A Study Of Handheld Devices for Application In e-Governance

A Thesis submitted towards partial fulfillment of the requirements for the degree of Master of Science by Research in Computer Science and Engineering

By
Yaswanth Narvaneni

International Institute of Information Technology, Gachibowli,
HYDERABAD, Andhra Pradesh, INDIA - 500 019

July, 2004

To my parents, teachers and friends

Acknowledgments

The graduate process has truly been a journey. I couldn't have reached this point without the support of family, friends and colleagues. When I first came to IIIT I never thought that I would write a thesis and graduate with a dual degree. Three years back when I met Dr. M. B. Srinivas regarding a course, he offered me a project to design a handheld device, I then thought that it would be impossible for a person with no electronics background to do it. I still remember his words when he said "You need no background except the enthusiasm to design one", from that day to this, his constant encouragement has helped me to do lot of work which I wouldn't have achieved otherwise. His constant guidance, support and continuous interest in my work made me achieve this goal. This work would not have been possible without his vision and advice. I thank him for giving me enough space and freedom for exploring new things.

I'm very grateful to Mr. Parthasarathi Nayani of Spectra Systems who helped me in my initial stages of the work and without whose support a major part of this work wouldn't have been possible.

Many friends and colleagues have helped me in the process, particularly Sunil Mohan Adapa for his valuable guidance in developing Telugu support for Casio PV, Neelima Adusumilli for her encouragement and for painstakingly proofreading my initial versions of this document. I thank my other friends, O. S. K. Chaitanya, O. V. L. Kiran Kumar, V R R Aditya and R S P Satish Kumar for their continuous encouragement. I'm very grateful to my friend, philosopher and guide Ms. A. Sharada for her continuous encouragement and support when I was going through tough times. I'm thankful to all the professors and students who made my stay at IIIT joyful and most memorable.

Finally, I am forever indebted to my father, mother and brother for their understanding, endless patience and encouragement when it was most required.

Synopsis

e-Governance is a concept that has recently come into regular use in political science, public administration and development management. e-Governance is meant to provide good governance to the citizens which is participatory, transparent and accountable in characteristic.

Personal Digital Assistant (PDA) is a term for a small hand-held device that provides small scale computing and information storage and retrieval capabilities for personal or business use. In National e-Governance Action Plan PDAs have been proposed to be used for data collection and for information dissemination.

There is a variety of PDAs available in the market, but most of them are expensive. From e-Governance perspective the following features have been identified to be important to be available on a PDA

- Facility for Application Development
- Capability for Remote Data Transfer
- Local Language Support

A detailed study of commercially available PDAs has been carried with respect to above features. While handheld PDAs have most of these features, they are found to be expensive. In this work a low cost platform has been identified and this thesis shows how the above mentioned features can be implemented on this platform.

Effort was also made to design a low-cost PDA mainly used for data collection and this thesis also presents and discusses the architecture.

Table of Contents

Chapter 1

e-Governance

e-Governance is a concept that has recently come into regular use in political science, public administration and development management. e-Governance is a method to provide good governance to the citizens which is participatory, transparent and accountable in characteristic.

Solutions to development issues often require changes to government processes, e.g. by decentralization. Objectives are generally to improve efficiency and effectiveness and to save costs. The driving force can also be public demand for online services and information that increase democratic participation, accountability, transparency, and the quality and speed of services. The implementation and use of Information and Communication Technology solutions (ICTs) can support governance reforms. e-Governance is more than just a government website on the internet. It means improving the present facilities using ICT solutions in the best way possible. While internationally most countries are in the early stages of e-governance, a good start has been made in several countries including India.

1.1 Definition

Many definitions exist for e-Governance. In the context of this work, e-Governance is defined as the "application of ICT to improve the governance and the interaction between the government and the governed".

Some of the other definitions are as follows:

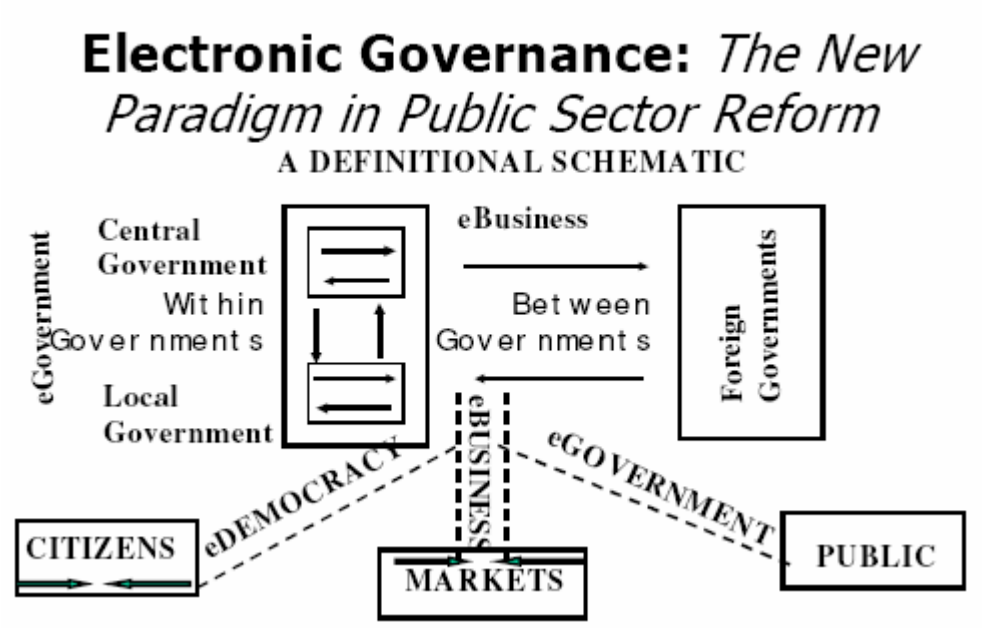
"e-Governance is defined as the application of *electronic means* in (1) the *interaction* between *government* and citizens and government and businesses, as well as (2) in internal *government operations* to simplify and improve democratic, government and business aspects of Governance." [1]

"Electronic Governance *(eGovernance)* incorporates all those processes and structures by means of which the new information and communication technologies (ICTs) can be deployed by government to enable the following:

- Administration of government (*eAdministration*) and delivery of services to the public (*eServices*). This generically constitutes electronic government (abbreviated *eGovernment*);
- Informing, vote-enabling, representation-enabling, consulting and involving the citizenry in, among others, broad consensus making in society in matters pertinent

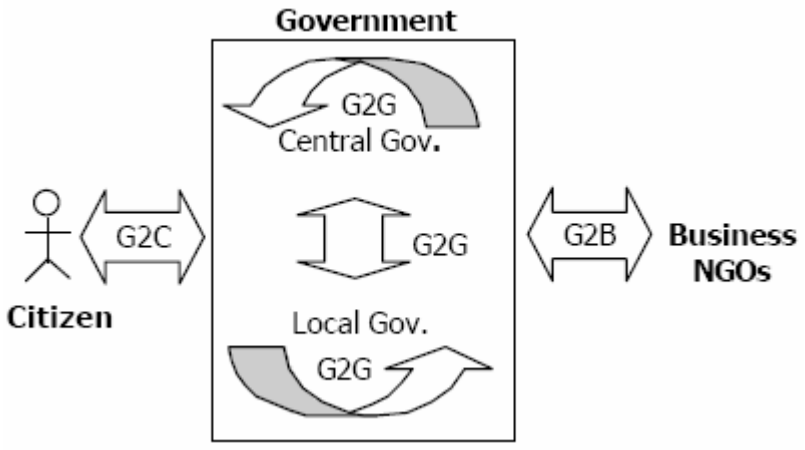
to decision making in political, social and economic priorities in government. This constitutes Electronic Democracy (abbreviated *eDemocracy*);
Transacting business with its "supply chain", namely, partners, clients and the markets. This constitutes Government Electronic Business (abbreviated simply *eBusiness*)." [2]

All the definitions are consolidated as follows:



A Broad Definitional Schematic for e-Governance [2]

A simpler version is as follows:



e-Governance [1]

1.2 Common Interaction in e-Governance

The most common interactions in e-governance are

- G2C – Government to Citizen
- G2B – Government to Business
- G2G – Government to Government

The objective of e-governance is to support and simplify governance for all parties - government, citizens and businesses. The use of ICT can connect all three parties and support processes and activities. In other words, e-governance uses electronic means to support and stimulate good governance. Therefore the objectives of e-governance are similar to the objectives of good governance. Good governance can be seen as an exercise of economic, political, and administrative authority to better manage affairs of a country at all levels, national and local. [1]

All government services can be classified under one of the three fundamental categories:

- Informational
- Interactive
- Transactional

Information is at the heart of every policy decision, response, activity, initiative, interaction and transaction between the government, citizens and businesses. Governments generate huge volumes of information which is organized and regularized. The internet and other advanced communications technologies can bring this information quickly and more directly to citizens. This includes publishing government information online, enabling the citizens and businesses to readily access the government information online. That is, disseminating the information about government and information compiled by the government to as wide an audience as possible.

Since services is the main work of the government, the primary objective of all government initiatives is to provide the citizen with an efficient alternative medium for interacting with public sector service providers. e-Government has the potential to involve citizens in the governance process by engaging them in interaction with policymakers throughout the policy cycle and at all levels of government. Interactive e-government involves two-way communications, starting with basic functions like email contact information for government officials or feedback forms that allow users to submit comments on legislative or policy proposals.

The next step involves transaction between the government and citizens. The government can create websites that allow users to conduct transactions online, just as private businesses do e-commerce. A transaction website offers direct link to government services available at any time. This will remove the citizen's hassle of standing in a long queue to pay bills or to renew ID cards etc. Innovations such as e-Seva centers [3] or Bhoomi projects [4] that bring e-government directly to the citizens are some examples.

1.3 e-Governance Models

Different institutions have defined their own model of e-Governance, some of the famous models are discussed here.

1.3.1 UN Model

There are 5 stages of e-Governance according to United Nations

- Emerging
- Enhanced
- Interactive
- Transactional
- Seamless

Emerging

A government web presence is established through a few independent official websites and the information will be limited, basic and static (G2C and G2B). The format of the first government websites is similar to that of a brochure or leaflet. The value to the public is that the information is publicly accessible. Internally(G2G) the government can also disseminate information with static electronic means.

Enhanced

The content and information on these government portals will be updated regularly to give the citizen the up to date information. In this phase the processes are described and thus become more transparent, which improves democracy and service.

Interactive

In this phase interaction between the government and the public (G2C and G2B) is stimulated with various applications. The citizens will be able to download forms, contact officials and make appointments and requests online which previously would have been only possible with long queues near counters. Internally (G2G) the government organizations can use LAN, WAN, intranet and e-mail to exchange information. The citizens can give their feedback and some suggestions on some policy decisions etc. This will bring a sense of participation for the citizen and improves the confidence of the citizen in the government.

Transactional

In this stage, users can actually pay for services or conduct financial transactions online. That is, all the transactions are done online without the citizen going to the office to finalize the decision. In this phase, internal (G2G) processes have to be redesigned to provide good service. Government needs to create new laws and legislation that will enable paperless transactions with legal certification. The bottom line is that now the complete process is online, including payments, digital signatures etc. This saves time, paper and money.

Seamless

A total seamless integration of e-functions and services across administrative and departmental boundaries. That is the public can get G2C and G2B services at one virtual counter. One single point of contact is the ultimate goal. This needs a drastic change in the culture, processes and responsibilities within the government (G2G). Government employees in different departments have to work together in a smooth and seamless way. In this phase cost savings, efficiency and customer satisfaction are reaching highest possible levels [5].

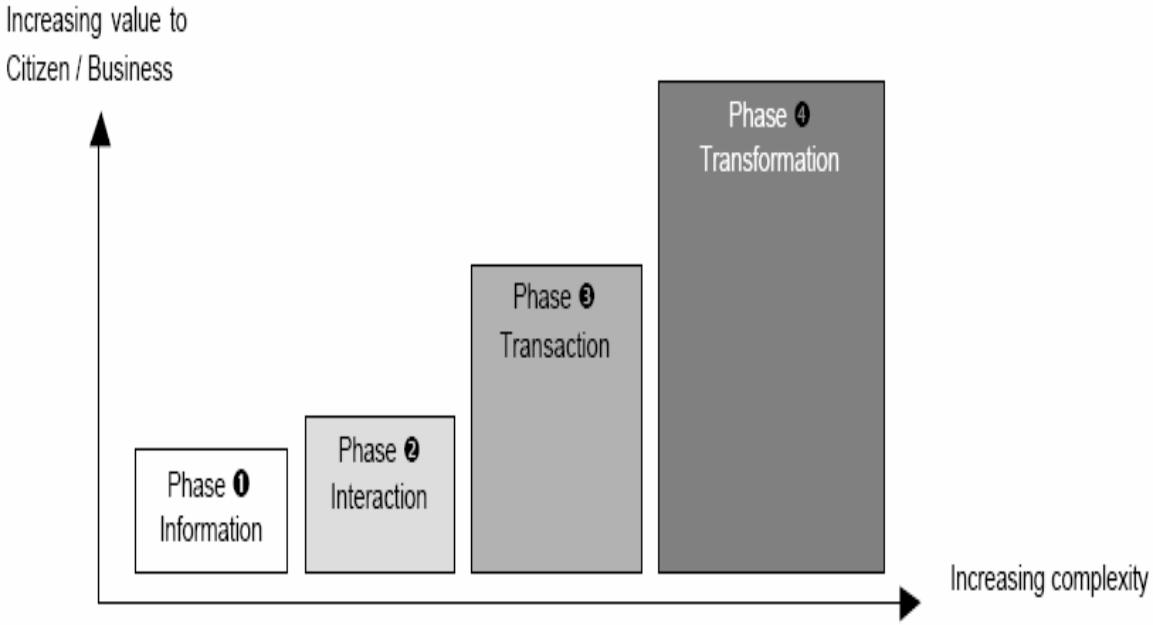
1.3.2 Gartner Model

There is a similar model by **Gartner** (an international e-business research consultancy firm) called the e-Governance maturity model which has four phases. These phases have been defined based on experiences with e-commerce and e-governance in Europe and other Western regions. The phases are as follows:

- Information
- Interaction
- Transaction
- Transformation

This is similar to the above model except that the “Emerging” and “Enhanced” phases is clubbed to form the Information phase in the Gartner model.

