Sushant.S.Sakhalkar, Parth Gala, Divya Mohan / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 3, May-Jun 2012, pp.1694-1698

Embedded Gaming

Sushant.S.Sakhalkar*, Parth Gala, ***Divya Mohan**
*(Department of Electronics, MUMBAI University, Mumbai-88)
** (Department of Electronics, MUMBAI University, Mumbai-88)

Abstract- Playing games is everyone's favorite childhood memory. Whether it be Tic-Tac-Toe, Pacman, Mario, Dave or even the latest craze- Angry Birds, we all have at one point or another dreamt about personalizing and tampering with the controls to create our very own gaming experience. Toying around with this thought, we decided to finally implement a gaming system with elementary concepts of embedded systems and real-time programming. In the paper we have presented a system developed using C programming on Code Blocks IDE. Our minimalistic embedded hardware platform includes AVR 64 microcontroller, USB programmer, joystick, Jtag port. The hardware also doubles up as our human interface and is pre-programmed to enable the user to have a unique gaming experience.

Keywords – **Sdl Code –Blocks, UniBoard, MMA7361L, AVR 64,**

1. Introduction

Embedded systems have a vast amount of useful applications throughout the world. This has provided a new fillip to many technical enthusiasts for exploring the amazing field of embedded technology. Tailor-made features are incorporated carefully by programming the system. The video games which we play on TV also comprise of embedded systems that are programmed a priority to improve the user's real- time interaction with the system. The controllers of PS2, Microsoft X-box, and Nintendo Wii, all have sophisticated embedded systems running them. This spurred us to develop our very own gaming system by perusing various bibliographic resources-both electronic and otherwise as also guidance from various knowledgeable personnel. Developing small embedded system presents several challenges, due to size and weight constraints. Size reduction implies the use of small microprocessors with limited memory and processing power, which represents a severe constraint for the software, which must be carefully designed to be efficiently exploiting the available resources. Moreover, if the system is powered by batteries, energy management policies must be applied at different levels of the architecture to reduce power .consumption and prolong the system lifetime as much as possible. The operating system has a crucial role in guaranteeing the application performance; because the timing behavior of the application strictly depends on task scheduling, interrupt handling, synchronization protocols, and resource management algorithms. The simplest way for enforcing timing constraints to the application is through a static schedule implemented by a cyclic executive. However, this solution is not flexible to changes and is very fragile under overload conditions. Hence, a priority-based kernel is more suitable to support dynamic control applications with variable computational requirements [1].

There are several real-time operating systems available in the market, both free and proprietary, but very few of them are suitable for small embedded microcontrollers with limited processing resources. VxWorks and QNX Neutrino are two examples of commercial kernels commonly used in real-time control applications. Another real-time kernel widely used in the industry is Micro C/OS[2], which is a preemptive real-time kernel written in C, available for more than twenty different microprocessors. Among the open source kernels related to Linux, RTLinux[3] and RTAI use a small real-time executive as a base and execute Linux as a thread in this executive, whereas Linux-RK directly modifies the Linux internals. Most of these kernels, however, do not implement the state-of-the-art features derived from the real- time scientific literature, and are not easy to modify. More flexible kernels are MarteOS (written in ADA), allowing the user to specify the scheduler at the application level, and Shark) (written in C), which is a modular kernel handling tasks with different criticality and allowing the user to select the algorithms for task scheduling and shared resource management. However, all the kernels mentioned above are designed for medium size applications and are not suited for small microcontrollers. On the other hand, very small kernels have limited real-time features. For example, TinyOS is widely used in sensor networks to support the tasks running on the Motes, but it uses a FIFO queue for task scheduling and it cannot handle timing constraints.

This system employs Mu Cos II kernel [2] which

is developed by Micrium technology. Then the coding part is done in code blocks software language I used for coding. For advanced gaming features chipmunk's physics engine is used which has been optimized for giving real physics functionality to game objects.

The paper is organized as follows: Section II introduces the hardware system, including the microcontroller, accelerometer, rtc, joystick, and the real-time kernel; Section III deals with the software part; Section IV deals with conclusion.

