CADE9,
Design and Implementation of Embedded Real-Time System

CS-3404: MINOR PROJECT

Project Report Submitted by:

Anshul Omar (07000037)
Nitin Yadav (07020008)
Rohit Yadav (07020003)

Project Guide:

Prof. A. K. Agrawal
(Professor and Head)
DEPARTMENT OF COMPUTER ENGINEERING
INSTITUTE OF TECHNOLOGY
BANARAS HINDU UNIVERSITY
VARANASI

DEPARTMENT OF
COMPUTER ENGINEERING
INSTITUTE OF TECHNOLOGY
BANARAS HINDU UNIVERSITY
VARANASI-221005, (U.P.)
INDIA

CERTIFICATE

This is to certify that **Mr. Anshul Omar, Mr. Nitin Yadav** and **Mr. Rohit Yadav,** third year undergraduate students in the Department of Computer Engineering, Institute of Technology, Banaras Hindu University, have worked on their project, during the period Dec 2009-Apr 2010, entitled **"CADE9, Design and Implementation of Embedded Real-Time System"** under my direct supervision and guidance, the findings of which have been incorporated in this report. They have worked diligently, meticulously and methodically. The report submitted by them embodies the original work done by them during the development of the project. Their project report has been found satisfactory and is approved for submission.

Prof. A. K. Agrawal
Professor and Head,
Department of Computer Engineering,
Institute of Technology,
Banaras Hindu University

ACKNOWLEDGEMENT

We would like to convey our deepest gratitude to **Prof. A. K. Agrawal**, Professor and Head, Department of Computer Engineering, Institute of Technology, Banaras Hindu University, who guided us throughout the project. His superb guidance and constant encouragement were the motive force behind this project work.

We are very thankful to all the technical and non-technical staffs of the Department of Computer Engineering for their assistance and co-operation.

Lastly, we are thankful to: Mr. Peter Eckstrand, author of *cocoOS*, for his support via email; our friends and fellow engineering students of ITBHU: Mr. Abhishek Modi, Mr. Heeralal, Mr. Ritesh, and Mr. Divakar Bari for helping us with the electronic equipments, circuit parts and overall concepts used in the design and implementation of our project.

Anshul Omar
07000037

Nitin Yadav
07020008

Rohit Yadav
07020003

B.Tech/IDD Part III,
Computer Science and Engineering
Department of Computer Engineering,
Institute of Technology,
Banaras Hindu University

Table of Contents

ABSTRACT

We describe here CADE9, Design and Implementation of Embedded Real-Time System.

CADE9 is an embedded real-time system on which arcade games like pong, snakes etc. can be written and played. For designing such as system we used *cocoOS*, an open source cooperative (round-round) task scheduler based on co routines, on ATmega32 microcontroller. To reduce the overall cost of the project, individual circuit elements were bought, soldered and interfaced according to our circuit design.

KEYWORDS

Real-Time, RT, Embedded System, Task, Task Scheduling, Round-Robin Algorithm, Non-Preemptive, Microcontroller, RTOS, Arcade Games

1. Introduction

Real-time systems are finding increasing applications nowadays. According to a recent estimate, the number of computers deployed in real-time applications vastly out number those in ordinary applications. Many of these computers are embedded. Embedded systems are ubiquitous and control many devices in common use today.

Real-time (RT) is a quantitative notion of time measured using a physical clock. A system is called real-time whenever we need to quantitatively express time in order to describe its behavior. Behavior of a system is described by listing the inputs to the system and the corresponding response of the systems.

Real-time systems find application in industrial applications like process control systems, industrial automation systems, SCADA applications, test and measurement equipments, and robotic equipments; in telecommunication applications like cellular systems, video conferencing, and cable modems; in aerospace like avionics, flight simulation, airline cabin management systems, satellite tracking systems and computer on-board an aircraft; in consumer electronics like set-top boxes, audio equipment, internet telephony, microwave ovens, intelligent washing machines, home security systems, air conditioning and refrigeration, toys and cell phones; in medical areas like robot used in recovery of displaced radioactive material; in peripheral equipments like laser printer; in transportation like Multi-Point Fuel Injection (MPFI) system; in computers and internet like switches, routers etc; in multimedia applications like video conferencing; and in defense applications like missile guidance system.

1.1 Purpose

The purpose of our project is to study the design and implementation of an embedded real-time system, and using those concepts, develop a low cost prototype of an embedded real-time system based on round-robin (cooperative) scheduling algorithm that may be extended to applications in real life as discussed above.

1.2 Scope

The RTC (Real Time Computing) or reactive computing is powered by the embedded system (hardware) together with the real-time task scheduler (software) in the RTOS (Real Time Operating System) that runs the system.

