CERTIFICATE

This is to certify that the seminar report titled ''EMBEDDED SYSTEMS'' submitted by RAJAT MOHAN AGARWAL and PAWAN KATHURIA is a bonafide work done by her under our supervision is excellent.

INTERNAL EXAMINER:

Mr. MOHAN SINGH

SUBMITTED BY:

RAJAT MOHAN AGARWAL (0823131058)

PAWAN KATHURIA (0823131050)

ACKNOWLEDGEMENT

Interdependence is a higher value than independence. The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of people, who made it possible, whose constant guidance and encouragement crowned our efforts of success.

First and foremost, we thank Almighty for all His blessings He showered on us. I express my deep sense of gratitude and sincere thanks to **Mr.Mohan singh** without whose contribution I would be not able to complete this report on "**EMBEDDED SYSTEMS**".

I would like to express my heartfelt gratitude to all Teachers and Staffs of R.D. Engineering College for their kind co-operation . I also extend my thanks to all my friends for their moral support and encouragement. Last but not the least we thank our parents and benefactors who inspired us always to do my best.

RAJAT MOHAN AGARWAL

PAWAN KATHURIA

ABSTRACT

Many embedded systems have substantially different design constraints than desktop computing applications. No single characterization applies to the diverse spectrum of embedded systems. However, some combination of cost pressure, long life-cycle, real-time requirements, reliability requirements, and design culture dysfunction can make it difficult to be successful applying traditional computer design methodologies and tools to embedded applications. Embedded systems in many cases must be optimized for life-cycle and business-driven factors rather than for maximum computing throughput. There is currently little tool support for expanding embedded computer design to the scope of holistic embedded system design. However, knowing the strengths and weaknesses of current approaches can set expectations appropriately, identify risk areas to tool adopters, and suggest ways in which tool builders can meet industrial needs.

CONTENTS

INTRODUCTION

If we look around us, today we see numerous appliances which we use daily, be it our refrigerator , the microwave oven, cars, PDAs etc. Most appliances today are powered by something beneath the sheath that makes them do what they do. These are tiny microprocessors, which respond to various keystrokes or inputs. These tiny microprocessors, working on basic assembly languages, are the heart of the appliances. We call them embedded systems. Of all the semiconductor industries, the embedded systems market place is the most conservative, and engineering decisions here usually lean towards established, low risk solutions. Welcome to the world of embedded systems, of computers that will not look like computers and won't function like any thing we are familiar with.

As the name signifies, an 'embedded system' is built into a noncomputing device, say a car, TV or toy. We can define an embedded system as "a computing device, built in to a device that is not a computer, and meant for doing specific computing tasks". In general engineering terms, embedded systems are used for the control of industrial or physical processes. In computer science terms, embedded systems are distributed reactive systems. Typically these systems have to react to stimuli from their environment in real time. This can be of high importance in situations where a lot of signal

processing must be carried out on the inputs inorder to compute the outputs. e.g., multimedia applications.

Embedded systems have been around us for about as long as computer themselves. These were first used in the late 1960s to control electro mechanical telephone switches. As computer industry has moved towards smaller systems over the last decade or so, embedded systems have also moved along with this trend.

An embedded system is a computer system designed to do one or a few dedicated and/or specific functions often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts. By contrast, a general-purpose computer, such as a personal computer (PC), is designed to be flexible and to meet a wide range of end-user needs. Embedded systems control many devices in common use today.

Embedded systems are controlled by one or more main processing cores that are typically either microcontrollers or digital signal processors (DSP). For example, air traffic control systems may usefully be viewed as embedded, even though they involve mainframe computers and dedicated regional and national networks between airports and radar sites (each radar probably includes one or more embedded systems of its own).

Since the embedded system is dedicated to specific tasks, design engineers can optimize it to reduce the size and cost of the product and increase the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.

Physically, embedded systems range from portable devices such as digital watches and MP3 players, to large stationary installations like traffic lights, factory controllers, or the systems controlling nuclear power plants. Complexity varies from low, with a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure.

In general, "embedded system" is not a strictly definable term, as most systems have some element of extensibility or programmability. For example, handheld computers share some elements with embedded systems such as the operating systems and microprocessors which power them, but they allow different applications to be loaded and peripherals to be connected. Moreover, even systems which do not expose programmability as a primary feature generally need to support software updates. On a continuum from "general purpose" to "embedded", large application systems will have subcomponents at most points even if the system as a whole is "designed to perform one or a few dedicated functions", and is thus appropriate to call "embedded".

Variety of embedded systems

PC Engines' ALIX.1C Mini-ITX embedded board with an x86 AMD Geode LX 800 together with Compact Flash, miniPCI and PCI slots, 44-pin IDE interface, audio, USB and 256MB RAM

An embedded RouterBoard 112 with U.FL-RSMA pigtail and R52 miniPCI Wi-Fi card widely used by wireless Internet service providers (WISPs) in the Czech Republic.

Embedded systems span all aspects of modern life and there are many examples of their use. Telecommunications systems employ numerous embedded systems from telephone switches for the network to mobile phones at the end-user. Computer networking uses dedicated routers and network bridges to route data.

Consumer electronics include personal digital assistants (PDAs), mp3 players, mobile phones, videogame consoles, digital cameras, DVD players, GPS receivers, and printers. Many household appliances, such as microwave ovens, washing machines and dishwashers, are including embedded systems to provide flexibility, efficiency and features. Advanced HVAC systems use networked thermostats to more accurately and efficiently control

temperature that can change by time of day and season. Home automation uses wired- and wireless-networking that can be used to control lights, climate, security, audio/visual, surveillance, etc., all of which use embedded devices for sensing and controlling.

Transportation systems from flight to automobiles increasingly use embedded systems. New airplanes contain advanced avionics such as inertial guidance systems and GPS receivers that also have considerable safety requirements. Various electric motors — brushless DC motors, induction motors and DC motors — are using electric/electronic motor controllers. Automobiles, electric vehicles, and hybrid vehicles are increasingly using embedded systems to maximize efficiency and reduce pollution. Other automotive safety systems include anti-lock braking system (ABS), Electronic Stability Control (ESC/ESP), traction control (TCS) and automatic four-wheel drive.

Medical equipment is continuing to advance with more embedded systems for vital signs monitoring, electronic stethoscopes for amplifying sounds, and various medical imaging (PET, SPECT, CT, MRI) for non-invasive internal inspections.