2. Hardware System

The controller board is illustrated in Figure 1. It performs interfacing functions with the game; it provides us with joystick and accelerometer by which we can control the game.

2.1 Controller Board

The controller board is UniBoard, which consists of AVR 64 microcontroller. It is 8 bit microcontroller.

2.2 Advanced RISC Architecture

It has 130 Powerful Instructions, most of them having a single clock cycle execution. It is also packed with 32 x 8 General Purpose Working Registers as well as Peripheral Control Registers. It has fully static Operation up to 16 MIPS Throughput at 16 MHz.It has on-chip 2-cycle Multiplier.

Fig. 1 Pin Diagram of AVR 64

2.3 JTAG (IEEE std. 1149.1 Compliant) interface

Boundary-scan Capabilities According to the JTAG Standard. It has Extensive On-chip Debug Support Programming of Flash, EEPROM, Fuses, and Lock Bits are through the JTAG Interface.

2.4 Peripheral Components

It has Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes. Also available are two Expanded 16-bit Timer/Counters with Separate Prescaler, Compare Mode, and Capture Mode, Real Time Counter with a Separate Oscillator. It has two 8-bit PWM Channels with Byte-oriented Two-wire Serial Interface. Dual Programmable Serial USARTs and Master/Slave SPI Serial Interface with Programmable Watchdog Timer with On-chip Oscillator and On-chip Analog Comparator are additional attractive components.

2.5 Special microcontroller Features

Some of these are - Power-on Reset, Programmable Brown-out Detection, internal calibrated RC external oscillator, Internal Interrupt Sources with Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby. It has Software Selectable Clock Frequency. The feature of ATmega103 Compatibility Mode Selected by a Fuse with Global Pull-up Disable Differential Channels with Programmable Gain (1x, 10x, 200x).Also Byte-oriented Two-wire Serial Interface and Dual Programmable Serial USARTs and Master/Slave SPI Serial Interface. Programmable Watchdog Timer with On-chip Oscillator and On-chip Analog Comparator are additional provisions.

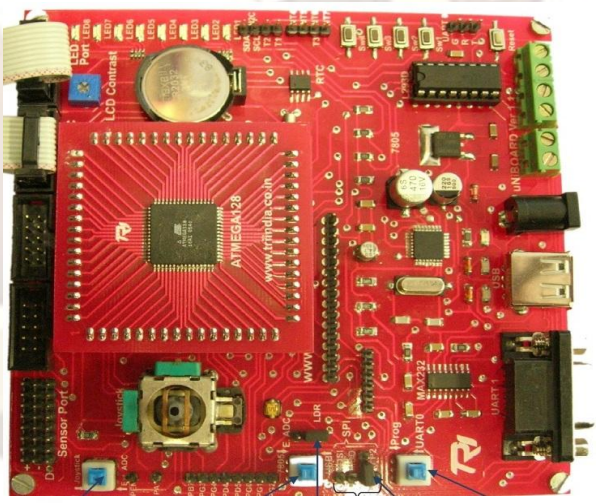

Fig.2 UniBoard

2.6 Real time kernel

Ucos II is a kernel developed by Micrium and Jean Labrosse [2]. Also it is optimized for Linux and Windows operating systems it has a portable kernel

code which generally a code is written in ANSI C which can be ported to any platform fulfilling certain requirements like memory footprint, computational power, number of registers etc. BSP or port files are specific to an architecture, so for every architecture / board there is a unique BSP / port files.
Configuration files can be used to optimize the kernel and reduce the memory footprint of the kernel image based on what features of the kernel would be used by any application program. Finally the application programme is generally written on top of the kernel code and utilizes kernel API's and functions exported by the OS for the users of the RTOS.

