Study And Implementation Of Real Time Operating Systems For 8-bit Microcontrollers

Akhil Guliani

Date

Acknowledgment

I thank the Global Internship Program in Engineering Design and Innovation of the Airtel IIT Delhi Centre of Excellence in Telecommunication (GIPEDI.AICET), Bharti School of Telecommunication Technology and Management, Indian Institute of Technology Delhi for giving me the opportunity to participate in the Program.

I thank Prof. Subrat Kar for being my Faculty Mentor and for sharing his experience and wisdom. I also thank Mr Akshat Bisht who, as Student Mentors, provided valuable and timely help.

Akhil Guliani
Date: 19 July, 2011

Certificate of Internship

This is to certify that Akhil Guliani of Netaji Subhas Institute of Technology has completed the work on 'Study And Implementation Of Real Time Operating Systems For 8-bit Microcontrollers' and has submitted this report in partial fulfillment of the requirements of the Global Internship Program in Engineering Design and Innovation under the aegis of the Foundation for Innovation and Technology Transfer, Indian Institute of Technology Delhi.

Prof. Subrat Kar
Program Coordinator,
GIPEDI Global Internship Program in Engineering Design and Innovation
under the aegis of Foundation of Innovation and Technology Transfer
Indian Institute of Technology Delhi Hauz Khas, New Delhi 110016, INDIA
gipedi@ee.iitd.ac.in
+91.11.2659 7164

Date:

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction to Real Time Systems

This summer I went to work at Bharti School of Telecommunication Technology and Management under the aegis of GIPEDI.AICET. The area of work chosen by me was Real Time systems.

In computer science, real-time computing (RTC), or reactive computing, is the study of hardware and software systems that are subject to a "real-time constraint"—i.e., operational deadlines from event to system response. Real-time programs must execute within strict constraints on response time.[3]

A real-time system may be one where its application can be considered to be of critical importance to the application. The anti-lock brakes on a car are a simple example of a real-time system here the real-time constraint in this system is the time in which the brakes must be released to prevent the wheel from locking. Real-time computations can be said to have failed if they are not completed before their deadline, where their deadline is relative to an event. A real-time deadline must be met, regardless of system load.

1.2 Problem statement and proposed methodology

The problem statement given to me was to find a suitable RTOS for AVR ATmega32 and to write a bit-banged I2C module for the same. The figure 1.2.1 gives the schematic diagram of the enire project.

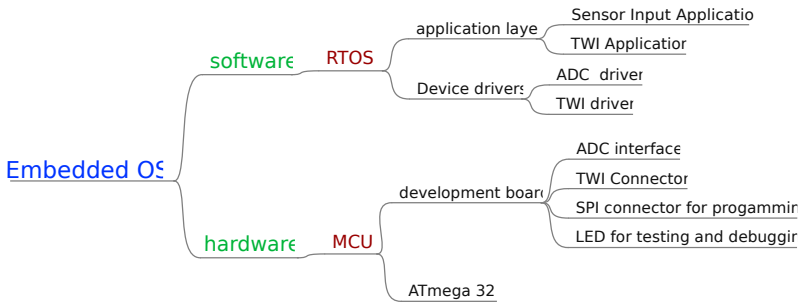

Figure 1.2.1: Schematic diagram of entire project

1.3 Organization of report

The report has been organized as follows:

1. Chapter 2 Gives an introduction to the AVR-core and ATmega32.
2. Chapter 3 Talks about Real Time Operating Systems(RTOS) and their applications. It also talks about the RTOSs used by me during the study.
3. Chapter 4 Talks about the I2C Protocol and its implementation as a Bit-Banged module.
4. Chapter 5 Talks about the methodology and results of the study.
5. Appendix A Gives a list of RTOSs for the AVR core.
6. Appendix B Gives the I2C Library developed during the study
7. Appendix C Gives a How to on writing applications for Femto OS
8. Appendix D Gives A sample schematic for Femto OS on an ATmega32

Chapter 2

Atmel AVR

2.1 Introduction

AVR or Atmel AVR are RISC based family of microcontroller produced by Atmel Corporations. The AVR is a modified Harvard architecture which was developed in 1996. The AVR was one of the first microcontroller families to use on-chip flash memory for program storage[**?**][1]

The original AVR MCU was developed at a local ASIC house in Trondheim Norway, where the two founders of Atmel Norway,Alf-Egil Bogen and Vegard Wollan were working as students.

2.2 Features

The main features of the AVR core are :

1. It has a RISC Architecture with 131 powerful instructions with most of them having single-clock cycle execution.
2. It has 32 x 8-bit General Purpose Working Registers
3. The AVR is a Harvard architecture machine with programs and data stored separately
4. Each instruction takes 16-bits with 8-bits for the instruction and 8-bits for the data
5. The core has a single level pipeline design, this means the next machine instruction is fetched while the current one is executing[**?**]

2.3 ATmega 32

Figure 2.3.1: AVR ATmega32[1]

2.3.1 Overview

The Atmel ® AVR® ATmega32 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega32 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.[4]

2.3.2 Features

The ATmega32 is a power packed microcontroller with the following features[4]:

1. High-performance, Low-power Atmel ® AVR® 8-bit Microcontroller
2. Has a High Endurance Non-volatile Memory segments i.e. 32Kbytes of In-System Self-programmable Flash program memory, 1024Bytes EEPROM and 2Kbyte Internal SRAM with an Optional Boot Code Section with Independent Lock Bits
3. Has In-System Programming by On-chip Boot Program True Read-While-Write Operation – Programming Lock for Software Security
4. Has a JTAG (IEEE std. 1149.1 Compliant) Interface
5. Peripheral Features
   (a) Two 8-bit Timer/Counters and one 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
   (b) Real Time Counter with Separate Oscillator and four PWM Channels

(c) 10-bit ADC 8 Single-ended Channels, 7 Differential Channels and 2 Differential Channels with Programmable Gain at 1x, 10x, or 200x

(d) Byte-oriented Two-wire Serial Interface, Programmable Serial USART and Master/Slave SPI Serial Interface

(e) Programmable Watchdog Timer with Separate On-chip Oscillator

(f) On-chip Analog Comparator

6. Special Microcontroller Features

(a) Power-on Reset and Programmable Brown-out Detection

(b) Internal Calibrated RC Oscillator – External and Internal Interrupt Sources

(c) Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby • I/O and Packages

(d) 32 Programmable I/O Lines

Chapter 3

Real Time Operating Systems

3.1 Introduction

A Real-Time Operating System (RTOS) is a computing environment that reacts to input within a specific time period. To achieve time reliability, real-time programs and their operating system environment must prioritize deadline actualization before anything else.[5, 3, 6, 2, 7, 8]

A key characteristic of a RTOS is the level of its consistency concerning the amount of time it takes to accept and complete an application's task; the variability is jitter. A hard real-time operating system has less jitter than a soft real-time operating system. The chief design goal is not high throughput, but rather a guarantee of a soft or hard performance category. A RTOS that can usually or generally meet a deadline is a soft real-time OS, but if it can meet a deadline deterministically it is a hard real-time OS.