Embedded systems are especially suited for use in transportation, fire safety, safety and security, medical applications and life critical systems as these systems can be isolated from hacking and thus be more reliable. For fire safety, the systems can be designed to be have

greater ability to handle higher temperatures and continue to operate. In dealing with security, the embedded systems can be self sufficient and be able to deal with cut electrical and communication systems.

In addition to commonly described embedded systems based on small computers, a new class of miniature wireless devices called motes are quickly gaining popularity as the field of wireless sensor networking rises. Wireless sensor networking, WSN, makes use of miniaturization made possible by advanced IC design to couple full wireless subsystems to sophisticated sensors, enabling people and companies to measure a myriad of things in the physical world and act on this information through IT monitoring and control systems. These motes are completely self contained, and will typically run off a battery source for many years before the batteries need to be changed or charged.

Classification:

Embedded systems are divided into autonomous, realtime, networked & mobile categories.

Autonomous systems:

They function in standalone mode. Many embedded systems used for process control in manufacturing units& automobiles fall under this category.

Realtime embedded systems:

These are required to carry out specific tasks in a specified amount of time. These systems are extensively used to carryout time critical tasks in process control.

Networked embedded systems:

They monitor plant parameters such as temperature, pressure and humidity and send the data over the network to a centralized system for on line monitoring.

Mobile gadgets:

Mobile gadgets need to store databases locally in their memory. These gadgets imbibe powerful computing & communication capabilities to perform realtime as well as nonrealtime tasks and handle multimedia applications.

The embedded system is a combination of computer hardware, software, firmware and perhaps additional mechanical parts, designed to perform a specific function. A good example is an automatic washing machine or a microwave oven. Such a system is in direct contrast to a personal computer, which is not designed to do only a specific task. But an

embedded system is designed to do a specific task with in a given timeframe, repeatedly, endlessly, with or without human interaction.

Hardware:

Good software design in embedded systems stems from a good understanding of the hardware behind it. All embedded systems need a microprocessor, and the kinds of microprocessors used in them are quite varied. A list of some of the common microprocessors families are: The Zilog Z8 family, Intel 8051/X86 family, Motorola 68K family and the power PC family. For processing of information and execution of programs, embedded system incorporates microprocessor or micro- controller. In an embedded system the microprocessor is a part of final product and is not available for reprogramming to the end user. An embedded system also needs memory for two purposes, to store its program and to

store its data. Unlike normal desktops in which data and programs are stored at the same place, embedded systems store data and programs in different memories. This is simply because the embedded system does not have a hard drive and the program must be stored in memory even when the power is turned off. This type of memory is called ROM. Embedded applications commonly employ a special type of ROM that can be programmed or reprogrammed with the help of special devices.

UMS Hardware:

A common obstacle for developers has been the need to develop different sets of hardware and software for different devices. An intelligent washing machine uses a hardware chip different from that used by an intelligent wrist watch. In addition, the software running on the hardware chip is different. This often results in increased costs and time taken for development.

The Universal Micro System(UMS) from cradle technologies is a solution for this problem. UMS is a general purpose chip built around a simple instruction set. It can be used to develop applications for embedded devices because all the functionality required for a specific device can be modeled in the software. Since the major functionality provided in

UMS is through software, the processor and memory units must be very fast and the input, output units must be programmable and versatile. UMS uses a large number of high speed, low power and small RISC based processors on a single chip. Each processing element coupled with two digital signal processors form a multi stream processor(MSP), which processes voluminous chunks of data. Developing on UMS represents a significant infrastructure cost savings, dramatic decrease in time to market and an unprecedented opportunity to combine many functions and redefine products in the market.

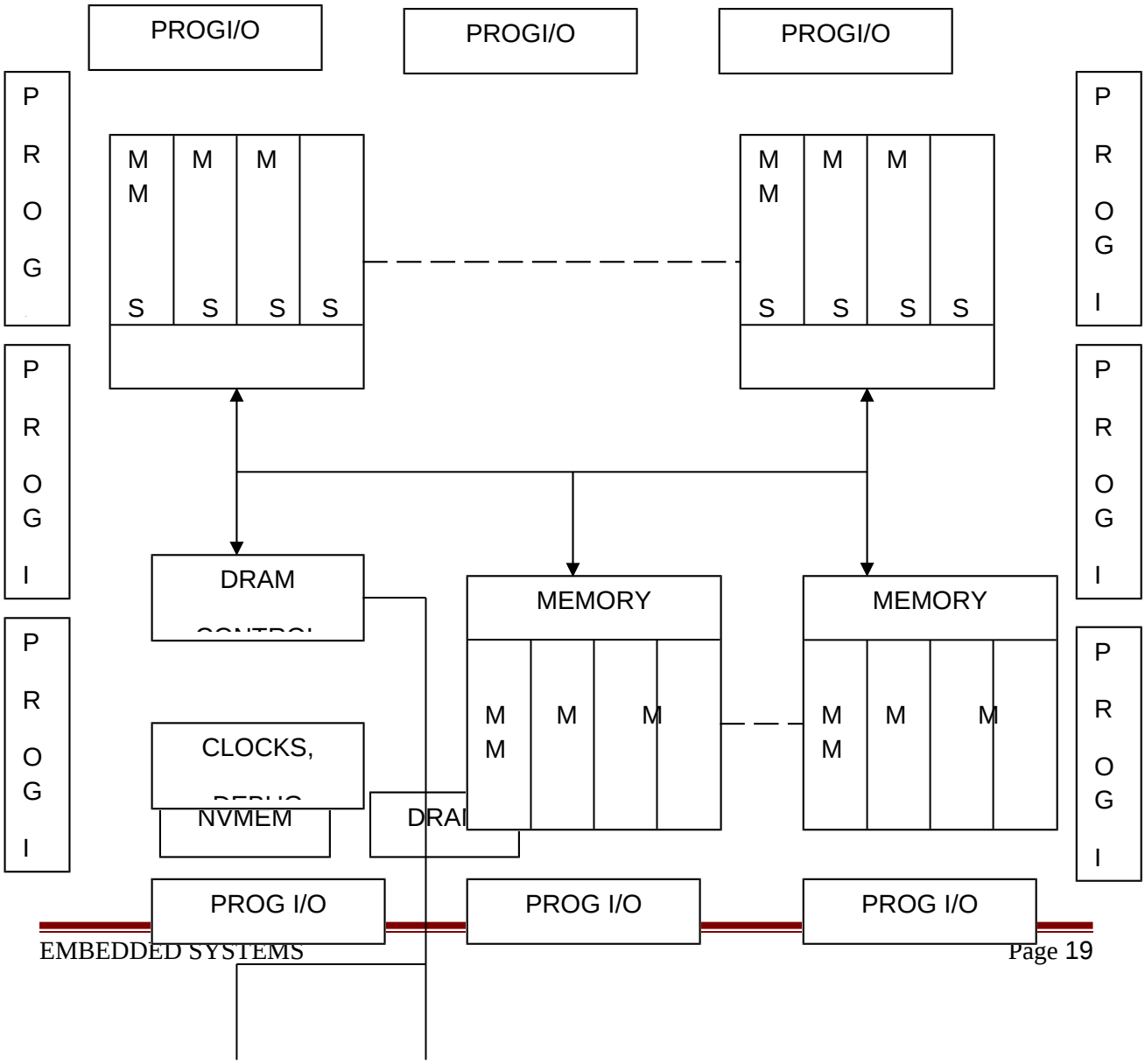
WATCHDOG TIMERS:

watchdog timer that resets the computer unless the software periodically notifies the watchdog subsystems with redundant spares that can be switched over to software "limp modes" that provide partial function Designing with a Trusted Computing Base (TCB) architecture[6] ensures a highly secure & reliable system environment.

An Embedded Hypervisor is able to provide secure encapsulation for any subsystem component, so that a compromised software component cannot interfere with other subsystems, or privileged-level system software. This encapsulation keeps faults from

propagating from one subsystem to another, improving reliability. This may also allow a subsystem to be automatically shut down and restarted on fault detection.

UMS hardware architecture:



Software:

Embedded software has grown complex and pervasive enough to attract the attention of computer scientists. Embedded software's main task is to engage the physical world interacting directly with sensors & actuators. The most pressing problem is how to adapt existing software techniques to meet the challenges of the physical world. Software for embedded systems must handle problems beyond those found in application software for desktops or mainframe computers. Embedded software often has several things to do at once - respond to external events, cope with unusual conditions without human intervention while being subjected to deadlines. So embedded software is harder to design. Embedded systems are increasingly networked which introduced significant complications such as downloadable modules that dynamically reconfigure the system. Moreover, consumers demand ever more elaborate functionality which greatly increases software complexity. These systems can no longer be designed by a single engineer, fine tuning tens of kilobytes of assembly code. Embedded software design is art as much as it is science. We must know how fast our system

operates and know how critical it is to meet each deadline. If deadlines are absolute, then it is a hard real time system else it is a soft real time system.

Functionality has steadily shifted from hardware to software. C has become the language of choice for embedded programmers, because it has the benefit of processor independence. Languages such as C++, Java are also used. Home devices will flourish best if they present a uniform API to application programmers, and an ideal interface is Java. Open Service Gateway Initiative(OSGI) , a java based system that combines services with event on security mechanisms, is defining a set of open standard software application interfaces for building Open Service Gateways including Residential Gateways . Embedded Graphical User Interface's (GUI) are growing more elaborate day by day. Developers now have to contend with such arcana as a color pallet. In some applications, an embedded GUI has to compete with mechanical controls. In luxury cars, for e.g, Graphical displays are starting to replace mechanical speedo meters and tacho meters.Visualization tools are designed to find bugs that are hard to find with theusual " Breakpoint and explore " debugging paradigm. They work beautifully on " Heisenberg bugs " – so called because they disappear when we start looking for them.

Any source code written in C or C++ or assembly language must be converted into an executable image that can be loaded onto a ROM chip. For this purpose three distinct steps are involved.

Cross compiled or assembled to generate object files.

Object files must be linked into a relocatable program.

Physical memory address must be assigned to the relocatable program.

RTOS:

To run any software we need operating system. Embedded systems do not require a complete operating system, which may make the system bulky, but only the basic functionalities of the operating system in a real time environment – RTOS. Off-the-shelf operating systems for these systems began to appear in the late 1970's, and today several dozen viable options are available.Embedded operating systems are available in variety of flavours: Windows NT, LINUX, Windows CE 3.0, PalmOS, QNX, ROMDOS, JBED, RT kernel, Tiny BIOS, Turbo task, Nucleus plus/Tasking, Diamond,ThreadX (#$@%) etc. Out of these , a few major players have emerged , such as Vxworks, PSOS, neculeus, windows CE,

ThreadX and linux. ‘Inferno’ and ‘Chai’ are the two popular environments that are used to develop applications.

Embedded Databases:

Embedded Databases are in software applications and in hardware devices, both mobile and fixed. The purpose of Embedded Database is data storage and retrieval with minimum intervention and minimum system impact. The rapid increase in the number of Telecomputers, the explosive growth of E-commerce and general migration to wireless technologies have put Embedded Database development on the IT short list. In a world of mobile computers and smart devices , size matters because memory and storage are very limited. A key factor is a small memory foot print. Embedded database vendors and developers tend to focus on smallness of their achievements. Sybase’s Ultra Lite, a version of SQL which is any where portable database has a foot print of 50 kilobytes.

‘e2B’:

The internet gives machines a way to communicate. For embedded internet, communication channels are seen in technologies such as blue tooth, CEbus, upnp and TCP/IP; linguistic mechanisms in notions such as java RMI; HTTP & SNMP. Embedding internet technologies is also called as 'e2B' or embedded to business.

