

Implementing RTC in Linux Based ARM9 Using I2C Bus

¹V.V. Ashok, ²B. Srinivasulu

¹PG Scholar, Dept of ECE, Malla Reddy Engineering College, India
[An Autonomous Institute]

²associate professor, Malla Reddy Engineering College, India

Abstract: In this paper we describe about designing and development of I2C driver for ARM9 evaluation board running Linux specifically, attention is given to understand the i2c protocol by running a RTC application where master driver namely I2C-S3C2440.C and slave driver namely i2c.dev generic slave driver is used. I2c is an interface between processor and devices. It is a simple and low bandwidth protocol which allows for short distance on board communication, while being extremely modest in its hardware resource requirements. The intended out-come of this work is to give a contribution in under-standing how to access I2C devices on any Linux based embedded systems.

Keywords: i2c overview, master driver, client driver, i2c sub-system, arm9, porting Linux, RTC slave device.

1. INTRODUCTION

A general embedded system may contain one or more microcontrollers and other peripheral devices like sensors, LCD drivers, converters, I/O expanders, memories etc. the connecting cost of all these devices and the complexity combined should be kept minimum. The design of the system should be done in such a way that the slower device doesn't slow down the faster ones when communicating with the system. In order to implement these requirements, we need a serial bus.

A meaning of the bus is the specifications for the addresses, protocols, procedures, formats and connections which define the rules in the bus. The I2C bus will exactly define these specifications.

1.1. I2c Protocol Overview

The Inter-Integrated Circuit, or I2C, is a synchronous master-slave messaging protocol designed to connect a pool of devices by means of a two-wire bus. It is a simple and low-bandwidth protocol which allows for short distance on board communications, while being extremely modest in its hardware resource requirements. The original standard specified a standard clock rate of 100 KHz. Later by updates to this standard produced a faster speed of 400 KHz and a higher speed of 1.7 or 3.4 MHz's. The I2C bus consists of two bi-directional lines, one line for data (SDA) and one for clock (SCL), by means of which a single master device can send information serially to one or more slave devices (Figure 1). To prevent any conflict every device connected to the bus have their own unique address. The standard I2C specifies two different addressing schemas, 7 and 10 bits, allowing at most 128 and 1024 devices connected at the same time.

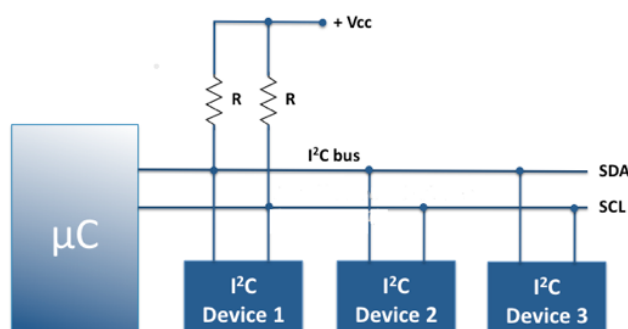


Fig1. Sample I2C Implementation

Each I2C transaction is always initiated by the master who is in charge of the bus for the entire duration of the transaction, meaning that it controls the clock and generates the START and STOP sequences. The start and stop sequences will mark the beginning and the end of a transaction and are the only places where the SDA line is allowed to change while the SCL is high. All data are transferred one byte at a time. In 7-bit addressing mode, the slave address occupies the seven most significant bits of the first byte, with the least significant bit serving as a read/write flag to indicate whether data will be written to the slave ('0') or data will be read from the slave ('1'). For every byte received, the slave device sends back an acknowledge bit. Figure2 shows an example of a typical I2C transaction.