A real-time OS has an advanced algorithm for scheduling. Scheduler flexibility enables a wider, computer-system orchestration of process priorities, but a real-time OS is more frequently dedicated to a narrow set of applications. Key factors in a real-time OS are minimal interrupt latency and minimal thread switching latency, but a real-time OS is valued more for how quickly or how predictably it can respond than for the amount of work it can perform in a given period of time.

There are two basic design philosophies for RTOS as follows:

1. Event-driven which switches tasks only when an event of higher priority needs service, called preemptive priority, or priority scheduling. Time-sharing designs switch tasks on a regular clock interrupt, and on events, called round robin.

2. Time-sharing designs switch tasks more often than strictly needed, but give smoother multitasking, giving the illusion that a process or user has sole use of a machine.

The RTOS also needs to provide a mechanism for reliable intertask communication.

Most of the RTOSs require a Board Support Package which makes them target specific.

3.2 FreeRTOS

3.2.1 Introduction

FreeRTOS includes official ports to 27 architectures and receives more than 77,500 downloads a year. It is market leading, robust, supported, portable, open source, free to download, and free to deploy. FreeRTOS can be used in commercial applications without any requirement to expose your proprietary source code. With a growing ecosystem, FreeRTOS is commonly integrated with both open source and commercial TCP/IP, file system, and USB components.

Each official port includes a pre-configured example application that demonstrates the kernel features, expedites learning, and enables 'out of the box' development. [7]

3.2.2 Features

FreeRTOS is a scale-able real time kernel designed specifically for small embedded systems. Highlights include[7]:

1. Free RTOS kernel is a preemptive, cooperative and hybrid configuration options.
2. The SafeRTOS derivative product provides a high level of confidence in the code integrity.
3. Official support for 27 architectures, including ARM, AVR, MIPS,etc. With free development tools for most.
4. Designed to be small, simple and easy to use. The typical kernel binary image will be of 4K to 9K bytes in size.
5. Very portable code structure predominantly written in C.
6. Supports both tasks and co-routines.
7. Queues, binary semaphores, counting semaphores, recursive semaphores and mutexes for communication and synchronisation between tasks, or between tasks and interrupts. The mutexes even support priority inheritance.

8. Supports efficient software timers.
9. Has a powerful execution trace functionality.
10. FreeRTOS has options for stack overflow detection.
11. Pre-configured demo applications for selected single board computers allowing 'out of the box' operation and fast learning curve.
12. Has free forum support, or optional commercial support and licensing.
13. There are no software restriction on the number of tasks that can be created and on the number of priorities that can be used.
14. There are no restrictions imposed on priority assignment; more than one task can be assigned the same priority.
15. Free embedded software source code. Royalty free. Cross development from a standard Windows host.

3.2.3 Limitations

The Major limitation of FreeRTOS are

1. The typical kernel binary image will be of 4K to 9K bytes in size[7]. Which is too large a footprint for AT-tiny devices and in some cases even for Mega devices
2. Due to the large footprint the complexity of application code becomes RAM limited

3.3 XMK

3.3.1 Introduction

XMK was designed from the ground up to be very small with respect to ROM, RAM and CPU processing resources. The original targeted platforms were 8bit microcontrollers with only 4K to 8K of ROM and 512 bytes of RAM. XMK has since been scaled up to 16bit and 32bit platforms. Originally it was only a preemptive scheduler with thread synchronization primitives. Now it includes such features as mailboxes, memory pools, file descriptors, hardware device drivers, and TCP/IP networking. Even as XMK has expanded its functionality and scope, it has maintained its extreme minimal philosophy and footprint. Its minimum size is still less than 340 bytes of ROM and 18 bytes of RAM[9].

3.3.2 Features

The main features of XMK are[9] :

Small. XMK was specifically designed to run on microcontrollers using on the only onboard ROM and RAM.

Configurable. The application only includes the kernel services that are needed.

Scalable. XMK will run on virtually any platform, 8bit, 16bit, and 32bit processors.

Portable. XMK is written in C with only a small amount of target specific assembler for speed and efficiency.

Free. The source can be used freely for commercial and proprietary applications (BSD licensing agreement).

3.3.3 Limitations

The major limitations of XMK are[9]:

1. It is does not have a Board Support Package (BSP). The burden of defining the vector table, C start-up code, booting the board, initializing/configuring RAM, etc. falls on the developer and/or the target's development system.

2. There is no Debug/Run-time environment. Targets must be loaded and debugged with the tools supplied by their respective development system.

3. It is not a C/C++ Run-time Library replacement. If the C/C++ Run time libraries are needed, they must be provided by the target's development system.

3.4 Femto OS

3.4.1 Introduction

The Femto OS is a very concise portable real time - preemptive operating system (RTOS) for embedded microcontrollers with minimal RAM and flash, say 2KB .. 16KB flash and 128 .. 1024 bytes RAM. The main target is the Atmel AVR architecture, such as the ATtiny or smaller ATmega series. The OS runs well on larger hardware also. The system is written in C with a separate port file. Porting has been done for 44 AVR devices.

The typical footprint is between 1K and 4K flash, depending on the functions used. Somewhere around 2K for the OS itself is realistic figure for normal applications. The OS takes between 10 and 20 bytes of RAM, tasks can take as little as 6 bytes of RAM, but approximately 20 to 40 bytes is more realistic for real applications. Figures are including the stack. It is perfectly possible to run 4 or more tasks on an ATtiny261. There is no separate idle task and most api function calls run in OS space. The OS is interruptible most of the time, if needed. There are tools to watch and protect the use of the stack of the tasks and the OS.