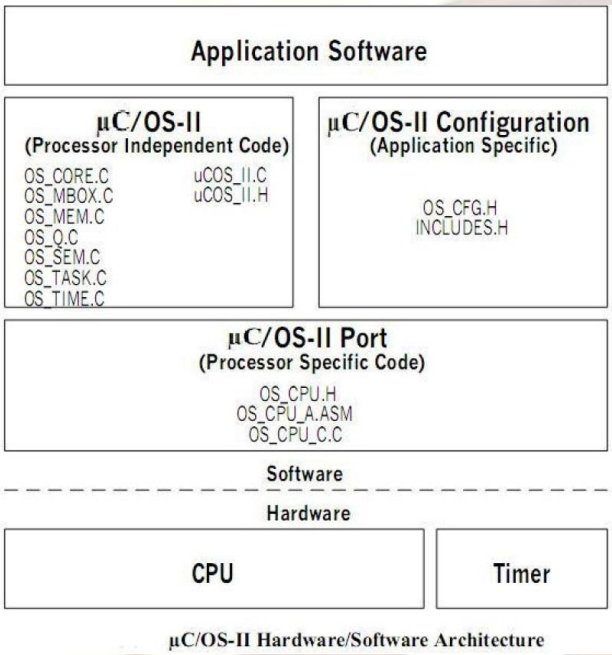

Fig.3 uCos II Architecture

2.7 SPI

There are two programs given for SPI, one for Master and another for Slave. You can communicate between two uNiBoard by connecting wires on SPI lines and making ground common of both the boards. The Output is debugged using the serial terminal whatever character typed on master's serial terminal is transferred to the slave's serial terminal using SPI.

2.8 (RTC)

The DS1307 is the RTC (Real Time Clock) chip interfaced on the I2C lines of the controller. The API's can be used for reading the date and time from RTC. Before reading you need to call this API Update_RTC_variables () which actually updates local variables that data and time API's returns. You can also set date and time for RTC using Write_RTC () API and date and time values for writing to RTC registers are taken from RTC_def_cfg.h.The output can be seen on the UART.

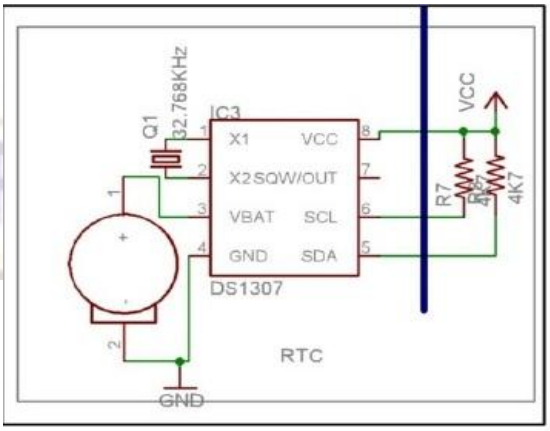

Fig.4 RTC Interfacing

2.9 Joystick and VT library

The Joystick (analog joystick of PS2 fame) is used as a mouse on the serial window in this sample example. The Joystick X axis is connected to the on-chip ADC channel 1 and Y axis to channel 2. The ADC is configured such that it gives an interrupt after the conversion is over. Since only one ADC channel conversion is done at a time so we need to reconfigure it for other channel. The ADC conversion value is stored in ADC's data register. The VT 100 is video terminal made by DEC. It became a de facto standard used by terminal emulators. Digital's first ANSI-compliant terminal, introduced in August 1978. The VT100 was more of architecture than a simple terminal. There are two display formats: 80 columns by 24 lines and 132 columns by 14 lines. A separate advanced video option was required to display 24 lines in 132-column mode. This was
The standard specified for VT102 and VT131.We will use the same VT102 commands for serial terminal. The header file contains API's which uses VT102 standard sequence characters for changing the configurations of the serial terminal. The cursor on the hyper terminal / Gtkterm can be controlled using the joystick and the joystick sensitivity/cursor speed can also be changed by pressing SW4. There are 3 different speeds programmed.