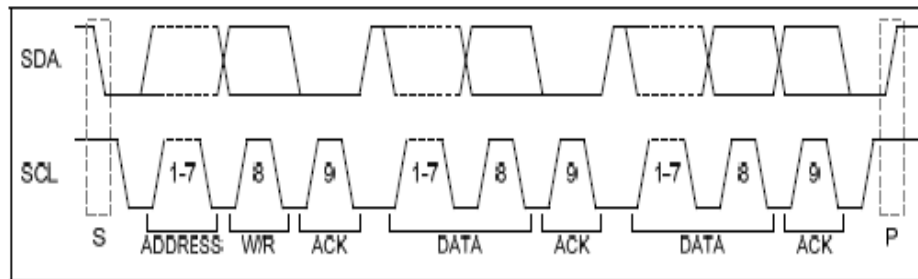


Fig2. I2C Transaction

The I2C protocol supports multiple masters. In a multi master environment two or more masters may simultaneously attempt to initiate a data transfer. In such a scenario, each master must be able to detect a collision and to follow the arbitration logic that leads to the election of a winner master. The winner master can then safely begin its transaction. The friendly arm mini2440 platform, like most system designs, operates in a single-master environment.

1.2. The Linux I2c Subsystem

The Linux kernel I2C framework consists of a core layer where resides all the routines and data structures available to bus drivers and client drivers (Figure 3). The core also provides a level of indirection that allows the underlying drivers to change from one system to another without affecting I2C subsystem that relies on them.

This philosophy of a core layer and its attendant benefits is an example of how Linux helps portability. For instance, enabling I2C on an new platform (which is precisely the objective of this project) requires only to design the hardware-dependent components, namely the bus driver and the client drivers, whereas the core layer needs not to be changed.

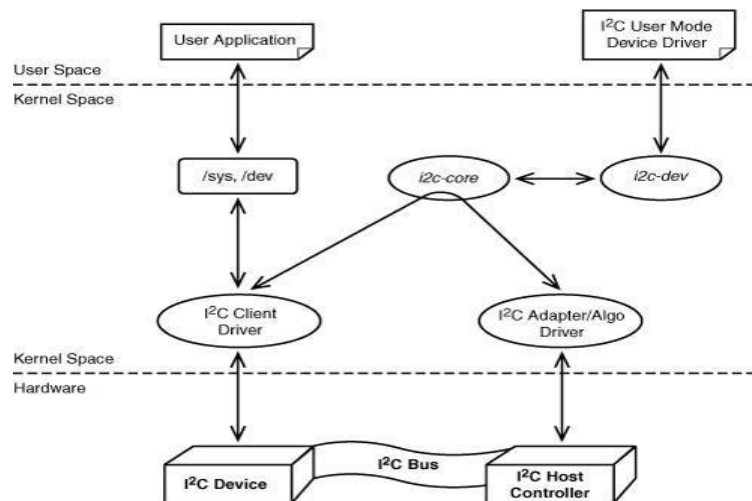


Fig3. Linux I2C Subsystem

1.3. Client Driver

Every hardware device will have a driver. The generic slave driver i2c.dev.c is used as a slave driver in our project. This slave driver is stored in kernel space at the location Linux 2.6.32.2 drivers/i2c/i2c.dev.c this slave driver is used to access the rtc application and calls the master driver through i2c-core.c.

1.4. Master Driver

S3C2440 which is the processor of the board min2440 acts a master as its master driver is I2C-S3C2440.C. This master driver is present at kernel Space at the location Linux 2.6.32.drivers/i2c/buses/i2c-S3C2440.c this initiates the master controller which controls the slave devices i.e., hardware devices through i2c bus.

2. ARM9: FRIENDLY ARM (MINI2440) PLATFORM

ARM9 is an architecture 32-bit RISC CPU family. Mini2440 | S3C2440 ARM9 Board Friendly ARM: Mini 2440 SBC (Single-Board Computer) 400 MHz Samsung S3C2440 ARM9 processor. Samsung ARM9 S3C2440 Module MINI2440 which is an ARM9-based, small form factor, OS Enable module populated with the S3C2440A.

