MPLAB 8.X

MPLAB 8.X is the last version of the legacy MPLAB IDE technology, custom built by Microchip Technology in Microsoft Visual C++. MPLAB supports project management, editing, debugging and programming of Microchip 8-bit, 16-bit and 32-bit PIC microcontrollers. MPLAB only works on Microsoft Windows. MPLAB is still available from Microchip's archives, but is not recommended for new projects.

MPLAB supports the following compilers:

- MPLAB MPASM Assembler
- MPLAB ASM30 Assembler
- MPLAB C Compiler for PIC18

Thus the software tools required for the implementation of the project is discussed in this chapter and the overall operation of the project is explained in chapter in chapter 4.

CHAPTER 4

OPERATION

INTRODUCTION

The operation of the system is discussed below which is entirely directed by the microprocessor**.** The flowchart of the project is shown in the Fig 4.1

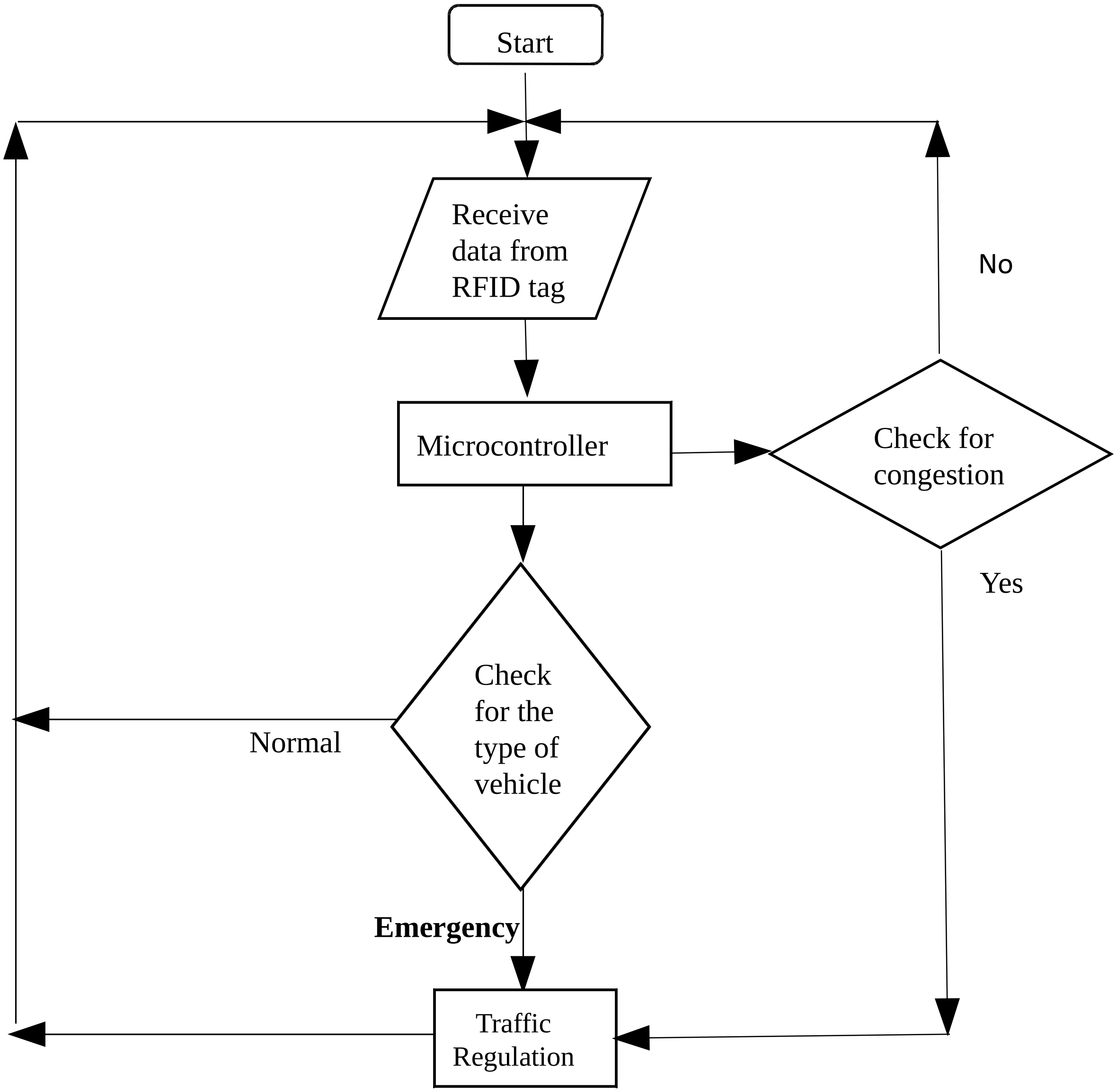


Fig 4.1 Operational flowchart of ATMS

ALGORITHM

The heart of the ATMS is the Raspberry Pi micro controller. It gives directions on how the traffic signals are changed in response to the varying traffic congestion.

1. The RFID tags placed in the vehicles are read by the readers which are installed in the traffic signals.

2. The microcontroller will detect the type of vehicle based on the tag read.

3. If the vehicle detected is the emergency vehicle then the traffic signal is changed.

4. But if the vehicle detected is the normal vehicle then the information of the vehicle is stored in the file.

5. Once the scanning process is done and if the police men is in need of the vehicle history he can obtain it from the database.

6. The second part of the project is the traffic management. If the traffic in the particular lane is high the reader detects the presence of congestion and automatically redirects the traffic to the lane where there is lesser amount of congestion.

7. An algorithm called ANTI COLLISION algorithm is used by the RFID reader which efficiently detects even more than two tags at the same instant.

8. The traffic signal are connected to the micro controller so that in response to the traffic congestion the signals are diverted.

STEPS INVOLVED IN ENABLING SPI

As the SPI is not enabled by default you will need to edit the raspi-blacklsit.conf in order to enable the SPI interface;

Add '#' in front of the line *spi-bcm2708* to comment it out of the blacklist. Save the file, and you will need to reboot the Raspberry PI, after which the *lsmod* command should show the spi device (spi_bcm2708) enabled.