Firmware:

Many authors use software and firmware in the same sense. Actually firmware consists of microcode programs executed from very high speed control storage. Commonly used object programs placed in ROMs and PROMs are also some times referred to as firmware. The problem with any approach to the field firmware updates is that if the upgrade contains a flaw, the target system may become an expensive doorstop. Many of the pitfalls are obvious and straight forward, but some insidious defects don't appear until after a product has been deployed. Any well designed firmware upgrade system must be able to recover from user errors and other catastrophic events to the fullest extent possible. The best way to accomplish this is to implement a fundamentally sound firmware update strategy that avoids these problems entirely. A microprogrammer is a system level description of how an embedded system beheves before, during and after the firmware update process. This is

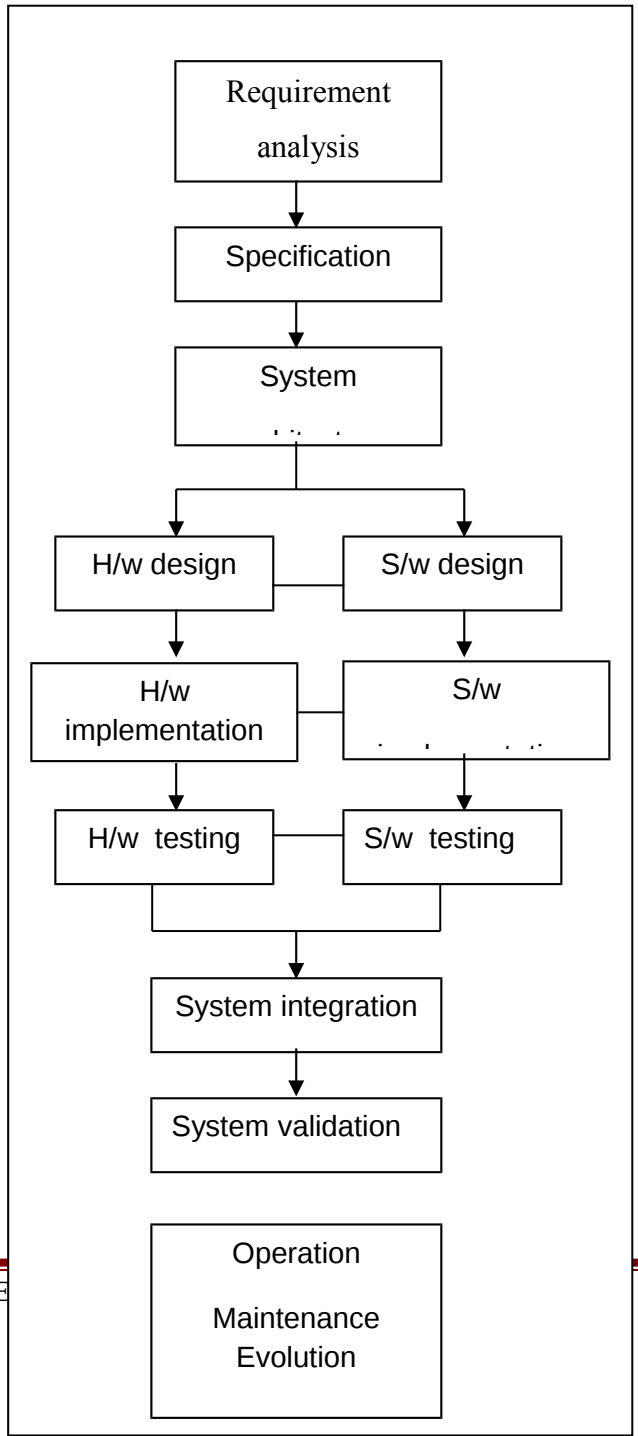
designed to help avoid many of the problems to downloadable firmware. 'Flexware' is a flexible firmware development environment.

Design process:

Embedded system design is a quantitative job. The pillars of the system design methodology are the separation between function and architecture, is an essential step from conception to implementation. In recent past, the search and industrial community has paid significant attention to the topic of hardware-software (HW/SW) codesign and has tackled the problem of coordinating the design of the parts to be implemented as software and the parts to be implemented as hardware avoiding the HW/SW integration problem marred the electronics system industry so long. In any large scale embedded systems design methodology, concurrency must be considered as a first class citizen at all levels of abstraction and in both hardware and software. Formal models & transformations in system design are used so that verification and synthesis can be applied to advantage in the design methodology. Simulation tools are used for exploring the design space for validating the functional and timing behaviors of embedded systems.

Hardware can be simulated at different levels such as electrical circuits, logic gates, RTL e.t.c. using VHDL description. In some environments software development tools can be coupled with hardware simulators, while in others the software is executed on the simulated hardware. The later approach is feasible only for small parts of embedded systems.

DEVELPOMENT CYCLE



Design of an embedded system using Intel's 80C188EB chip is shown in the figure. Inorder to reduce complexity, the design process is divided in four major steps: specification, system synthesis, implementation synthesis and performance evaluation of the prototype.

Specification:

During this part of the design process, the informal requirements of the analysis are transformed to formal specification using SDL.

System synthesis:

For performing an automatic HW/SW partitioning, the system synthesis step translates the SDL specification to an internal system model switch contains problem graph& architecture graph. After system synthesis, the resulting system model is translated back to SDL.

Implementation synthesis:

SDL specification is then translated into conventional implementation languages such as VHDL for hardware modules and C for software parts of the system.

Prototyping:

On a prototyping platform, the implementation of the system under development is executed with the software parts running on multiprocessor unit and the hardware part running on a FPGA board known as phoenix, prototype hardware for Embedded Network Interconnect Accelerators.

Applications:

Embedded systems are finding their way into robotic toys and electronic pets, intelligent cars and remote controllable home appliances. All the major toy makers across the world have been coming out with advanced interactive toys that can become our friends for life. 'Furby' and 'AIBO' are good examples at this kind. Furbies have a distinct life cycle just like human beings, starting from being a baby and growing to an adult one. In AIBO first two letters stands for Artificial Intelligence. Next two letters represents robot . The AIBO is robotic dog. Embedded systems in cars also known as Telematic Systems are used to provide navigational security communication & entertinment services using GPS, satellite. Home appliances are

going the embedded way. LG electronics digital DIOS refrigerator can be used for surfing the net, checking e-mail, making video phone calls and watching TV.IBM is developing an air conditioner that we can control over the net. Embedded systems cover such a broad range of products that generalization is difficult. Here are some broad categories.

CATEGORIES

Aerospace and defence electronics:

fire control, radar, robotics/sensors, sonar.

Automotive:

autobody electronics, auto power train, auto safety, car information systems.

Broadcast & entertainment:

Analog and digital sound products, camaras, DVDs, Set top boxes, virtual reality systems, graphic products.

Consumer/internet appliances:

Business handheld computers, business network computers/terminals, electronic books, internet smart handheld devices, PDAs.

Data communications:

Analog modems, ATM switches, cable modems, XDSL modems, Ethernet switches, concentrators.