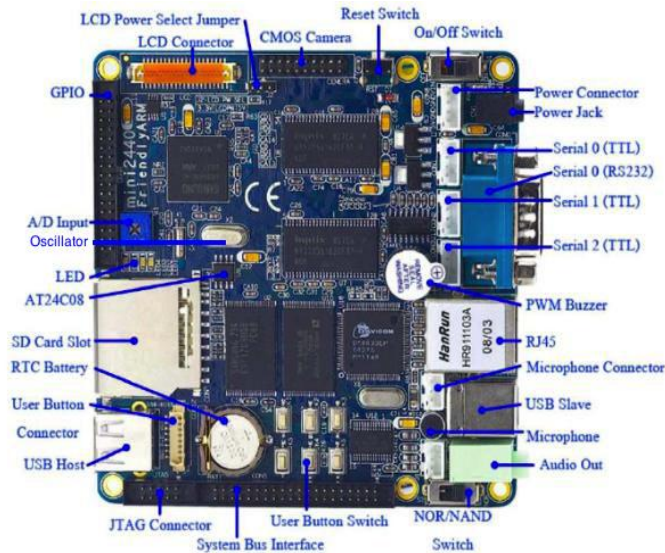


Fig4. Friendly arm mini2440 Board

Rich peripherals such as integrated LCD controller, USB and Ethernet make this module the suitable candidate for embedded applications which require high performance and low power consumption. The MMU on-board supports major operating systems including Windows Embedded CE and Linux. Another chip-level feature includes three UARTs, I2C, SPI, a real-time clock with a separate power domain, DDR memory controllers, and NAND Flash. These features make the devices particularly suitable for industrial control application and automotive as well as medical systems.

2.1. RTC Slave Device

The controller designed controls the RTC device through the I2C protocol. Here the act of a master device is done by I2C controller and control the RTC which act as a slave. The read operation will be accomplished by sending a set of control signals including the address and/or data bits. The control signals should be followed with proper clock signals.

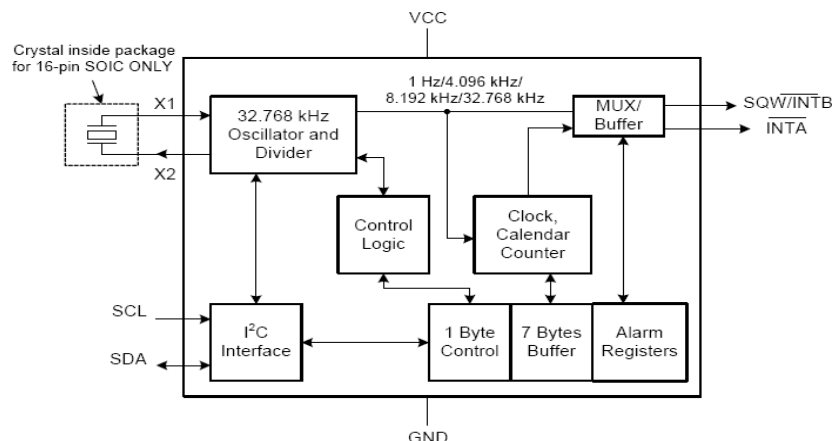


Fig5. Interfacing of I2C-RTC

2.2 RTC Application

In our project we are running an rtc application through i2c protocol. Applications are generally performed or run in user space. Application cannot directly communicate with the hardware devices. We use IOCTL calls to communicate with drivers in kernel space. This application performs read/write operations we compile this application by `$/hwclock.c`. Whenever a driver is loaded it is registered as character driver which can be verified in `/dev/i2c`.

3. PROJECT SETUP

Porting Linux on mini2440:

Step 1: Installing a tool chain at host side.

Step2: Hardware setup.

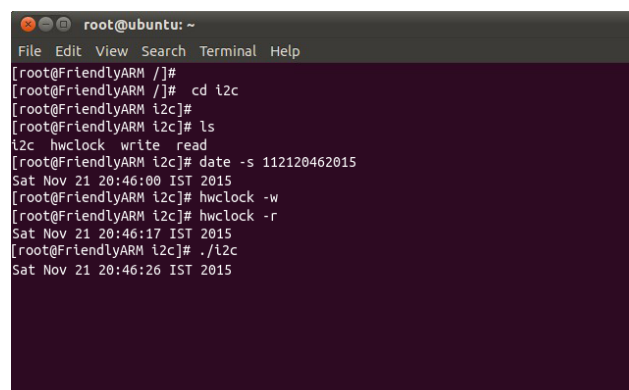
- Connect serial port 0 to PC COM port by a modem cable i.e., straight cable (This Connection is responsible for communication between Host and Target).
- Connect DC 5V power adapter to power supply in.
- Connect with PC by a USB cable (This connection is used for transferring boot loader, Kernel, RootFs images from Host machine to target). Boot mode can be selected by S2, according words on silk screen. S2 connect to Nor Flash side-system will boot from Nor Flash S2 connect to Nand Flash side-system will boot from Nand Flash BIOS which pre-loaded in Nor Flash.