CONNECTION DIAGRAM

The connections to be made for interfacing Raspberry Pi with the RFID module is shown in the Fig 4.2

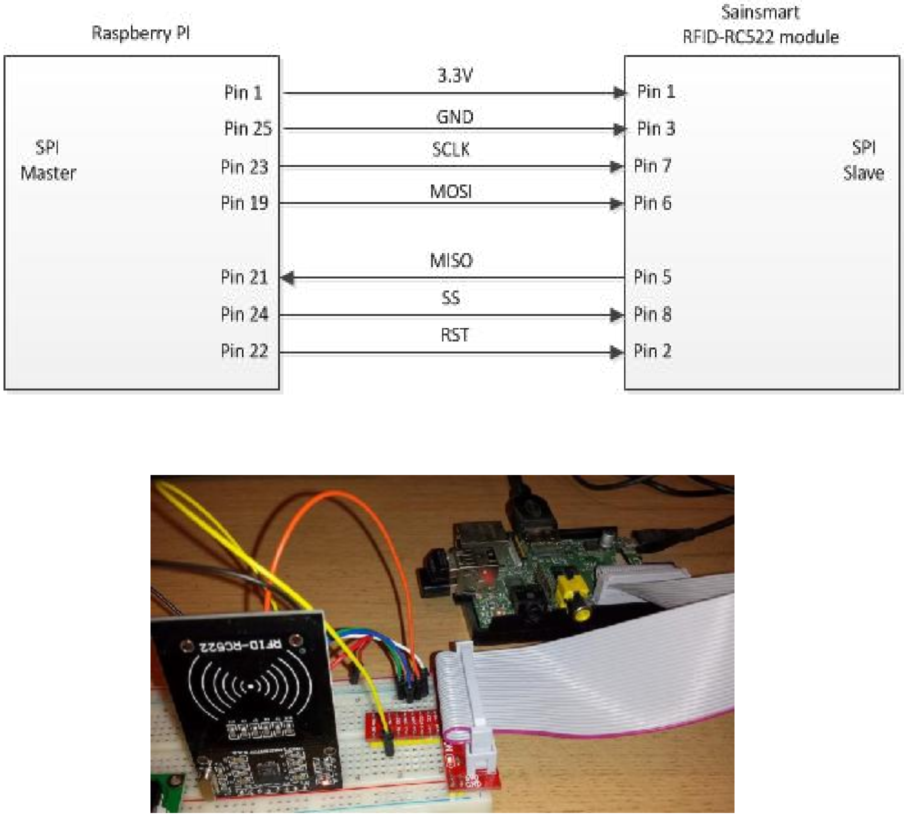


Fig.4.2 Interfacing raspberry Pi with RFID module

STEPS INVOLVED IN INTERFACING

1. To use the module from Python, need to load a SPI wrapper, however we need to install ‘python-dev’ to enable us to install the SPI wrapper.

2. To install ‘python-dev’, use the command *pseudo apt-get install python-dev*.

3. In order to read data from the SPI bus in Python we need a set of routines, a suitable set is SPI- Py, available form Git hub.

4. To do the install, clone the SPI- Py git repository.

5. Install the SPI- Py module by using the command *cd SPI-Pseudo python setup.py install.*

By following the above steps the RFID module is made to communicate with the Raspberry Pi.

CHAPTER 5
CONCLUSION

Using ATMS, the most important problem in the transportation i.e. traffic is eliminated. Also, the entire vehicles can be consolidated in a database so that illegal use of the vehicles can be identified. ATMS is user friendly where there is smart usage of the traffic so that major traffic threats are avoided.

ATMS can be implemented using a GPS module where in the module is fitted inside the vehicle. Program is made in the GPS module so that VIP vehicles and emergency vehicles can be given special priority. Ambulance can be directed to the nearby hospital and the hospital can be informed about the nature of emergency. Vehicle tracking can be easily done and with the help of the GPS system the vehicle can be easily located.

REFERENCES:

1.http://www.raspberrypi.org/forums/

2.https://github.com/mj3052/MFRC522-Pi

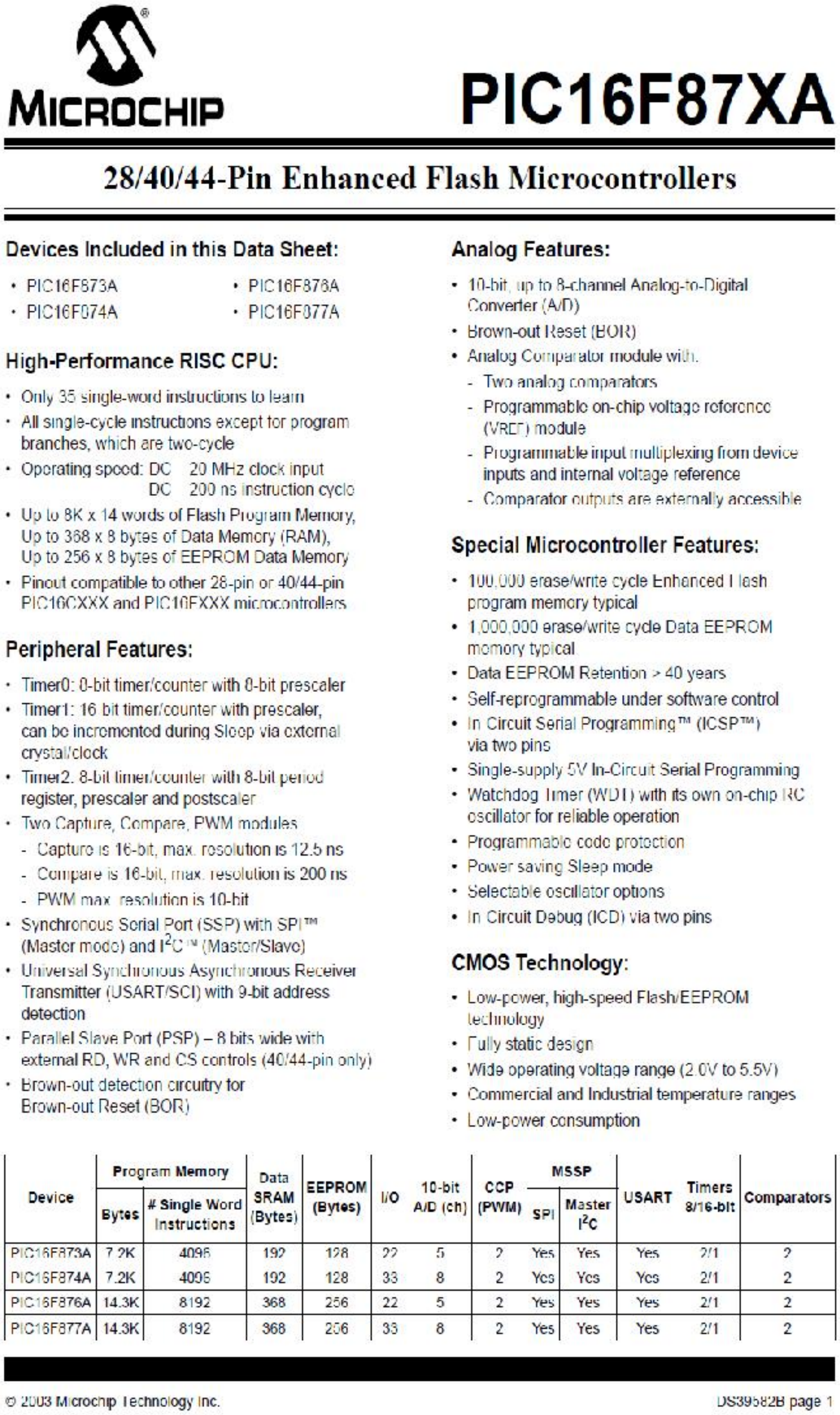
3.http://www.raspberrypi.org/forums/viewtopic.php?f=91&t=70756