Digital imaging:

Copiers, digital still cameras, Fax machines, printers, scanners.

Industrial measurement and control:

Hydro electric utility research & management traffic management systems, train marine vessel management systems.

Medical electronics:

Diagnostic devices, real time medical imaging systems, surgical devices, critical care systems.

Server I/O:

Embedded servers, enterprise PC servers, PCI LAN/NIC controllers, RAID devices, SCSI devices.

Telecommunications :

ATM communication products, base stations, networking switches, SONET/SDH cross connect, multiplexer.

Mobile data infrastructures:

Mobile data terminals, pagers, VSATs, Wireless LANs, Wireless phones.

History

One of the first recognizably modern embedded systems was the Apollo Guidance Computer, developed by Charles Stark Draper at the MIT Instrumentation Laboratory. At the project's inception, the Apollo guidance computer was considered the riskiest item in the Apollo project as it employed the then newly developed monolithic integrated circuits to reduce the size and weight. An early mass-produced embedded system was the Autonetics D-17 guidance computer for the Minuteman missile, released in 1961. It was built from transistor logic and had a hard disk for main memory. When the Minuteman II went into production in 1966, the D-17 was replaced with a new computer that was the first high-volume use of integrated circuits. This program alone reduced prices on quad nand gate ICs from $1000/each to $3/each, permitting their use in commercial products.

Since these early applications in the 1960s, embedded systems have come down in price and there has been a dramatic rise in processing power and functionality. The first microprocessor

for example, the Intel 4004, was designed for calculators and other small systems but still required many external memory and support chips. In 1978 National Engineering Manufacturers Association released a "standard" for programmable microcontrollers, including almost any computer-based controllers, such as single board computers, numerical, and event-based controllers.

As the cost of microprocessors and microcontrollers fell it became feasible to replace expensive knob-based analog components such as potentiometers and variable capacitors with up/down buttons or knobs read out by a microprocessor even in some consumer products. By the mid-1980s, most of the common previously external system components had been integrated into the same chip as the processor and this modern form of the microcontroller allowed an even more widespread use, which by the end of the decade were the norm rather than the exception for almost all electronics devices.

The integration of microcontrollers has further increased the applications for which embedded systems are used into areas where traditionally a computer would not have been considered. A general purpose and comparatively low-cost microcontroller may often be programmed to fulfill the same role as a large number of separate components. Although in this context an embedded system is usually more complex than a traditional solution, most of

the complexity is contained within the microcontroller itself. Very few additional components may be needed and most of the design effort is in the software. The intangible nature of software makes it much easier to prototype and test new revisions compared with the design and construction of a new circuit not using an embedded processor.

CHARACTERISTICS

Gumstix Overo COM, a tiny, OMAP-based embedded computer-on-module with Wifi and Bluetooth.

1. Embedded systems are designed to do some specific task, rather than be a general-purpose computer for multiple tasks. Some also have real-time performance constraints that must be met, for reasons such as safety and usability; others may have low or no performance requirements, allowing the system hardware to be simplified to reduce costs.

2. Embedded systems are not always standalone devices. Many embedded systems consist of small, computerized parts within a larger device that serves a more general purpose. For example, the Gibson Robot Guitar features an embedded system for tuning the strings, but the overall purpose of the Robot Guitar is, of course, to play music.[5] Similarly, an embedded system in an automobile provides a specific function as a subsystem of the car itself.

e-con Systems eSOM270 & eSOM300 Computer on Modules

3. The program instructions written for embedded systems are referred to as firmware, and are stored in read-only memory or Flash memory chips. They run with limited computer hardware resources: little memory, small or non-existent keyboard and/or screen.

User interface

Embedded system text user interface using MicroVGA

Embedded systems range from no user interface at all — dedicated only to one task — to complex graphical user interfaces that resemble modern computer desktop operating systems. Simple embedded devices use buttons, LEDs, graphic or character LCDs (for example popular HD44780 LCD) with a simple menu system.

More sophisticated devices use graphical screen with touch sensing or screen-edge buttons provide flexibility while minimizing space used: the meaning of the buttons can change with the screen, and selection involves the natural behavior of pointing at what's desired. Handheld systems often have a screen with a "joystick button" for a pointing device.

Some systems provide user interface remotely with the help of a serial (e.g. RS-232, USB, I²C, etc.) or network (e.g. Ethernet) connection. In spite of the potentially necessary proprietary client software and/or specialist cables that are needed, this approach usually gives a lot of advantages: extends the capabilities of embedded system, avoids the cost of a display, simplifies BSP, allows to build rich user interface on the PC. A good example of this is the combination of an embedded web server running on an embedded device (such as an IP camera) or a network routers. The user interface is displayed in a web browser on a PC connected to the device, therefore needing no bespoke software to be installed.

Processors in embedded systems

Secondly, Embedded processors can be broken into two broad categories: ordinary microprocessors (µP) and microcontrollers (µC), which have many more peripherals on chip, reducing cost and size. Contrasting to the personal computer and server markets, a fairly large number of basic CPU architectures are used; there are Von Neumann as well as various degrees of Harvard architectures, RISC as well as non-RISC and VLIW; word lengths vary from 4-bit to 64-bits and beyond (mainly in DSP processors) although the most typical remain 8/16-bit. Most architectures come in a large number of different variants and shapes, many of which are also manufactured by several different companies.

A long but still not exhaustive list of common architectures are: 65816, 65C02, 68HC08, 68HC11, 68k, 78K0R/78K0, 8051, ARM, AVR, AVR32, Blackfin, C167, Coldfire, COP8, Cortus APS3, eZ8, eZ80, FR-V, H8, HT48, M16C, M32C, MIPS, MSP430, PIC, PowerPC, R8C, RL78, SHARC, SPARC, ST6, SuperH, TLCS-47, TLCS-870, TLCS-900, TriCore, V850, x86, XE8000, Z80, AsAP etc.

Ready made computer boards

PC/104 and PC/104+ are examples of standards for ready made computer boards intended for small, low-volume embedded and ruggedized systems, mostly x86-based. These are often physically small compared to a standard PC, although still quite large compared to most simple (8/16-bit) embedded systems. They often use MSDOS, Linux, NetBSD, or an embedded real-time operating system such as MicroC/OS-II, QNX or VxWorks. Sometimes these boards use non-x86 processors.