For design and implementation of the embedded system, we use microcontrollers. A microcontroller (μC) is a small computer on a single integrated circuit consisting internally of a relatively simple CPU, clock, timers, I/O ports, and memory. Program memory in the form of NOR/NAND flash ROM is also often included on chip, as well as a typically small amount of RAM. μCs are very commonly used in embedded real-time systems as they reduce the cost of building real-time systems at the same time provide an easy platform for engineers to develop applications on them. We chose ATmega32 which is a low cost, low power, high-performance 8-bit RISC μC with maximum throughput at 16Mhz and has 32KB of In-System Self-programmable Flash program memory and 2KB internal SRAM.

For RTOS, which provides a real-time task scheduler, we explored many open source implementations like FreeRTOS, FemtoOS, Yavrtos but finally used cocoOS. Typically, in an RTOS the task scheduler serves as the over all in-charge.

In cocoOS, the task scheduler is a priority based round-robin (cooperative) scheduler, based on coroutines. Task procedures are scheduled to run till the completion of tasks. They execute from the first line of the procedure to the last line. The use of coroutines enables us to implement task procedures that does not have to execute all the way down to the last line of code. Instead execution can end in the middle e.g waiting for a semaphore to be released. When execution is resumed, it continues at that line. The fine thing with coroutines, is that this can be done without having to save the complete task context when switching to another task. Hence, cocoOS is non-preemptive in nature. Also, in round-robin algorithm both the response time and CPU utilization is high with medium throughput and medium turn around time, and most importantly it is starvation free. The real-time system we designed here according to our application is of type firm real-time system because in case a deadline is missed, the system will not fail.

Using the above mentioned hardware and software parts form the building blocks of our project that power the RTC in our embedded real-time system.

1.3 General Model

In a typical real-time system, a sensor converts some physical characteristic of its environment into electrical signals. The collected data from the user or the environment which are fed to an input device of the real-time computer. The real-time computer acts accordingly and directs the actuators. An actuator takes converts electrical signals from a computer into some physical actions. The physical actions may be motion, change of thermal, electrical, pneumatic, or physical characteristics of some objects. Also, a human computer interface is provided to monitor, diagnose and control the system.

Typical example of sensors include a photo-voltaic cell converts light energy into electrical energy, a temperature sensor typically operates based on the principle of a thermocouple, and a pressure sensor typically operates based on the piezoelectricity principle. Typical examples of actuators are: motors, heaters, hydraulic and pneumatic actuators.

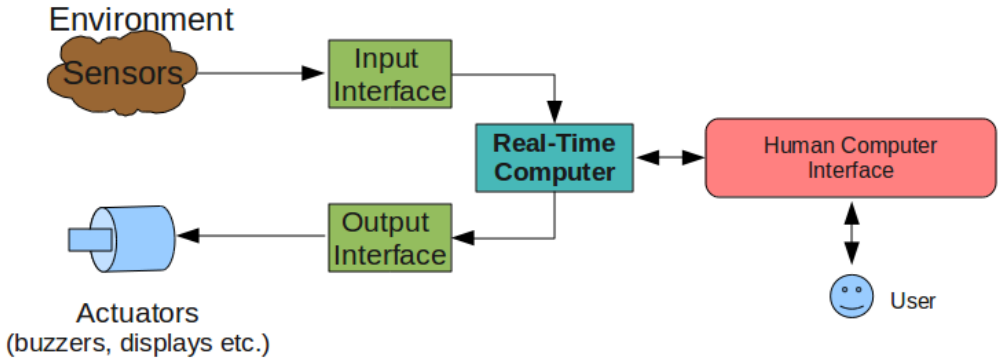


Fig 1.3.1 Typical Real-Time System

The real-time system we incorporated in our project is reactive i.e. it processes interaction between environment and user. For the matter, a feedback system is used.

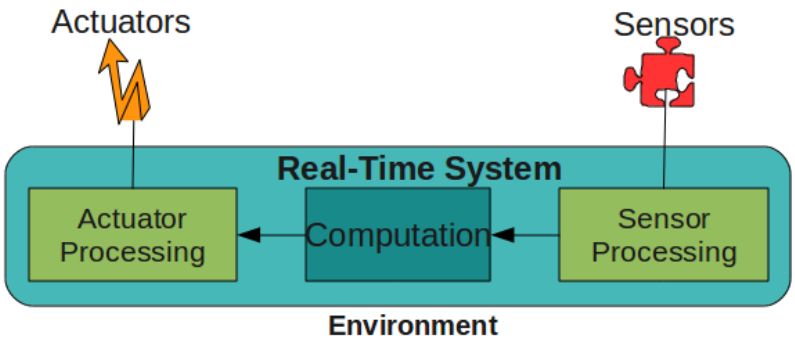


Fig 1.3.2 Feedback Mechanism of RT Systems

In our RT system, we consider user input via push buttons as the sensory data to be processed and display and buzzer outputs as actuators. The player playing the game, gets the data from the actuators (display and buzzer) and acts accordingly via the sensors (push buttons), sends feedback to the sensors after analyzing game state from the actuators.