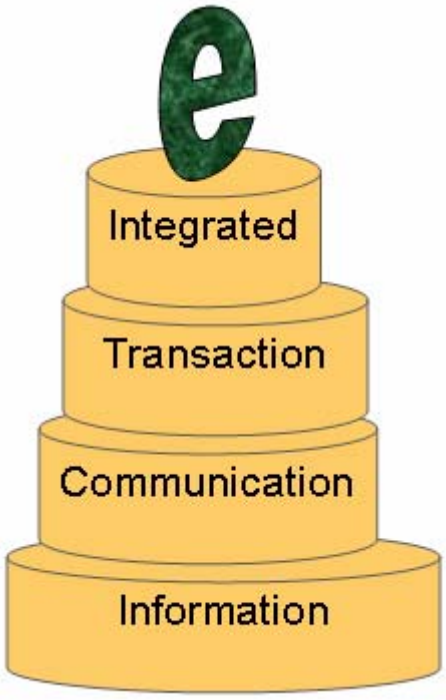
The complexity will increase when going from one phase to the other. The complexity graph of the Gartner model is as follows [1] :



Complexity of Different Phases (Gartner Model)

1.4. National e-Governance Perspective

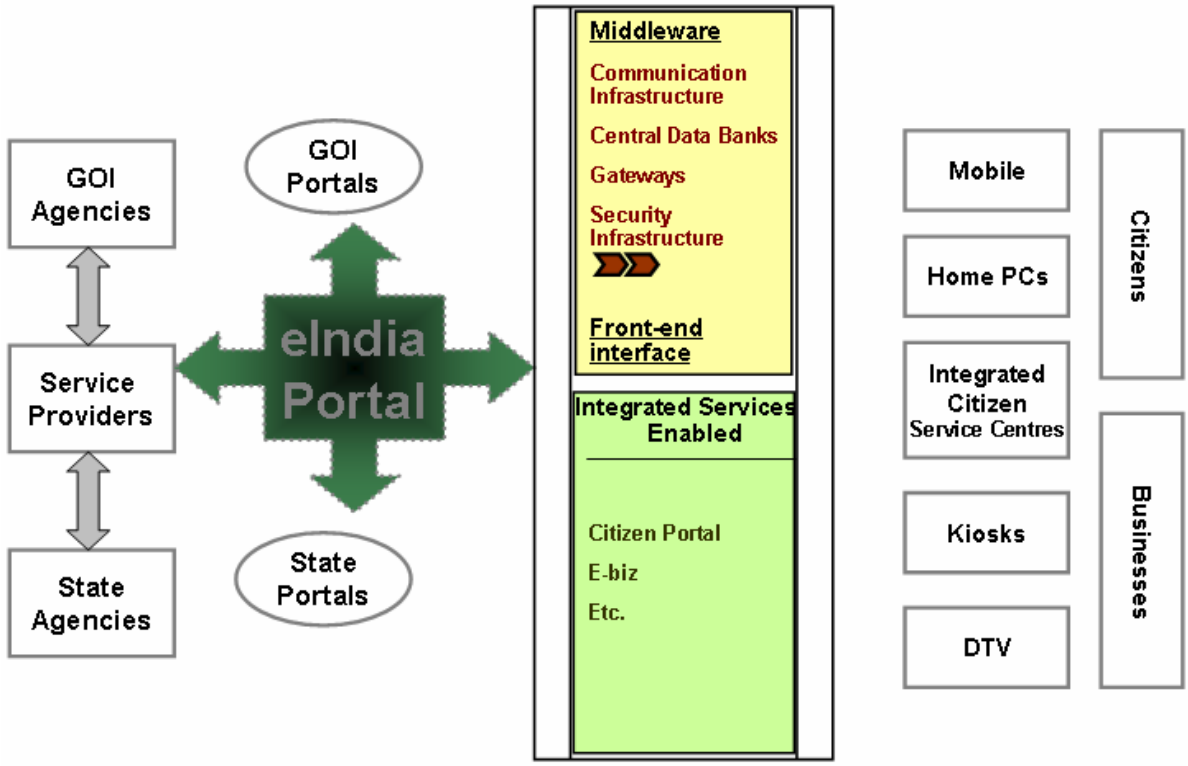
Government of India has taken a pro-active role to provide e-enabled services to the citizens through e-Governance. Government of India has also come up with a similar model as that of Gartner's for its e-Governance initiative. According to the National e-Governance Action Plan, the e-Governance model adopted is as follows [5] :



e-Governance model adopted by Government of India

In the first stage, Information is collected and is made available to the Citizens in the form of websites etc., this information will be regularly collected and updated. Then comes the communication stage, that is, the citizens will be able to download forms, contact officials and make appointments and requests online which previously would have been only possible with long queues near counters. Internally (G2G) the government organizations can use LAN, WAN, intranet and e-mail to exchange information. The third stage is the transaction stage where the citizens can do transactions with the government online, that is pay bills, reserve tickets online, finalize decisions etc., without going to the government offices. The fourth and the final stage is the Integrated stage where a total seamless integration of e-functions and services across administrative and departmental boundaries takes place. That is the citizen can interact with the government at a single point and can transact with the government. Currently India is somewhere between Stage 2 and Stage 3, that is, some of the government transactions can be done online and most of the information about different departments is available online for the citizens.

The Department of Information Technology has presented the following e-Governance framework for India. According to the National e-Governance Action Plan, the Overall framework is as follows:



e-Governance in India Overall Framework according to "National e-Governance Action Plan"

All the information regarding the Government and its transactions will be available on central portal called “The e-India portal”. This portal will be internally connected to

different Government of India portals and different state portals. All the G2G, G2C and G2B transactions will take place through this portal. The connectivity to this portal will be through Internet, LAN, WAN, Intranet for the Government organizations, where as the connectivity will be through internet, mobile telephone networks, wireless networks, home PCs, Integrated Citizen service centers, kiosks and DTVs for citizens and businesses [6].

Chapter 2

Personal Digital Assistants

Personal Digital Assistant (PDA) is a term for a small hand-held device that provides small scale computing and information storage and retrieval capabilities for personal or business use. Evolving from the digital assistant, the basic functionality allows for keeping schedule calendars, address book information and note-entering handy. PDAs are not self-contained — they are designed to synchronize with a desktop PC and keep information up to date on both sides.

2.1. PDAs in e-Governance

PDAs, as mentioned in the National e-Governance Action Plan, play an important role in e-Governance projects for the last mile connectivity, as they are used as data collection and dissemination devices. All government services, as discussed earlier can be classified under three fundamental categories, and PDAs can be used in all these categories

- Informational
- Interactive
- Transactional

PDAs play a major role in gathering information, which includes details like land records, health records, population census etc. There are many villages in the country where connectivity is not available, therefore portable and low cost devices like PDAs can be used for data dissemination. They are preferred over laptops or PCs because of their portability, low cost and high performance.

PDAs are already being used in different e-Governance projects like Student MAP in Andhra Pradesh for bringing out of school children back to school. Where they are being used as information collection devices. In Bhoomi-Suggi[4] project in Karnataka, they are being used for collecting land record information, where it acts as an interactive device since it also gives the land record information to the farmer. In projects by BESCOM[22][36], PDAs are being used for collecting electricity bills and such projects place them in transactional services too.

PDAs are primarily data collection devices and therefore those which have a database management system as well as communication support for transferring data remotely will be of great use. Most users at field level use local language than English and hence, PDAs with local language support will be extremely useful. Thus, PDAs that are meant to be used in e-Governance should most importantly have features that support application development, data communication and local language support. In the following, a

comprehensive analysis of most popular PDAs that are suitable for use in e-Governance is carried out.

2.2. A Study of Commercial PDAs

In order to decide upon the best PDA to be used for e-Governance, its necessary to carryout a comprehensive study to identify the device that has superior performance at low price. There is a large variety of PDAs with varying configuration, processor speed, price, performance and power consumption, available in the market. The present work focuses on the following models

- Simputer
- Palm Zire
- Pocket PC
- Casio PV S-600
- Analogic PDA

2.2.1. Simputer™

Simputer™ stands for Simple, Inexpensive, Multilingual, People's Computer. It was conceived as a Linux based 'platform for social change' that could inexpensively bring easy-to-use computers to rural Indian villages. Simputer is being used in some e-Governance projects like land record procurement, spot billing etc. The low cost model of Simputer, which has the following features, is discussed [7].

Features

Hardware

- CPU 32-bit Strong Arm SA-1100 RISC CPU running at 200MHz
- 16 MB of DRAM
- 8 MB Flash for Permanent Storage (DOC)
- Display I/F 320x240 Monochrome LCD Display Panel

Interfaces

- Touch-panel Overlay on LCD Display used with a plastic stylus
- Speaker and MIC Jacks, Smartcard Connector
- RJ-11 Telephone Jack
- USB Connector

System Software

- Operating System: GNU/Linux
- Soft-Modem Algorithms V.34/V.17 Data/Fax Modem Technology
- Perl/Tk scripting environment

Application Software

- imli: an IML browser
- Tapatap: Input method
- Internet access (Browser, Email, etc.)
- Dhvani: Text-to-Speech Software

The lowest model doesn't have any in-built modem but a soft-modem algorithm to compensate the hardware. It has a built in battery charger, a real-time clock and support for J2ME. The device doesn't have a keyboard or handwriting-recognition software but in certain applications the user can select letters or numbers from a software-generated keyboard that pops up on the screen. It has support to specialized cradles with micro-printer and keyboard.

It has some in-built special programs like

- Tapatap
- Text to Speech
- Smart cards

Special Software

Tapatap

Simputer has a program called Tapatap that displays a three-by-three grid; one can input a letter or number by tapping on the squares of the grid in a particular sequence. Although this method is easier than hunting and pecking on a software keyboard, it is still somewhat laborious, so the Simputer's applications have been carefully designed to minimize the need for tapping in text.

Text to Speech

The Simputer holds a database of phonemes-- basic linguistic sounds-- and from these it can generate an audio representation of any word as long as it is spelled phonetically and in characters from the Roman alphabet. It will work for various Indian languages, including Hindi, Kannada and Tamil, allowing the Simputer to read the text aloud on its tiny built-in speakers.

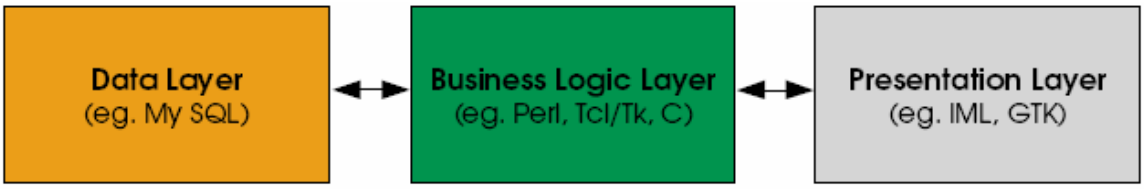
Smart Cards

The Simputer also has a slot for " smart" cards, a feature that its makers see as crucial. Because the device lacks a hard drive, smart cards will act as the device's portable storage units. In this way, many people will be able to share a single Simputer without having to share their private information with one another.

Programming Environment

Simputer runs on Linux operating system and the development environment is also in Linux. It has support for Perl and various Perl module extensions. It has support for Tcl/Tk scripting environment with GUI widget set and Tcl/Tk extensions, and Gtk/Gdk 1.2 GUI widget. Simputer has a three tier architecture, in the manner of all modern client-

server architectures, and allows for clean separation of the data, business logic and presentation layers. Simputer applications typically output IML mark-up code, which is then rendered in the Malacca interface.



Drawbacks

Simputer is slow, taking about 15 seconds to boot up and often needing several seconds to digest the information that the user inputs. And it sometimes crashes when it is left idle for a while, making it necessary to reboot the machine. Also, powering the device may be a daunting task in areas that do not have a reliable electricity supply. Although the Simputer can run on three AAA batteries, it can operate for only a few hours before draining them[8].

2.2.2. Palm Zire 21

Palm Zire 21 is an entry level model by Palm® and its low cost and usability add to its advantage. It has all the features of a entry level handheld device that replaces the paper, post-it notes and daytimers. It uses Palm OS® v5.2.1 as the operating system and has the following features [9] :

Features

Hardware

- 126MHz TI MAP 311 Processor
- 2 MB Flash for Permanent Storage (7.2MB actual storage capacity)
- Display 160x160 pixel display, 4-bit grayscale (No backlight)
- Rechargeable lithium ion battery (internal, non user removable)

Interfaces

- Touch-panel Overlay on LCD Display used with a plastic stylus
- Mini USB (for HotSync® operation)
- Infrared

System Software

- Operating System: Palm OS® Software v5.2.1
- Grafitti® 2: Handwriting recognition

Special Software

Grafitti®

Graffiti 2 allows the user to enter data quickly into Palm™ handheld. Writing with Graffiti 2 is similar to writing on paper. It takes one stroke of the stylus to enter some characters, and two strokes to enter others.

Programming Environment

The Palm OS Developer Suite is a new tool chain from PalmSource that provides software developers with an easier and faster path to develop robust applications. With Palm OS Developer Suite, programmers will have a parallel development path for creating ARM-native software applications, and 68K software applications that run on Palm OS 4, Palm OS Garnet and Palm OS Cobalt, through PACE (the Palm Application Compatibility Environment).

The Palm OS Developer Suite integrates Palm OS SDKs and tools into the open source Eclipse IDE. The Eclipse platform is a foundation for constructing and running integrated end-to-end software development tools. Eclipse is an open-source Integrated Development Environment (IDE) originally developed by IBM. It is a IDE written originally for and in Java, which now supports development in a variety of languages, including C, C++, Java, and COBOL. IBM released their source code to the public under a very liberal "BSD-like" license, and passed ownership to the Eclipse Organization, which is the community-run group dedicated to the ongoing development of Eclipse.

Programming Palm

Palm OS applications are generally single-threaded, event-driven programs. Applications run one at a time. Users do not quit or exit an application; they simply choose to run a different application. In response, Palm OS stops the current application and launches the newly selected one. While Palm OS applications are event-driven, they may perform work outside of the application's main event loop in response to other system requests.

- Each application has a `PilotMain` function that is equivalent to `main` in C programs. To launch an application, the system calls PilotMain and sends it a launch code. The launch code may specify that the application is to become active and display its user interface (called a normal launch), or it may specify that the application should simply perform a small task and exit without displaying its user interface. The sole purpose of the PilotMain function is to receive launch codes and respond to them.

- Palm OS is an event-based operating system, so Palm OS applications contain an event loop; however, this event loop is only started in response to the normal launch. The application may perform work outside the event loop in response to other launch codes.

- Most Palm OS applications contain a user interface made up of “forms”, which are analogous to windows in a desktop application. The user interface may contain both predefined UI elements (sometimes referred to as UI objects), and custom UI elements.

- All applications should use the memory and data management facilities provided by the system.

- The application's features are implemented by calling Palm OS functions. Palm OS consists of several managers, which are groups of functions that work together to implement a feature. As a rule, all functions that belong to one manager use the same prefix and work together to implement a certain aspect of functionality. Managers are available to, for example, generate sounds, send alarms, perform network communication, and beam information through an infrared port.

- The ANSI C libraries are not part of the Palm development platform. In many cases, one can perform the same function using a Palm OS API call as one does with a call to a ANSI C function. For example, the Palm OS provides a string manager that performs many of the string functions one would expect to be able to perform in an ANSI C program. If a standard C function is used, the code for the function is linked into the application and results in a bigger executable.

2.2.3. Pocket PC

Features[10]

Hardware

- Processor: 206 MHz Intel StrongARM 32 bit RISC Processor
- Display Type: 240 x 320, .24 mm Pixel Pitch, 4096 color, Color reflective thin film transistor (TFT) LCD
- RAM: 32 MB
- ROM: 16 MB Flash
- Battery: 950 mAh Lithium Polymer (up to 12hrs of life)

Interfaces

- Touch-panel Overlay on LCD Display used with a plastic stylus
- Communications Port: Interface with USB cradle
- Infrared Port: 115 Kbps
- Audio out: 3.5 mm Stereo

System / Application Software

- Operating System: Windows® Powered Pocket PC (in Flash memory)
- Pocket Excel
- Pocket Word
- Voice Recorder

- Pocket Internet Explorer
- Media Player

Programming Environment

Pocket PC runs on Windows Mobile. Windows Mobile is Microsoft's software platform for Pocket PCs and Smartphones. Windows Mobile extends the familiarity of the Windows desktop to personal devices. To build native C++ applications for the Pocket PC 2003, one will need Microsoft® eMbedded Visual C++® 4.0, Microsoft® eMbedded Visual C++® 4.0 Service Pack 3, and the Pocket PC 2003 Software Development Kit. To build managed code applications for Pocket PC 2003, one will need Visual Studio .NET 2003 and the Pocket PC 2003 SDK.

Applications on Pocket PC are developed using Microsoft® eMbedded Visual C++. The Microsoft® eMbedded Visual C++ tool delivers a complete desktop development environment for creating applications and system components for Windows® CE .NET-powered devices. The capabilities of embedded VC++ include C++ exception handling, Run Time Type Information (RTTI), it includes STL library components and a debugger. There are emulators that are provided to test the code on the PC.

Programming Pocket PC [11]

The most common tools for developing native applications for Windows Mobile-based devices are the eMbedded Visual Tools 3.0 for Pocket PC 2000 and 2002, eMbedded Visual C++ 3.0 for Smartphone 2002, and eMbedded Visual C++ 4.0 for Pocket PC 2003 and Smartphone 2003 including Pocket PC 2003 Second Edition and Smartphone 2003 Second Edition.