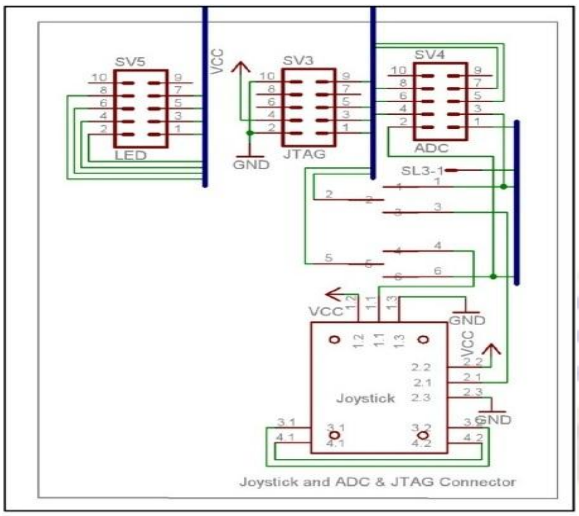

Fig.5 Joystick interface

2.10 USB programmer

USB programmer is available on board which is used to embed the program in the microcontroller and also for providing power.

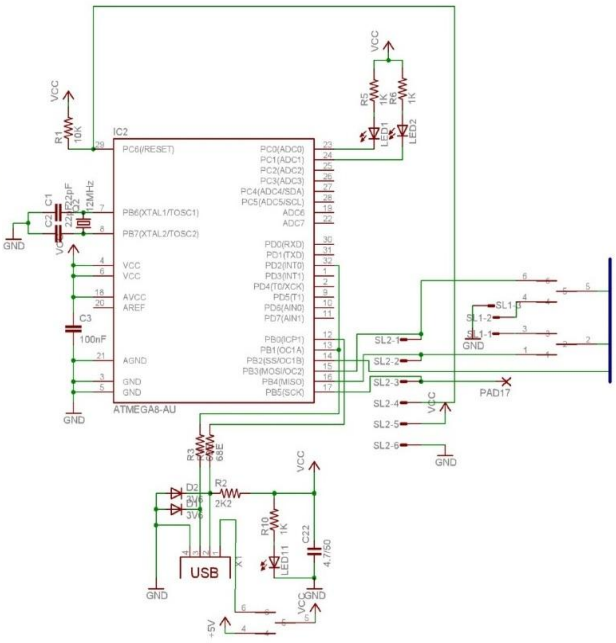

Fig.6 USB Interface

3. Software System

The software part consists of two types of code. The first part of the code was written for the controller board and the second was the code written for gaming.

3.1 Code written for controlling game

We have to configure the board according to our game and write the appropriate code. The code is written in C language [5]. If any interrupts are required for example: if have to exit a game then we can directly press the button which will interrupt the game and exit. Also we can glow the Led's according to the joystick movements. We write the code in AVR Programmers software. We can also configure the accelerometer as per the requirement [5]. The code is written in such a way that according to the movement of joystick or accelerometer we get the corresponding values on HyperTerminal through which we are able to know whether proper synchronizing between the game control and our control joystick and accelerometer has been done.

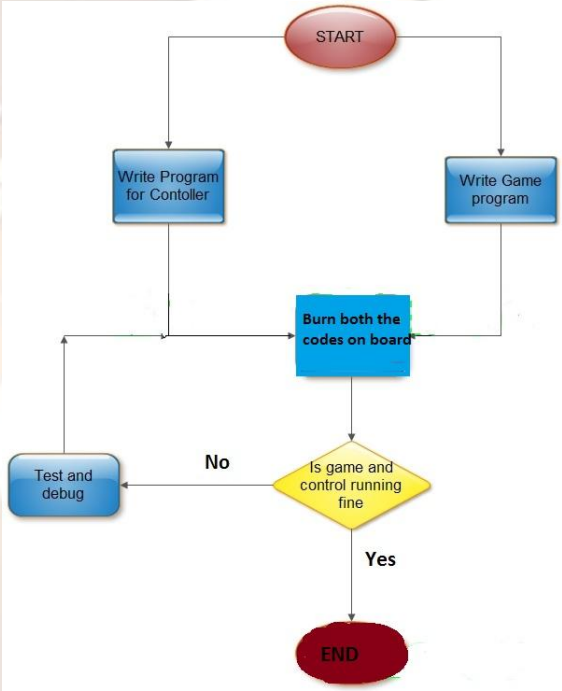

Fig.7 Flow Chart of program flow

3.2 Code written for Game

Using SDL Code Blocks software we can design awesome games for the system [7]. Also integrating the chipmunks' physics engine will further enhance our game by achieving the tasks with optimized codes.

