International Journal of Information & Computation Technology.
ISSN 0974-2239 Volume 4, Number 17 (2014), pp. 1861-1868

http://www. irphouse.com

Real Time Operating System for Robotics using AVR Microcontrollers

Srishti Dubey, Devendra Singh Bais and Ankit Chouhan

Electronics and Communication
Medi-Caps Institute of Engineering & Technology, Indore, India

Abstract

The objective of this paper is to create an operating system for real-time Robotics based on AVR microcontroller operates in the multitask environments. In addition to this, creating a single hardware that performs the multiple tasks without recompiling and/or burning the programs invariably into the controller's memory. The operating system 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 uploaded 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 Term-RTOS, ATmega32, SD card, RISC.

Introduction

A real-time operating system (RTOS) is an operating system (OS) that intends to serve real-time application requests. The main function of RTOS, if programmed correctly; it can guarantee that a program will run with consistent timing [1] and always run the current program. The need of an RTOS is to run the tasks in different environment in minimum time. The purpose of this paper is to design a real time operating system for robotics which is based on AVR microcontroller. RTOS can be incorporated into the applications of industrial automation and robotics on a large scale with the help of an SD card in order to perform multiple functions. This would also cater the problem of absence of a computer as different SD cards can be fed with different programs which may be used in robotics without the support of a computer.

The software design of this paper is broken up into two parts:-

1. Operating system
2. 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 microcontroller, so the main OS program read the data of the card via SPI, and place it into the flash memory at run time using boot loader program. After that our system will perform the task and even if we remove the card from the hardware kit, the same program will still run on the kit [2].

Problem

Whenever a new task is performed on the hardware, it has to be connected with the computer to erase the current program and burn the new program into the microcontroller and this process has to be followed for every new program.

Solution

One of the solutions of the above problem is that, the program in the portable memory (SD card) can be uploaded in the .hex file without connecting computer with the hardware. Then, the card can be inserted in the hardware and it will perform the respective task.

The AVR program coded and compiled in Code Vision AVR Studio 4, before installing the AVR studio 4, winavr is first installed because the winavr includes GCC compiler for the AVR target for C and C++ [3].

The O.S. is put 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, which is too big to fit in this space, so some functions in high flash addresses are fixed next to the boot-loader, keeping the functions that program the flash actually in the boot-loader. The program is then tested in the AVR universal testing kit. There are various kits available in the market like STK500.

About AVR ("Advanced Virtual RISC") controller

Introducing AVR family of microcontrollers and Atmel AVR 8 and 32 bit microcontrollers delivering 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 get from market faster [4].

Overview of AVR Microcontroller

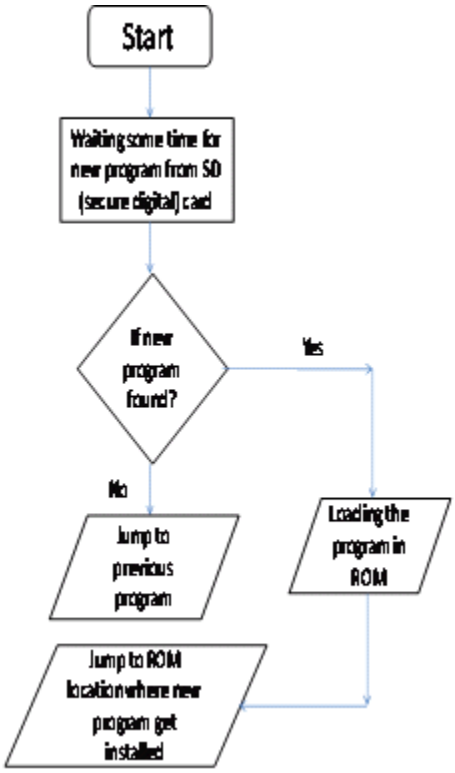
- AVR RISC Architecture.

- 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].

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.

Flowchart



Flowchart [7]

Theory oriented:-Design idea

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 [8].

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.

Hardware used for O.S testing

To run this programs on an Atmel Mega32 development board; the SD card slot is designed in the board. The SD card wss connected through PORTB with a jumper cable, which contains the Mega32's SPI interface. Also the SD card connector's clock was wired, 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, a simple voltage regulator was used [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.

Creating 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.

Working of system hardware for an RTOS

The code of RTOS was loaded into the AVR development board, consisting of card reader. Hardware waited for a secure digital card to be inserted and a reset button was pressed; at this point a program was loaded from the card to flash memory and also

got executed. A new card with a new program could be inserted and run, at any time. Executing a new program didn't require reprogramming the Atmel processor.