Microsoft eMbedded Visual Tools 3.0 provide an entry-level integrated development environment for building mobile applications, including necessary compilers, debugging facilities and platform documentation. The eMbedded Visual Tools 3.0 include Microsoft eMbedded Visual C++® 3.0 and eMbedded Visual Basic® 3.0. Both tools are separate development environments and do not require any other development environment, such as Microsoft Visual Studio or Visual Studio .NET, to run. The Microsoft eMbedded Visual C++ 3.0 development system offers the native code C++ compiler for Windows Mobile-based development. The eMbedded Visual C++ debugger works via Microsoft ActiveSync®, through a wired LAN, wireless LAN or cradle connection.

Microsoft eMbedded Visual C++ 4.0 supports development for Windows CE .NET 4.2 based devices and, via service pack 3, supports Windows Mobile 2003 Second Edition software for Pocket PCs and Smartphones. eMbedded Visual C++ 4.0 is ideal for targeting mobile or embedded devices based on Windows CE .NET 4.2 using native code. Enhancements of eMbedded Visual C++ 4.0 include:

- Just-In-Time debugging for diagnosing unhandled exceptions
- C++ structured exception handling

- "Attach-to" a process for improved process debugging
- Better integration with the new emulator

The Smart Device Programmability (SDP) features of Visual Studio .NET 2003 help developers write mobile applications that take advantage of the Microsoft .NET Compact Framework, enabling distributed mobile computing in either connected or disconnected scenarios. The extensive class library available through the .NET Compact Framework allows applications to be written much faster than with traditional tools.

2.2.4. Analogic Intelligent Hand Held Terminal

Analogic Intelligent Hand Held Terminal (IHHT) was developed by Analogic Technomatics (P) Ltd a Hyderabad based company as a hand held unit for different industries ranging from FMCG to e-Governance. Analogic has different custom devices for different operations like ticket issuing, meter reading, spot billing, point of sale terminal, stores inventory. All these devices are similar with some features added or removed. Intelligent Hand Held Terminal is a general device which has almost all the features that are present in the other devices. These devices have been used in different e-Governance projects for Ticketing, Electricity meter reading etc. The features of IHHT are as follows [12]:

Features

Hardware

- High speed processor with 16 bit internal Architecture
- 256K (Optional 512 K) Flash, 128K (Optional 512 K) SRAM with battery backup, 8MB Flash Disk (Optional)
- High-Clarity LCD with backlight (4 Rows of 20 Characters)
- Real Time Clock: Available with battery backup
- Built in 1.55 Ahr NiMH Battery with intelligent Charger
- Charger / Adapter : 65 to 265 V AC, 45 to 65 Hz

Interfaces

- Membrane or Elastomeric Keypad 45, 32, 30, 20 or 15 keys, Standard or Custom Keypad layouts
- Up to 115.2 k bps Communication
- 24Col. Alphanumeric high speed impact printer(2.7 lines/sec) Paper width 57mm
- Serial Ports (RS 232)
    - i) Standard one port (115K band)
    - ii) Programming port : available (806 K baud)
- Telephone Line Port: RJ Connector with fixed 2900 bps
- **GSM Modem**
    - Multi-Band GSM/GPRS, Dual Band EGSM900/800MHZ,
    - Quad Band GSM850/900/1800/1900MH2,
    - Compliant with ETSI GSM Phase2+Standerd,

- Class4 (2 W @ 250 / 900 MHZ)
- Class 1 (1 W @ 1800 / 1900 MHZ)

- **PSTN modem**
  - Data modem formats: V.22 bis 2400 bps. V.22/Bell
  - 212 A 120Dphs,
  - Security Modem formats,
  - Caller ID detection & decode,
  - AT Command Set,
  - HDLC formatting in H/W, Integrated DAA,
  - Parallel Handset detection.

System / Application Software

- Operating System: No Operating System
- 'C' programmable by user with PC link.

Programming Environment

Analogic has developed a state-of-the-art Hand Held Terminal with built in printer, modem (PSTN/GSM) and built-in Smart Card Reader. The Terminal is an user programmable, using the proprietary 'C'. The unit consists of wide range of optional Hardware components like: Custom Key Pad, Display (4x20), 256K Flash (optional 512K), 128K SRAM (optional 512 K), Built in 1.55 Ahr NiMH battery Intelligent Battery Charger, high speed COM port, programming port, high speed Impact Printer (2.7 lines/sec), Analogic V.22 Bis Modem and Smart Card Reader. The user need to develop his application software on a PC and on successful debugging, the same need to be loaded on to terminal's Flash memory.

The development is done using a proprietary 'C' called "Dynamic C", an integrated development environment. Dynamic C, an interactive development environment with editor, source-level debugger and the Terminal connected to a COM port on the PC (COM 1 by default) whose default operation is at 115,200 baud rate, forms a complete application development environment. The IHHT can handle C language applications of approximately 1 Megabyte (50,000+ C statements) [13].

Programming IHHT

Dynamic C is an integrated development environment (IDE) for writing embedded software. It runs on an IBM-compatible PC and is designed for use with Analogic Intelligent Hand Held Terminals (IHHT).

Dynamic C also supports assembly language programming. It is not necessary to leave C or the development system to write assembly language code. C and assembly language may be mixed together.

Dynamic C provides extensions to the C language (such as shared and protected variables, costatements and cofunctions) that support real-world embedded system development. Interrupt service routines may be written in C. Dynamic C supports cooperative and preemptive multitasking. Dynamic C comes with many function libraries, all in source code. These libraries support real-time programming, machine level I/O, and provides standard string and math functions.

Dynamic C compiles directly to memory. Functions and libraries are complied and linked and downloaded on-the-fly. On a fast PC, Dynamic C might load 30,000 bytes code in 5 seconds at a baud rate of 115,200 bps.

The IHHT is connected to the development system (PC) using a special programming cable.

2.2.5. Casio PVS-600

Casio PV® is a low-end PDA (Personal Digital Assistant) developed by Casio Inc. It has a very thin layer of proprietary operating system developed by Casio Inc called the PVOS (Pocket Viewer Operating System). The PDA connects to the software on the server running Microsoft© Windows® Operating System. It has all the features like contacts, notes etc., expected from an entry level PDA [14].

Features

Hardware

- CPU: NC3022 a 8-bit processor
- Core/Bus Clock: 20MHz
- 6MB Flash (OS & Application and User Data)
- 4 MB SD-RAM (for work (Program and data))
- LCD 160 x 160 mono-chrome Full dots with EL Back light.
- Power Supply: AAA-size battery (LR03) x 2 (120 hours of battery life)

Interfaces

- Touch-panel Overlay on LCD Display used with a plastic stylus
- USB Connector 1.1

System Software

- Operating System: Casio PVOS

Programming Environment

Casio Inc. provides an SDK for application development on Casio PV. The SDK comes with a Windows® application named “Casio Sim-SH” a simulator for NC3022 processor. There is a third party software called as PVIDE an integrated development environment for Casio PV, as well as a GUI designing tool called “PV Designer” that gives a VB like interface developing interfaces on Casio PV.

Programming Casio PV

Casio PV applications run one at a time, as the operating system is not a multitasking OS. When a new application has to run, the existing application has to be quit to run the new application. Casio Inc. has developed some libraries to use the calls of the PVOS and write the application.

- Most of the ANSI C libraries like "string" libraries can be used for writing applications.

- The Casio PV C is not completely compatible with ANSI C because of its library functions. Most frequently used functions in ANSI C like "printf()" are not available, instead some inbuilt library functions can be used for this purpose.

- As in ANSI C, the program starts with a "main" function. But the main function first clears the screen to draw the GUI of the new application and then initializes the touch areas.

- Unlike Embedded VC++ or any other advanced GUI development tool on desktop or any other PDA, it doesn't have advanced graphic libraries. Every GUI object like button or text box has to be drawn using lines and dots then a touch area has to be defined so that the respective action takes place.

- Touch areas are defined using a touch table. The touch table contains all the touch related data like the touch area, the function/operation(like displaying the keyboard) that has to take place when a particular touch takes place.

Some GUI based applications on Casio PV are explained in the Appendix section.

2.3. Evaluating PDAs

The above mentioned PDAs have been chosen depending on the potential for their use in different e-Governance projects. They are evaluated based on the following aspects:

- Price
- Performance
- Memory availability
- Ease of Application development
- Power consumption etc.

A Comparative Evaluation of PDAs

	Simputer	Palm Zire	Pocket PC	Analogic IHHT	Casio PVS-1600
Price	Rs 10,000	Rs 6000	Rs 20,000 (Approx)	Rs 16000 (Approx)	Rs 4,000
Processor	32-bit Strong Arm SA-1100 RISC CPU running at 200MHz	126MHz TI MAP 311 Processor	206 MHz Intel StrongARM 32 bit RISC Processor	High speed processor with 16 bit internal Architecture	NC3022 a 8-bit processor, 20MHz
Memory	16 MB of DRAM 8 MB Flash	2 MB Flash for Permanent Storage	RAM: 32 MB ROM: 16 MB Flash	256K Flash, 128K SRAM, 8MB Flash Disk (Optional)	6MB Flash
Ease of Application Development	Similar to developing applications with Tcl/TK on desktops	Availability of many tools makes the application development easier	Similar to that of applications for desktop	No OS, system tasks have to be taken care of by the program	Number of tools are less, tough compared to Simputer, Palm or Pocket PC
Power Consumption	6 to 8 hrs (Rechargeable)	6 to 8 hrs (Rechargeable)	12hrs (Rechargeable)	8hrs (Rechargeable)	180hrs (from 2 AAA batteries)
Local Language Support	Available	Available	Available (Some Indian Languages have not been developed)	Not Available as the Display does not support	Not available
Database Support	Available	Available	Available	Not Available.	Third Party tools are available
Communication Support	TCP/IP Stack, in-built modem	TCP/IP Stack, IR connectivity	TCP/IP stack, Wireless network connecivity	GSM, PSTN Modems in built	Not Available

2.4 Applications Development on PDAs

Designing applications for PDAs requires a different mindset than designing for a high-powered desktop computer with a large color screen whose users typically sit down and work for hours at a time. PDAs are low power, small, battery-operated devices that are used frequently but intermittently. Following are some of the considerations that has to be taken into account while developing an application on handheld devices:[15]

- **Screen Size:** Most PDAs have a small screen-as small as 160x160 pixels-limiting the information that the application can display at one time. Desktop applications often provide many up-front choices using tool bars. On PDAs, the main screen should contain only the most-wanted information, and nothing extraneous.

- **Usage Pattern:** The typical usage pattern for a PDA is many times a day for short periods of time. It is not uncommon for a handheld to be turned on 40 times a day for a total of 1 hour. PDA is not a small laptop, so one has to maximize application to these short bursts of user activity. Requiring a user to spend an extra 30 seconds to find the information they most often need is excusable when they are sitting down for hours at a desktop computer but cumbersome on a handheld when that is the only thing they are going to do before they turn it off.

- **Limited Input:** PDAs do not require users to enter a lot of data. Handwritten strokes, the on-screen keyboard, or even the miniature keyboards found on some handhelds are not as efficient as the keyboard and mouse on a typical desktop computer. Full-sized attachable keyboards are available, but they should not be made a requirement. Interface has to be provided to the user so that he/she can enter most of the information on the desktop and then transfer it onto the handheld later.

- **Power:** Although the processing speeds of some of the newer PDAs are approaching the desktop computer range, handhelds typically aren't designed to perform the same type of processing as a desktop PC. The complex tasks which use more processing power have to be offloaded onto a more high-powered desktop. The device-side application has to be limited to that of the information and tools that user will need when away from desks.

- **Battery:** Batteries are relatively small on a handheld, yet one of the keys to the success of PDAs is the device's outstanding battery life. With regard to battery power, a handheld is much more like a good cell phone than a laptop. Users depend on being able to use their device without needing to worry about whether or not it is charged. The more complex tasks like animation and other tasks that occupy the CPU for long periods of time have to be done away with if they can be.

- **Memory:** PDAs have a relatively limited amount storage space, and expansion is often limited (or not available). Optimization is extremely important, and has to be looked in the order of, heap space first, speed second, and code size third.

- **Data Storage:** PDAs like Casio PV and Palm store data in memory chunks called records, which are grouped into databases to maximize the use of the limited storage space. A database is analogous to a file, but the breaking down of a database into records allows the data to be stored in non-contiguous memory chunks. To save space, one has to edit data in the storage heap in place rather than creating or buffering it in RAM and later writing it to the storage heap.

Chapter 3

Low-cost PDA for Applications in e-Governance (Casio PVS-600)

As detailed in the last chapter a variety of PDAs with a good number of features are available commercially. The price of these PDAs is also proportional to features available on them. However in applications related to e-Governance, where a large number of PDAs is expected to be used, cost is the major consideration and thus there is a need to identify the low-cost device and develop potential applications useful for e-Governance.

In this work the PDA, Casio PVS-600, has been considered as a platform for development of applications related to e-Governance while this device is the most inexpensive of all PDAs considered, it has very limited features thereby limiting the applications that can be developed. As mentioned earlier, from e-Governance perspective, the following are the most important features that any PDA should have:

1. Database Management System for Application development

2. Communications support for remote data transfer

3. Local language support for developing applications in local languages

Accordingly, this chapter focuses on these features and demonstrates how they can be developed even in an extremely low-end PDA like Casio PV S-600 that is built on an 8-bit CPU.

3.1 Database Support of Casio PV

PDA are generally used as data collection units, that is they help the user to collect personal data, organize it in a particular fashion and transfer it onto a central server or a home PC. But unlike personal information on the PDA, the data for e-Governance tasks are huge, some time going into few tens of mega bytes. Organizing this data in a particular format and its efficient retrieval is required for most applications. For this reason database support is very essential for any PDA that is being used for e-Governance tasks.

Some e-Governance applications might have data which have nearly 10,000 records (as in case of patient records in a village). Searching through this humongous amount of data and retrieving the data to the user is a highly processor intensive task. The more processor intensive a task is the more time is required to retrieve the data. So a database system has to be made so that the retrieval times are less. In the same way, storing the

data should also be faster for efficient use of the device. Therefore the essential features of the database system that should exist for a PDA are:

- Fast data insertion
- Fast data retrieval
- Organization of data so that there will not be any data loss

Keeping these things in mind an attempt to design a database system for Casio PV has been made. Before knowing the internals and the implementation details of Casio PV it is essential to know the file system of Casio PV.

3.1.1. File System on Casio PV

The database format that Casio PV follows to store the data is called ADT format. As Casio PV stores the data in a database that is analogous to a file, ADT format can be treated as a file format. The ADT file format is as discussed below.

3.1.2. ADT File Format

Every ADT file has a header and a set of records. The header has a constant size of 256 bytes and the structure of the header is as follows:

Offset	Length	Description
0	22	Signature: "CASIO ADDIN DATAFILE"+0x1A+0x00
22	4	Length of file
26	4	Number of records in file
30	16	Name of file (we see this name on PV)
46	210	Unused tail (0x00)

The rest of the file contains the data in the form of records. Every record can be up to 30720 bytes. Before each record there is an associated length of the record which is 4-byte long. Every record is crypted, this crypted data can be retrieved using a 20-byte mask. That is, every byte of data has to be decrypted with the corresponding byte in the mask, it is more clearly explained later, as for now, the 20-byte long mask is as follows:

```
0x83,0x4A,0x83,0x56,0x83,0x49,0x8C,0x76,0x8E,0x5A,
0x8B,0x40,0x8A,0x94,0x8E,0xAE,0x89,0xEF,0x8E,0xD0
```

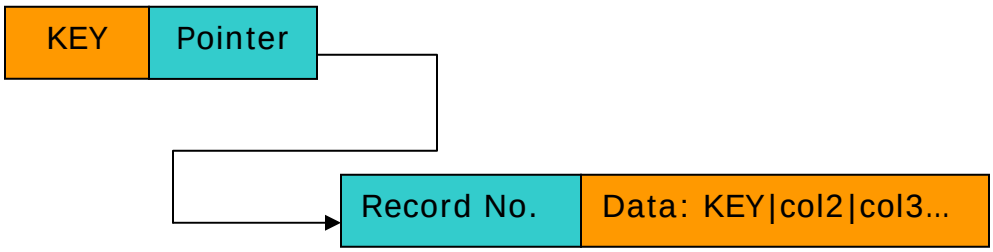
While retrieving the data, one must XOR every byte from the record with this mask. For example: First byte of the record must be xored with first byte of mask (`0x83`) second byte of record - with second byte of mask (`0x4A`) and so on and the 21st byte of record – with first byte of mask.