The code itself is heavily documented, and comes with a number of example applications, (which are minimally documented, but are intended to speak for themselves). Demo's run on all devices for which a port is available. Most of the demo's run directly on the STK500/STK501/STK503 boards, some require extra (but very simple) hardware. Femto OS is distributed under GPLv3 but a commercial license and support are available[2].

3.4.2 Features

The main features of Femto OS are[2]:

1. Round Robin Scheduling per Priority ; The scheduler runs through all tasks in the highest priority. The task which has not yet run, but is able to, gets the turn. If no task is able to run, a lower priority is examined. If no task can be started, the idle task is started. The idle task does not consume resources.

2. Preemptive and cooperative; Per task it can be specified if the task should be preemptible or not. Tasks which are run on a cooperative basis do not need to save any registers.

3. Shared Stacks for tasks that do not run very often, for example that are watching some switches, can share their stack space. Among each other they run cooperatively, but with the remaining tasks they still run preemtive.

4. Register Compression; Registers which are not used in a specific tasks, or registers only used in area's where no context switch is possible do not need to be saved. Especially in the AVR architecture which has 32 registers, this can save a lot of stack space. This also makes the context switch quicker.

5. Separate OS/ISR Stack Space; Regular operating systems have no special stack for the OS. The code for save/restore context is simple, but on every task stack a copy of variables used in the OS appear.But, in the Femto OS a separate stack is used for the OS. Something similar applies to the interrupt service routines. You may give it its own space or let it make use of the OS stack.

6. Power save on Idle; The idle task is a portable method, in which you can call a power save mode of the device. Of course this is implemented in the port on the AVR's.

7. Honest Time Slicing; Many OS's have a tick interrupt with a fixed time interval. There are situations however where this may starve particular tasks, especially if they are at the end of the cycle. With Honest Time Slicing you can ensure every task has at most a given number of sub ticks to run. The sub tick timer is reset at every task switch. Since this time slice can be set per application and can also be extended, this is particularly useful for implementing communication tasks. If you know some communication can be completed in say 5 ms, just set the time slice for that task to 5 ms. As default, the Femto OS uses equidistant tick interrupts.

8. OS interruptible; Interrupts can never occur in routines where the stack is changed, the context save/restore for example, and in the event handlers. Apart from this, the OS and kernel calls can be made interruptible if needed. However, it is also possible to work in a OS protected mode. Thus, depending on your needs the system can be quite responsive. Note that nested interrupts are not allowed per default. Kernel calls are not preemptable. For calls that are handled in OS space, this makes no real sense, and calls that are handles in OS space are usually very short.

9. Resource Tracking; Tasks can be terminated by the OS if an error occurs. Usually in the testing phase because the stack overflows or not all used registers are saved on the context. Tasks can also be terminated by an kernel call. The OS makes sure all hold locks on for example the file system, mutexes etc are released when such an event takes place.

10. Rendez Vous, Mutexes, Queues; Several synchronization primitives are available. You can let tasks (any number) wait for each other, you can define a (reentrant) mutex, and you can make use of queues. Queues are locked by stating the number of bytes involved. A lock is obtained if the queue is ready for reading or writing. At task can hold several locks simultaneously. It is also possible to request the possession of two mutexes, queues or a combination simultaneously. The task remains blocked until both resources become available.

11. Priority Lifting; Also called priority inheritance. If a high priority task blocks on a low priority task, the low priority task is increased in priority. If the synchronization is over, the low priority task is set to its original

priority. What differs from the standard implementation is that the low priority task gets its the priority stored in flash, instead of the priority it had before the block occurred. This is done to save RAM.

12. Precision Delays; Femto OS is driven by a 16 bit timer, and the system is quick enough to run a 1000Hz timer interrupt on a ATtiny861. You have the possibility to wake every task a fixed number of ticks after its last wake, which makes precision timing possible.

13. Watchdog per Task; For every task a watchdog can be defined which keeps track of the activity of your task. (This is not related to the device watchdog itself) If your task gets stuck, it will bark and give you the opportunity to act in a particular way. This is most convenient when implementing communication busses. In this way you don't have to check on every spot if the timeout has passed, simply let the watchdog take care of that.

14. High Resolution Load Monitor; Using the sub tick timer it is possible to monitor the time spent in every task, the OS and the ISR. This functionality can be compared with the well known windows taskmanager. It also keeps track of the use of the registers and stack using a watermark technique.

15. Parameter Checking; All methods that take parameters have optional parameter checking in place. This protects you, for example, from passing non existent task numbers, priorities, reading from empty queues and so on. Of course, when you go to production, you switch these checks off.

16. Stack Overflow Protection; When living on the edge, you must have tools to rescue when you fall over. On every context switch the system calculates if the context will actually fit on the remaining stack. If not, the context is not saved, to prevent overflow, and an error message is generated. Of course, when your code goes into production this can be switched off.

17. Error Callback; The port file contains a method that is called in case of an error. The current implementation is to present that information on PORTA by blinking the leds. It shows an error code, and subsequently which task or slot caused the problem. If the error is not fatal, Femto OS will terminate the offending task and resume operations. The Femto OS also contains a method for recreating the particular context of a task and restart it.

18. Configuration Checking; Since there are numerous options that must be configured, a mistake is easily made. That in itself is not a problem, but it becomes a problem when the code still compiles and runs. Therefore there

is a special configuration check facility (all standard preprocessor code) which checks the typo's made most often.

19. Lots of Example code; At this moment we have 12 demo applications demonstrating what this system is capable of. At the same time you can learn from it, how the functions should be used. It is often a lot quicker to modify already working code than start from scratch.

20. Source available and free; Whether this is a reason to choose for the Femto OS is a matter of taste. In any case, you can easily modify the code if something is not to your liking. And if you are willing to publish your code and embedded upgrade instructions it may be suited for commercial applications as well. If not, well, there still is the commercial license.

21. Minimal Footprint; Only those parts you really need are compiled into the Femto OS, this not only applies to the methods implementing the API but also to the core of the system itself. This can all be fine tuned with a number of configuration parameters.

22. Heavily Documented; The Femto OS core code and port file is heavily documented. The overall design is pretty straightforward OS system design, so anyone with some knowledge in that area should find it easy to understand and modify. The core file contains 43% of lines with code, 50% with comments, 7% of blank lines. Or, if you count the characters, 62% of them is devoted to comments.