2. Design and Analysis

2.1 Requirements

For this project, we chose to make a custom handheld real-time arcade gaming console on which classic arcade games like pong, snakes and breakout can be implemented and played. For the matter, we specifically designed and implemented one-player pong game. Pong or Ping-Pong was the world's first video arcade game. We used 7-segment displays that can display a maximum score of nine, for a particular arcade game, to display score of each player. Hence, we named it *CADE9*.

2.1.1 Sensor

To gather data from the environment and the user we require an input module or sensor. For the matter, we used buttons to interact with the user that assume the status of a sensor. Five buttons are suited for implementing an arcade gaming device, namely: UP, DOWN, LEFT, RIGHT and FIRE.

2.1.2 Actuators

For output modules or actuators we are required to develop custom hardware modules. A buzzer is used as an audio unit. A 12 x 7 (3mm) LED matrix and 7-segment displays serve as display units.

2.2 Architecture

The architecture is shown below. User provide commands to the device by using the buttons. Each button is assigned different pin on the µC. A button task checks which button is pressed and processes the command accordingly. The RT system actuates and sends signal to the output modules i.e. display and buzzer units.

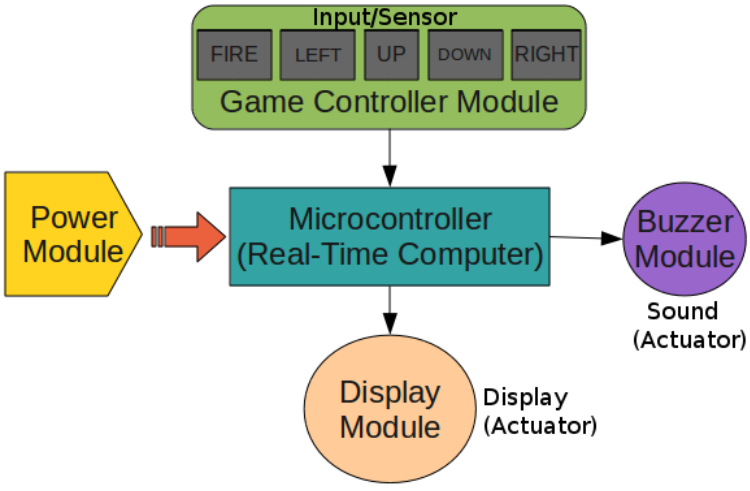


Fig 2.2a CADE9 RT Architecture

2.2.1 Implementation

For hardware implementation, we worked on a circuit that works according to our requirements. Then, we solder and interface the parts on a matrix circuit board and test the integrity by using a electronic multimeter. In such a RT system, the hardware and software is tightly coupled.

ATmega32 μC serves as the real-time computer. The pins of the μC are powered by a DC power module, clocked by a low-cost 16 Mhz ceramic oscillator. The μC is then interfaced with input and output hardware modules. A programmer port is interfaced with the μC, through which this embedded system can be connected to a workstation and the built software system can be burned on the flash program memory of the μC.

3. Design and Implementation

3.1 Hardware

The hardware constitutes the following parts:

• A matrix circuit board to hold all the hardware.
• A soldering iron, solder and solder flux to connect the hardware parts.
• 5V DC Power Supply provided by IC 7805 and two low pass-filters (capacitors).
• Real-time computer provided by ATmega32 µC.
• Real Time Clock for the µC provided by low-cost 16 Mhz ceramic oscillator with two 22pF low-pass filters (capacitors) to reduce noise in the voltage supply.
• Programmer Port, for programming the RT system, provided by a 6-pin jumper.
• Input module constituting of five buttons which are interfaced to the µC.
• Output module constituting of a buzzer, a display made by a custom made 12x7 LED matrix and two 7-segment displays.
• Two buffer IC 7406 to act as a constant current source and sink for the LED matrix display.
• Decoder IC 74138 to implement raster scanning on the display.
• Resistors to control voltage and power in the circuitry.
• Capacitors to act as low-pass filters to reduce noise in voltage supply.
• IC beds to hold ICs, enabling users to replace them in case they get damage.
• A USB Programmer that takes the machine code, compiled by the compiler and assembler, from a computer and burns it through the Programmer Port on the µC.

According to the circuit diagram, parts were connected, soldered and interfaced. Shown below is a picture of the hardware that constitutes the embedded RT system we developed.

