Implementation of Real Time Operating System for AVR Microcontrollers

Srishti Dubey
Electronics and Communication
Medi-Caps Institute of Engineering & Technology
Indore, India
shrishti1301@gmail.com

Ankit Chouhan
Electronics and Communication
Medi-Caps Institute of Engineering & Technology
Indore, India
chouhan4ankit@gmail.com

Guided By:
Devendra Singh Bais
Electronics and Communication
Medi-Caps Institute of Engineering & Technology
Indore, India
devendrabais@gmail.com

Abstract-
The objective of this paper is to create the operating system for real-time Robotics based on AVR microcontroller that operates in the multitask environments. In addition to this, a single hardware performs the multiple tasks without recompiling and/or burning the programs into the controller’s memory every time (invariably). The operating system in the AVR controller will read the program through secure digital (SD) card and execute it; if we insert the new card with the new program then same set of hardware performs the new task. The main advantage of this technique is that we don’t have to connect the computer with the hardware, however the program can be uploading in the flash disk (SD card) as the .hex file and when we insert the card in the hardware, it performs the task accordingly.

Index Terms- RTOS, ATmega32, SD card, RISC.

I. Introduction
A real-time operating system (RTOS) is an operating system (OS) that intended to serve real-time application requests. The main point of RTOS is, if programmed correctly; an RTOS can guarantee that a program will run with very consistent timing [1] and RTOS will always run the current program and execute it. Need of RTOS, to run the system in different environment with minimum speed. The purpose of this paper is to design a real time operating system which is based on AVR microcontroller. The software design of this paper is broken up into two parts-
A. Operating system
B. Card/boot loader program, which load the data into the SD/SC card.

The card loader program is accessed via the Atmel's SPI interface, which places the program into flash memory. When we interface the card with the controller, so the main OS program read the data of the card via SPI, and place it into the ROM memory at run time using boot loader program. After that our system will perform the task and if we remove the card from the hardware kit, then also the program will still run on the kit [2].

II. Problem Domain
In this paper, whenever we will perform the new task with the hardware so, we have to connect the hardware with the computer and recompiling or burning the controller and upload the new program in the controller.

SOLUTION Domain
The advantage of this paper is that, we can upload the program in the flash disk (SD card) in the .hex file without connecting computer with the hardware. Then, we insert the card in the hardware and hardware will perform the respective task. The AVR program coded and compiled in Code Vision AVR Studio 4, before installing the AVR studio 4, we first have to install the winavr because the winavr includes GCC compiler for the AVR target for C and C++ [3].

RESEARCH ARTICLE
OPEN ACCESS
We will put our O.S. in the boot-loader section of flash, because the boot loader resides at the highest addresses of flash memory. The maximum boot-loader size is 2048 instructions, is too big to fit in this space, so I fixed some functions in high flash addresses next to the boot-loader, keeping the functions that program the flash actually in the boot-loader. We will test our program in the AVR kit. There are various kits available in the market like STK500 for example.

III. ABOUT AVR ("ADVANCED VIRTUAL RISC") CONTROLLER

In this particular paper, introducing AVR family of microcontrollers and Atmel AVR 8 and 32 bit microcontrollers deliver a unique combination of performance, power efficiency, and as well design flexibility. Optimized speed time to market and also they are based on the industry's most code-efficient architecture for C and assembly programming. On the other hand, no other microcontrollers deliver more computing performance with better power efficiency and industry-leading development tools and design support let you get to market faster [4].

A. Overview of AVR Microcontroller
- AVR RISC Architecture.
- 32K bytes of in-system self-programmable flash program memory.
- We can access ROM at run time.
- It can deliver 16 million intrusions per second.
- 2K of SRAM.
- In-system programming by on-chip boot program.
- SPI - serial peripheral interface.
- Speed - 16 MHz
- 1024 bytes EEPROM [5].

B. Typical Hardware Support in AVR (ATmega32)
- Internal or External Oscillator/Clock [6].
- 10bit ADC.
- Real time clock.
- One or more USART.
- EEPROM.
- One or more timers
- External interrupts.