4.https://www.pantechsolutions.net/microcontroller-boards/7segment-interfacing-with-pic16f877a-primer

5.https://www.youtube.com/watch?v=9aSL_FAF-Gg

6.https://www.youtube.com/watch?v=x5PZDOp-148

APPENDIX:



MICROCHIP

PIC16F87XA

28/40/44-Pin Enhanced Flash Microcontrollers

Devices Included in this Data Sheet:

- PIC16F873A
- PIC16F874A
- PIC16F876A
- PIC16F877A

High-Performance RISC CPU:

- Only 35 single-word instructions to learn
- All single-cycle instructions except for program branches, which are two-cycle
- Operating speed: DC – 20 MHz clock input
DC – 200 ns instruction cycle
- Up to 8K x 14 words of Flash Program Memory, Up to 368 x 8 bytes of Data Memory (RAM), Up to 256 x 8 bytes of EEPROM Data Memory
- Pinout compatible to other 28-pin or 40/44-pin PIC16CXXX and PIC16FXXX microcontrollers

Peripheral Features:

- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler, can be incremented during Sleep via external crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Two Capture, Compare, PWM modules
  - Capture is 16-bit, max. resolution is 12.5 ns
  - Compare is 16-bit, max. resolution is 200 ns
  - PWM max. resolution is 10-bit
- Synchronous Serial Port (SSP) with SPI™ (Master mode) and I^2C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
- Parallel Slave Port (PSP) – 8 bits wide with external RD, WR and CS controls (40/44-pin only)
- Brown-out detection circuitry for Brown-out Reset (BOR)

Analog Features:

- 10-bit, up to 8-channel Analog-to-Digital Converter (A/D)
- Brown-out Reset (BOR)
- Analog Comparator module with:
  - Two analog comparators
  - Programmable on-chip voltage reference (VREF) module
  - Programmable input multiplexing from device inputs and internal voltage reference
  - Comparator outputs are externally accessible

Special Microcontroller Features:

- 100,000 erase/write cycle Enhanced Flash program memory typical
- 1,000,000 erase/write cycle Data EEPROM memory typical
- Data EEPROM Retention > 40 years
- Self-reprogrammable under software control
- In-Circuit Serial Programming™ (ICSP™) via two pins
- Single-supply 5V In-Circuit Serial Programming
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code protection
- Power saving Sleep mode
- Selectable oscillator options
- In-Circuit Debug (ICD) via two pins

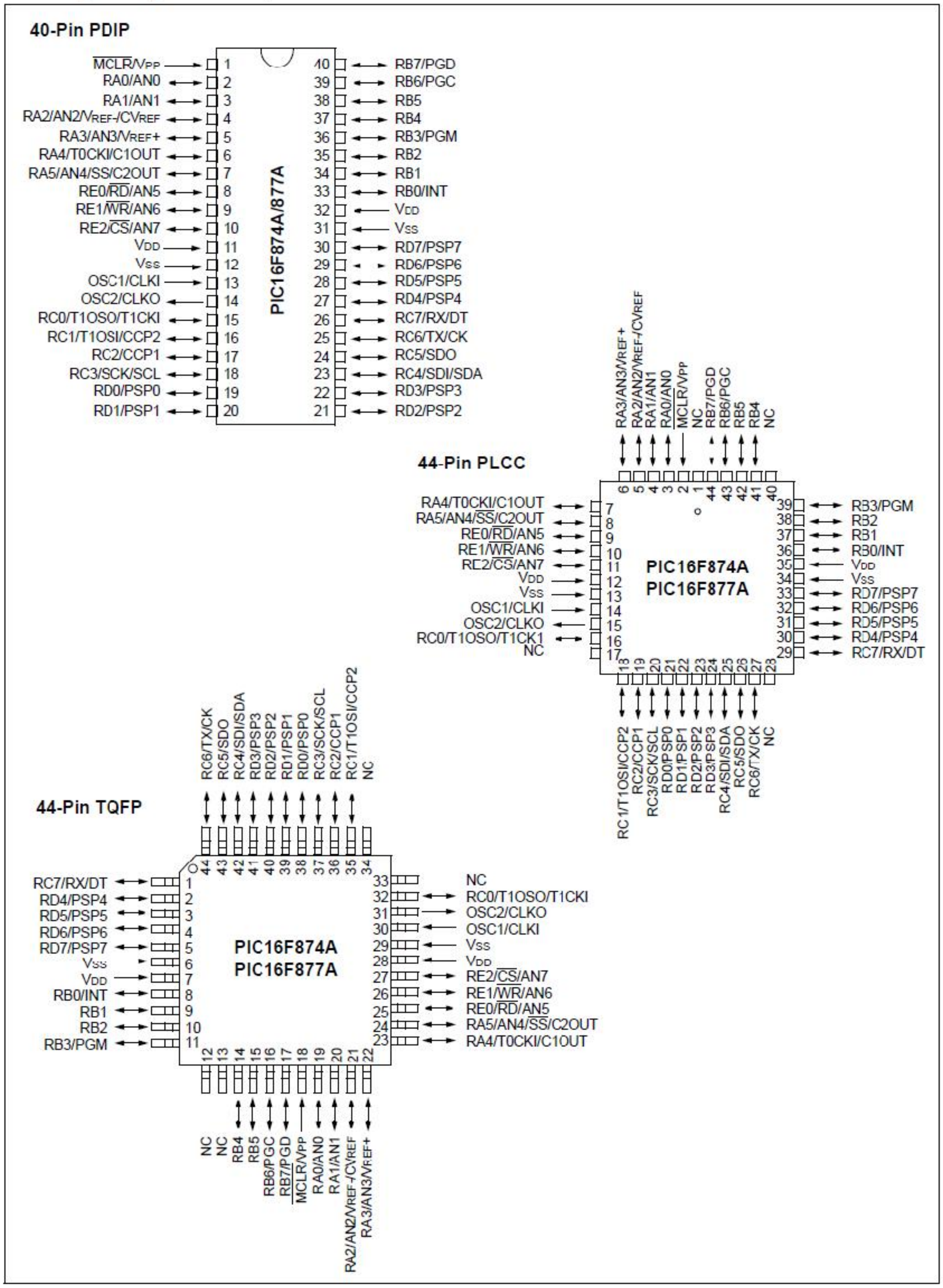
CMOS Technology:

- Low-power, high-speed Flash/EEPROM technology
- Fully static design
- Wide operating voltage range (2.0V to 5.5V)
- Commercial and Industrial temperature ranges
- Low-power consumption

Device	Program Memory		Data SRAM (Bytes)	EEPROM (Bytes)	I/O	10-bit A/D (ch)	CCP (PWM)	MSSP		USART	Timers 8/16-bit	Comparators
	Bytes	# Single Word Instructions						SPI	Master I^2C			
PIC16F873A	7.2K	4096	192	128	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F874A	7.2K	4096	192	128	33	8	2	Yes	Yes	Yes	2/1	2
PIC16F876A	14.3K	8192	368	256	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F877A	14.3K	8192	368	256	33	8	2	Yes	Yes	Yes	2/1	2

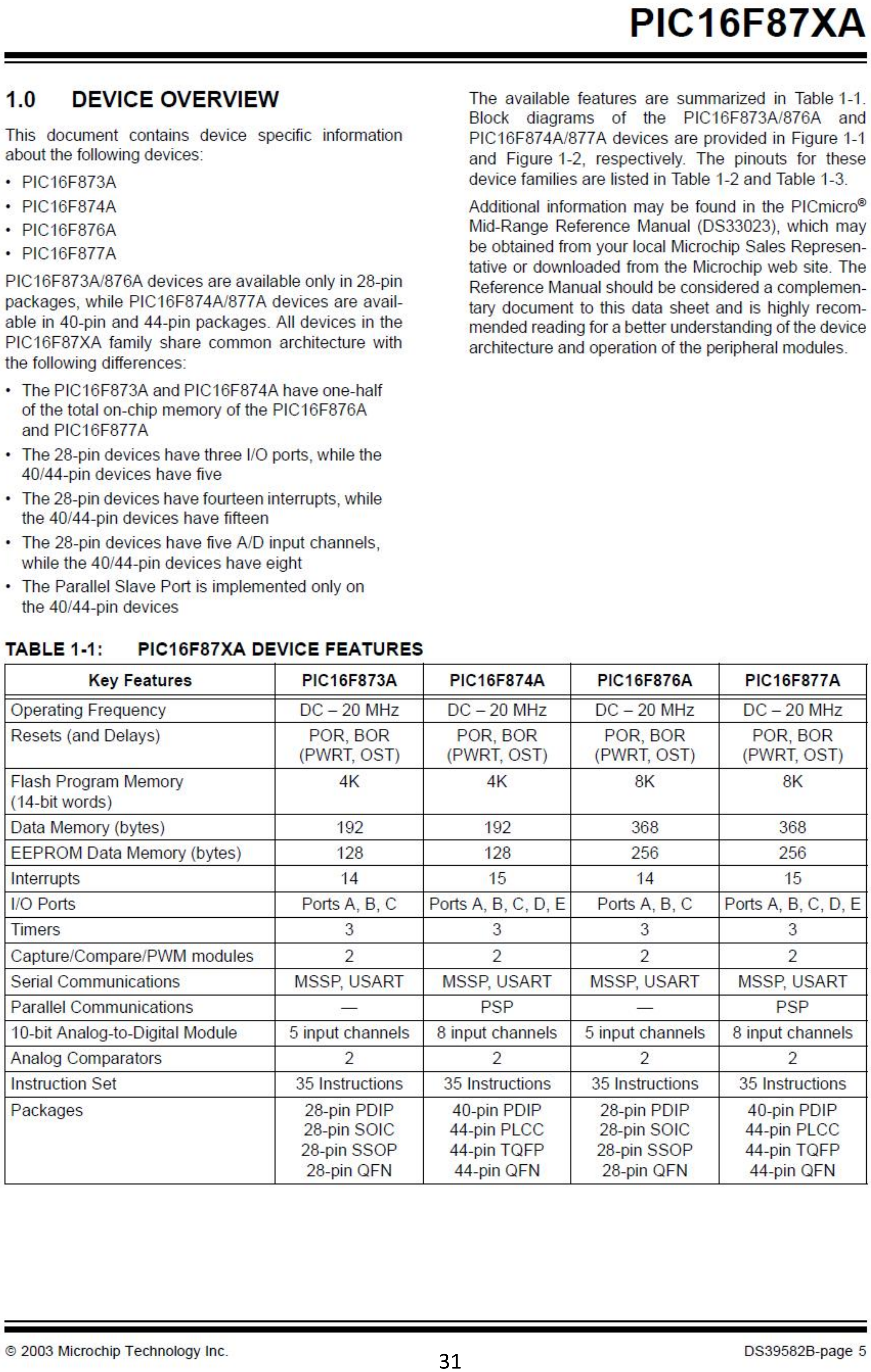
© 2003 Microchip Technology Inc. DS39582B page 1

Pin Diagrams (Continued)


40-Pin PDIP

44-Pin PLCC

44-Pin TQFP



1.0 DEVICE OVERVIEW

This document contains device specific information about the following devices:

- PIC16F873A
- PIC16F874A
- PIC16F876A
- PIC16F877A

PIC16F873A/876A devices are available only in 28-pin packages, while PIC16F874A/877A devices are available in 40-pin and 44-pin packages. All devices in the PIC16F87XA family share common architecture with the following differences:

- The PIC16F873A and PIC16F874A have one-half of the total on-chip memory of the PIC16F876A and PIC16F877A
- The 28-pin devices have three I/O ports, while the 40/44-pin devices have five
- The 28-pin devices have fourteen interrupts, while the 40/44-pin devices have fifteen
- The 28-pin devices have five A/D input channels, while the 40/44-pin devices have eight
- The Parallel Slave Port is implemented only on the 40/44-pin devices