23. Over 40 ports; For the AVR, Femto OS has been ported to 44 devices. All demo applications have been tested on the hardware, except in those cases where the hardware does not accommodate the application. Where possible, the preprocessor reduces footprint by removal of the higher stack pointer etc. Also, the ATmega256x is supported, albeit with the limitations imposed by gcc. Adding a new port is easy since all device information is contained within one file.

24. X-platform toolchain builder; Included in the package is a toolchain builder which installs the latest versions of binutils, gcc, avr-libc and avrdude on your system. It even solves the gmp and mpfr library problem for you. It has been tested on Gentoo Linux, Ubuntu Linux, Mac OSX, Windows 2000 and Windows XP.

3.4.3 Limitations

Femto OS is specially designed for gcc, and the generated executable depends on preprocessor processing of the source code. Therefore, Femto OS is bound to the limits given in the table 3.4.1. If you need to run more than 16 tasks, or more synchronization primitives, you need to use an other OS for those applications[2].

Parameter type	Value
Maximum number of tasks	16
Maximum number of priorities	8
Maximum number of locks	15
Minimum TCB size per task	2 bytes
Smallest appk size per task	4 bytes
Minimum OS stack size	9 bytes
Minimum OS other RAM usage	3 bytes
Smallest application	258 bytes

Table 3.4.1: Femto OS Parameters[2]

Chapter 4

I2C Protocol

4.1 Introduction

Inter-Integrated Circuit; generically referred to as "two-wire interface") is a multi-master serial single-ended computer bus invented by Philips that is used to attach low-speed peripherals to a motherboard, embedded system, or cellphone or other electronics.[10]

I²C uses only two bidirectional open-drain lines, Serial Data Line (SDA) and Serial Clock (SCL), pulled up with resistors. The significance of such an arrangement is that it allows for all the devices on the bus to work at different voltage levels and as the devices can only pull the line low and the high is handled by the pull ups.

4.2 Reference Design

The reference design for I2C is a bus with a clock (SCL) and data (SDA) lines with 7-bit addressing. The bus has two types of nodes[10]:

1. Master node : The node that issues the clock and addresses the slaves
2. Slave node : The node that receives the clock line and address.

The I2C bus has a multi-master architechture which means any number

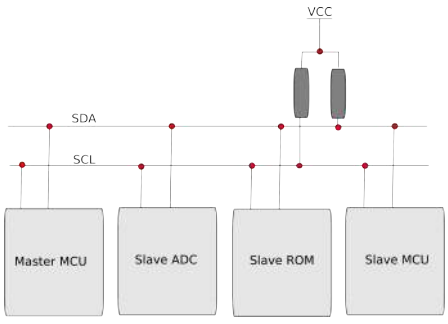

Figure 4.1.1: A sample I2C schematic

of master nodes can be present. Additionally, master and slave roles can be changed between messages.

There are four potential modes of operation for a given bus device:

1. Master transmit : The master is sending data to a slave
2. Master receive : The master is receiving data from a slave
3. Slave transmit : The slave is sending data to the master
4. Slave receive : The slave node is receiving data from the master

The master is initially in master transmit mode by sending a start bit followed by the 7-bit address of the slave it wishes to communicate with, which is followed by a single bit representing whether it wishes to write(0) to or read(1) from the slave.

If the slave exists on the bus then it will respond with an ACK bit (active low for acknowledged) for that address. The master then continues in either transmit or receive mode (according to the read/write bit it sent), and the slave continues in its complementary mode (receive or transmit, respectively).

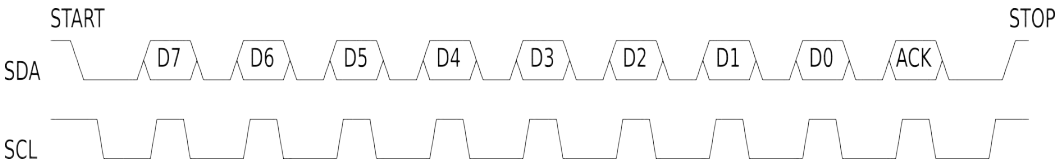

Figure 4.2.1: I2C Timing Diagram

The address and the data bytes are sent most significant bit first. The start bit is indicated by a high-to-low transition of SDA with SCL high; the stop bit is indicated by a low-to-high transition of SDA with SCL high.

If the master wishes to write to the slave then it repeatedly sends a byte with the slave sending an ACK bit. (In this situation, the master is in master transmit mode and the slave is in slave receive mode.)

If the master wishes to read from the slave then it repeatedly receives a byte from the slave, the master sending an ACK bit after every byte but the last one. (In this situation, the master is in master receive mode and the slave is in slave transmit mode.)

The master then ends transmission with a stop bit, or it may send another start bit if it wishes to retain control of the bus for another transfer.

4.3 Bit-Banged I2C Module

4.3.1 Introduction

In most AVR micro-controllers we have a hardware I2C module, but there are some AVR MCUs which do not have this module. The I2C Reference design allows for software emulation or ”bit-banging” of the I2C hardware. The only requirements for such an I2C module is the presence of two open drain i/o pins in the MCU.

4.3.2 Application Design

The application flow is given in the Figure 4.3.1. The application and its libraries are given in Appendix B.

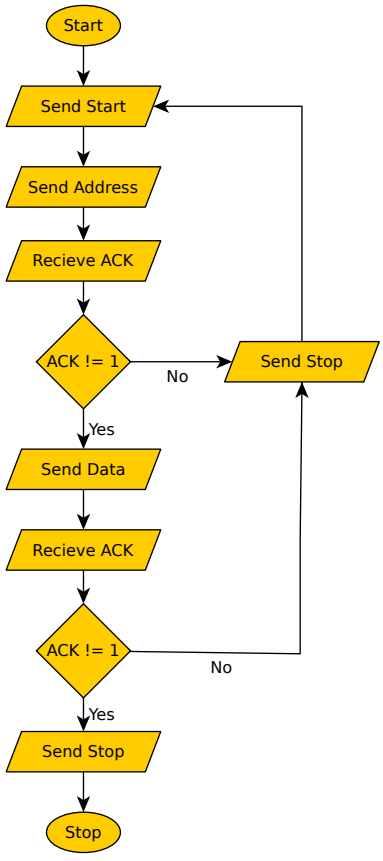

Figure 4.3.1: Application Flow

Chapter 5

Methodology and Results

5.1 Methodology

This is how we did what was needed.