All programs on the Atmel processor 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 some functions were fixed in high flash addresses next to the boot loader, keeping the functions that program the flash memory actually in the boot loader.

All the programs created got coded and compiled in AVR Studio 4. AVR studio 4 creates .hex files; these .hex files were taken and used 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, the hardware is used 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 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 overlaps, so we can use the same memory for both.

Time required to run the programs of different sizes

The time taken by the boot loader code to prepare itself for the insertion of the SD card before being reset may be considered as constant.

The time taken for a program to be loaded and jump back to its main routine is its first run time as can be seen in "fig. 2".

First run time: Program load time + time to jump over the main routine.

The time taken by the program to check for new program and to jump over the main routine is its second run time as can be seen in "fig. 3".

Second run time: Time to check for new program + time to jump over the main routine.

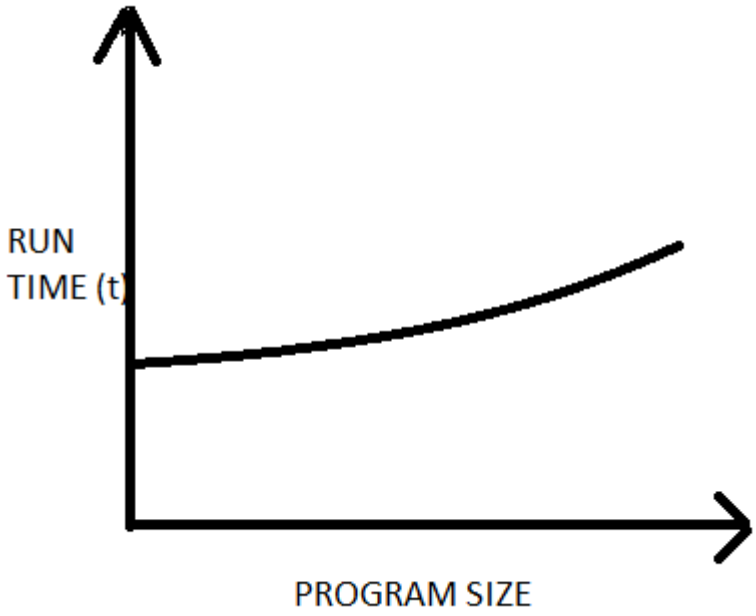


Fig. 2: Graph of first run time

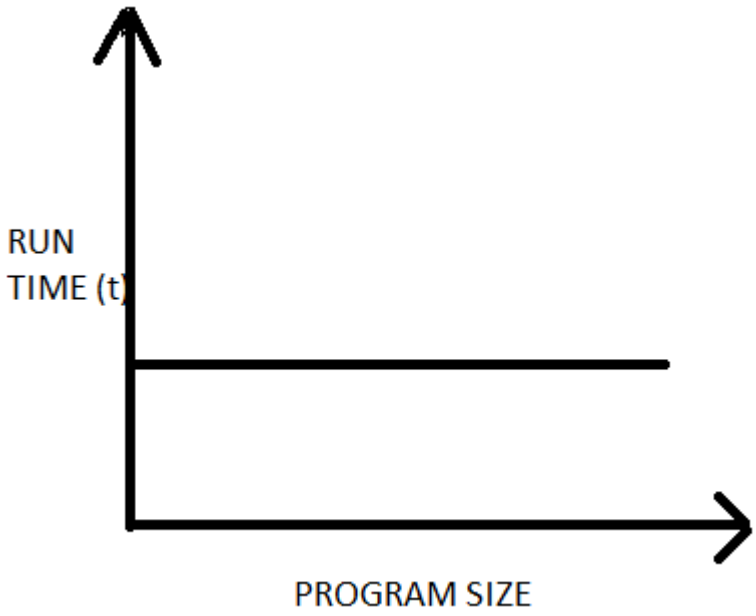


Fig. 3. Graph of second run time

HARDWARE DESIGN

ATmega32 is used for testing the designed RTOS. The circuit is as “fig. 4”,

ATmega32 microcontroller is used in 16 MHz clock, Max232 i.e.) for serial peripheral interface with PC, capacitors and LEDs.