3.1.3. Database with ADT File

While the current ADT file format suites most of the requirements for a file database, there is one important component of a database which will help in inserting the data in an organized fashion and makes the retrieval of the data fast, that is indexing. There are different indexing schemes that are implemented depending on the size of the data and the usage of the application. The indexing scheme that is implemented in the current work is array indexing which is discussed below.

Array Indexing

This is a basic indexing scheme that can be used to store large amounts of data. Every file has a header and the database part. The header has information regarding the size of the data record size. The database part of the file has two parts, one is the index data and the second part is the data itself. Every record of the index data has two parts, one is the "Key" and the other is the "Pointer" to the record. Every record in the data part is of equal size. The pointer in the record part points to the respective record in the data part.



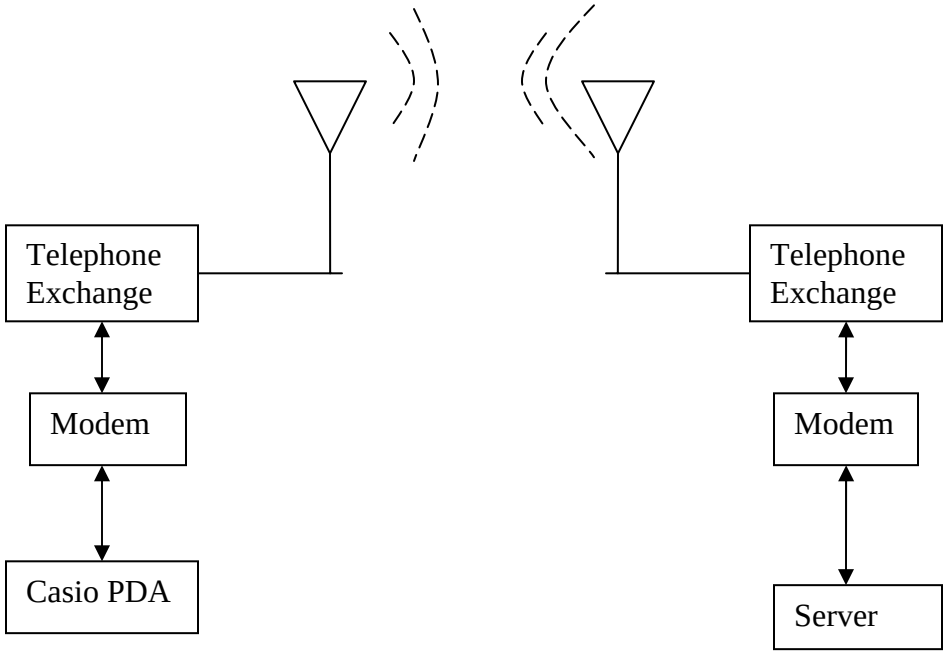
Inserting the data is also simple. The data is inserted at the end of the file and the corresponding key and pointer values are inserted into the index field. Once the index field is filled a new index field is created at the end of the file and the end of the first indexing data will point to the second indexing part. There can be two kinds of searching that can be implemented on the database, one is the indexed searching and other is sequential searching. In indexed searching as the application starts, the index data will be first loaded into the memory and when searched for a specific data, the index is searched and the data is retrieved. In sequential searching the file is searched completely byte-wise to retrieve the data. The deletion scheme that is used is a lazy deletion scheme. The assumption that was made before developing this scheme is that the number of deletions and insertions will be minimal and the number of deletions compared to the number of insertions will be lesser.

A pseudo code for implementation of array indexing is given below.

```
LoadIndexData()
{
     /*Traverse the whole file*/
     While (LibFileFindNext (&fb, &finf, 0x00))
     {

          /*Till all the indexing data is read in that
area*/
          While(LibFileFindNext(&fb, &finf, 0x00) == TRUE
&& i < fb.fbuf.bin.char_num)
          {
               Index[i].Pointer = fb.fbuf.bin.bin_buf;
          }
          LastIndexPointer = Index[i];
     }
}

ReadData(int Index[i].Pointer)
{
     LibFileFindNext(&fb, &finf, Index[i].Pointer);
}

InsertNewData(char *data)
{
     memcpy(fb.fbuf.bin.bin_buf, data, strlen(data))
     /*This writes data at the end of the file*/
     f_handle = LibFileWrite(&fb, &finf);
     LibFileFindNext(&fb, LastIndexPointer, 0x00);

     /*If the index end has reached, then start writing
     index in a new location*/
     if(fb.fbuf.bin.bin_buf == INDEX_END);
     memcpy(fb.fbug.bin.bin_buf, finf.fp, strlen(finf.fp);
     /*Writes data at the end of the file*/
     LibFileWrite(&fb,&finf);
}
```

3.2 Communication Support for Casio PV

Casio PV does not have an option to connect and work over a network. In this work, it is demonstrated how modem support can be provided with the help of a specially made male-male serial cable (details in the appendix) and using a very low-end communication protocol to enable remote data transfer. Casio PV code for setting serial port is explained in the appendix. The block diagram of the communication is as follows:



3.2.1. The Communication Protocol

- A standard command set by name AT Command Set or Hayes Command Set is used for the PDA to communicate with the Modem. A brief description to the Hayes command set is provided in the appendix [20].

- A new protocol, partly based on low-end SLIP protocol, has been designed and implemented to enable remote data transfer.

The protocol for transmitting the data over serial cable using Casio PV is as follows [21].

- The whole protocol runs on low-end SLIP protocol. The data in the packets is bound between ESC and END (Refer SLIP protocol in Appendix).

- The client (here Casio PV) sends the login and password in a single packet, the server receives the info and checks it with the database. If the info is correct the server sends an acknowledgement packet indicating that the login and password combinations are correct and connection is established. Else it sends another acknowledgement saying that the combination is wrong.

- The client then sends the name of the file/table that has to be received by the server. And the server checks whether the table name exists in its database or not. If it doesn't exist then it sends an acknowledgement asking for resending the name again. If the name is correct the server sends an acceptance acknowledgement.

- After this, the client sends each segment (in .adt file) as a packet. The server receives and sends an acknowledgement packet.

- After the data is sent, the server sends a termination packet (the empty packet with SLIP header and footer looks as follows: ESC ESC END END), i.e., an END in the packet, is sent.

- After this is done, the server sends a termination packet. Indicating that it has received the last packet and that no other data is present.

- If the server has to send any data, it sends it after receiving the termination packet and the data sent will be as follows

    - The server sends a start packet similar to ESC ESC ESC END. After this is received the client sends and acknowledgement packet and the data communication is established.

    - The server sends the name of the file/table that has to be sent.

    - The client receives the name checks if the name already exists. If so, the user is prompted to over-write the file and then over writes the file. If the name doesn't exist then the client checks whether it has enough labels to create a new file (Casio PV allows only 16 files to be written). If there are no labels or if the user doesn't want the old file to be written, a termination packet is sent and the communication ends.

Server to Client (Casio PV)

The server is continuously connected to the data modem. When the client dials the server, the server lifts the phone and initial settings are adjusted based on the above protocol. After the connection is established, the data transmission takes place as per the protocol explained above. After the client closes the connection, the following are the steps followed.

- The ADT file that is sent by the client (by default the client sends the file in ADT format) is converted to a flat file (.DAT or .TXT)

- Data is extracted from this flat file, (data might have table contents separated by a de-limiter).

- After the extraction of data, the data is entered into the data base.

Pseudo code for Modem Initialization

The following is the pseudo code used for initializing the modem.

```
StartCommunication()
{
     Char *mdmData;

     SetSerialPort();    /*Function explained in appendix*/
     SendData("ATE0");   /*Set Echo to zero*/
     RecieveData(mdmData);

     if(!strstr(mdmData, "OK")) /*Echo is not set to zero
successfully*/
     {
          LibPutMsgDlg("Error Setting Modem");
          Exit(0);
     }

     SendData("ATV1Q0"); /*Initializing Modem*/
     RecieveData(mdmData);

     if(!strstr(mdmData, "OK")) /*Modem is not Initialized
*/
     {
          LibPutMsgDlg("Error Initializing Modem");
          Exit(0);
     }

     SendData("ATXS10=255");  /*Set Register values,
without this modem communication won't work on some
telephone lines*/
     RecieveData(mdmData);

     if(!strstr(mdmData, "OK")) /*Modem settings are not
set*/
     {
          LibPutMsgDlg("Error Initializing Modem");
          Exit(0);
```

```
     }

     SendData(“ATDT123”) /*Dial telephone number 123*/

     for(i = 0; i < TIME_OUT; i++)
     {
       srl_err = RecieveData(mdmData);

       if(strlen(mdmData) > 0) /* If no data */
            break;
       else
            LibWait(IB_125MWAIT);      /* Wait for 125ms */
     }

     If(strstr(mdmData, "NO DIALTONE")) /*If no dialtone*/
     Print(“ERROR”);

     If(strstr(mdmData, "NO CARRIER"))  /*If no line is
connected*/
     Print(“ERROR”);

     If(strstr(mdmData, "CONNECT"))     /*If the call is
received by the server*/
     Print(“ERROR”);
}
```

3.2.2. Implementation

The remote data transfer has successfully been implemented by connecting PDA/Modem to a telephone line. The IIIT server was dialed from a remote village (Medchal, Ranga Reddy Dist.) and the data was successfully transmitted to the server.

3.3 Local Language Support of Casio PV

Casio PV has support for European languages like German, English, Spanish, French and Italian, but unlike European/Roman languages Indian languages have a complex script. While in this section implementation of Telugu script on Casio PDA is discussed, support for other Indian languages can be approached in the same way. A generic engine has been developed so that implementation of any language (including Arabic like scripts that are written from left to right and Chinese scripts that are written from top to bottom) needs only modification of few settings. A new font format is designed as the current font format doesn't support Indian Language fonts. The complexity of the Telugu script is described in the appendix.

3.3.1. Indian Language Fonts

The major difference between roman and Indic alphabets is that the latter has glyphs whose size is highly variable and characters are compositional. In roman fonts one can see one to one association between a character from the alphabet and the corresponding glyph. Not only is it hard to find such an association but in actual practice, as in Telugu, glyphs are often reduced to fewer number of primitive geometric shapes from which characters can be composed. So the size of the glyph inventory of Telugu from one font to another is so vast that it is often difficult to accomplish [31].

In roman script the internal representation of text in the editors are the same numeric codes (ASCII) assigned to glyph locations containing the letters. When it comes to fonts for Indian languages, the display has to be built up with more than one glyph for many aksharas (alphabets) and hence the internal representation of the aksharas is purely a function of where the glyphs for the aksharas are located within the font. The stored text is not in a format that can be viewed on different computer systems because, the encoding standard may not be supported in the system. Also, glyph codes are the choice of the font designer and will bear no relationship to the ordering of the aksharas in the scripts. Thus, linguistic processing of the stored text is a formidable task as the text is font dependent even for the same script. Shown below are the glyphs of two Devanagari script for two different fonts, which show that ASCII string required to display one font differ from the other [27].

Font 1 [27]

Font 2 [27]

From above figures we can notice that the number of glyphs required to display an akshara depend on the akshara and so variable number of bytes is required to represent each akshara. This is not the case with the roman/western alphabets. This shows that there is a need for some correlating table to convert data in one font to data in another font. The solution to one akshara to one glyph for Indian languages is not possible with the current 8-bit codes as there are only 256 glyphs that can be displayed but the number of aksharas are nearly 13000 (Because each akshara is a combination of consonants and vowels).

To solve the problem of correlating with different sequences of the same data for different fonts, a new standard called ISCII (Indian Script Code for Information Interchange) has been suggested on lines of ASCII.

In Indian languages the display of an akshara depends on the different combination of glyphs. Each glyph image is associated with various metrics which are used to describe how they must be placed and managed when rendering text. These relate to glyph placement, cursor advances as well as text layout. They are extremely important to compute the flow of text when rendering a string of text. Each scalable format also contains some global metrics, expressed in notional units, to describe some properties of all glyphs in the same face. Examples for global metrics are the maximum glyph bounding box, the ascender, descender and text height for the font. Though these metrics also exist for non-scalable formats, they only apply for a set of given character dimensions and resolutions, and are usually expressed in pixels.

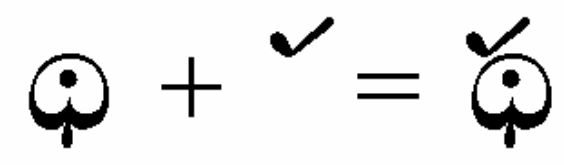
Rendering Indian language text using the bitmap fonts is not possible unless we have all the 13000 glyphs at our disposal. This is because the font metrics are very important when rendering Indian language text. The metrics help us to decide on how and where the glyphs have to be placed and managed.

Half Ha + Talakattu = Full Ha

The above figure shows Half Ha and Talakattu (matra) forming a full Ha. The different glyphs in the font are:

Half Ha and Talakattu

which do not have any meaning individually. Both the glyphs have the above mentioned metrics. There will be different kinds of Talakattu with different metrics to help the proper formation of different alphabets. It can be seen from below that full Dha is not properly formed because the metrics of the Talakattu do not suite for the formation of full Dha, so a new Talakattu glyph with different metrics is needed.



Half Dha + Talakattu = Full Dha

The metrics of Talakattu defines the co-ordinates where it has to start drawing the glyph and whether the cursor has to be advanced or not and if so in which direction and how many points. The coordinates help the glyph positioning properly and the cursor advancing property tells the word processor, how many points the cursor has to be advanced. In case of Talakattu it is zero.

Why a new font format?

The work for Indian Languages on Casio PV cannot be progressed with the current font format available in Casio PV as they do not have any means of defining the metrics of the font. The metrics include:

- Coordinates where the glyph has to be placed[29]
- The number of units the cursor has to be advanced, both in x and y direction.

Therefore this thesis suggests a new but simple font format which has the glyph bitmaps along with the metrics [29].

New font format

The file consists of glyphs (bitmaps) along with the metrics. Each glyph is defined as follows:

```
{
     bitmap_left, bitmap_top, advance_x, advance_y,
     {
      GSIZE (width_of_the_glyph, height_of_the_glyph),
          0xff, 0xc0,
          0xff, 0xc0,
          0xc0, 0xc0,
          0xc0, 0xc0,
          0xc0, 0xc0,
          0xc0, 0xc0,
          0xc0, 0xc0,
          0xc0, 0xc0,
          0xc0, 0xc0,
          0xc0, 0xc0,
          0xc0, 0xc0,
          0xc0, 0xc0,
          0xc0, 0xc0,
          0xc0, 0xc0,
```

Bitmap Data
(In Hex Bytes)

```
          0xff, 0xc0,
          0xff, 0xc0,
     }
}
```

Each glyph is separated by the curly braces that define the glyph boundaries.

The bitmap_left and bitmap_top define where the glyph has to be placed from the glyph origin (which will be explained later). The data advance_x and advance_y define where the cursor has to be placed from the glyph origin and they help in deciding whether the cursor has to be advanced or not, if it has to be then in which direction and how many units. Mostly these values will not be negative for Telugu Fonts. The values in GSIZE (width_of_the_glyph, height_of_the_glyph)tell us the size of the glyphs. The function GSIZE() is an inbuilt function in Casio PV SDK. It is defined as follows:

```
#define   GSIZE(x, y) ((x) & 0x00ff), ((unsigned int)(x) >>
8), ((y) & 0x00ff), ((unsigned int)(y) >> 8)
```

This function returns the size of bitmap.