In certain applications, where small size or power efficiency are not primary concerns, the components used may be compatible with those used in general purpose x86 personal computers. Boards such as the VIA EPIA range help to bridge the gap by being PC-compatible but highly integrated, physically smaller or have other attributes making them attractive to embedded engineers. The advantage of this approach is that low-cost commodity components may be used along with the same software development tools used for general software development. Systems built in this way are still regarded as embedded since they are integrated into larger devices and fulfill a single role. Examples of devices that may adopt this approach are ATMs and arcade machines, which contain code specific to the application.

However, most ready-made embedded systems boards are not PC-centered and do not use the ISA or PCI busses. When a System-on-a-chip processor is involved, there may be little

benefit to having a standarized bus connecting discrete components, and the environment for both hardware and software tools may be very different.

One common design style uses a small system module, perhaps the size of a business card, holding high density BGA chips such as an ARM-based System-on-a-chip processor and peripherals, external flash memory for storage, and DRAM for runtime memory. The module vendor will usually provide boot software and make sure there is a selection of operating systems, usually including Linux and some real time choices. These modules can be manufactured in high volume, by organizations familiar with their specialized testing issues, and combined with much lower volume custom mainboards with application-specific external peripherals. Gumstix product lines are a Linux-centric example of this model.

ASIC and FPGA solutions

A common array of n configuration for very-high-volume embedded systems is the system on a chip (SoC) which contains a complete system consisting of multiple processors, multipliers, caches and interfaces on a single chip. SoCs can be implemented as an application-specific integrated circuit (ASIC) or using a field-programmable gate array (FPGA).

Peripherals

Embedded Systems talk with the outside world via peripherals, such as:

Serial Communication Interfaces (SCI): RS-232, RS-422, RS-485 etc.

Synchronous Serial Communication Interface:

I2C, SPI, SSC and ESSI (Enhanced Synchronous Serial Interface)

Universal Serial Bus (USB)

Multi Media Cards (SD Cards, Compact Flash etc.)

Networks:

Ethernet, LonWorks, etc.

Fieldbuses:

CAN-Bus, LIN-Bus, PROFIBUS, etc.

Timers:

PLL(s), Capture/Compare and Time Processing Units

Discrete IO:

aka General Purpose Input/Output (GPIO)

Analog to Digital/Digital to Analog (ADC/DAC)

Debugging:

JTAG, ISP, ICSP, BDM Port, BITP, and DP9 ports

Tools

As with other software, embedded system designers use compilers, assemblers, and debuggers to develop embedded system software. However, they may also use some more specific tools:

Utilities to add a checksum or CRC to a program, so the embedded system can check if the program is valid.

For systems using digital signal processing, developers may use a math workbench such as Scilab / Scicos, MATLAB / Simulink, EICASLAB, MathCad, Mathematica,or FlowStone DSP to simulate the mathematics. They might also use libraries for both the host and target which eliminates developing DSP routines as done in DSPnano RTOS and Unison Operating System.

Custom compilers and linkers may be used to improve optimisation for the particular hardware.

An embedded system may have its own special language or design tool, or add enhancements to an existing language such as Forth or Basic.

Another alternative is to add a real-time operating system or embedded operating system, which may have DSP capabilities like DSPnano RTOS.

Sometimes, development tools for a personal computer can be used if the embedded processor is a close relative to a common PC processor.

As the complexity of embedded systems grows, higher level tools and operating systems are migrating into machinery where it makes sense. For example, cellphones, personal digital assistants and other consumer computers often need significant software that is purchased or provided by a person other than the manufacturer of the electronics. In these systems, an open programming environment such as Linux, NetBSD, OSGi or Embedded Java is required so that the third-party software provider can sell to a large market.

DEBUGGING

Embedded debugging may be performed at different levels, depending on the facilities available. From simplest to most sophisticated they can be roughly grouped into the following areas:

Interactive resident debugging, using the simple shell provided by the embedded operating system (e.g. Forth and Basic)

External debugging using logging or serial port output to trace operation using either a monitor in flash or using a debug server like the Remedy Debugger which even works for heterogeneous multicore systems.

An in-circuit debugger (ICD), a hardware device that connects to the microprocessor via a JTAG or Nexus interface. This allows the operation of the microprocessor to be controlled externally, but is typically restricted to specific debugging capabilities in the processor.

An in-circuit emulator (ICE) replaces the microprocessor with a simulated equivalent, providing full control over all aspects of the microprocessor.

A complete emulator provides a simulation of all aspects of the hardware, allowing all of it to be controlled and modified, and allowing debugging on a normal PC.

Unless restricted to external debugging, the programmer can typically load and run software through the tools, view the code running in the processor, and start or stop its operation. The view of the code may be as HLL source-code, assembly code or mixture of both.

Because an embedded system is often composed of a wide variety of elements, the debugging strategy may vary. For instance, debugging a software- (and microprocessor-) centric embedded system is different from debugging an embedded system where most of the processing is performed by peripherals (DSP, FPGA, co-processor). An increasing number of

embedded systems today use more than one single processor core. A common problem with multi-core development is the proper synchronization of software execution. In such a case, the embedded system design may wish to check the data traffic on the busses between the processor cores, which requires very low-level debugging, at signal/bus level, with a logic analyzer, for instance.

Tracing Real-time operating systems (RTOS) often supports tracing of operating system events. A graphical view is presented by a host PC tool, based on a recording of the system behavior. The trace recording can be performed in software, by the RTOS, or by special tracing hardware. RTOS tracing allows developers to understand timing and performance issues of the software system and gives a good understanding of the high-level system behavior. A good example is RTXCview, for RTXC Quadros by Quadros Systems, Inc..

RELIABILITY

Embedded systems often reside in machines that are expected to run continuously for years without errors, and in some cases recover by themselves if an error occurs. Therefore the software is usually developed and tested more carefully than that for personal computers, and unreliable mechanical moving parts such as disk drives, switches or buttons are avoided.

Specific reliability issues may include:

The system cannot safely be shut down for repair, or it is too inaccessible to repair. Examples include space systems, undersea cables, navigational beacons, bore-hole systems, and automobiles.

The system must be kept running for safety reasons. "Limp modes" are less tolerable. Often backups are selected by an operator. Examples include aircraft navigation, reactor control systems, safety-critical chemical factory controls, train signals, engines on single-engine aircraft.