1. The AVR 8-bit architecture was studied
2. Then OS Fundamentals were studied
3. A list of suitable RTOSes was found
4. Femto OS as a suitable candidate was identified
5. Sample applications for Femto OS were implementd on ATmega32
6. An Implementation of the Bit-Banged I2C Master Library was made

5.2 Results

These are the results we got.

1. Femto OS as a suitable candidate was identified and implemented.
2. An Implementation of the Bit-Banged I2C Master Library for Femto OS was made.
3. The waveforms shown in figure were observed

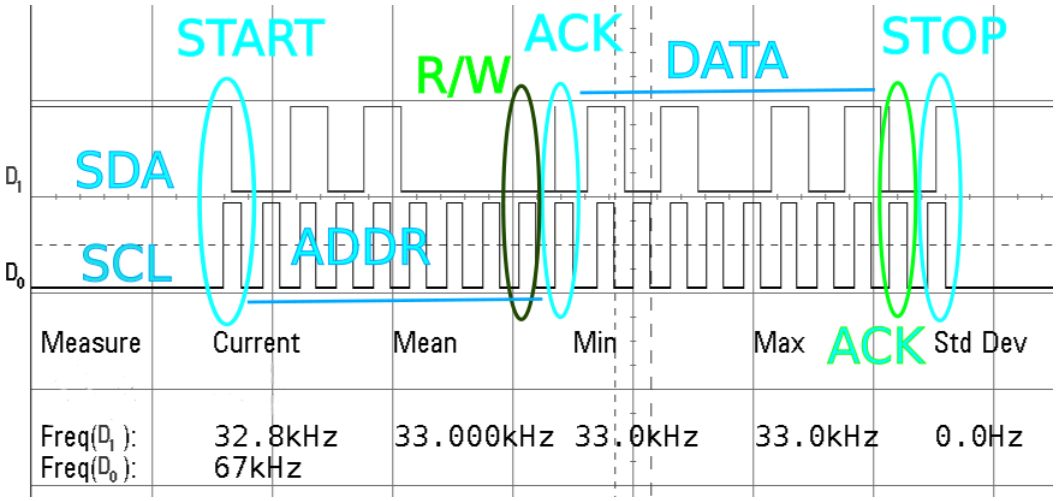

Figure 5.2.1: I2C Master Implementation waveforms

5.3 Conclusions

These are the main conclusions of the study:

1. A scalable RTOS for AVR-8 Bit Microcontrollers was found.
2. A Bit-Banged I2C Module for AVR RTOSs was written.

5.4 Future work

The study can be taken forward as follows:

1. To further test and improve the I2C module
2. To write an ADC driver for the RTOS

Appendix A

Full list of RTOS for AVR

The following is the list of RTOSes for AVR as compiled by AVRFreaks.net[6] on the thread `http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=66371&start=0&postdays=0&postorder=asc&highlight=` :

1. FreeRTOS: A Free RTOS for micro-controllers;
2. AvrX: Real-Time Kernel for AVR processors;
3. YAVRTOS: Yet Another Atmel AVR Real-Time Operating System;
4. AVRAsmOS: A tiny OS for small AVRs
5. pc/OS RTOS Kernel (for larger AVR processors >= mega128)
6. uSmartX: Non-Preemptive Priority-based Multitask RTOS
7. COMATOS
8. Task dispatcher
9. Opex
10. csRTOS: one of the longest 'freaks threads ever
11. Adam Dunkel's Protothreads
12. Femto OS
13. TinyOS
14. Contiki
15. pico OS

16. scmRTOS: a C++ OS with Mit license
17. iRTOS: a C preemptive OS with LGPL license
18. DuinOS
19. NUTOS (also Ethernut)
20. SST - Super Simple Tasker
21. FunkOS
22. AtomThreads
23. XMK (may no longer be supported)
24. BeRTOS (with lots of peripheral support libraries)
25. ChibiOS from Giovanni de Sirio.
26. mthreads
27. RTK

Appendix B

Bit Banged I2C Library

The following is the Header file ; ”i2c_test.h” for the I2C library :

```
/*! I2C Bit-Banged Master Module "Header File"
 *  Author : Akhil Guliani
 *  Written for : Femto OS
 *  Designed for : AVR 8 bit Core
 */

/*************************************************************************/
/****** Fixed definitions used
/*************************************************************************/

/*! I2C frequency selection */
#define sclfreq                                         7

/*! The I2C Port
 * The definition for the AVR Port to be used for the I2C Bus
 */
#define i2c_PORT                                        PORTB
#define i2c_PIN                                         PINB
#define i2c_DDR                                         DDRB

/*!I2C Pin Position
 * The two pins used as the I2C Data and Clock line defined here
 */
#define i2c_SDA                                          3
#define i2c_SCL                                          4
```

```
/*! I2C Address Definitions
 *  All the address used by the I2C BUS are defined here
 */
#define write_target_addr                0xAA
#define gen_call_addr                    0x00
/*****************************************************************/
/***** Functions For the I2C Bit-Banged Master
/*****************************************************************/

void h_Delay(void)__attribute__ ( (always_inline ) ) ;
/*! fn void h_Delay(void)
 * It is used to provide a Delay used for driving the clock
 * It is attributed an inline function
 */

void  i2c_start(void)__attribute__ ( (always_inline ) );
/*! \fn void  i2c_start(void)
 * This function sends an I2C Start condition
 * It is attributed an inline function
 */

void  i2c_stop(void)__attribute__ ( (always_inline ) );
/*! fn void  i2c_stop(void)
 * This function sends an I2C Stop condition
 * It is attributed an inline function
 */

Tbool i2c_master_write_byte(Tuint08 byte);
/*! fn Tbool i2c_master_write_byte(Tuint08 byte)
 * This function is used to write a byte to the Serial Data line
 * this function is also responsible for reading the Acknowledge
 * and returning it to the calling function
 */

unsigned char i2c_master_read_byte(void);
/*! fn unsigned char i2c_master_read_byte(void)
 * It is used to read a byte from the Serial Data line
 * It is also responsible for sending the Acknowledge
 * and returning it to the calling function
 * It is also responsible for driving the Clock line while reading
 */
```

```
/*****************************************************************/
/*Include Files Having Function Definitions for the above prototypes
/*****************************************************************/

#include "i2c_test2.c"

/*****************************************************************/
/*****************************************************************/
```