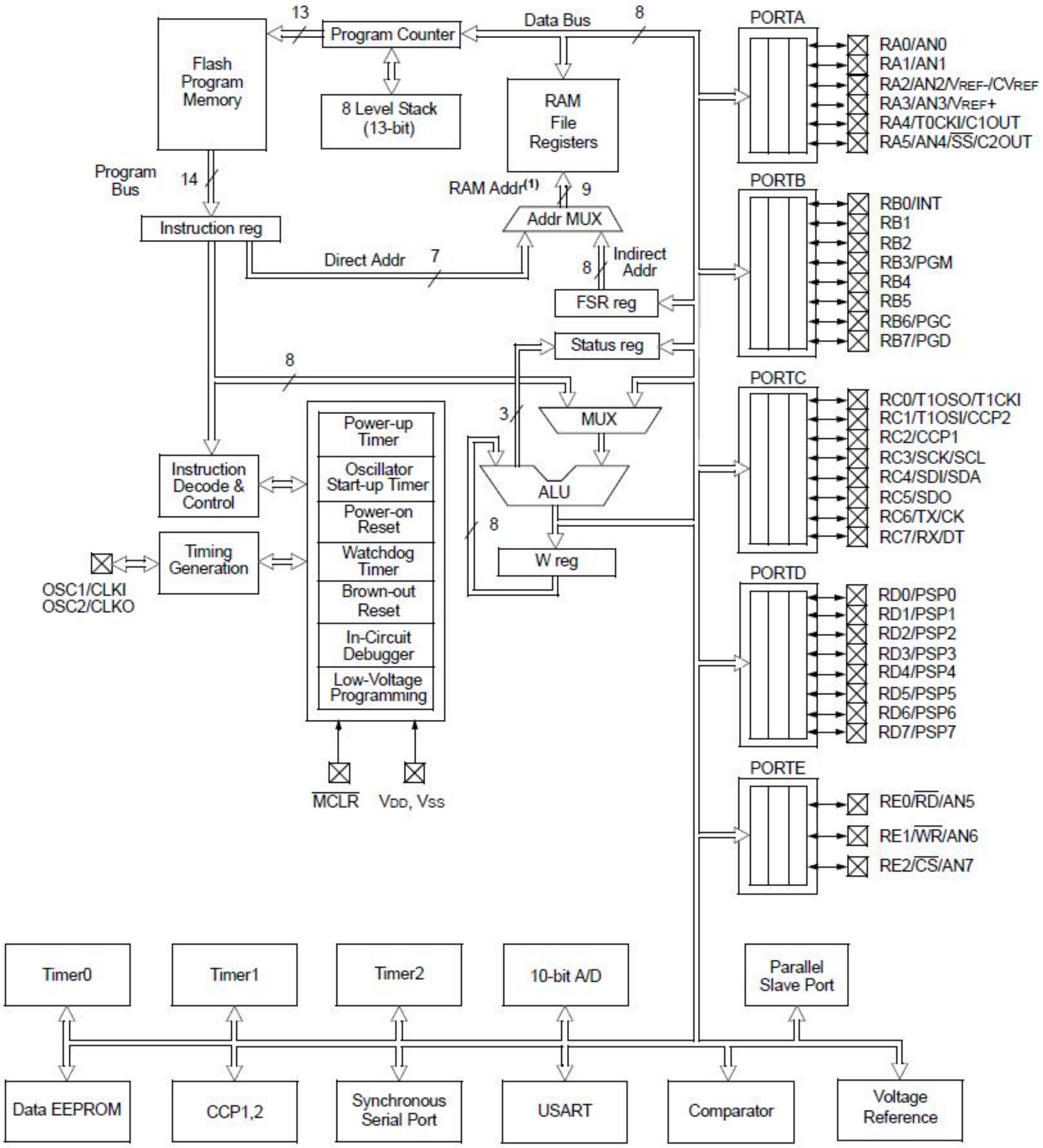
The available features are summarized in Table 1-1. Block diagrams of the PIC16F873A/876A and PIC16F874A/877A devices are provided in Figure 1-1 and Figure 1-2, respectively. The pinouts for these device families are listed in Table 1-2 and Table 1-3.

Additional information may be found in the PICmicro® Mid-Range Reference Manual (DS33023), which may be obtained from your local Microchip Sales Representative or downloaded from the Microchip web site. The Reference Manual should be considered a complementary document to this data sheet and is highly recommended reading for a better understanding of the device architecture and operation of the peripheral modules.

TABLE 1-1: PIC16F87XA DEVICE FEATURES

Key Features	PIC16F873A	PIC16F874A	PIC16F876A	PIC16F877A
Operating Frequency	DC – 20 MHz	DC – 20 MHz	DC – 20 MHz	DC – 20 MHz
Resets (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
Flash Program Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory (bytes)	128	128	256	256
Interrupts	14	15	14	15
I/O Ports	Ports A, B, C	Ports A, B, C, D, E	Ports A, B, C	Ports A, B, C, D, E
Timers	3	3	3	3
Capture/Compare/PWM modules	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	—	PSP	—	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels	5 input channels	8 input channels
Analog Comparators	2	2	2	2
Instruction Set	35 Instructions	35 Instructions	35 Instructions	35 Instructions
Packages	28-pin PDIP 28-pin SOIC 28-pin SSOP 28-pin QFN	40-pin PDIP 44-pin PLCC 44-pin TQFP 44-pin QFN	28-pin PDIP 28-pin SOIC 28-pin SSOP 28-pin QFN	40-pin PDIP 44-pin PLCC 44-pin TQFP 44-pin QFN

FIGURE 1-2: PIC16F874A/877A BLOCK DIAGRAM



Device	Program Flash	Data Memory	Data EEPROM
PIC16F874A	4K words	192 Bytes	128 Bytes
PIC16F877A	8K words	368 Bytes	256 Bytes

Note 1: Higher order bits are from the Status register.

TABLE 1-3: PIC16F874A/877A PINOUT DESCRIPTION

Pin Name	PDIP Pin#	PLCC Pin#	TQFP Pin#	QFN Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKI	13	14	30	32		ST/CMOS(4)	Oscillator crystal or external clock input.
OSC1					I		Oscillator crystal input or external clock source input. ST buffer when configured in RC mode; otherwise CMOS.
CLKI					I		External clock source input. Always associated with pin function OSC1 (see OSC1/CLKI, OSC2/CLKO pins).
OSC2/CLKO	14	15	31	33		—	Oscillator crystal or clock output.
OSC2					O		Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode.
CLKO					O		In RC mode, OSC2 pin outputs CLKO, which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate.
$\overline{\text{MCLR}}$/VPP	1	2	18	18		ST	Master Clear (input) or programming voltage (output).
$\overline{\text{MCLR}}$					I		Master Clear (Reset) input. This pin is an active low Reset to the device.
VPP					P		Programming voltage input.
							PORTA is a bidirectional I/O port.
RA0/AN0	2	3	19	19		TTL	
RA0					I/O		Digital I/O.
AN0					I		Analog input 0.
RA1/AN1	3	4	20	20		TTL	
RA1					I/O		Digital I/O.
AN1					I		Analog input 1.
RA2/AN2/VREF-/CVREF	4	5	21	21		TTL	
RA2					I/O		Digital I/O.
AN2					I		Analog input 2.
VREF-					I		A/D reference voltage (Low) input.
CVREF					O		Comparator VREF output.
RA3/AN3/VREF+	5	6	22	22		TTL	
RA3					I/O		Digital I/O.
AN3					I		Analog input 3.
VREF+					I		A/D reference voltage (High) input.
RA4/T0CKI/C1OUT	6	7	23	23		ST	
RA4					I/O		Digital I/O – Open-drain when configured as output.
T0CKI					I		Timer0 external clock input.
C1OUT					O		Comparator 1 output.
RA5/AN4/$\overline{\text{SS}}$/C2OUT	7	8	24	24		TTL	
RA5					I/O		Digital I/O.
AN4					I		Analog input 4.
$\overline{\text{SS}}$					I		SPI slave select input.
C2OUT					O		Comparator 2 output.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC Oscillator mode and a CMOS input otherwise.