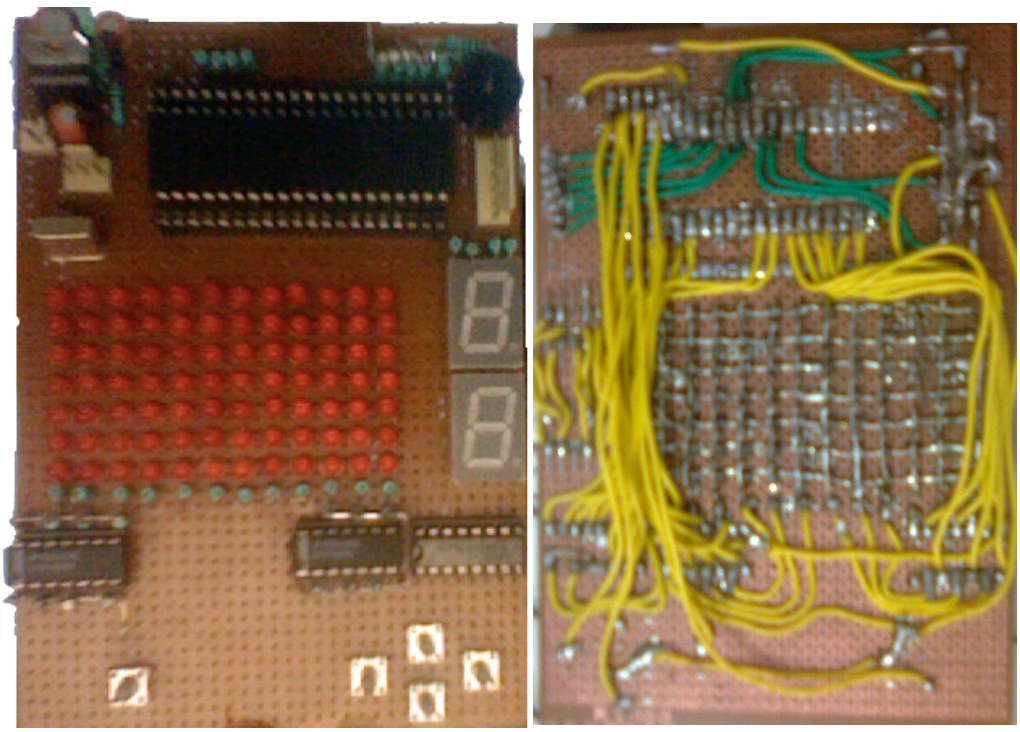


Fig 3.1a CADE9 (Front and Back) RT Arcade Game Console

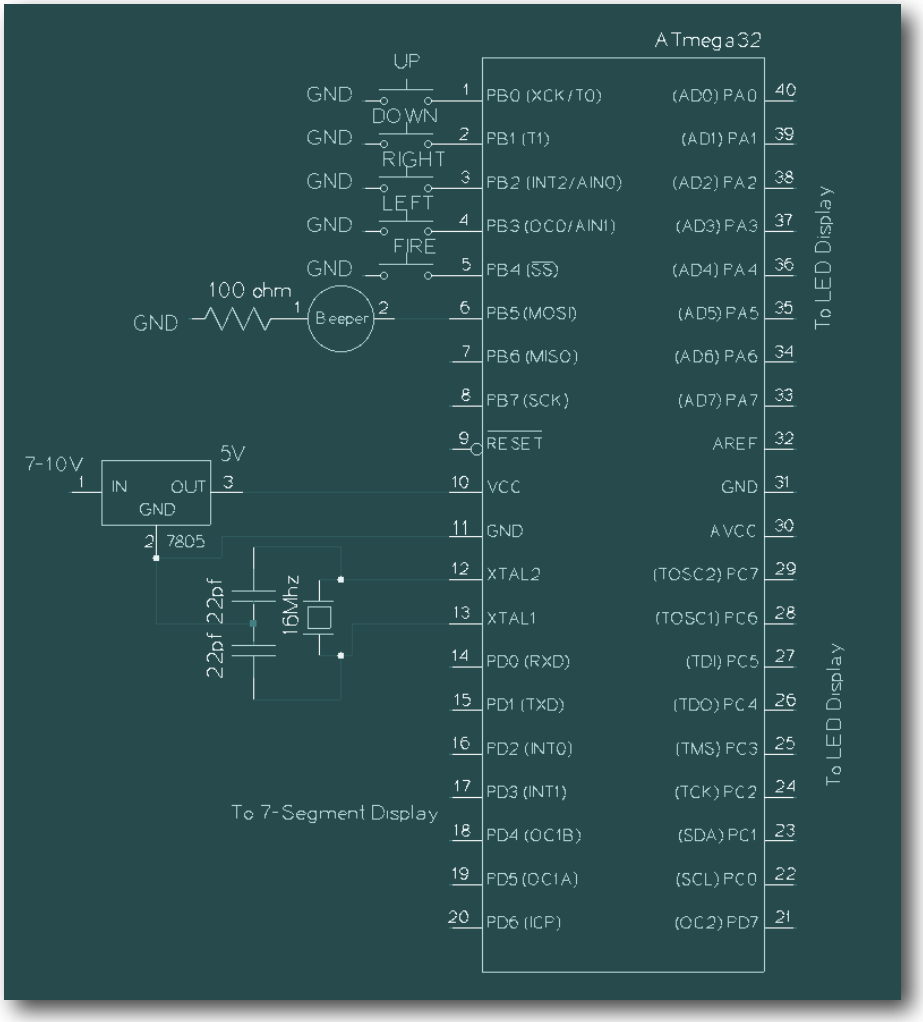


Fig 3.1b Circuit Diagram I

The above circuit was used for implementing the hardware. Circuit diagram for LED matrix display is discussed in following sections.