Mainly the game is developed by integrating and combination of various images [6]. By Blitting the image we can make required game. Likewise there is a predefined library of SDL Code Blocks [7]. These can be readily implemented by us. There are various codes as per specific functions. Some of the methods are described below.

- SDL_Init--Initializes SDL.

- SDL_Quit--Shut down SDL.
- SDL_InitSubsystem—initializes the subsystem.
- SDL_VideoDriverName—returns the name of video driver.
- SDL_Flip—swaps screen buffers.
- SetVideoMode—set a video mode with specific height, width, and bits-per-pixel.
- SDL_OpenAudio—opens required audio parameters.
- SDL_LoadWAV-- loads a wav file.
- SDL_JoystickName---returns the joystick name.
- SDL_JoystickIndex---returns the index of joystick.
- SDL_JoystickOpen---opens a joystick for use.
- SDL_JoystickClose---closes a previously opened joystick.

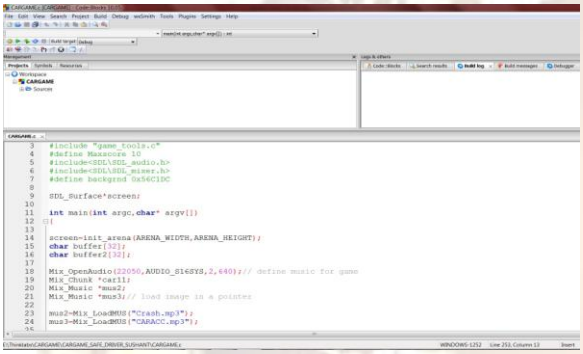

Fig.8 Code written in Code blocks

Fig.9 One of the racing games developed.

3. Conclusion

The process of actual implementation of the project has given us ample exposure to the various advancements in the field of embedded gaming. We have successfully implemented a low-cost, low-power embedded gaming system at the heart of which is an AVR microcontroller. From the very basic design of gaming controllers to the actual synchronization between the game and the controller, every step has strengthened our basic concepts and enhanced our programming skills. Further scope for development includes employment of HMI interface akin to Microsoft Kinect and Nintendo Wii. Wi-Fi enabled gaming controllers may also be designed to provide effortless gaming experience from a distance. Augmented reality, which is still in the nascent stage of its development, may also be incorporated like in Nintendo 3DS and PlayStation Vita to enable gaming enthusiasts to have a real-life feel whilst playing.

ACKNOWLEDGEMENTS

Thinklabs Technosolutions Pvt.Ltd.

References

[1] J. Turley, "Embedded processors," *ExtremeTech, http://www.extremetech.com*, January 2002.

[2] J. J. Labrosse, *Micro C/OS-II: The Real-Time Kernel*. CMP Books, 2002.

[3] "RTLinux RTOS," fSMLabs Inc., URL: http://www.fsmlabs.com.

[4] W. Kim, D. Shin, H. Yun, J. Kim, and S. Min, "Performance comparison of dynamic voltage scaling algorithms for hard real-time systems," in *Proc. 8th IEEE Real-Time and Embedded Technology and Applications Symp.*, San Jose, California, September 2002, pp. 219–228

[5] Muhammad Ali Mazidi, Janice Mazidi, Sarmad Naimi, *Avr Microcontroller and Embedded Systems: Using Assembly and C* (Prentice-Hall, 2010-1-21).

[6] Erik Yuzwa & Francois Dominic Laramee, *Learn C++ by making games* (Thomson, 2010)

[7] Ernest Pazera, Andre Lamothe, *Focus on SDL Game Development*, (Premier Press)

[8] http://lazyfoo.net