The system will lose large amounts of money when shut down: Telephone switches, factory controls, bridge and elevator controls, funds transfer and market making, automated sales and service.

A variety of techniques are used, sometimes in combination, to recover from errors—both software bugs such as memory leaks, and also soft errors in the

Immunity Aware Programming

High vs low volume

For high volume systems such as portable music players or mobile phones, minimizing cost is usually the primary design consideration. Engineers typically select hardware that is just "good enough" to implement the necessary functions.

For low-volume or prototype embedded systems, general purpose computers may be adapted by limiting the programs or by replacing the operating system with a real-time operating system.

Embedded software architectures

There are several different types of software architecture in common use.

Simple control loop

In this design, the software simply has a loop. The loop calls subroutines, each of which manages a part of the hardware or software.

Interrupt controlled system

Some embedded systems are predominantly interrupt controlled. This means that tasks performed by the system are triggered by different kinds of events. An interrupt could be generated for example by a timer in a predefined frequency, or by a serial port controller receiving a byte.

These kinds of systems are used if event handlers need low latency and the event handlers are short and simple.

Usually these kinds of systems run a simple task in a main loop also, but this task is not very sensitive to unexpected delays.

Sometimes the interrupt handler will add longer tasks to a queue structure. Later, after the interrupt handler has finished, these tasks are executed by the main loop. This method brings the system close to a multitasking kernel with discrete processes.

Cooperative multitasking

A nonpreemptive multitasking system is very similar to the simple control loop scheme, except that the loop is hidden in an API. The programmer defines a series of tasks, and each task gets its own environment to “run” in. When a task is idle, it calls an idle routine, usually called “pause”, “wait”, “yield”, “nop” (stands for no operation), etc.

The advantages and disadvantages are very similar to the control loop, except that adding new software is easier, by simply writing a new task, or adding to the queue-interpreter.

Preemptive multitasking or multi-threading

In this type of system, a low-level piece of code switches between tasks or threads based on a timer (connected to an interrupt). This is the level at which the system is generally considered to have an "operating system" kernel. Depending on how much functionality is required, it

introduces more or less of the complexities of managing multiple tasks running conceptually in parallel.

As any code can potentially damage the data of another task (except in larger systems using an MMU) programs must be carefully designed and tested, and access to shared data must be controlled by some synchronization strategy, such as message queues, semaphores or a non-blocking synchronization scheme.

Because of these complexities, it is common for organizations to use a real-time operating system (RTOS), allowing the application programmers to concentrate on device functionality rather than operating system services, at least for large systems; smaller systems often cannot afford the overhead associated with a generic real time system, due to limitations regarding memory size, performance, and/or battery life.The choice that a RTOS is required brings in its own issues however as the selection must be done prior to starting to the application development process. This timing forces developers to choose the embedded operating system for their device based upon current requirements and so restricts future options to a large extent. The restriction of future options becomes more of an issue as product life decreases. Additionally the level of complexity is continuously growing as devices are required to manage many variables such as serial, USB, TCP/IP, Bluetooth, Wireless LAN,

trunk radio, multiple channels, data and voice, enhanced graphics, multiple states, multiple threads, numerous wait states and so on. These trends are leading to the uptake of embedded middleware in addition to a real time operating system.

Microkernels and exokernels

A microkernel is a logical step up from a real-time OS. The usual arrangement is that the operating system kernel allocates memory and switches the CPU to different threads of execution. User mode processes implement major functions such as file systems, network interfaces, etc.

In general, microkernels succeed when the task switching and intertask communication is fast, and fail when they are slow.

Exokernels communicate efficiently by normal subroutine calls. The hardware, and all the software in the system are available to, and extensible by application programmers.

Monolithic kernels

In this case, a relatively large kernel with sophisticated capabilities is adapted to suit an embedded environment. This gives programmers an environment similar to a desktop

operating system like Linux or Microsoft Windows, and is therefore very productive for development; on the downside, it requires considerably more hardware resources, is often more expensive, and because of the complexity of these kernels can be less predictable and reliable.

Common examples of embedded monolithic kernels are Embedded Linux and Windows CE.

Despite the increased cost in hardware, this type of embedded system is increasing in popularity, especially on the more powerful embedded devices such as Wireless Routers and GPS Navigation Systems. Here are some of the reasons:

- Ports to common embedded chip sets are available.
- They permit re-use of publicly available code for Device Drivers, Web Servers, Firewalls, and other code.
- Development systems can start out with broad feature-sets, and then the distribution can be configured to exclude unneeded functionality, and save the expense of the memory that it would consume.

- Many engineers believe that running application code in user mode is more reliable, easier to debug and that therefore the development process is easier and the code more portable.
- Many embedded systems lack the tight real time requirements of a control system. Although a system such as Embedded Linux may be fast enough in order to respond to many other applications.

Features requiring faster response than can be guaranteed can often be placed in hardware.

Many RTOS systems have a per-unit cost. When used on a product that is or will become a commodity, that cost is significant.

Exotic custom operating systems

A small fraction of embedded systems require safe, timely, reliable or efficient behavior unobtainable with the one of the above architectures. In this case an organization builds a system to suit. In some cases, the system may be partitioned into a "mechanism controller" using special techniques, and a "display controller" with a conventional operating system. A communication system passes data between the two.

Additional software components

In addition to the core operating system, many embedded systems have additional upper-layer software components. These components consist of networking protocol stacks like CAN, TCP/IP, FTP, HTTP, and HTTPS, and also included storage capabilities like FAT and flash memory management systems. If the embedded devices has audio and video capabilities, then the appropriate drivers and codecs will be present in the system. In the case of the monolithic kernels, many of these software layers are included. In the RTOS category, the availability of the additional software components depends upon the commercial offering.

Embedded Development Environment

The embedded system may not have a keyboard, a screen, a disk drive and other peripheral devices required for programming and development tasks. Therefore most of the programming for embedded systems is done on a host, which is a computer system with all the programming tools. Only after the program has been written, compiled, assembled and linked is it to move to the target or the system that is shipped to the customers.

After writing source file compiling, linking, relocating and porting the executable image into the ROM, you need to test and debug the application. Once you have an executable image stored as a file on the host computer, you need a way to download that image into a memory device on the target board or development board and execute it from there. And if you have the right tools at your disposal, it will be possible to set breakpoints in the program or set break points in the program or observe its execution. These various tools could be a remote debugger, simulator, emulator or an in-circuit emulator.