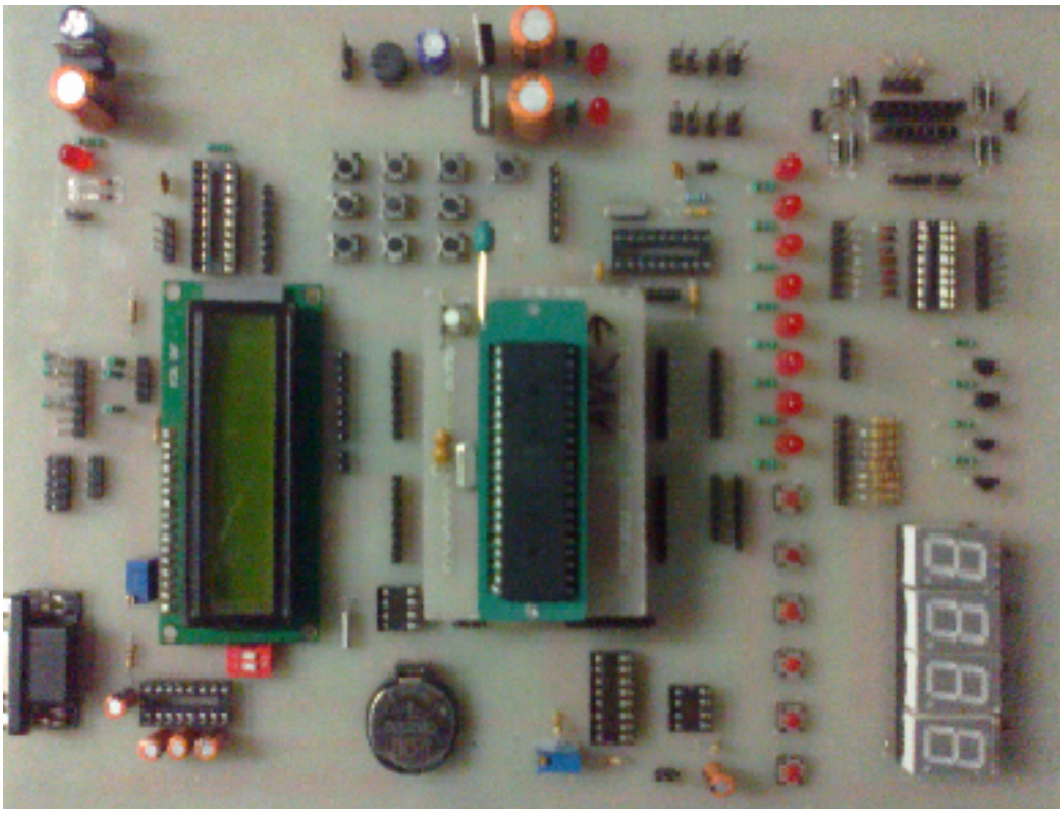


Fig 4: Circuit to test RTOS

Conclusion AND Future Implementation

At the end, we successfully accomplished the objective, to create the operating system for real-time robotics 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, the second objective was also achieved 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

[1] Tran Nguyen,Bao Anh, “REAL-TIME OPERATING SYSTEM FOR SMALL MICROCONTROLLERS”, Published by the IEEE Computer Society, pp.30-46, September 2009.

[2] AHMAD, S. Real-time multiprocessor based robot control. In Proceedings of the IEEE International Conference on Robotics and Automation (San Francisco, Calif., Apr. 1986). IEEE, NewYork, pp. 858-863.

[3] Taghi.Mohamadi, “Real Time Operating System for AVR Microcontrollers,” pp.376 – 380, 9-12 Sep. 2011, ISBN: 978-1-4577-1957-8

[4] http://www.atmel.com/products/microcontrollers/avr/default.aspx

[5] Atmega32 datasheet, www.datasheetcatalog.com

[6] PIC16F91X28/40/44-Pin Flash-Based, 8-Bit CMOS MCU sw/LCD & nanoWatt Technology DataSheet. Microchip Technology Inc., http://ww1.microchip.com/downloads/en/DeviceDoc/41250E.pdf, E edition, 2005.

[7] Dubey, Srishti; Chouhan, Ankit; Bais, Devendra Singh; , “Real Time Operating System based on AVR Microcontroller,” International Journal of Engineering Research & Technology (IJERT), ISSN: 2278-0181, Vol. 3 Issue 1, January-2014

[8] www.electroons.com/downloads/tuts/Lect2-ATmega16.pdf

[9] Baynes, K.; Collins, C.; Fiterman, E.; Brinda Ganesh; Kohout, P.; Smit, C.; Zhang, T.; Jacob, B.; , "The performance and energy consumption of embedded real-time operating systems," Computers, IEEE Transactions on , vol.52, no.11, pp. 1454-1469, Nov. 2003 doi: 10.1109/TC.2003.1244943

[10] Hessel, F.; da Rosa, V.M.; Reis, I.M.; Planner, R.; Marcon, C.A.M.; Susin, A.A.; , "Abstract RTOS modeling for embedded systems," Rapid System Prototyping, 2004. Proceedings. 15th IEEE International Workshop, pp. 210-216, 28-30 June 2004 doi:10.1109/IWRSP.2004.1311119