A typical font file looks as follows:

```
glyph far telugu_font[3] = {
/* For char : 0 */
{
     0, 0,  0, 0,
     {
     GSIZE (0,0),
     }
},
/* For char : 1 */
{
     2, 15,  13, 0,
     {
     GSIZE (10,15),
     0xff, 0xc0,
     0xff, 0xc0,
     0xc0, 0xc0,
     0xc0, 0xc0,
     0xc0, 0xc0,
     0xc0, 0xc0,
     0xc0, 0xc0,
     0xc0, 0xc0,
```

```
      0xc0, 0xc0,
      0xc0, 0xc0,
      0xc0, 0xc0,
      0xc0, 0xc0,
      0xc0, 0xc0,
      0xff, 0xc0,
      0xff, 0xc0,
      }
},
/* For char : 2 */
{
      2, 15,  13, 0,
      {
      GSIZE (10,15),
      0xff, 0xc0,
      0xff, 0xc0,
      0xc0, 0xc0,
      0xc0, 0xc0,
      0xc0, 0xc0,
      0xc0, 0xc0,
      0xc0, 0xc0,
      0xc0, 0xc0,
      0xc0, 0xc0,
      0xc0, 0xc0,
      0xc0, 0xc0,
      0xc0, 0xc0,
      0xc0, 0xc0,
      0xff, 0xc0,
      0xff, 0xc0,
      }
}
};
```

The number in `glyph far telugu_font[3]` (here 3) defines the number of glyphs in the font.

3.3.2. Keyboard Layout

Keyboard is the main input device where different characters are typed in to display it on the screen. In normal keyboards, the keyboard gives the ASCII value of each key that is pressed. As discussed above, Indian languages are represented in ISCII. As the ISCII values range from 0 to 255 but only few of them can be entered using the standard keyboard, there are different keyboard layouts that have been designed keeping in view different styles of typing. For the data entry in Indian languages, the default option is INSCRIPT (INdian SCRIPT) layout, which uses the standard 101 keyboard. The

mapping of the characters is such that it remains common for all the Indian languages (written left to right).[30]

As discussed one can divide the characters of Indian language alphabets into Consonants, Vowels, Nasals and Conjuncts. Every consonant represents a combination of a particular sound and a vowel. The vowels are representations of pure sounds. The Nasals are characters representing nasal sounds along with vowels. The conjuncts are combinations of two or more characters. The Indian language alphabet table is divided into Vowels (Swar) and Consonants (Vyanjan). The vowels are divided into long and short vowels and the consonants are divided into vargs.

The INSCRIPT layout takes advantage of these observations and thus the organization is simple. In the INSCRIPT keyboard layout, all the vowels are placed on the left side of the keyboard layout and the consonants, on the right side. The placement is such that the characters of one varg are split over two keys. This keyboard layout has been adopted for local language support on Casio PV. The INSCRIPT layout for Telugu is as shown below.

Inscript Keyboard Layout for Telugu

3.3.3. Indian Languages on Casio PV: The Display and The Flow

There are two major parts which deal with the display and arrangement of the script on the screen. They are the rendering engine and layout engine. The rendering engine deals with the display part of the font on the screen depending on the properties of each glyph. Layout engine deals with the layout of the text on the display, which includes the display of fonts along with the proper cursor positioning (this is done with syllable splitting) and dealing with new lines, paragraphs and storage of data in a standard formats like ISCII or Unicode. Detailed description of each of these is given in the subsequent sections.

ISCII to Glyph Mapping and Keyboard Mapping

This deals with display of the font given an ISCII string. This is specific for a given font as the Glyph codes change from font to font and includes the mapping of an ISCII character or a combination of two or more to a particular glyph or glyphs.

As the font that is being used is Tikkana font and as the font designer can assign his/her own values to each of the glyphs (glyph indices) a table has to be maintained between ISCII and Tikkana glyph indices. Before that a table is required to convert the INSCRIPT keystrokes to ISCII values. The former is called a conversion table (or conversion chart) and the latter is called the Keytable.

Every character that is typed(taken in INSCRIPT) is converted to ISCII using the Keytable. This character along with the previously typed characters is sent for conversion from ISCII to Tikkana. This conversion module checks for the string occurrences in the conversion table and returns the corresponding string that contains the glyph indices. The rendering engine searches the font file for these glyph indices and displays the them on the screen. There are different issues like cursor positioning, line, paragraph splitting etc. These can be much efficiently implemented using syllable splitting. That is in the current scenario of searching the conversion table, the whole input string is taken and then given to the conversion module, which a tedious task if the string is very long. Instead if the string is split into syllables, then only syllable boundaries can be taken into consideration than the sentence boundaries.

Syllable Splitting

Syllable splitting is done using a standard state table. Almost all the Indian languages have the same character set and the language behaviors and taking this into account a regular expression is constructed. From this regular expression a state table or an NFA is generated with 38 different states and 7 final states. The details of the regular expression are discussed in the appendix.

After obtaining the NFA, the string is given to the NFA and the syllable boundaries are noted and only that syllable that is being currently edited will be sent to the conversion module. This improves the efficiency of the glyph conversion. Syllable splitting also helps in cursor position, i.e., once the syllable boundaries are noted, the cursor will be displayed only after the syllable boundaries and not between them, which is the main problem if the syllable splitting is not done.[32]

3.3.4. Rendering Engine

Once the font is ready, it has to be displayed on the screen. Which is handled by the rendering engine. The rendering engine uses the properties of each glyph and displays it on the screen.

There are two types of font rendering

- Stroke font rendering
- Bitmap font rendering

Stroke font rendering deals with rendering each character as a set of line segments. Bitmap font rendering is where each character is a bit map and is placed at the desired location on the screen. As the fonts that are used here are not stroke fonts but bitmap fonts, the rendering engine is a bitmap font rendering. As bitmaps are nothing but small images, all the rendering engine has to do is to display the glyph at the desired location. This desired location depends on the properties of the glyph, the screen size, the position of the previous glyph etc.

Usually font rendering includes different steps like

- Displaying the Font
- Font Hinting[28]
- Anti-aliasing[28]

Displaying the font deals with placing the bitmaps on the screen. Font Hinting, deals with fitting of the glyphs to the pixel grid. Anti-aliasing deals with the smudging of the glyph outlines to give a pleasant look to the eye. As hinting is used for scalable fonts and not for bitmapped fonts, font hinting is not used here. And as anti-aliasing requires more than 2 colors (at least grey scale) and Casio PV supports only two colors (black and white), anti-aliasing is also not used here.

As discussed above, after the conversion is done from ISCII to glyph indices, these respective glyphs have to be displayed on the screen. As glyphs here are bitmaps, a Casio PV library function is used to display the glyph on the screen.

```
void LibPutGraph(int x, int y, const byte far *graph)
```

The `x`, `y` are the co-ordinates of the display and the `graph` is the bitmap. The bitmap format in Casio PV is similar to that of the glyph bitmap that is defined above(font section). So the same bitmap data can be given to the function to display the glyph at the mentioned co-ordinates. To give a seamless effect of the display, the glyph that is being displayed in some cases has to be merged with the glyph is already at that position.

In the above figure, **LLa** appears at a stroke of a key, to make **LLa** to **LLaa** "Dheergham" has to be added. This can be done by erasing the first **LLa** and replacing it

by **LLaa** but this will require more computing power as we need to replace more pixels than needed. It is enough if we replace the "Talakattu" with the "Dheergham". The area where the "Dheergham" will be displayed has to be ORed with the glyph data of "Dheergham". That is, the area where "Dheergham" will be displayed will be captured, this data will be ORed with the "Dheergham" bitmap, this will make the combination of two glyphs look like one glyph. For capturing the bitmap data, a library function of Casio PV is used.

```
void LibGetGraph(int x, int y, int xsize, int ysize, byte
far *ubfptr)
```

The `x`, `y`, `xsize` and `ysize` define the bounding box of the bitmap that has to be captured, the captured bitmap will be stored in `ubfptr`. This data as discussed above is ORed with the glyph data that has to be displayed (here "Dheergham") and the resultant bitmap is displayed at the given co-ordinates (here `x` and `y)`.

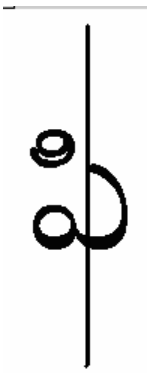
3.3.5. Layout Engine

The layout engine deals with the layout of the text on the display that includes the display of fonts along with the proper cursor positioning (this is done with syllable splitting) and dealing with new lines, paragraphs and storage of data in a standard formats like ISCII or Unicode.

To implement the layout engine, the main part is the syllable splitting. As discussed above, in syllable splitting the whole text is divided into syllables and the syllable boundaries are noted. Every thing the layout engine deals with is with the syllable boundaries. If a syllable boundary is crossing a screen boundary then the syllable is put in the next line. To do a perfect text editing, word boundaries have to be noted along with syllable boundaries, but here only syllable boundaries are noted.

Similarly, even the cursor positioning is handled. That is the cursor is only placed at the syllable boundaries, which is better than placing the cursor near the glyph boundaries. An example of placing the cursor near the glyph boundaries and syllable boundaries is show below.

Cursor on Syllable Boundary



Cursor on Glyph Boundary

The figure on the left is when the cursor is placed on the syllable boundary and the figure on the right is the cursor placed on the glyph boundary.

The current Layout engine can be adopted to any Indian Language that is written from left to write. With some minor changes languages that are written from right to left like Arabic and Urdu, can be supported. Similarly languages that are written vertically like Chinese and Japanese can be supported. Currently word splitting is not done but only syllable splitting is done. This will be a problem for writing multiple lines, because in some cases the word will be split at the line boundary making first part of the word in the above line and the next part in the below line. But taking care of word boundaries and implementing them increases the memory used by the program and hence the engine will become heavier for the target hardware to handle.

Chapter 4

A New PDA Architecture

Attempts have also been made to design a new, low cost, indigenous PDA based on an 8-bit processor that may be useful for e-Governance applications. This chapter discusses the architecture of this low-cost PDA.

4.1 8-bit PDA

The PDA was designed keeping in view of the demand for low cost hand held devices for data storage and retrieval. Unlike the other Data Organizers available presently in the market, it is a highly customizable data storing device. It has a Matrix Keyboard as an input device and a Liquid Crystal Display as an output device. The data is stored in 128KB Non-Volatile Static RAM. It has function keys to store different kinds of data and computer connectivity over serial port for transferring the data. The data template can be dynamically changed over a PC and then transmitted to the PDA over the serial port.

Hardware:

The following hardware was used to build the prototype PDA.

- Philips 89C51 RD+ Microcontroller
- Lampex 20x4 Liquid Crystal Display with back light
- A Custom made Matrix Keyboard
- 8255 connector for connecting the I/O devices
- 128KB Non-Volatile Static RAM
- RS232 Serial Port Interface
- Real Time Clock

4.1.1 Central Processing Unit: 89c51RD+ Micro controller

The 89C51RD+ MC contains a non-volatile FLASH program memory of 64 k bytes that is both parallel programmable and serial in-system programmable. In-System Programming allows devices to alter their own program memory, in the actual end product, under software control. This opens up a range of applications that can include the ability to field update the application firmware.

A default serial loader (boot loader) program in ROM allows In-System serial programming of the FLASH memory without the need for a loader in the FLASH code. User programs may erase and reprogram the FLASH memory at will through the use of standard routines contained in ROM.

Features of MC 89C51RD+

- 80C51 Central Processing Unit
- On-chip FLASH Program Memory with In-System Programming (ISP) capability
- Boot ROM contains low level FLASH programming routines and a default serial loader
- Speed up to 33 MHz
- Full static operation
- RAM expandable externally to 64 k bytes
- 4 level priority interrupt
- 7 interrupt sources, depending on device
- Four 8-bit I/O ports
- Full-duplex enhanced UART
    - Framing error detection
    - Automatic address recognition
- Power control modes
    - Clock can be stopped and resumed
    - Idle mode
    - Power down mode
- Programmable clock out
- Second DPTR register
- Asynchronous port reset
- Low EMI (inhibit ALE)
- Watchdog timer

4.1.2 Input Device: Matrix Key Board

Keyboard uses a matrix with the rows and columns made up of wires. Each key acts like a switch. When a key is pressed, a column wire makes contact with a row wire and completes a circuit. The keyboard controller detects this closed circuit and registers it as a key press. A more detailed description of the matrix key board is given in the appendix [34].

4.1.3 Output Device: Liquid Crystal Display

Lampex 20x4 LCD with Back Light is used as the display.

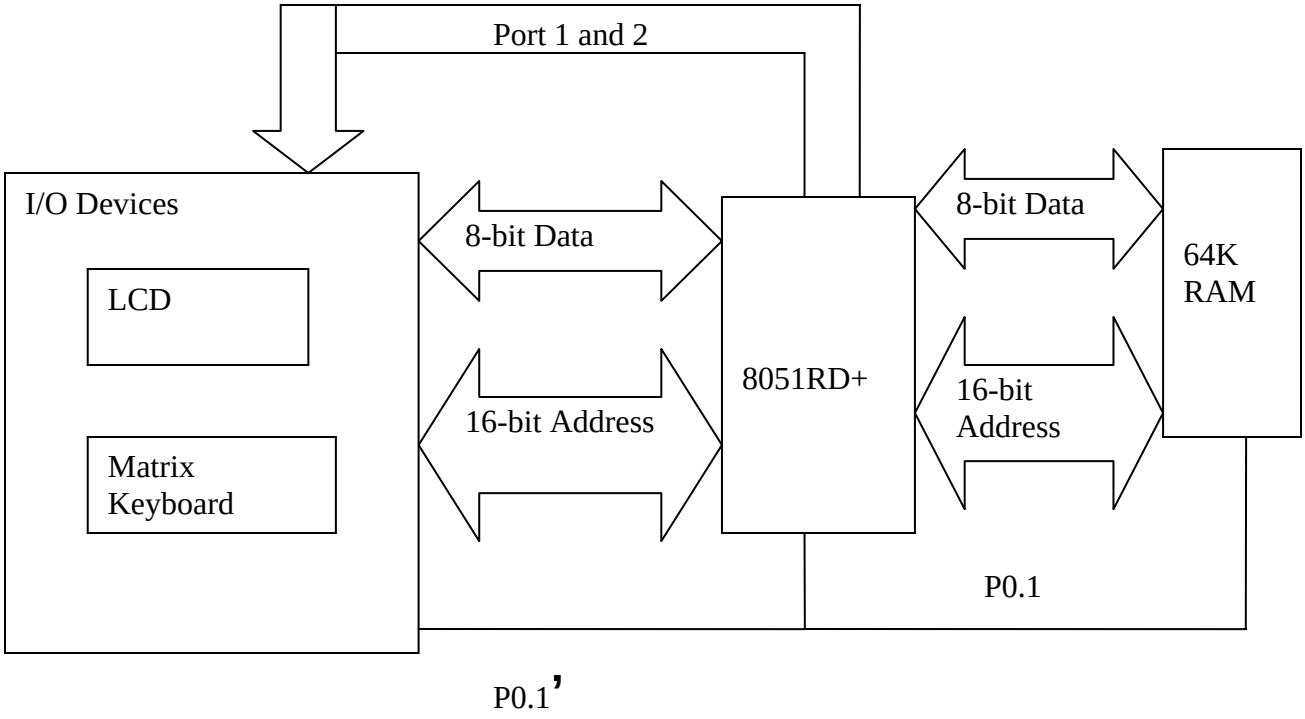
4.1.4 RS232 Serial Port Interface

The 8-bit PDA has a Serial Port Interface to communicate with the server side software on the PC. This helps in transferring data from the PDA to the PC for backing up data and also for transferring the template in which the data has to be collected.

8051 provides a transmit channel and a receive channel of serial communication. The transmit data pin (TXD) is specified at P3.1, and the receive data pin (RXD) is at P3.0. The serial signals provided on these pins are TTL signal levels and must be *boosted* and *inverted* through a suitable converter to comply with RS232 standard. All modes are controlled through SCON, the Serial CONtrol register. The SCON bits are defined as SM0, SM1, SM2, REN, TB8, RB8, TI, RI from MSB to LSB. The timers are controlled using TMOD, the Timer MODe register, and TCON, the Timer CONtrol register [35].