TABLE 1-3: PIC16F874A/877A PINOUT DESCRIPTION (CONTINUED)

Pin Name	PDIP Pin#	PLCC Pin#	TQFP Pin#	QFN Pin#	I/O/P Type	Buffer Type	Description
							PORTB is a bidirectional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.
RB0/INT	33	36	8	9		TTL/ST$^{(1)}$	
RB0					I/O		Digital I/O.
INT					I		External interrupt.
RB1	34	37	9	10	I/O	TTL	Digital I/O.
RB2	35	38	10	11	I/O	TTL	Digital I/O.
RB3/PGM	36	39	11	12		TTL	
RB3					I/O		Digital I/O.
PGM					I		Low-voltage ICSP programming enable pin.
RB4	37	41	14	14	I/O	TTL	Digital I/O.
RB5	38	42	15	15	I/O	TTL	Digital I/O.
RB6/PGC	39	43	16	16		TTL/ST$^{(2)}$	
RB6					I/O		Digital I/O.
PGC					I		In-circuit debugger and ICSP programming clock.
RB7/PGD	40	44	17	17		TTL/ST$^{(2)}$	
RB7					I/O		Digital I/O.
PGD					I/O		In-circuit debugger and ICSP programming data.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC Oscillator mode and a CMOS input otherwise.

TABLE 1-3: PIC16F874A/877A PINOUT DESCRIPTION (CONTINUED)

Pin Name	PDIP Pin#	PLCC Pin#	TQFP Pin#	QFN Pin#	I/O/P Type	Buffer Type	Description
							PORTC is a bidirectional I/O port.
RC0/T1OSO/T1CKI	15	16	32	34		ST	
RC0					I/O		Digital I/O.
T1OSO					O		Timer1 oscillator output.
T1CKI					I		Timer1 external clock input.
RC1/T1OSI/CCP2	16	18	35	35		ST	
RC1					I/O		Digital I/O.
T1OSI					I		Timer1 oscillator input.
CCP2					I/O		Capture2 input, Compare2 output, PWM2 output.
RC2/CCP1	17	19	36	36		ST	
RC2					I/O		Digital I/O.
CCP1					I/O		Capture1 input, Compare1 output, PWM1 output.
RC3/SCK/SCL	18	20	37	37		ST	
RC3					I/O		Digital I/O.
SCK					I/O		Synchronous serial clock input/output for SPI mode.
SCL					I/O		Synchronous serial clock input/output for I^{2}C mode.
RC4/SDI/SDA	23	25	42	42		ST	
RC4					I/O		Digital I/O.
SDI					I		SPI data in.
SDA					I/O		I^{2}C data I/O.
RC5/SDO	24	26	43	43		ST	
RC5					I/O		Digital I/O.
SDO					O		SPI data out.
RC6/TX/CK	25	27	44	44		ST	
RC6					I/O		Digital I/O.
TX					O		USART asynchronous transmit.
CK					I/O		USART1 synchronous clock.
RC7/RX/DT	26	29	1	1		ST	
RC7					I/O		Digital I/O.
RX					I		USART asynchronous receive.
DT					I/O		USART synchronous data.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC Oscillator mode and a CMOS input otherwise.