A remote debugger can be used to download, execute, and debug embedded software over the serial port or network connection between the host and the target. In case of embedded systems, the debugger executes on two different computer systems – a remote debugger consists of two pieces of software. The front-end runs on the host computer and provides the human interface, and the hidden back-end runs on the target processor and communicates

with the front-end over a communication link. The back-end provides low-level control of the target processor and is usually called debug monitor.

The debug monitor resides in the ROM and is automatically started whenever the target processor is reset. It monitors the communication link to the host computer and responds to the request from the remote debugger running there. Remote debuggers are the most commonly used tools for downloading and testing tools during the development of embedded software – mainly because of there low cost.

Remote debuggers are helpful in monitoring and controlling the state of embedded software, but only in in-circuit emulators (ICEs) allow you to examine the state of the processor on which that program is running. In fact an ICE actually takes the place of the processor on your target board, or in other words, emulates the work of the processor and provides the human interface with what exactly is happening on the board in real-time. This also allows the ICE to support powerful debugging features such as hardware breakpoints and real-time tracing.

Many other debugging tools – such as simulators, logic analysers and oscilloscopes – are also available in embedded systems. A simulator is a completely host-based program that simulates the functionality and instruction set of the target processor. Although simulators

have many disadvantages, they are quite valuable in the early stages of the project when there isn't as yet any actual hardware for the programmers to experiment with. The biggest disadvantage of a simulator is that it simulates only the processor. And embedded systems frequently contain one or more other peripherals. Interaction with these devices can only sometimes be imitated. You may not do much with the simulator once you have the actual embedded hardware available to you.

Once the target hardware is available, you can use logic analysers and oscilloscopes as debugging tools. These are very useful for debugging the interactions between the processor and other chips on the board. These tools only view signals that lie outside the processor, and cannot control the flow of execution of your software like debuggers or emulators can.

A logic analyser is equipment that is designed to find whether the electrical signal it is attached to is currently to logic level 1 or 0. An oscilloscope so another piece of equipment for hardware debugging, and is used to examine any electrical signal, analogue signal, or digital signal on the hardware.

Harsh environment

Many embedded systems do not operate in a controlled environment. Excessive heat is often a problem, especially in applications involving combustion (e.g., many transportation applications). Additional problems can be caused for embedded computing by a need for protection from vibration, shock, lightning, power supply fluctuations, water, corrosion, fire, and general physical abuse. For example, in the Mission Critical example application the computer must function for a guaranteed, but brief, period of time even under non-survivable fire conditions.

Cost sensitivity

Even though embedded computers have stringent requirements, cost is almost always an issue (even increasingly for military systems). Although designers of systems large and small may talk about the importance of cost with equal urgency, their sensitivity to cost changes can vary dramatically. A reason for this may be that the effect of computer costs on profitability is more a function of the proportion of cost changes compared to the total system cost, rather than compared to the digital electronics cost alone

Information Appliance

In the past, embedded systems allowed information appliances to carry out simple and specific functions only. But with the penetration of the Internet into the homes of many ordinary families, it was realized that electric appliances could make human life easier and more convenient if they could access Internet information. Electric appliances can now access the Internet, compute and do what they were not able to do earlier. In other words, electric appliances are being transformed into information appliances (IA) or what may also be called 'embedded IA'.

Like the traditional embedded systems, the embedded information appliance needs only the least amount of hardware to operate. It can operate even without a hard disk, or with low power and small footprint.

IA product can be classified into four mainstream products:-

- Set-Top Boxes (STB)
- Personal Access Device (PAD)
- Thin Client (TC)
- Residential Gateway (or Home Gateway)

Most IA products may be derived, with little or some modifications, from these four types of products.

Set-Top Boxes

The set-top box is driving the digital revolution right into your living room. Your fingertips now command a wealth of high quality digital information and digital entertainment, right from your favorite armchair. The set-top box revolutionizes home entertainment by providing vibrant television images with crystal clear sound, along with e-mail, Web surfing, along with customized information such as stock quotes, weather and traffic updates, on-line shopping, and video-on-demand, right through a traditional television.

Personal Access Devices

Personal Access Devices (PADs) are web terminals that feature convenient Web browsing, email, and information access capabilities in a lightweight, mobile form.

Thin Client

A thin client is an information access drive that provides users with remote access to applications and data that are maintained and executed on a central server. The thin-client computing environment consists of an application server, a network, and thin-client devices. By centralizing deployment and updates of corporate applications, thin clients allow for simplified Information Systems (IS) management with dramatically increased security.

Residential Gateway

The RG mainly provides various kinds of interfaces that link all the electronic devices. The RG unlike the PC, is a very small, slim and light piece of hardware and may soon be incorporated inside other popular electronic appliances. It will play the role of an information hub responsible for the exchange of information between all kinds of electronic devices in an ordinary home.

CONCLUSION:

We are standing on the threshold of an exciting new age of information technology that will change our lives and the future forever. Soon we shall see more and more digitization of appliances, and these will be fuelled by human need. Embedded systems and Information Appliances have virtually entered every sphere of our life and they will truly change the way we live.

Embedded systems have virtually entered every sphere of our life, right from the time we work out on tread mills to the cars that we drive today. The possibilities in this field are only limited by our imagination.Many of the embedded systems are managed by human controllers by some sort of man machine interface-for example a cash register,a cell phone, a TV screen or a PC interface.It is this MMI that often represents the most costly investment in the system's development,interms of both time and money.

REFERENCES

[1] Embedded Systems Programming. Miller Freeman, San Francisco, ISSN 1040-3272.

[2] Daniel D. Gajski. Frank Vahid, Sanjiv Narayan & Jie Gong, Specification and Design of Embedded Systems. PTR Prentice Hall, Englewood Cliffs NJ, 1994.

[3J Jack Ganssle, Art of programming Embedded Systems, Academic Press, San Diego. 1992.

[4J Shem-Tov Levi & Ashok Agrawala, Fault Tolerant System Design, McGrawHill, New York, 1994.

[5] Nancy Leveson, Safeware: system safety and computers, Addison-Wesle,1994..

[6] WIKKIPEDIA

[7] WWW.SEMINARPROJECTS.COM

[8] WWW.SCRIBD.COM