4.1.5 PDA Block Diagram

The block diagram of the prototype PDA is as shown below. The microcontroller is interfaced to a 64KB RAM and I/O devices (LCD, Keyboard, Serial port) both through the data and address buses. The P0.1 bit is used to select between the I/O devices and the RAM. Port1 and 2 are connected to the Keyboard to scan the keypad for the keypresses.



4.1.6 Software

Software has been developed both on the PC and the PDA to customize the PDA to the user's needs. The software developed can be classified as follows.

- Software on the PC
- Software on the PDA

Software on the PC

The software on the PC is used to define the interface of the software that is used on PDA, it helps the user to download this definition onto the PDA and also upload the data from the PDA.

Software on the PDA

The software on the PDA interprets the definition that is downloaded to give an interface for the user for data entry. The software also stores the data in the RAM in the defined format, which can later be uploaded on to the PC.

The software can be divided into the following modules:

a) Key Board scanning
b) LCD displaying module

c) Loading the template
d) Storing the records
e) Scanning the records one on one.
f) Searching the records based on fields of the template.

The LINKED LIST implementation:

The data on the PDA is directly stored on the RAM. The PDA doesn't have any file system to store the data efficiently, to avoid wastage of memory when using multiple records the concept of linked list is used. There is a pointer at the end of each record which tells us the location of the next record. The advantages of using linked lists are:

a) Enhances search efficiency

b) Optimum usage of memory

Chapter 5

Conclusions

National e-Governance action plan proposes to use handheld devices (PDAs) on a large scale to collect citizen related data as well as to bridge digital divide between haves and have-nots. A large variety of handheld devices with a variety of features exist in the market that suite the purpose. However most of the commercially available PDAs are expensive and therefore implementation of such an action plan is necessarily going to be expensive. Thus there is an urgent need to identify a low-cost platform and design/implement features that are relevant for e-Governance.

One of the contributions of this work is to identify such a low-cost PDA (Casio PV S-600) and demonstrate how tools/applications that are most relevant to e-Governance can be designed/developed. The features implemented include Database support, Communication support for remote data transfer and Local Language support. Also since this PDA runs on AAA batteries with long life, it is also ideally suitable for Indian conditions where large areas are without any electric power.

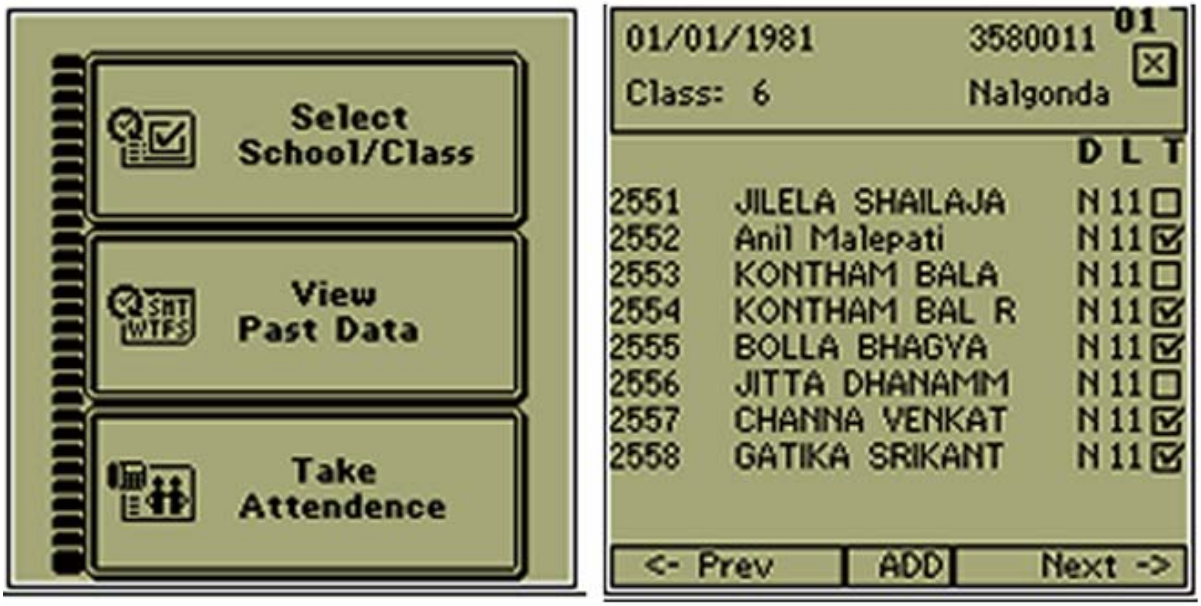
The second contribution of this work is to design/develop a prototype PDA that can be built indigenously. However much work needs to be done to make it commercially viable.

Appendix 1

PDA based Applications

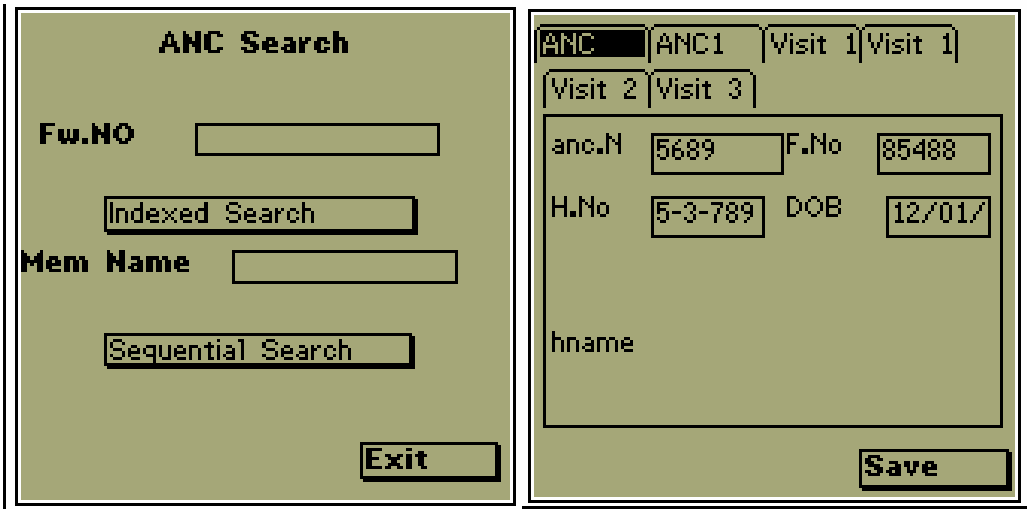
Attendance Monitoring in Schools using PDAs

The following figures show a few applications that have been developed on Casio PV S-600.



Attendance Monitoring in Schools

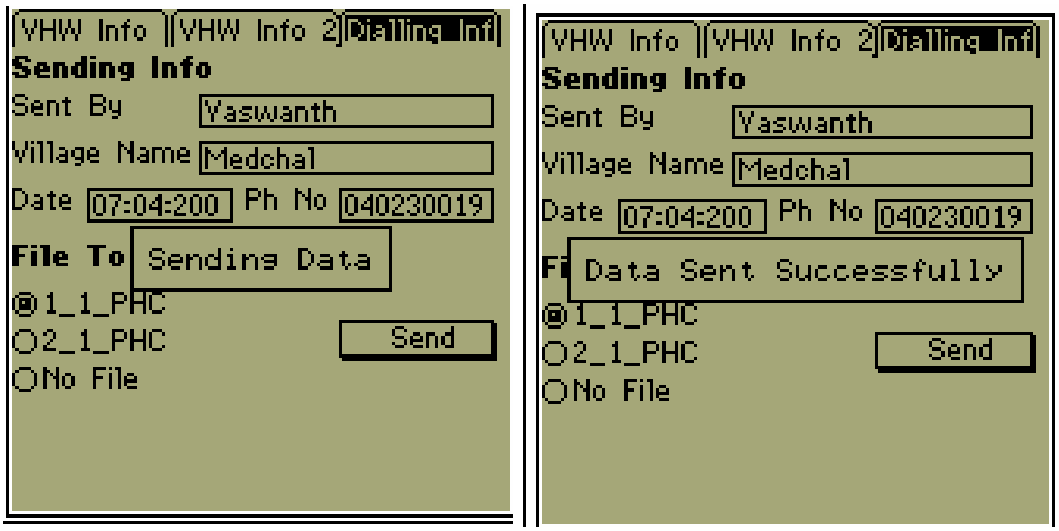
Patient Monitoring in PHCs



Anti Natal Care (ANC) Search **ANC Results**

Modem Communication using Casio PV

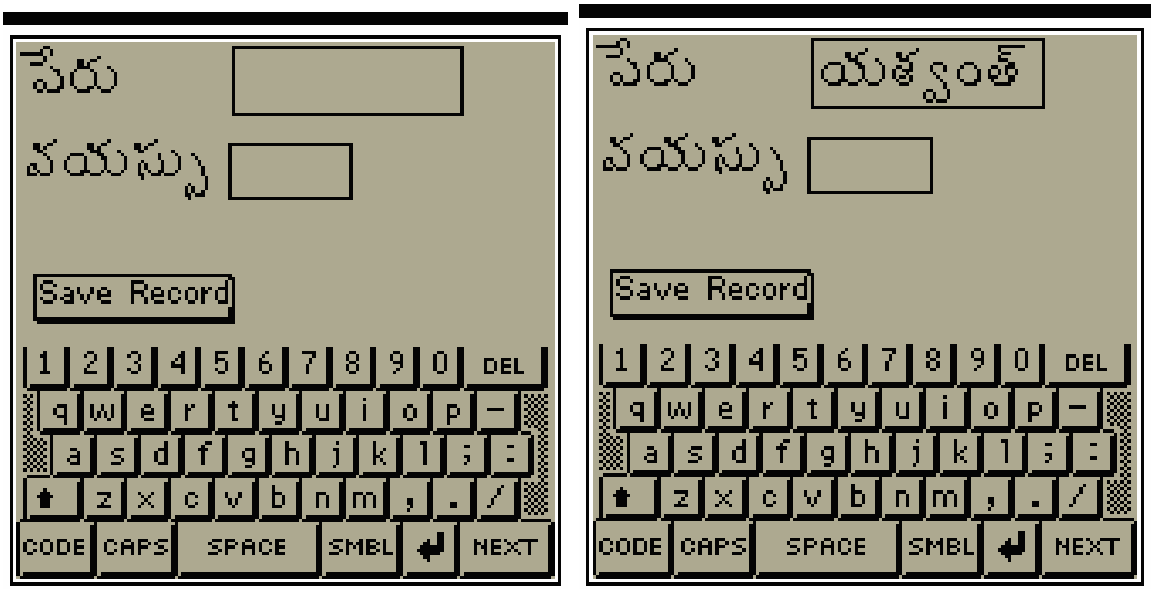
In a demo application a modem has been connected to Casio PV and the data has been successfully transferred from a remote village to a server in IIIT



Modem Communication: Started **Successfully Completed**

Local Language Support for Casio PV

Local language support was developed for Casio PV. Screenshots of an application are as follows:



Telugu Font Demo Application

Appendix 2

Modem Communication

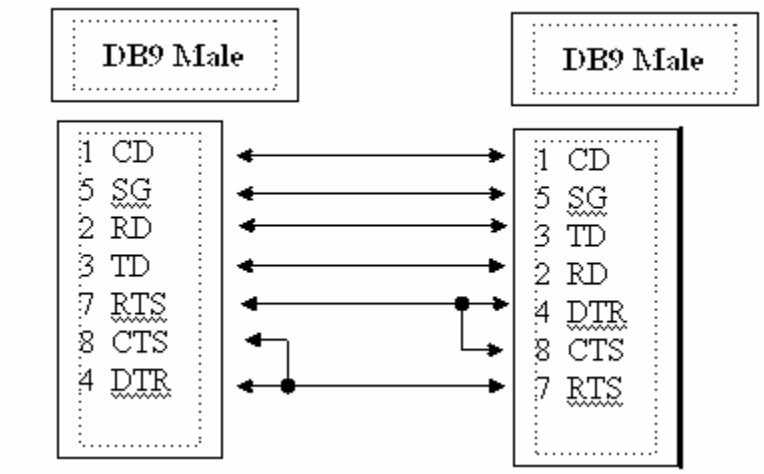
General rules for using AT commands

- Type 'AT' before each command and '/r' after each command. **Note:** The exceptions are A/, A> and +++, which require neither AT nor '/r'.
- Leave zeroes off the end of AT commands. A missing numeric parameter is assumed to be a zero. For example, ATE is equivalent to ATE0.
- Either use 'AT' (all caps) or at. Mixed case, as in At for example, is unacceptable.
- One can create compound commands of up to 56 characters between AT and '/r'.

The command AT informs the modem that a command is coming. AT must precede all commands [20].

Modem connection to Casio PV-S660

The cradle of Casio PV-S660 is not made to connect it directly to the modem. So a specialized cable is needed to connect the cradle of the PV to Modem. The following pin configuration is used for connecting Casio PC cradle to modem (RS-232 Converter). RTS is connected to DTR and CTS to make the terminal and the controller to know that they are ready to send and receive the data. To make the modem and Casio PV talk, the TD of the first is connected to the RD of the second and vice-versa. The principal behind this is when we speak, the other person hears with his ear, which is connecting the mouth of first person to the ear of second and vice-versa.



Max connection speed = 38400 bps, but 9600 bps will be more reliable.

SLIP Protocol

SLIP has its origins in the 3COM UNET TCP/IP implementation from the early 1980's. It is merely a packet framing protocol: SLIP defines a sequence of characters that frame IP packets on a serial line, and nothing more. It provides no addressing, packet type identification, error detection/correction or compression mechanisms. Because the protocol does so little, though, it is usually very easy to implement.

The SLIP protocol defines two special characters: END and ESC. END is octal 300 (decimal 192) and ESC is octal 333 (decimal 219) not to be confused with the ASCII ESCape character. To send a packet, a SLIP host simply starts sending the data in the packet. If a data byte is the same code as END character, a two byte sequence of ESC and octal 334 (decimal 220) is sent instead. If it the same as an ESC character, an two byte sequence of ESC and octal 335 (decimal 221) is sent instead. When the last byte in the packet has been sent, an END character is then transmitted. Phil Karn suggests a simple change to the algorithm, which is to begin as well as end packets with an END character. This will flush any erroneous bytes which have been caused by line noise. In the normal case, the receiver will simply see two back-to-back END characters, which will generate a bad IP packet. If the SLIP implementation does not throw away the zero-length IP packet, the IP implementation certainly will. If there was line noise, the data received due to it will be discarded without affecting the following packet. Because there is no 'standard' SLIP specification, there is no real defined maximum packet size for SLIP. Deficiencies of SLIP are:

- Addressing: both computers don’t know the IP address of the other, so routing will be a problem

- Type identification: Only one protocol like TCP/IP can run and two protocols cannot share a serial line.

- Error Detection and Correction: Noisy phone lines will corrupt the packets and there is no way to detect an error. But we can implement higher lever protocols which will use error detection and correction mechanism.

- Compression: There is no default way of implementing compression over SLIP. We can use our own compression mechanisms to compress the data. As we are transmitting data with Casio PV, compression process might take more time than it reduces.

(Refer rfc1055 for detailed description and SLIP drivers).