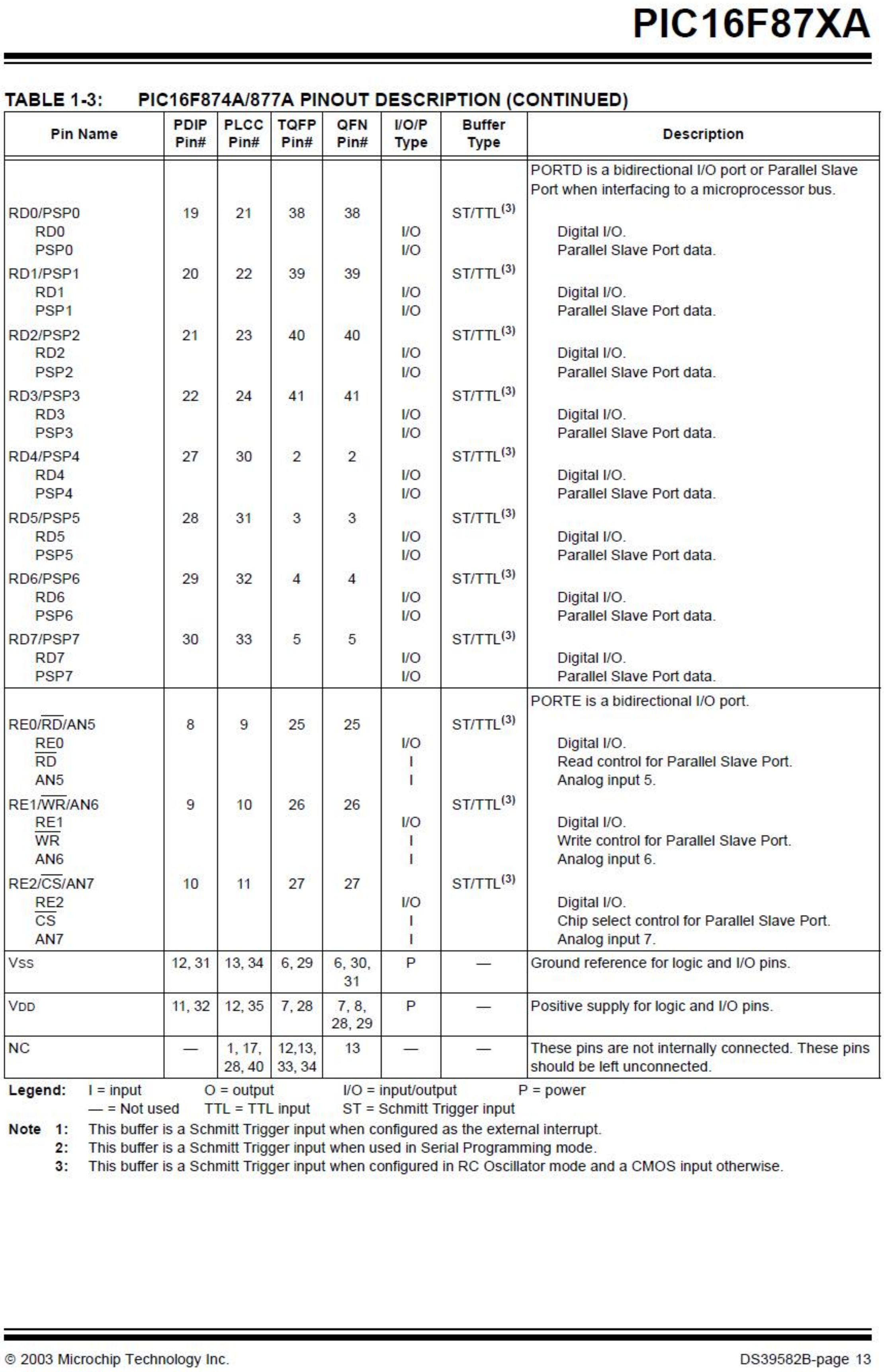


TABLE 1-3: PIC16F874A/877A PINOUT DESCRIPTION (CONTINUED)

Pin Name	PDIP Pin#	PLCC Pin#	TQFP Pin#	QFN Pin#	I/O/P Type	Buffer Type	Description
							PORTD is a bidirectional I/O port or Parallel Slave Port when interfacing to a microprocessor bus.
RD0/PSP0	19	21	38	38		ST/TTL[3]	
RD0					I/O		Digital I/O.
PSP0					I/O		Parallel Slave Port data.
RD1/PSP1	20	22	39	39		ST/TTL[3]	
RD1					I/O		Digital I/O.
PSP1					I/O		Parallel Slave Port data.
RD2/PSP2	21	23	40	40		ST/TTL[3]	
RD2					I/O		Digital I/O.
PSP2					I/O		Parallel Slave Port data.
RD3/PSP3	22	24	41	41		ST/TTL[3]	
RD3					I/O		Digital I/O.
PSP3					I/O		Parallel Slave Port data.
RD4/PSP4	27	30	2	2		ST/TTL[3]	
RD4					I/O		Digital I/O.
PSP4					I/O		Parallel Slave Port data.
RD5/PSP5	28	31	3	3		ST/TTL[3]	
RD5					I/O		Digital I/O.
PSP5					I/O		Parallel Slave Port data.
RD6/PSP6	29	32	4	4		ST/TTL[3]	
RD6					I/O		Digital I/O.
PSP6					I/O		Parallel Slave Port data.
RD7/PSP7	30	33	5	5		ST/TTL[3]	
RD7					I/O		Digital I/O.
PSP7					I/O		Parallel Slave Port data.
							PORTE is a bidirectional I/O port.
RE0/$\overline{RD}$/AN5	8	9	25	25		ST/TTL[3]	
RE0					I/O		Digital I/O.
$\overline{RD}$					I		Read control for Parallel Slave Port.
AN5					I		Analog input 5.
RE1/$\overline{WR}$/AN6	9	10	26	26		ST/TTL[3]	
RE1					I/O		Digital I/O.
$\overline{WR}$					I		Write control for Parallel Slave Port.
AN6					I		Analog input 6.
RE2/$\overline{CS}$/AN7	10	11	27	27		ST/TTL[3]	
RE2					I/O		Digital I/O.
$\overline{CS}$					I		Chip select control for Parallel Slave Port.
AN7					I		Analog input 7.
VSS	12, 31	13, 34	6, 29	6, 30, 31	P	—	Ground reference for logic and I/O pins.
VDD	11, 32	12, 35	7, 28	7, 8, 28, 29	P	—	Positive supply for logic and I/O pins.
NC	—	1, 17, 28, 40	12,13, 33, 34	13	—	—	These pins are not internally connected. These pins should be left unconnected.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC Oscillator mode and a CMOS input otherwise.

TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 1											
80h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	31, 150
81h	OPTION_REG	$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	23, 150
82h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	30, 150
83h(3)	STATUS	IRP	RP1	RP0	$\overline{\text{TO}}$	$\overline{\text{PD}}$	Z	DC	C	0001 1xxx	22, 150
84h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	43, 150
86h	TRISB	PORTB Data Direction Register								1111 1111	45, 150
87h	TRISC	PORTC Data Direction Register								1111 1111	47, 150
88h(4)	TRISD	PORTD Data Direction Register								1111 1111	48, 151
89h(4)	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			0000 -111	50, 151
8Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	30, 150
8Bh(3)	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	24, 150
8Ch	PIE1	PSPIE(2)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	25, 151
8Dh	PIE2	—	CMIE	—	EEIE	BCLIE	—	—	CCP2IE	-0-0 0--0	27, 151
8Eh	PCON	—	—	—	—	—	—	$\overline{\text{POR}}$	$\overline{\text{BOR}}$	---- --qq	29, 151
8Fh	—	Unimplemented								—	—
90h	—	Unimplemented								—	—
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000	83, 151
92h	PR2	Timer2 Period Register								1111 1111	62, 151
93h	SSPADD	Synchronous Serial Port (I^{2}C mode) Address Register								0000 0000	79, 151
94h	SSPSTAT	SMP	CKE	D/$\overline{\text{A}}$	P	S	R/$\overline{\text{W}}$	UA	BF	0000 0000	79, 151
95h	—	Unimplemented								—	—
96h	—	Unimplemented								—	—
97h	—	Unimplemented								—	—
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	111, 151
99h	SPBRG	Baud Rate Generator Register								0000 0000	113, 151
9Ah	—	Unimplemented								—	—
9Bh	—	Unimplemented								—	—
9Ch	CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0111	135, 151
9Dh	CVRCON	CVREN	CVROE	CVRR	—	CVR3	CVR2	CVR1	CVR0	000- 0000	141, 151
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	133, 151
9Fh	ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	128, 151

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8>, whose contents are transferred to the upper byte of the program counter.

2: Bits PSPIE and PSPIF are reserved on PIC16F873A/876A devices; always maintain these bits clear.

3: These registers can be addressed from any bank.

4: PORTD, PORTE, TRISD and TRISE are not implemented on PIC16F873A/876A devices, read as '0'.

5: Bit 4 of EEADRH implemented only on the PIC16F876A/877A devices.