Step3: Installing boot loader (supervivi), Linux image (zImage), RootFs.

Step4: Off the power and change to NAND mode from nor mode. It is the file system on which all the other file systems are mounted (i.e., logically attached to the system) as the system is booted up (i.e., started).

4. RESULT ANALYSIS

- Connect a udisk in which executable files of required application are copied.
- Mount the files in udisk to the directory created on board `[root@friendlyArm/]#Mount /dev/udisk /file name`
- Run the required RTC application file. To connect the Linux clock use `hwclock -w`. To read the data from device use `hwclock -r`. To implement the rtc application through i2c we use `./i2c`.



```

root@ubuntu: ~
File Edit View Search Terminal Help
[root@FriendlyARM /]#
[root@FriendlyARM /]# cd i2c
[root@FriendlyARM i2c]#
[root@FriendlyARM i2c]# ls
i2c hwclock write read
[root@FriendlyARM i2c]# date -s 112120462015
Sat Nov 21 20:46:00 IST 2015
[root@FriendlyARM i2c]# hwclock -w
[root@FriendlyARM i2c]# hwclock -r
Sat Nov 21 20:46:17 IST 2015
[root@FriendlyARM i2c]# ./i2c
Sat Nov 21 20:46:26 IST 2015

```

Fig6. *Displaying the RTC Application*

5. CONCLUSION

This paper has dealt with the I2C device driver design and running an rtc application in a Linux embedded system environment. Due to the tight coupling between these types of drivers and the underlying hardware, usually their development is undertaken by the manufacturer of the embedded system. Therefore, most of the commercial boards ship with a tailored Linux distribution which has all the drivers necessary for the devices on the board. This way the engineers can focus on developing the application specific software rather than building kernel components. Nevertheless, this paper gives an insight into how the Linux kernel supports I2C, making it helpful for those who need to modify an existing implementation to fit their own needs.

REFERENCES

- [1] NXP Semiconductors N.V. 2014. UM10204 “I2C-bus specification and user manual” Rev. 6 — 4 April 2014. User manual pp 1-18, 2014.
- [2] [Bhavani K R, Dr Kurahatti N G, Ravindra Prasad “Implementation and application of i2c device driver in Embedded Linux,” ijitest, volume 1 and issue 5, pp 1-7, 2014.
- [3] Waqar S. Qureshi, Matthew N. Dailey “I2C interface tutorial”, Asian institute of technology, Praxis II, pp 1-15, January 2014.
- [4] Regu Archana, Mr V J Rao “Implementation of I2C master bus protocol on FPGA”, Dept Vignan Institute of Technology & Science Deshmukhi, Hyderabad, Vol. 4, Issue 10 (Version 5), October 2014, pp.06-10.
- [5] Raj Kamal, “Devices and Communication Buses for Devices Network,” in Embedded system: Architecture programming and Design, Shalini Jha Ed. New Delhi, India: Tata McGraw-Hill Education, 2008, pp.160- 165.
- [6] Jordi Cortina Guardia and Ian Grout “Designing and implementing a remote location system for farm animals” university of Limerick, February 2013, pp 1-99.
- [7] Muhammed Ali Mazidi, Janice Gillispie Mazidi, Rolin D. Mcklinlay, “The 8051 Microcontroller and Embedded Systems Using Assembly and C,” 2nd ed., Pearson Prentice Hall, pp. 407-429, 2009.
- [8] Micheal Kerrisck “The Linux programming interface – A Linux and Unix system programming handbook”, William Pollock, Chapter 10, pp 185-210, 2013.

AUTHOR’S BIOGRAPHY



V.V. Ashok received degree in Electronics and Communication Engineering from Sri Krishna Devarayah University, Ananthapur in 2013. He is pursuing his post graduation in Embedded Systems from Malla Reddy Engineering College, Hyderabad.