Casio PV code explained

```
void SetSerialPort()
/*Function to set the serial port settings*/
{
  word err;
/* Startup stuff*/
  srl_status.port = IB_SRL_COM2;   /*9Pin Serial Port*/
  srl_status.speed = IB_SRL_9600BPS;   /*9600 bps*/
  srl_status.parit = IX_SRL_NONE;  /*No Parity*/
  srl_status.datab = IX_SRL_8DATA; /*8 bit length data*/
  srl_status.stopb = IX_SRL_1STOP; /*1bit Stop*/
  srl_status.fctrl = IX_SRL_NOFLOW;  /*RS/CS control*/
  err = LibSrlPortOpen(&srl_status); /*Opening the Serial
Port*/
  if (err != IW_SRL_NOERR)    /*Checking for Error*/
  {
   return;
  }
}

/*Inbuilt function which receives a block of data,
noofrecvbytes tells the no of bytes received,
LibSrlSendBlock is similar function to send data*/

word LibSrlRecvBlock(InBuffer, BUFFER_SIZE,
&noofrecvbytes);

word LibSrlRxBufClr(); /*Clears the receive buffer*/
/*Sends a byte when serial port is not busy, LibSrlRecvByte
is a similar function to receive data*/

word LibSrlSendByte(IB_FOLLOW_BUSY, byte);
```

Appendix 3

Local Language Support: Telugu

Telugu Script Complexity

Telugu Script is a descendant of the ancient Brahmi script. Telugu script is written from left to right and consists of sequences of simple and/or complex characters. The script is largely syllabic in nature - the basic units of writing are syllables. Since the number of possible syllables is very large, syllables are composed of more basic units such as vowels (“achchu” or “swar”) and consonants (“hallu” or “vyanjan”). Consonants in consonant clusters take shapes which are very different from the shapes they take elsewhere. Consonants are presumed to be pure consonants, that is, without any vowel sound in them. However, it is traditional to write and read consonants with an implied 'a' vowel sound. When consonants combine with other vowel signs, the vowel part is indicated orthographically using signs known as vowel maatras. The shapes of vowel maatras are also very different from the shapes of the corresponding vowels. The overall pattern consists of 60 symbols, of which 16 are vowels, 3 vowel modifiers, and 41 consonants. Spaces are used between words as word separators. The sentence ends with either a single (“purna virama”) or a double bar (“deergh virama”). They also have a set of symbols for numerals, though Arabic numbers are typically used. These are shown in the illustration below. For reference, the aksharas are shown along with the equivalent Roman Diacritic representations.

అ	ఆ	ఇ	ఈ	ఉ	ఊ	ఋ	ఎ	ఏ	ఐ	ఒ	ఓ	ఔ	అం	అః
a	ā	i	ī	u	ū	ṛ	e	ē	ai	o	ō	au	aṃ	aḥ

Vowels

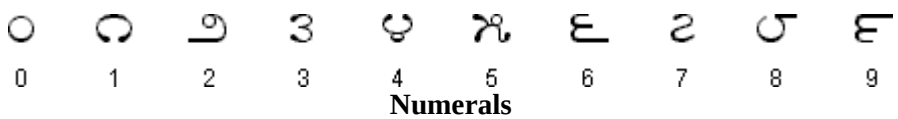
క	ఖ	గ	ఘ	ఙ	చ	ఛ	జ	ఝ	ఞ
ka	kha	ga	gha	ṅa	ca	cha	ja	jha	ña

ట	ఠ	డ	ఢ	ణ	త	థ	ద	ధ	న
ṭa	ṭha	ḍa	ḍha	ṇa	ta	tha	da	dha	na

ప	ఫ	బ	భ	మ		క్ష	జ్ఞ	ఱ	ళ
pa	pha	ba	bha	ma		kṣa	jña	rra	lla

య	ర	ల	వ	శ	ష	స	హ		
ya	ra	la	va	śa	ṣa	sa	ha		

Consonants



Numerals

Casio PV Font Format

Casio Fonts are developed by Casio Inc. for its low-end handheld devices called Casio PV. The fonts are in the form of bit maps. The detailed format is as follows: Fonts in Casio PV are defined in 3 separate files

- FONT00.SRC – Defines the normal fonts
- FONT01.SRC – Defines the bold fonts
- FONT02.SRC – Defines the title fonts

A font can be added only as a font for displaying a fixed message. Code positions of fonts that can be added are:
0Ah – 0Ch, 0EH-0FH, 15H-17H and E8H-EFH (16 in total).
The structure of the font file is as follows (Taken from the Casio PV 1600 Localize SDK):

A font file consists of information that describes identification (ID), version information, font size, etc. at the top and a font table. For font size, one has to specify the width and height in dots. If the width is set to 0, the target font is considered to be a proportional font, and one must set the width according to data of each character.

The font table has a structure that is composed of two tables. In PV, one table that starts from **LTABLE00:** in the source codes of each font stores offset addresses to font data including undefined fonts from **0x20 to 0x7E**. And the other table that starts from **LTABLE00:** stores offset addresses to font data including undefined fonts from **0x00 to 0x1F** and from **0x7F to 0xFF**.

Each of the two tables contains 256 offset address definitions and stores addresses corresponding to their respective numbers. However, when the value of an offset address set here is **0xFFFF**, there shall be no font that corresponds to it.

We explain the matter using redrawing of a font in the normal font (**FONT00.SRC**) file.

In each of LTABLE00 and LTABLEF8 tables, address T00XX (XX is the hexadecimal character number) to font data of the corresponding character and an offset address calculated from its base address BASE00xx with BASE00xx as 0 are set. To T00XX, data of the font is set in the format explained in 2) below. If you want to change the font shape of an existing code, modify the data on which the applicable label is set, recompile the resources and reconstruct the OS to reflect the change on the system.

The procedure of adding fonts

If one is adding a font that has the character code 0xE8, for example, the offset table for the font that one is changing is LTABLEF8, because 0xE8 is a character that is in the range between 0x7F and 0xFF.

Therefore, one has to set T00E8-BASE00xx for the table of E8.

Then, one has to set the label T00E8: at the end of main data of the font, and describe there the width data and bitmap data of the font as explained in “Structure of font data” below.

Finally, one must execute MAKERES.BAT and MKOS.BAT to reconstruct the OS. By this, a font with character code E8 is added to the device.

Structure of font data

Font data is defined with the MSB on the left side of the screen.
Data is represented with the digits aligned to the MSB side.

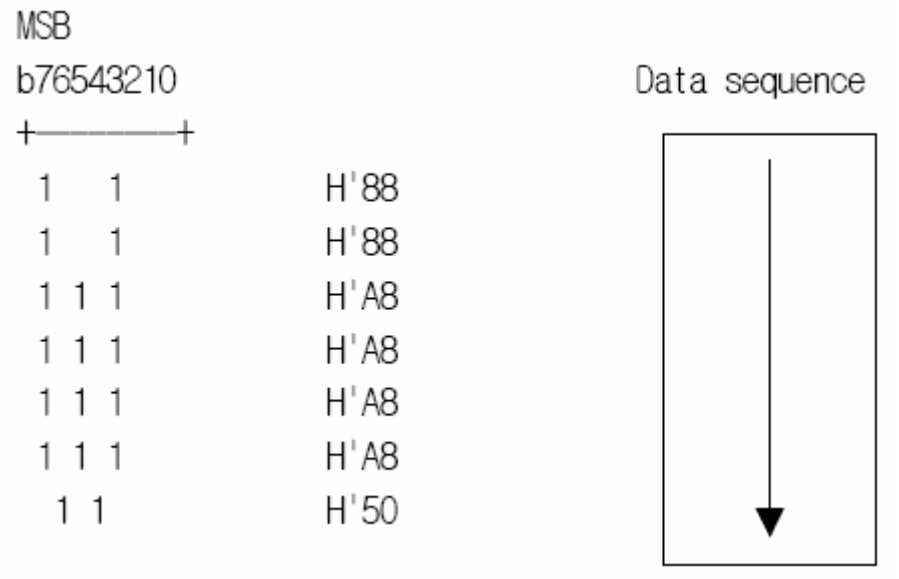
The structure is explained by taking an alphabet “W” as an example below.

Fixed-width font:

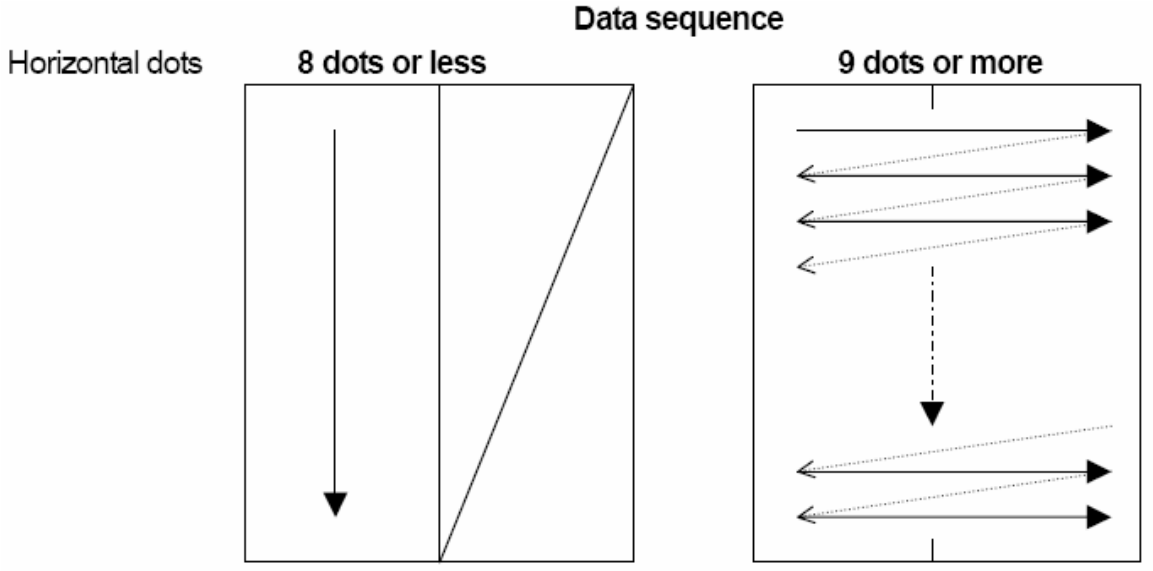
Font in FONT57.SRC:

Because the font is a fixed width font, all the glyphs will have the same width. And as the width value is not 0, the value of the width is used as font width. In this case the data of font width is not included in the actual glyph data.

```
WIDTH: .DATA.B H'06
HIGHT: .DATA.B H'07
.............
T057: .DATA.B H'88,H'88,H'A8,H'A8,H'A8,H'A8,H'50, ; Shape data of font of W
```



Proportional font:



Font in FONT00.SRC:

Because WIDTH is 0, meaning the font is a proportional font, width data is added for each font on the actual data side.

```
WIDTH: .DATA.B H'00
HIGHT: .DATA.B H'08
.........................
T0057: .DATA.B H'08 ; Font width of W
.DATA.B H'92,H'92,H'92,H'AA,H'AA,H'44,H'44,H'00 ; Shape
data of font of W
```

```
MSB
b76543210 76543210
+--------+--------+
 1   1   1                  H'92
 1   1   1                  H'92
 1   1   1                  H'92
 1 1 1 1                    H'AA
 1 1 1 1                    H'AA
  1     1                   H'44
  1     1                   H'44
                            H'00
```

Data in other files viz. FONT01.SRC and FONT02.SRC are added in the same fashion.

```
WIDTH:   .DATA.B  H'00
HIGHT:   .DATA.B  H'08

          ..........................

T0057:   .DATA.B  H'0A                                          ; Font width of W
         .DATA.B  H'C9,H'80,H'C9,H'80,H'DD,H'80,H'DD,H'80       ; Shape data of font of W
         .DATA.B  H'77,H'00,H'77,H'00,H'22,H'00,H'00,H'00

                  MSB
                  b76543210 76543210
                  +--------+--------+
                   11   1   1 1              H'C9 H'80
                   11   1   1 1              H'C9 H'80
                   11 111 1 1                H'DD H'80
                   11 111 1 1                H'DD H'80
                    111 111                  H'77 H'00
                    111 111                  H'77 H'00
                     1   1                   H'22 H'00
                                             H'00 H'00
```

Font in FONT01.SRC

```
WIDTH:   .DATA.B  H'00
HIGHT:   .DATA.B  H'0c

          ..............................

T0057:   .DATA.B  H'0A      ; Font width of W
         ; Shape data of font of W
         .DATA.B  H'C9,H'80,H'C9,H'80,H'C9,H'80,H'DD,H'80,H'DD,H'80,H'D5,H'80
         .DATA.B  H'D5,H'80,H'77,H'00,H'63,H'00,H'63,H'00,H'00,H'00,H'00,H'00

                  MSB
                  b76543210 76543210
                  +--------+--------+
                   11   1   1 1              H'C9 H'80
                   11   1   1 1              H'C9 H'80
                   11   1   1 1              H'C9 H'80
                   11 111 1 1                H'DD H'80
                   11 111 1 1                H'DD H'80
                   11 1 1 1 1                H'D5 H'80
                   11 1 1 1 1                H'D5 H'80
                    111 111                  H'77 H'00
                    11   11                  H'63 H'00
                    11   11                  H'63 H'00
                                             H'00 H'00
                                             H'00 H'00
```

Font in FONT02.SRC

The above described font format made by Casio Inc. is not useful for displaying Indian Languages.

Syllable Splitting

Syllable splitting as discussed earlier helps in increasing the efficiency of dealing with the text. Syllable splitting is done using a state machine implementation, the regular expression for the syllables in Indian Languages are as follows:

```
DHEERGHAM (D)=[\xa1\xa2\xa3]
VOWELS (V) =
([\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\
xb2]|[\xa6\xa7\xaa]\xe9)
CONSONANTS (C) =
[\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\x
c1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\
xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9]
MATRAS (M) =
([\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7]
|[\xdb\xdc\xdf]\xe9)
HALANT (H) = [\xe8]
NUKTA (N) = [\xe9]
OM = (\xa1\xe9) /* Specific for Devanagari Script) */
AVAGRAH = (\xea\xe9)
FULL_STOP = (\xea)
DIGIT = [\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa]
LATIN =
[\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x
2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\
x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c
\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5
b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x
6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\
x79\x7a\x7b\x7c\x7d\x7e]
CONJUNCT = ({C}{N}?({H}{C}{N}?)+{M}{D})
VOWEL_SYLLABLE = ({V}{D}?{AVAGRAH}?)
CONSONANT_SYLLABLE =
({C}{N}?({H}{C}{N}?)*{M}?{D}?{AVAGRAH}?)
```

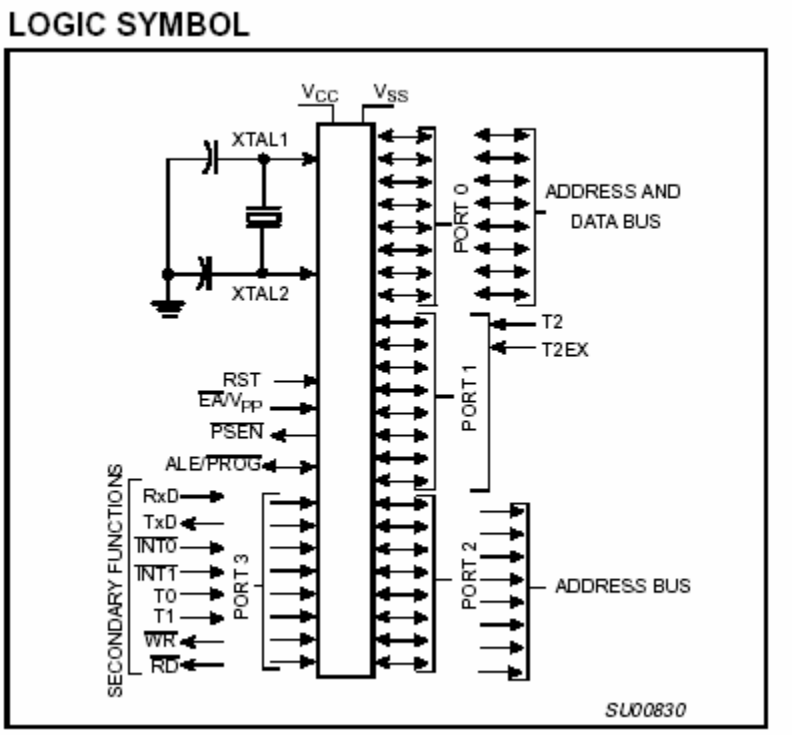
The DFA for the above regular expression is generated and implemented, the DFA that is generated has 38 states and 7 are stop or accept states.