The following is the C-file; i2c_test2.c containing the function definitions for the I2C library :

```
/*! I2C Bit-Banged Master Module "C-File"
 *  Author : Akhil Guliani
 *  Written for : Femto OS
 *  Designed for : AVR 8 bit Core
 */

/*****************************************************************/
/**** Pre-Processor Functions
/*****************************************************************/
// I2C Data Related Functions

#define set_SDA()                       i2c_PORT |= (1 << i2c_SDA)
#define clear_SDA()                     i2c_PORT &=  ~(1 << i2c_SDA)
#define read_SDA()                      ((i2c_PIN >> i2c_SDA) & 1)
#define write_DDR_SDA()                 i2c_DDR |= (1 << i2c_SDA)
/// write_DDR_SDA()
//sets the i2c_DDR position for I2C Data line

#define read_DDR_SDA()                  i2c_DDR &= ~(1 << i2c_SDA)
/// read_DDR_SDA()
// clears the i2c_DDR position for I2C Data line

// I2C Clock Related Functions

#define set_SCL()                       i2c_PORT |= (1 << i2c_SCL)
#define clear_SCL()                     i2c_PORT &=  ~(1 << i2c_SCL)
#define read_SCL()                      ((i2c_PIN >> i2c_SCL) & 1)
```

```
#define write_DDR_SCL()                 i2c_DDR |= (1 << i2c_SCL)
/// write_DDR_SCL()
//sets the i2c_DDR position for I2C Clock line

#define read_DDR_SCL()                  i2c_DDR &= ~(1 << i2c_SCL)
/// read_DDR_SCL()
// clears the i2c_DDR position for I2C Clock line

/*************************************************************/
/***** Function definitions
/*************************************************************/

void h_Delay()
{
        int j = sclfreq;
        for(; j > 0; j--){
                asm("nop");
        }
}
// for clock freqency of < 100Khz choose sclfreq >= 7
// for clock freqency of < 400khz choose sclfreq between 4 and 6

Tbool i2c_master_write_byte(Tuint08 byte)
{
        unsigned int i = 0;
        //unsigned int k = 0;
        Tbool nack = false;
        write_DDR_SCL();
        read_DDR_SDA();
        // due to this changes in port will enable/disable
        // internal pullups for the DATA line

        for(;i < 8; i++){
                if((byte & 0x80) != 0){
                set_SDA(); // enables pullups
                h_Delay();
                }
                else{
                        clear_SDA();// disables pullups
                        h_Delay();
                }
```

```
                set_SCL();// set the clock high
                h_Delay();
                byte=byte<<1;
                h_Delay();
                clear_SCL();// set the clock low
                h_Delay();
                // uncomment below to use clock streching
                // not tested
                /*while(read_SCL() == 0){
                        k++;
                        if(k>20) break;
                        }
                */
        }
        clear_SDA();
        // read_DDR_SDA(); //not needed
        h_Delay();
        set_SCL(); // set clock high
        h_Delay();
        nack = read_SDA(); // read the acknowledgement
        h_Delay();
        clear_SCL();// set clock low
        h_Delay();
        // write_DDR_SDA(); //not needed
        // uncomment below to use clock streching
        // not tested
        /*while(read_SCL() == 0){
                k++;
                if(k>20) break;
        }
        */
        clear_SDA();
        return nack; //return ACK value to the calling function
}

void  i2c_start()
{
        set_SDA();
        set_SCL();
        h_Delay();
        clear_SDA(); // gives negative edge while clock is high
```

```
        h_Delay();
        clear_SCL();
        h_Delay();
}
void  i2c_stop(void)
{
        h_Delay();
        set_SCL();
        h_Delay();
        set_SDA(); // gives positive edge while clock is high
        h_Delay();
        clear_SCL();
        h_Delay();
}

unsigned char i2c_master_read_byte(void);
{
        int l = 8;
        unsigned char byte = 0x00;
        write_DDR_SCL();
        read_DDR_SDA(); // prepare for reading SDA line
        set_SDA(); // Enable internal pull ups
        for(;l > 0; l--){
                h_Delay();
                set_SCL(); // set clock high
                h_Delay();
                // read data line on clock high
                if(read_SDA() != 0 ){
                        byte |= 0x01;
                        }
                else{
                        byte &= 0xFE;
                }
                h_Delay();
                clear_SCL();
            h_Delay();
            byte = byte << 1;
         }
         // send acknowledgement
         clear_SDA();
         h_Delay();
```

```
        set_SCL();
        h_Delay();
        h_Delay();
        clear_SCL();
        h_Delay();
}
/*****************************************************************/
/*****************************************************************/
```

The Femto Os Application is as follows ;

```
void appLoop_test(void)
{
        while(true){
                devLedPORT |= (1<<0);
                Tbool nack = false;
                Tuint08 packet;
                i2c_start();
                nack = i2c_master_write_byte(gen_call);
                if (nack == true){
                        i2c_stop();
                        taskDelayFromNow(10);
                        continue;
                }
                packet = 0xA5;
                nack = i2c_master_write_byte(packet);
                if (nack == true){
                        i2c_stop();
                        taskDelayFromNow(10);
                        continue;
                }
                i2c_stop();
                devLedPORT &= ~(1<<0);
                taskDelayFromNow(10);
        }
}
```

Appendix C

How to make a Femto OS Application

This is a short tutorial on making an application for Femto OS[2].

Step 1 : Install the tool chain using the following instruction in the terminal:

```
~$ cd FemtoOS_0.92/
~/FemtoOS_0.92$ sudo ./Install_Scripts/install_toolchain
```

Step 2 : Install the Eclipse Workspace using the following instruction in the terminal:

```
~/FemtoOS_0.92$ sudo ./Install_Scripts/install_eclipse_workspace
```

Step 3 : Go to the workspace/ folder in the IDE/ directory

Step 4 : Within the workspace/ folder go to FemtoOS_HelloWorld/

```
~/FemtoOS_0.92/IDE$ cd workspace/FemtoOS_HelloWorld/
~/FemtoOS_0.92/IDE/workspace/FemtoOS_HelloWorld$ ls
devices  include  Release  src
```

Step 5: The four sub directories in the folder contain the links to all the source files present in the MainCode directory

The most important files in this directory are the following:

- src/code_TestHelloWorld.c
- include/config_application.h

Now to write an application here are the things one needs to do

1. In code_TestHelloWorld.c:

```
#include "femtoos_code.h"
// only include file required by os

#if (preTaskDefined(<name>))
// the #if is used to check the task is defined
// or not in the config_application.h file
void appLoop_<name>(void)
// all applicaton space names start with app
// appLoop specifies that the following is
// the main loop for the application
{
                while(true)
                {
                        // code and all declarations here
                }
}
#endif
```