3.1.1 Constant DC Power Supply, IC 7805

IC 7805 is a 3 pin 1A voltage regulator and is an analog IC. On the input side a DC voltage greater than 5 V is applied and on the output side a constant DC voltage of 5 V is obtained. For reducing the noise, we use two low-pass filters or capacitors as shown in the circuit below.

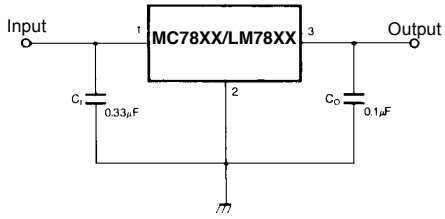


Fig 3.1.1a IC 7805 Circuit Diagram

3.1.2 The ATmega32 µC

The ATmega32 µC we used was 40 pin DIP (Dual In-line Package) and a low-power CMOS 8-bit µC 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.

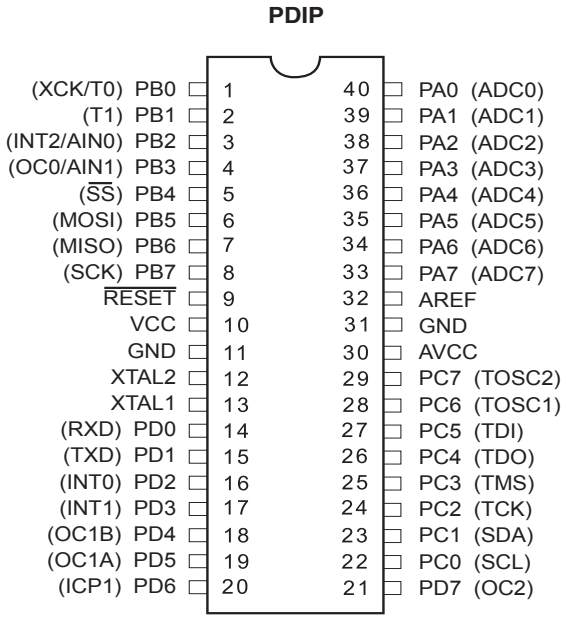


Fig 3.1.2a ATmega32 Pin Diagram

Some of its features are as follows:

• Advanced RISC Architecture
  – 131 Powerful Instructions – Most Single-clock Cycle Execution
  – 32 x 8 General Purpose Working Registers
  – Fully Static Operation
  – Up to 16 MIPS Throughput at 16 MHz
  – On-chip 2-cycle Multiplier
• High Endurance Non-volatile Memory segments
  – 32K Bytes of In-System Self-programmable Flash program memory
  – 1024 Bytes EEPROM
  – 2K Byte Internal SRAM
  – Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  – Data retention: 20 years at 85°C/100 years at 25°C
  – Optional Boot Code Section with Independent Lock Bits
  – True Read-While-Write Operation
  – Programming Lock for Software Security
• Peripheral Features
  – Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  – One 16-bit Timer/Counter with Separate Prescaler, Compare/Capture Mode
  – Real Time Counter with Separate Oscillator

– Four PWM Channels
– 8-channel, 10-bit ADC
– 8 Single-ended Channels
– On-chip Analog Comparator
• Special Microcontroller Features
– Power-on Reset and Programmable Brown-out Detection
– Internal Calibrated RC Oscillator
– External and Internal Interrupt Sources
– Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
• I/O and Packages
– 32 Programmable I/O Lines
– 40-pin PDIP
• Operating Voltages
– 4.5 - 5.5V for ATmega32
• Speed Grades
– 0 - 16 MHz
• Power Consumption at 1 MHz, 3V, 25°C for ATmega32L
– Active: 1.1 mA
– Idle Mode: 0.35 mA
– Power-down Mode: < 1 μA

3.1.3 Display and Sound Module

For making the 12x7 LED matrix, we soldered 84 LEDs such that we can control all the LEDs using 19 lines. For the matter we multiplexed the LEDs, to do that we connected all the anodes of the LEDs in a row and cathodes in a column. Consider a simple prototype of 3x3 LED matrix we made, shown below. We switched on only one row at a time to ground and turn on the required LEDs in that row and the process is repeated step by step for rest of the rows. This way if the switching is done fast enough, we can control the LEDs to display our game scene. This process is called raster scanning. Due persistence of vision, we see our game on the 12x7 LED display.