IV. Theory Oriented: - Design Idea Of Paper (Creating The Operating System For Real-Time Robotics On AVR Based System)

To create a real-time operating system for the Atmel mega32 microcontroller, the hardware waits for a card to be inserted and a reset button to be pressed; at that point a program is loaded from the SD card and gets executed [7]. The design of this particular hardware is divided into: the operating system itself and the card reader/program loader. Also the card reader is accessed via the Atmel's SPI interface by the program loader, which also places the program into flash memory and programs can be written similar to a standard Atmel Mega32 program, except that it must include the header file. The program loader resides in the Mega32's boot loader section of flash. Since, this gives it write access to other portions of flash memory so that it can write executables to program space.

a. Paper Specifications

To run this programs on an Atmel Mega32 development board; the SD card slot is designed in the board. We connect the SD card through PORTB with a jumper cable, which contains the Mega32's SPI interface. Also we wired the SD card connector's clock, Data In, and Chip Select pins to PORTB's pins 7, 5, and 0, respectively, and the Data Out to PORTB pin 6. The three inputs to the card each were fed
through separate step-down circuit to decrease the voltage from the power supply's 5V to the 3.3V required by the card. The output from the card was fed through a step-up circuit before going to B.6. To obtain the 3.3 volts needed for the HIGH voltage for the card, we used a simple voltage regulator [2]. Likewise, the card acts as a load on the 3.3V, drawing some current. The 3.3V was pulled down to 2.6V with the card's load, with the help of voltage dividers. Regulator is able to sustain a voltage of about 3.26V with the card's load, within the card's specifications.

We have 3 unused pins on PORTB after connecting to the card reader, due to that we added a pushbutton with a pull-up resistor and connected it to pin 1. This button is used as a reset that causes the OS to load the program from an SD card.

b. How to make an OS in the AVR Controller?
Few steps to achieve the task:
- Interfacing of SD/MMC card with ATmega32.
- Reading the data of card and accessing its data.
- Storing the program at run time in ROM location with the help of boot loader program.
- Setting the pointer to that stored program location.
- With the help of pointer running that installed program.
- Returning from that program location.

c. How the RTOS will work?
Writing the code for O.S. and creating the AVR development board with card reader. Hardware waits for a secure digital card to be inserted and a reset button to be pressed; at this point a program is loaded from the card to flash memory and also gets executed. A new card with a new program can be inserted and run, at any time. Executing a new program doesn't require reprogramming the Atmel processor. All programs on the Atmel have to be contained in flash memory, but only the boot loader can write to flash memory. Thus, it was decided to put the O.S. in the boot loader section of flash. This decision also helped to avoid address conflicts with user programs because the boot loader resides at the highest addresses of flash memory. The maximum boot loader size is 2048 instructions. It is too big to fit in this space, that's why we fixed some functions in high flash addresses next to the boot loader, keeping the functions that program the flash memory actually in the boot loader.

The Atmel programs created have coded and compiled in AVR Studio 4. AVR studio 4 creates .hex files when it makes a project; we took these .hex files and used them to program the SD card and write a programmer to load the data into the card, separate from the O.S. program, which programs the contents of a .hex file to the SD card.

Whenever, using the hardware in the new task environment the Operating system in AVR controller boots, it begins waiting for a new program in the SD card and a reset signal. At this point, it start begins reading the contents of the card into small RAM buffers, which subsequently transferred to flash, because RAM is limited [3]. Now, the buffers used to read data from the card overlap with O.S. Data structures that are not used until after the program is loaded. Since executing and loading never overlap, so we can use the same memory for both.

d. Time required to run the programs of different sizes
First run time: Program load time + time to jump over the main routine. Second run time: Time to check for new program + time to jump over the main routine.

Fig. 2. Graph of first run time

Fig. 3. Graph of second run time
VI. Conclusion And Future Implementation

At the end, we successfully accomplished the objective, to create the operating system for real-time based on AVR microcontroller that loads program dynamically from SD card and performs the task accordingly. Writing the code for O.S. and creating the AVR development board with card reader is done successfully. However, also completed the second objective, a single hardware performs the multiple tasks without recompiling and/or burning the programs into the microcontroller.

In designing the operating system, we first considered various library functions that could add to increase functionality that is, LCD and keypad drivers and support for multiple programs on a single card with a menu system to choose between them [3]. These kind of features, although useful bells and whistles are cut from the design as, they are extraneous to the core vision that creates dynamically-loading OS, there is some unnecessary complexity been added, and consumed extra flash and RAM, which may be needed for some user programs.

References