2. In config_application.h

```
/* ================================================== */
/* TASK NAMES ======================================== */
/* ================================================== */
#define  CN_00                                          <name>

/* ================================================== */
/* INCLUDE TASKS ===================================== */
/* ================================================== */
 //#define  TaskInclude_<name>                  cfgExclude
//won't compile
#define TaskInclude_<name>                  cfgStartRunning
//will compile

/* ================================================== */
/* STACK SIZES ======================================= */
/* ================================================== */

#define  StackSafety                                    4
#define  StackSizeOS                                    24
#define  StackSizeISR                                   0
#define  StackSizeShared                                0
```

```
#define  StackSizeOverride                          46
// here the last option can be chaged to cfgOverRideNon
// and then individual task stack sizes can be defined

/* ================================================== */
/* REGISTER USE ===================================== */
/* ================================================== */
#define  RegisterUse_<name>                  registersAll
//this gives an advanced optimization option by selecting
//only those registers that are used by the task ,
//the registers can only be selected in groups of four
```

Step 6: After the code has been written as stated above it can be compiled as below:

```
~/FemtoOS_0.92/IDE$ ./script/compile -h
Usage: ./script/compile [all|<Project>] [cross|<Device>] [compact]
[info] [system] [clean]
<Project> = one of the following projects:
    Bare          Minimal          FlashLeds      Watchdog
Rendezvous     Queus          Sleep     Hooks          Interrupt
Shell          Passon         Remember       HelloWorld

<Device> =  one of the following devices:
attiny2313     attiny24       attiny25       attiny261
attiny43u      attiny44       attiny45     attiny461      attiny48
attiny84       attiny85       attiny861      attiny88 atmega128
atmega1280     atmega1281     atmega1284p    atmega16
atmega162      atmega164p     atmega168     atmega2560
atmega168p     atmega2561     atmega32       atmega324p
atmega325p     atmega3250p     atmega328p     atmega325
atmega3250     atmega48       atmega48p      atmega64
atmega640     atmega644      atmega644p     atmega645
atmega6450     atmega8        atmega8515     atmega8535
atmega88       atmega88p
Option 'compact' compiles with cfySysOptimized set to cfgTrue
Option 'info' generates more information on display about the
compile result, but only if one project and one device is compiled.
Option 'system' call the AVR tools on your system rather then the
ones build by the toolchain. Of course, they must be installed.
Option 'clean' deletes the Release subrirectory of the project or
```

```
all projects is requested. No compile takes place.
If the Device or Project is not specified, the last (succesfull)
values are used. Options are not stored.
************************************************************
%% we use the following script
FemtoOS_0.92/IDE$ ./script/compile HelloWorld atmega32 compact info
%% here is what we get as result
  text    data     bss     dec     hex    device      project
  1668       0      78    1746    6d2    atmega32    HelloWorld

Function ==========  Stack ===   Regtable   Individual registers
use(changed)
:                        0 (  0)    ........
appLoop_Display:         0 (  0)    ........
HelloWorldStr:           0 (  0)    ........

Label ====  Name ======================================
Task  0     Display

Address     Fields ===================================
0x0060      xOS
0x0078      Stack00
0x00a7      tcb00
0x00ab      uxTickCount
0x00ad      uiOsStatus

Address     Size =    Unit ============================
0x0060      0x0046    ./femtoos_shared.o
0x00a6      0x0001    ./code_TestHelloWorld.o
0x00a7      0x0007    ./femtoos_shared.o

Errors and warnings === (see Maincode/binaries/compile_results
for more details)
************************************************************
```

Appendix D

Sample Schematics for Femto OS

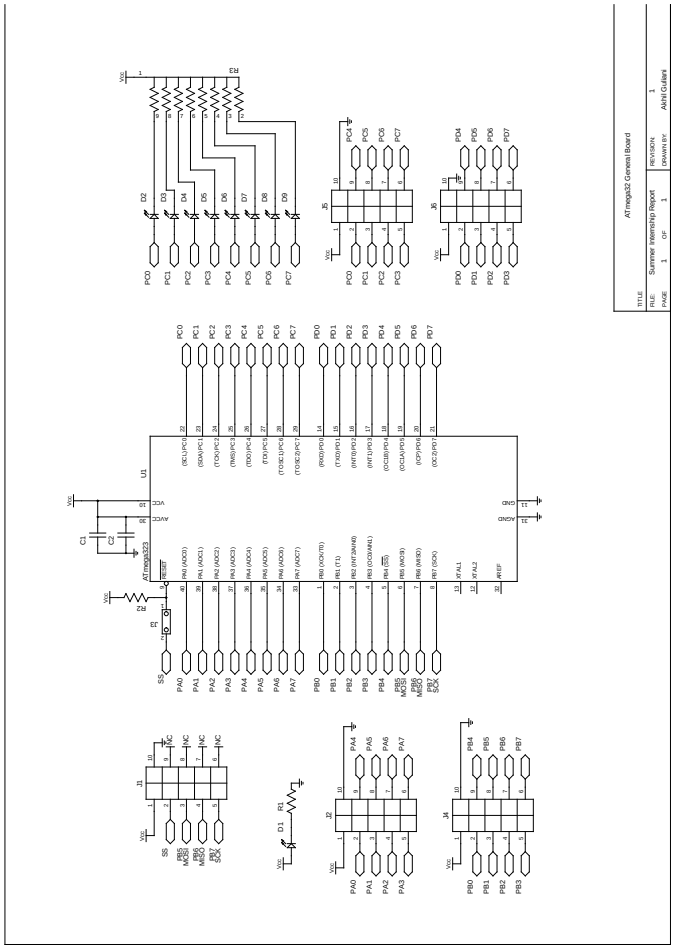

Figure D.0.1: Sample Schematics

Appendix E

About GIPEDI

The Airtel IIT Delhi Centre of Excellence in Telecommunication (AICET) within the Bharti School of Telecommunication Technology and Management, IIT Delhi has started an Internship Program in Engineering Innovation and Design for Undergraduate / Postgraduate students in any branch of the Electrical Sciences e.g. Electrical Engg, Electronics with Telecommunication / Communications, Computer Science, Instrumentation, Opto-electronics etc.