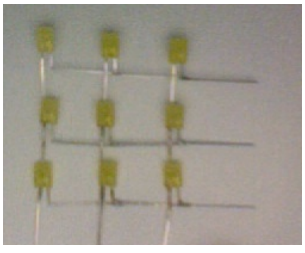


Fig 3.1.3a 3x3 LED matrix

The fast switching between the rows is done by decoder IC 74138 which is handled in software part of the system. For powering the 12 LEDs in a row, we are using two buffer IC 7406 which act as a constant current source and sink. This way current flows through the ICs and not the μC, preventing any heat damage of μC. The switching of these ICs are handled in the software too. The following circuit diagram shows interfacing and connections between these above discussed circuit parts.

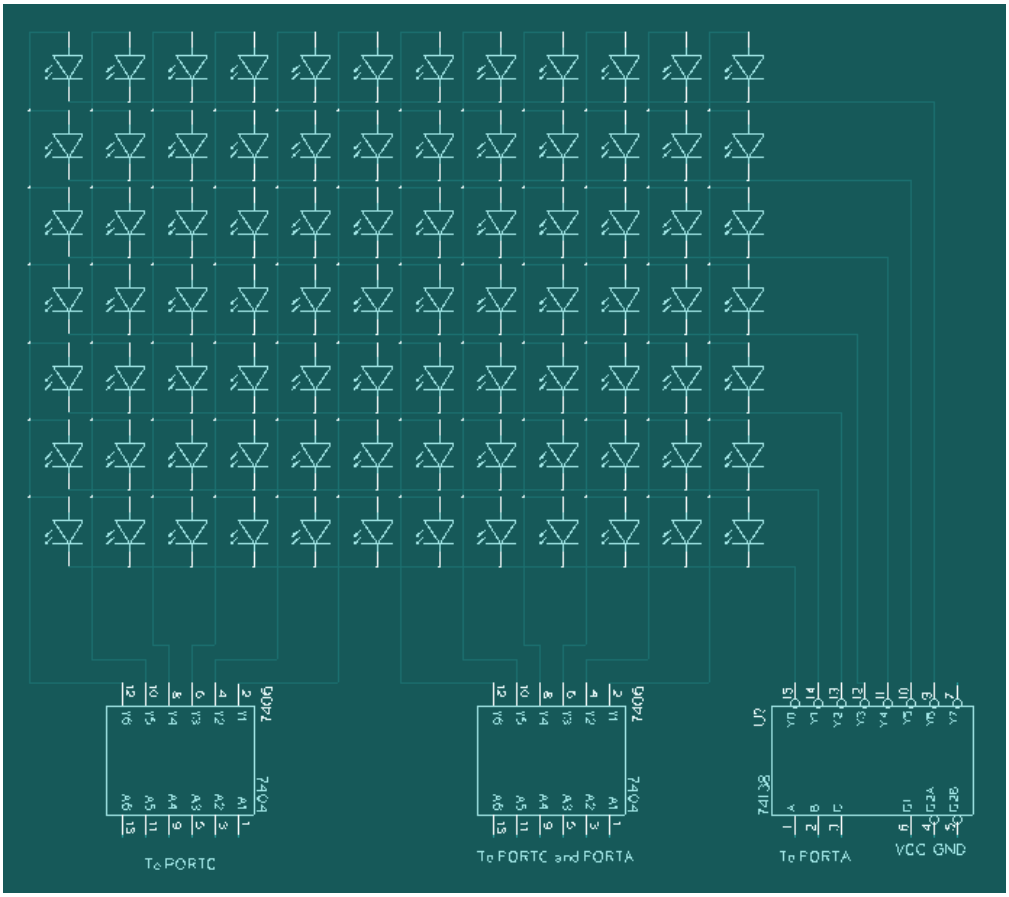


Fig 3.1.3b Circuit Diagram II

Apart from this display, we use two 7-segment displays to show scores of the players. That are also switched fast enough to display scores. For audio confirmation, we use a buzzer that is switched on according to a game implementation.

3.1.4 Game Controller

The game controller constitutes of five push buttons that are grounded when pressed. In the software part, a task checks value on the particular pin where the buttons are interfaced and corresponding action is taken.

3.2 Software

The software constituted *cocoOS,* an open source task scheduler written by Peter Eckstrand under the GPL license and our implementation of pong game according to the hardware interfacing. The code is completely written in portable C code and we used *WinAVR* package and the *AVR Studio 4* that provided us *avr-gcc* compiler, assembler etc. that compiles the code and produces machine code, a Intel HEX file. This HEX file is then written on the µC using an AVR USB Programmer. For this, the programmer bus on the AVR USB Programmer is connected to the Programmer Port on the embedded system and then the program is transferred. Interfacing is discussed in detail in sections below.