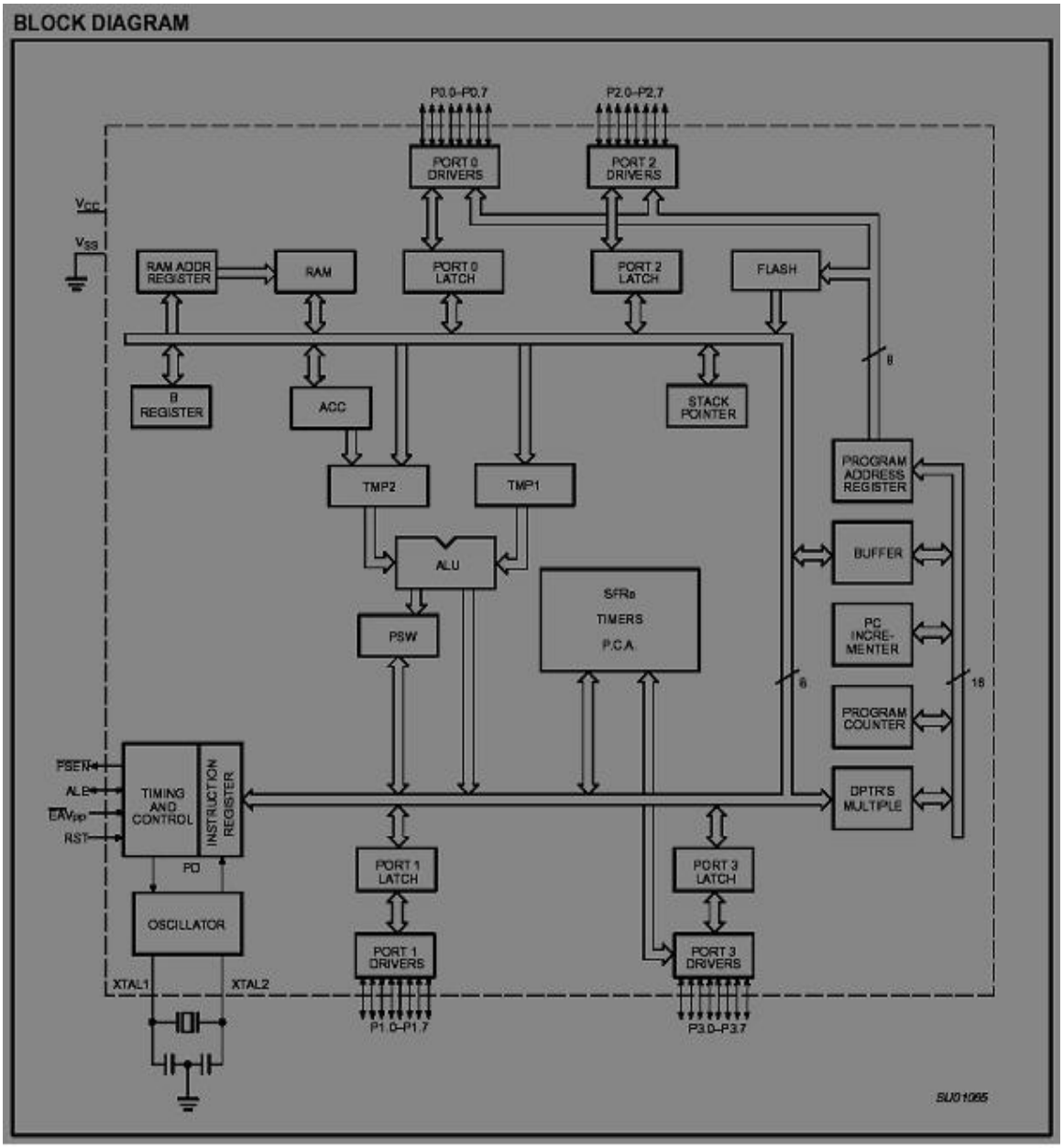
Appendix 4

PDA Design

8-bit PDA

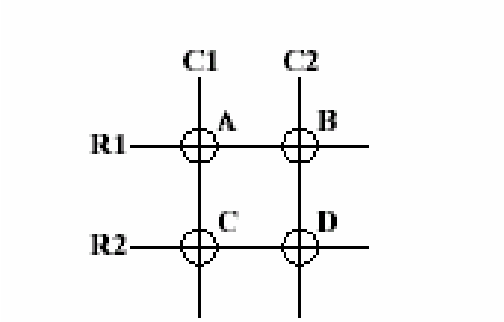
Central Processing Unit: 89c51RD+ Micro controller



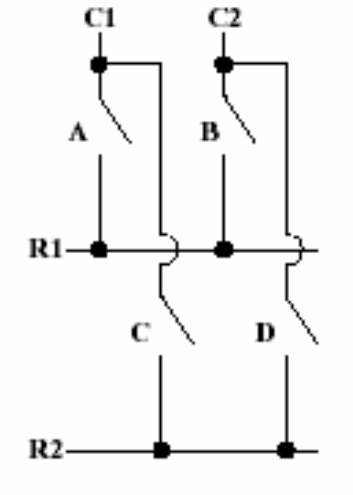

BLOCK DIAGRAM
P0.0–P0.7
P2.0–P2.7
PORT 0 DRIVERS
PORT 2 DRIVERS
VCC
VSS
RAM ADDR REGISTER
RAM
PORT 0 LATCH
PORT 2 LATCH
FLASH
B REGISTER
ACC
STACK POINTER
8
PROGRAM ADDRESS REGISTER
TMP2
TMP1
ALU
BUFFER
SFRs
TIMERS
P.C.A.
PSW
PC INCRE-MENTER
6
16
PROGRAM COUNTER
PSEN
ALE
EAVpp
RST
TIMING AND CONTROL
INSTRUCTION REGISTER
DPTR'S MULTIPLE
PD
PORT 1 LATCH
PORT 3 LATCH
OSCILLATOR
PORT 1 DRIVERS
PORT 3 DRIVERS
XTAL1
XTAL2
P1.0–P1.7
P3.0–P3.7
SU01065

Matrix Key Board

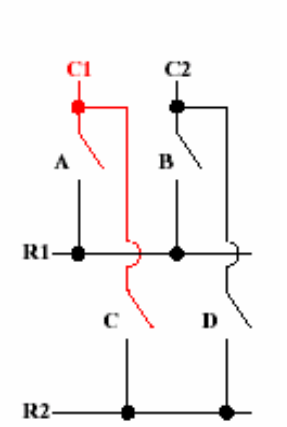
Here is a simple keyboard matrix:



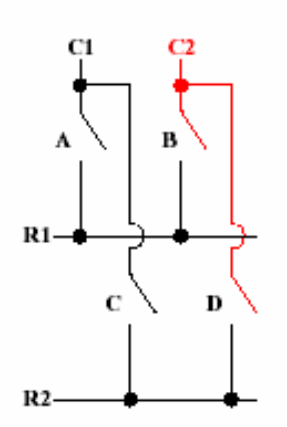
This keyboard only has 4 keys: *A*, *B*, *C*, and *D*. Each key has a unique grid location, much like points on a graph. Key *A* is at node C1R1, key *B* is at node C2R1, key *C* is at node C1R2, and key *D* is at node C2R2. The electronic circuit for this matrix looks something (not exactly) like this:



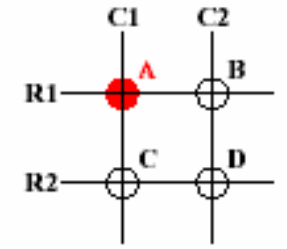
In order to detect key presses, the keyboard controller will scan all columns, activating each one by one. When a column is activated, the controller detects which rows are "activated". To step through this procedure, the controller activates column C1 and checks rows R1 and R2:
Scanning Column 1:



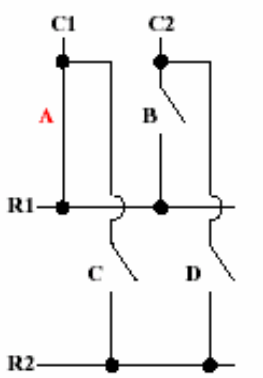
Neither key *A* nor *C* is pressed, so neither row R1 nor R2 is activated. The controller now knows that both nodes C1R1 and C1R2 are deactivated. Then it activates column C2 and checks rows R1 and R2 again:
Scanning Column 2:



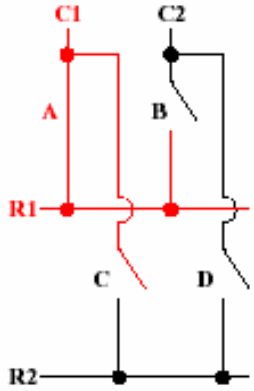
Single Key Press:
Assuming *A* key is pressed. The matrix will look like this:



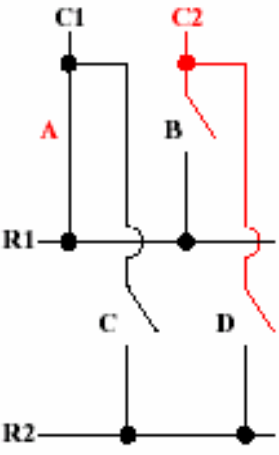
Key *A* corresponds to node C1R1. This is the circuit with switch A closed:



Walking through the scanning procedure again, the controller activates column C1 and detects which rows are activated:



This time, row R1 is activated. So the controller now knows that node C1R1 is pressed. Since C1R1 corresponds to the *A* key, the controller knows that the *A* key is pressed.
When the controller activates column C2, neither row R1 nor R2 is activated. Both switches B and D are open:

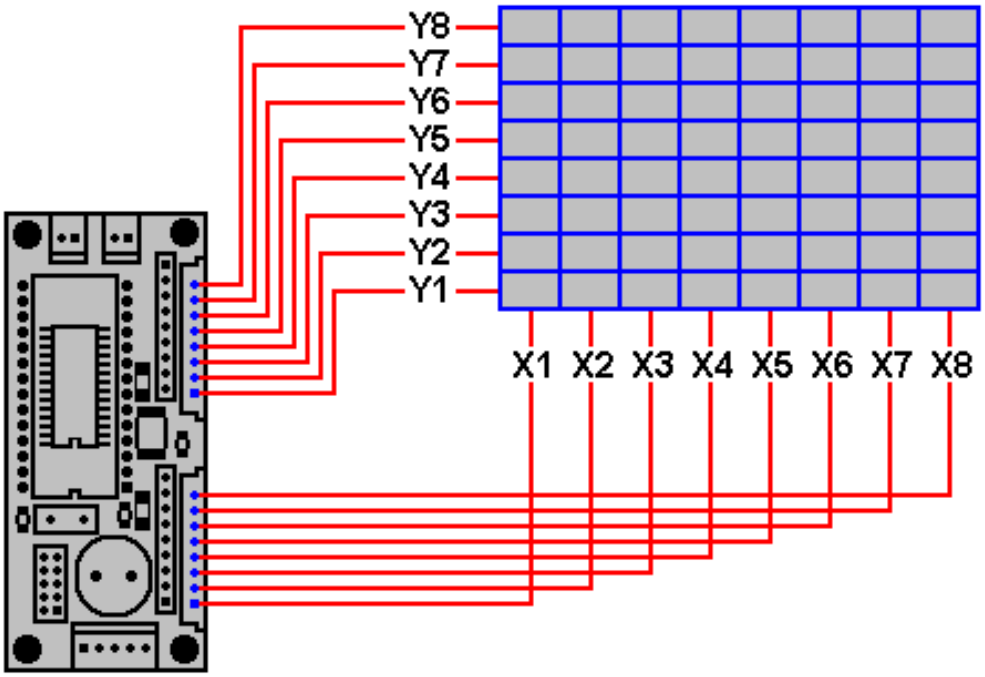


When the *A* key is released, the circuit goes back to the original, and the controller detects the node C1R1 is no longer activated.

When Multiple Keys are pressed, we can set the priority in the software. The priority can be set the order of the scanning.

The keyboard layout of the 8x8 Matrix Keypad is as follows:

The Keypad connectivity is as follows:



References

1. "E-Governance and Developing Countries, Introduction and examples" by Michiel Backus April 2001

2. "ELECTRONIC GOVERNANCE: *Abridged Definitive Conceptual Framework*" by Rogers W'O Okot-Uma, Commonwealth Secretariat London

3. e-Seva Centers: An initiative by Government of Andhra Pradesh for citizen services

4. Bhoomi Suggi Project: An initiative by Government of Karnataka for computerizing land record data

5. "Benchmarking E-Government: A Global Perspective. Assessing the Progress of UN Member States" combined report from "UN Division for Public Economics and Public Administration" and "American Society for Public Administration"

6. National e-Governance Action Plan (2003-07) (http://www.mit.gov.in/actionplan/about.asp)

7. Simputer (http://www.simputer.org)

8. "Computers for the Third Word, The Simputer is a Handheld Device Designed for Rural Villagers" by Scientific American, October, 2002 by Fiona Harvey

9. Palm Zire (http://www.palmone.com/us/products/compare/)

10. Pocket PC (http://www.pocketpcmag.com/default.asp)

11. Windows Mobile (http://msdn.microsoft.com/mobility/)

12. Analogic IHHT (http://www.analogicgroup.com/Products/Handheld.pdf)

13. Analogic PDA Programming (http://www.analogicgroup.com/Products/Prog_Manual.pdf)

14. Casio PV S-600 (http://www.pdasupport.com/PocketViewerS600Plus.htm)

15. Palm OS Development
(http://www.palmos.com/dev/start/intro.html)

16. Interfacing the Serial / RS232 Port
(http://www.beyondlogic.org/serial/serial.htm)

17. Basic Working of Serial Port
(http://computer.howstuffworks.com/serial-port1.htm)

18. Connector for Casio PV
(http://www.geocities.com/andretuly/CasioPV/converter.htm)

19. Communication Protocol for Casio Digital Diary v 3.8L
(http://pocket-viewer.com/Com-Protocol%20PV00-38L.zip)

20. AT Command Set
(http://www.lasat-usa.com/kap-3.htm)

21. SLIP Protocol
(http://www.faqs.org/rfcs/rfc1055.html)

22. "India Inc. takes to low-cost computing" – Article Express Computer.
(http://www.expresscomputeronline.com/20040112/focus01.shtml)

23. "The Virtual Ballot Box: A Survey of Digital Democracy in Europe" by Janet Caldow, Director, Institute for Electronic Government, IBM Corporation

24. "Information Age Government: Success Stories of Online Land Records & Revenue Governance from India" by Dr K M Baharul Islam, Chairman, State Rural Technology Promotion Council, Government of Assam, India

25. "Report of the Committee on India Vision 2020" by Dr.S.P.Gupta, Chairman, Planning Commission, Government of India

26. "THE E-GOVERNMENT HANDBOOK FOR DEVELOPING COUNTRIES" A Project of InfoDev and The Center for Democracy & Technology, The World Bank Group

27. Fonts for Indian languages
(http://acharya.iitm.ac.in/ind_fonts.html)

28. An Introduction to TrueType Fonts: A look inside the TTF format
(http://scripts.sil.org/cms/scripts/page.php?site_id=nrsi&item_id=IWS-Chapter08)

29. FreeType Glyph Conventions
(http://www.freetype.org/freetype2/docs/glyphs/index.html)

30. Inscript Keyboard
(http://tdil.mit.gov.in/keyoverlay.htm)

31. Telugu Script
(http://www.languagetechnologies.ac.in/lerc/telscr/telscr.htm)

32. Pango Documents
(http://www.pango.org/resources.shtml)

33. Philips MC 89C51RD+
(http://www.zlgmcu.com/philips/80c51/shouce/p89c5x/ePDF/89C51_89C52_89C54_89C58_en_1.pdf)

34. Matrix Keyboard
(http://www.dribin.org/dave/keyboard/keyboard.pdf)

35. 8051 Serial Communication
(http://www.8052.com/tutser.phtml)

36. BESCOM: Bangalore Electric Supply Company

37. Communication Protocol for Casio Digital Diary v 3.8L
(http://pocket-viewer.com/Com-Protocol%20PV00-38L.zip)