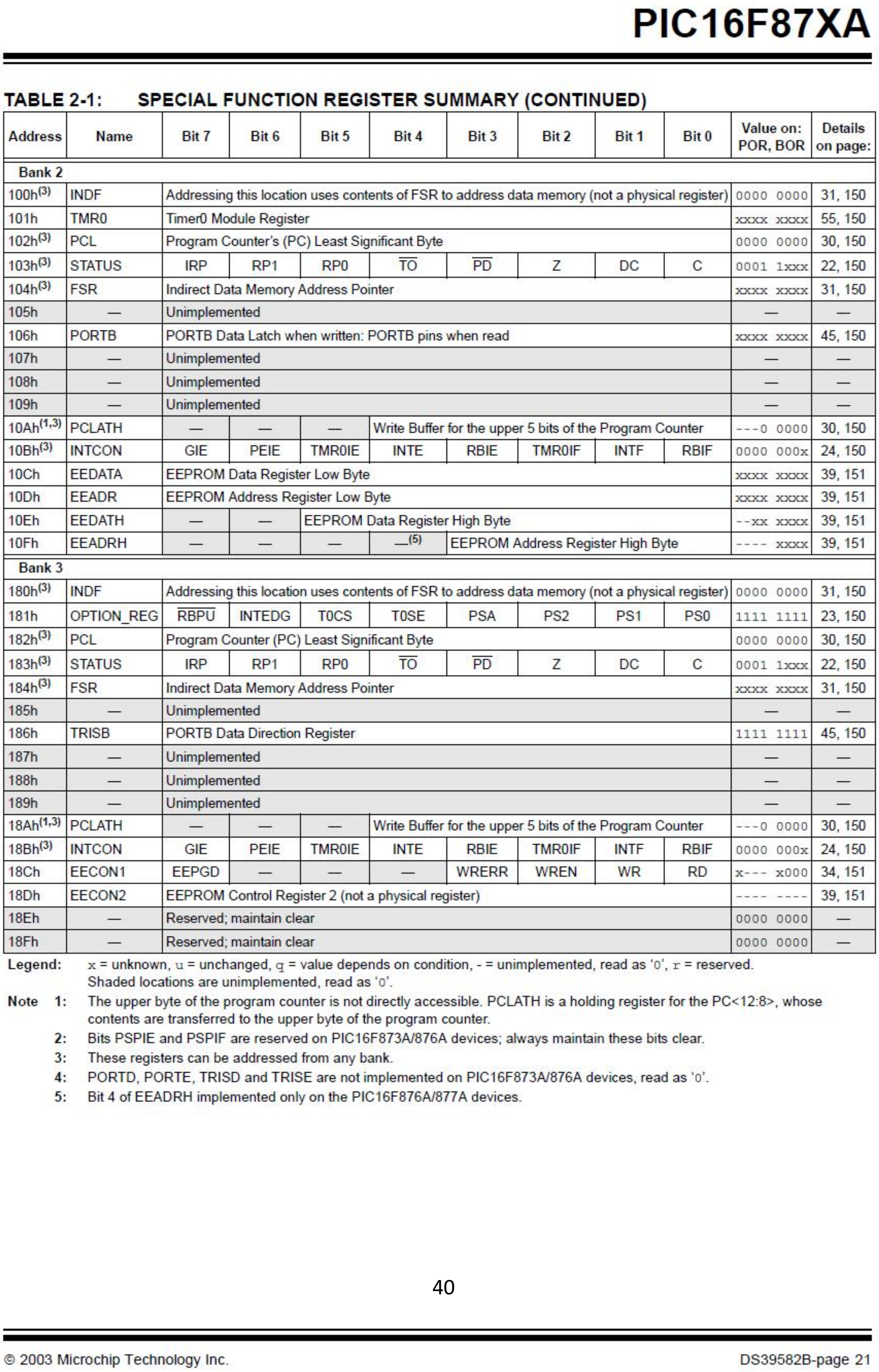


TABLE 2-1: SPECIAL FUNCTION REGISTER SUMMARY (CONTINUED)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 2											
100h[3]	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	31, 150
101h	TMR0	Timer0 Module Register								xxxx xxxx	55, 150
102h[3]	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	30, 150
103h[3]	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	22, 150
104h[3]	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
105h	—	Unimplemented								—	—
106h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	45, 150
107h	—	Unimplemented								—	—
108h	—	Unimplemented								—	—
109h	—	Unimplemented								—	—
10Ah[1,3]	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	30, 150
10Bh[3]	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	24, 150
10Ch	EEDATA	EEPROM Data Register Low Byte								xxxx xxxx	39, 151
10Dh	EEADR	EEPROM Address Register Low Byte								xxxx xxxx	39, 151
10Eh	EEDATH	—	—	EEPROM Data Register High Byte						--xx xxxx	39, 151
10Fh	EEADRH	—	—	—	—[5]	EEPROM Address Register High Byte				---- xxxx	39, 151
Bank 3											
180h[3]	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	31, 150
181h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	23, 150
182h[3]	PCL	Program Counter (PC) Least Significant Byte								0000 0000	30, 150
183h[3]	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	22, 150
184h[3]	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
185h	—	Unimplemented								—	—
186h	TRISB	PORTB Data Direction Register								1111 1111	45, 150
187h	—	Unimplemented								—	—
188h	—	Unimplemented								—	—
189h	—	Unimplemented								—	—
18Ah[1,3]	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	30, 150
18Bh[3]	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	24, 150
18Ch	EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000	34, 151
18Dh	EECON2	EEPROM Control Register 2 (not a physical register)								---- ----	39, 151
18Eh	—	Reserved; maintain clear								0000 0000	—
18Fh	—	Reserved; maintain clear								0000 0000	—

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved.
Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8>, whose contents are transferred to the upper byte of the program counter.

2: Bits PSPIE and PSPIF are reserved on PIC16F873A/876A devices; always maintain these bits clear.

3: These registers can be addressed from any bank.

4: PORTD, PORTE, TRISD and TRISE are not implemented on PIC16F873A/876A devices, read as '0'.

5: Bit 4 of EEADRH implemented only on the PIC16F876A/877A devices.

RC522

Specification

- Module Name:MF522-ED
- Working current：13 - 26mA / DC 3.3V
- Standby current：10 - 13mA / DC 3.3V
- Sleep current：<80uA
- Peak current：<30mA
- Working frequency：13.56MHz
- Card reading distance ：0～60mm （Mifare1 card）
- Protocol：SPI
- Data communication speed：10Mbit/s Max.
- Card types supported: Mifare1 S50, Mifare1 S70, Mifare UltraLight, Mifare Pro, Mifare Desfire
- Dimension：40mm × 60mm
- Working temperature：-20—80 degree
- Storage temperature：-40—85 degree
- Humidity：relevant humidity 5%—95%