3.2.1 AVR USB Programmer

The AVR USB Programmer is connected to computer and the bus on the Programmer is connected to the Programmer Port.

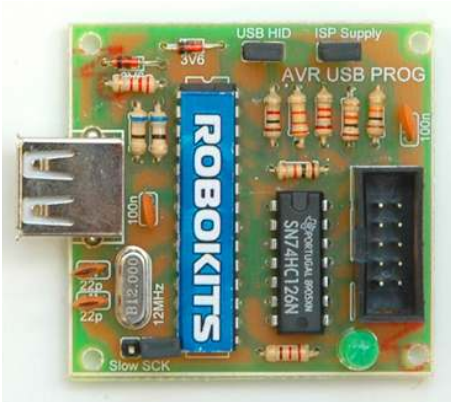


Fig 3.2.1a AVR USB Programmer

The built HEX file is then written on the µC using the software shown below. Other options like Fusebits and Lockbits can be set too. Once, the HEX file is successfully written on the µC, a constant DC power source is applied to start the RT system.

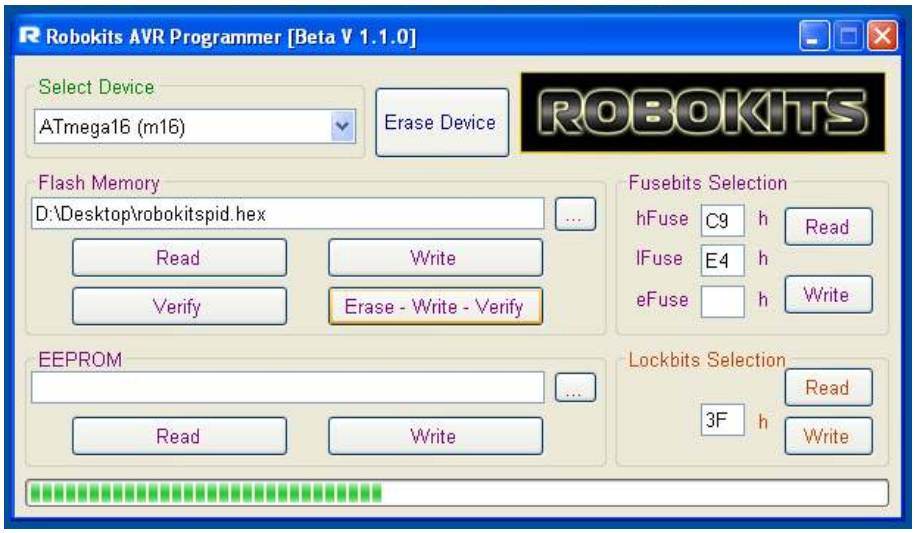


Fig 3.2.1b AVR Programmer Software

3.2.2 Source Code

The source code is made open source and the code resides in a public repository at: http://github.com/rohityadav/cade9. The main folder contains four folders and a *AVR Studio 4* project file:

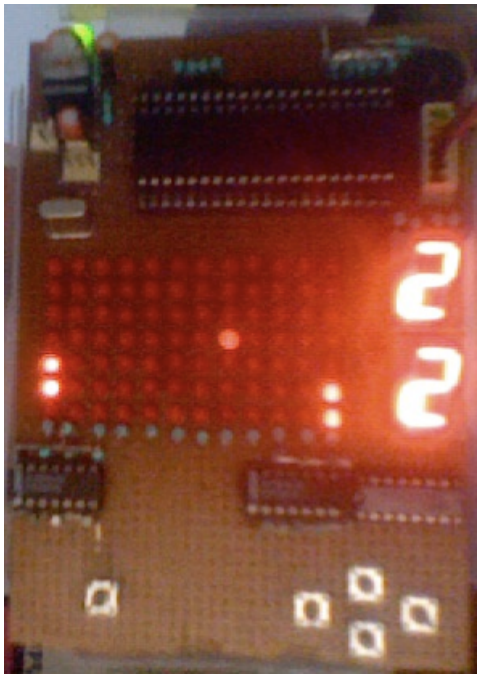
1. *build:* Here the HEX file is created and the project's Makefile resides.
2. *cocoOS:* Source code of cocoOS.
3. *port:* Hardware specific files. Here we set the CPU_CLOCK to the frequency of the oscillator used.
4. *cade9*: Contains main.c and main.h files that have task procedures that implement pong game on CADE9.

In cocoOS, in os_task.c file the os_task_tick routine searches all the waiting tasks and decrements waiting time. In case, a waiting task is found to be READY, it is executed then. The cooperative scheduler executes the highest available task, according to priority that is statically set, that in READY state. A task is created using os_create_task routine.
CADE9 has five tasks, that are statically scheduled:

1. button_task: This task processes user input waits for 15 clock ticks.
2. display_task: This task raster scan the LED display and waits only 1 clock tick.
3. ball_task: Updates ball position and movement, and waits 50 clock ticks.
4. ai_task: Calculates movement of computer's bat to win and waits 90 clock ticks.
5. score_task: Updates scores on 7-segment displays, waits 4 clock ticks.