This Internship Programme is a different and separate program from the IIT Delhi Summer Research Internship Program which has a limited intake. This internship program will accept any student selected under the IITD Summer Research Program with no charges (no application fee, no handling charges and no internship fee will be charged).

For application form and further information go to the following website:`http://sites.google.com/site/aicetinterns/`

Appendix F

Profile of Author / Resume

Curriculum Vitae

Name	:	Akhil Guliani
Date of Birth	:	15 / 5/ 1990
Gender	:	Male
Address (residential)	:	A-140 Yojana Vihar, Delhi – 110092
Telephone No	:	011 – 22149637 (residential)
		9990684442 (M)
E-mail id	:	akhil.guliani@gmail.com

Educational Qualifications :

1. School Education :

Class	Name of School	Name of Board	Year of Passing	Total Percentage	Subjects Taken
X (secondary)	Ryan International School, Trans Yamuna	CBSE , Delhi (Central Board for Secondary Education)	2006	92.6 %	Mathematics Science English Sanskrit Social studies
XII (Senior Secondary)			2008	90.6 %	Mathematics English Physics Chemistry Biology

2. University Education :

Currently enrolled in Seventh semester of BE (Instrumentation and Control Engineering) at Netaji Subhas Institute of Technology (NSIT), Delhi; under the Delhi University

Exams Given	Results
Semester 1	Passed in first division with 73%
Semester 2	Passed with distinction with 78%
Semester 3	Passed with distinction with 77%
Semester 4	Passed with distinction with 77%

Semester 5	Results Awaited
Semester 6	Results Awaited

Aggregate percentage so far : 76%

Internships undertaken :

1. Undertook Winter Internship at NSIT Delhi under Dr. Smriti Srivastava From 10/12/2009 to 10/1/2010
2. Under took Casual studentship at IIT, Delhi under Dr IP Singh from 1/6/2010 to 31/7/2010
3. Undertook Global Internship Program in Engineering Design and Innovation of the Airtel IIT Delhi Centre of Excellence in Telecommunication (GIPEDI.AICET) , Bharti School of Telecommunication Technology and Management, Indian Institute of Technology Delhi under Prof. Subrat Kar From 15/12/2010 to 15/1/2011
4. Undergoing Global Internship Program in Engineering Design and Innovation of the Airtel IIT Delhi Centre of Excellence in Telecommunication (GIPEDI.AICET) , Bharti School of Telecommunication Technology and Management, Indian Institute of Technology Delhi under Prof. Subrat Kar From 15/05/2011 to 15/07/2011

Conferences, Seminars and workshops Attended :

1. Attended session on Internet Security and Hacking by Ankit Fadia conducted by IEEE Delhi Section Computer Society Chapter at Netaji Subhas Institute of Technology, Delhi on 22nd September 2008

2. Attended and Participated in Robokriti: a Workshop conducted by Technophilia Solutions in association with Innovision 08 on Fundamentals of Robotics at Netaji Subhas Institute of Technology, Delhi on 23rd – 24th August 2008

3. Participated in the 2nd YUVA Meet 2010 on "Understanding Climate change through the Social Glass" organised by The Energy Research Institue in partnership with the Ministry of Youth Affairs & Sports, Government of India and British Council from 2-3 Feburary 2010 at New Delhi

4. Awarded a Certificate of Merit for participation of Renewable Energy Festival, held under the Rashtriya Urja Jana-Jagriti Abhiyan and the Orange Revolution organized by Vijnana Bharti in collaboration with Ministry of New and Renewable Energy at Netaji Subhas Institute of Technology on 18th January 2011

Events Organized in the College campus:

1. Innovision 2008 Participated as a Volunteer
2. Innovision 09-10 Certificate of Appreciation as Awarded for Organizing Kriti-Mechanical
3. Innovision 09-10 Certificate of Excellence as Awarded for Coordinating School Bag
4. Innovision 09-10 Certificate of Appreciation as Awarded for 3rd Position in B-Plan (Ultimo–Empressario)

5. Organized IEEE GOLD Workshop on UNIX by Ayush Jain in April 2010 at NSIT, Organizer
6. Renewable Energy Festival, January 2011 Held At NSIT, as Speaker
7. Awarded Outstanding Student Volunteer Award 2011 by IEEE Delhi-Section

Participation in Academic Competitions during School:

1. Awarded Certificate of Merit in 9th National Science Olympiad (final round held on 18th February 2007 in New Delhi
2. Successfully Participated in 8th National Science Olympiad held on 24th November 2005 in India and Abroad
3. Successfully Participated in 7th National Science Olympiad held on 2nd December 2004 in India and Abroad
4. Successfully Participated in 10th National Science Olympiad held on 24th February 2008 in New Delhi
5. Awarded a Certificate of Merit and First Prize for Science Quiz at Hillwoods Academy Delhi in session 2006-2007
6. Participated in Expressions 2005 in Power Point Presentation held at Bal Bharti School Brij Vihar, Ghaziabad

Skill Set

1. Software Skills:
    a. Multisim 10 and 11
    b. UltiBoard 10 and 11
    c. OrCAD PSPICE
    d. GEDA
    e. MATLAB
    f. C and C++
    g. Assembly
    h. GCC
    i. AutoCAD
2. Hardware Skills :
    a. MCS-51
    b. AVR-core
    c. PCB Designing

Areas of Interest:

1. Embedded Systems

2. Robotics
3. PCB Designing
4. Computer Graphics
5. Computer Vision
6. Operating Systems

Hobbies :

1. I love playing squash. I have played squashed up-till the district level during my school days.
2. Basketball is another sport I love to play.
3. Reading be it of any kind is a pleasure.
4. Working on specific problems.

Bibliography

[1] Atmel, AVR Website.

[2] F. O. Team, Femto OS website.

[3] Wikipedia, Real-time computing ; Wikipedia The Free Encyclopedia, 2011, [Online; accessed 18-July-2011].

[4] Atmel, AVR ATmega 32 Datasheet.

[5] Wikipedia, Real-time operating system ; Wikipedia The Free Encyclopedia, 2011, [Online; accessed 18-July-2011].

[6] Avrfreaks.net, AVR-Freaks Forum.

[7] F. Team, FreeRTOS website.

[8] Galvin, Gagne, and Silberschatz, *Operating Systems Concepts*, John Wiley & Sons, Inc., eighth edition, 2011.

[9] X. Team, XMK website.

[10] Wikipedia, I2C Wikipedia, The Free Encyclopedia, 2011, [Online; accessed 20-July-2011].