5. Conclusion and Further Work

The project was successfully implemented. The overall cost of implementation of the embedded RT system was less than INR 1k or $20. In mass production, the cost is estimated to decrease. The following image shows the one player pong implementation in action.



In CADE9, we used an open source non-preemptive cooperative scheduler. To extend the project, we can work on writing a preemptive EDF based RTOS. We may further, extend the RT system to implement a more real life application and make it a hard RT system.

Finally, we can conclude that though this project finds potential work to be done, a robust prototype is created for further research and development that can be easily used to build low-cost embedded real-time system.

6. References

- Real-Time Systems by Rajib Mall, ISBN: 81-317-0069-0
- Modern Operating Systems by AS Tanenbaum, ISBN: 978-81-203-3904-0
- Operating System Concepts by Silberschatz et al, ISBN: 978-81-265-2051-0
- Procedural Elements for Computer Graphics by David F. Rogers, ISBN-13: 978-0-07-047371-3
- cocoOS: Project page, source and documentation, http://sites.google.com/site/cocoosorg/home
- ATmega32 Datasheet: http://www.atmel.com
- AVR Freaks: Forum for AVR Projects, http://www.avrfreaks.net
- Wikipedia: Encyclopedia on web, http://en.wikipedia.org
- LED Cube: LED Multiplexing, http://www.lomont.org/Projects/LEDCube/LEDCube.php
- Robokits India: Hardware parts, http://www.robokits.co.in/

7. Glossary

Arcade Games: An arcade game is a coin-operated entertainment machine, usually installed in public businesses such as restaurants, public houses, and video arcades. Most arcade games are redemption games, merchandisers (such as claw crane), video games, or pinball machines.

Embedded System: An embedded system is a computer system designed to perform one or a few dedicated functions often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts.

IC: Integrated Circuit.

Microcontroller: A microcontroller (also microcomputer, MCU or μC) is a small computer on a single integrated circuit consisting internally of a relatively simple CPU, clock, timers, I/O ports, and memory. Program memory in the form of NOR flash or OTP ROM is also often included on chip, as well as a typically small amount of RAM. Microcontrollers are designed for small or dedicated applications.

Nonpreemptive multitasking: Nonpreemptive multitasking is a style of computer multitasking in which the operating system never initiates a context switch from a running process to another process. Such systems are either statically scheduled, most often periodic systems, or exhibit some form of cooperative multitasking, in which case the computational tasks can self-interrupt and voluntarily give control to other tasks. When non preemptive is used, a process that receives such resources can not be interrupted until it is finished.

Process: In computing, a process is an instance of a computer program that is being executed. It contains the program code and its current activity.

Real-Time (RT): RT or Real-time is a quantitative notion of time measured using a physical clock.

Real-Time Clock: A real-time clock (RTC) is a computer clock (most often in the form of an integrated circuit) that keeps track of the current time. Although the term often refers to the devices in personal computers, servers and embedded systems, RTCs are present in almost any electronic device which needs to keep accurate time.

Real-Time Computing: 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 System: A system is call real-time whenever we need to quantitatively express time in order to describe its behavior. Behavior of a system is described by listing the inputs to the system and the corresponding response of the systems. A real time system may be one where its application can be considered (within context) to be mission critical.

Round-Robin Algorithm: RR is one of the simplest scheduling algorithms for processes in an operating system, which assigns time slices to each process in equal portions and in

circular order, handling all processes without priority. Round-robin scheduling is both simple and easy to implement, and starvation-free.

RTOS: A real-time operating system (RTOS) is an operating system (OS) intended for real-time applications. Such operating systems serve application requests nearly real-time. A real-time operating system offers programmers more control over process priorities. An application's process priority level may exceed that of a system process. Real-time operating systems minimize critical sections of system code, so that the application's interruption is nearly critical.

SCADA: Supervisory Control and Data Acquisition.

Scheduling algorithm: Scheduling algorithm is the method by which threads, processes or data flows are given access to system resources (e.g. processor time, communications bandwidth). This is usually done to load balance a system effectively or achieve a target quality of service.

Task: A task is "an execution path through address space." In other words, it is a set of program instructions that are loaded in memory.

Task Scheduling: Task Scheduling is a key concept in computer multitasking, multiprocessing operating system and real-time operating system designs. Scheduling refers to the way processes are assigned to run on the available CPUs, since there are typically many more processes running than there are available CPUs. This assignment is carried out by softwares known as a scheduler and dispatcher.

Task Throughput: Number of processes that complete their execution per time unit.