IMPLEMENTATION OF LINUX BASED UART DEVICE DRIVER

Risma Rajan[1], V. Venkateswarlu[2]

[1]*Final Semester Student, M.Tech. – VLSI Design and Embedded Systems, Dept. of PG Studies, VTU Extension Centre, UTL Technologies Ltd. Bangalore, India*

[2]*Principal and Head of Department, Dept. of PG Studies, VTU Extension Centre, UTL Technologies Ltd. Bangalore, India*

Abstract

This paper deals with design, implementation and testing of device driver for a DUART TL16C2550 peripheral on a MDROADM board. The MDROADM board is used in optical networks for switching the optical signals between the nodes. The MDROADM board is designed with a MPC8308 PowerQUICC II Pro Processor as the main processor and the TL16C2550 peripheral is used for extending the serial interface on the board. The MDROADM board runs an embedded Linux operating system.

The device driver for the DUART TL16C2550 is designed based on the customized interfacing with the MPC8308 PowerQUICC II Pro Processor. The device driver is in compliance with Linux kernel V2.6.29.6

Keywords: *Device Driver, Linux, embedded system, DUART, TL16C2550.*

----------***----------

1. INTRODUCTION

Embedded systems are designed and developed to cater to a specific application and are characterized by its hardware and the software designed for the application[1]. The hardware components consist of a main controller and numerous peripheral integrated based on the system requirements. The software running in the embedded system provides the intelligence to it. The software running in the system coordinates the functioning and usage of the various hardware modules and implements core logic pertaining to the target usage of the system.

Based on the complexity of the system, the software ranges from a non OS based simple application to a well-structured organization of software into operating system and user applications. The operating system consists of the hardware abstraction layer that is dependent on the architecture of the main processor, a kernel that contains various system management modules that manages the resources available on the system and coordinates the execution of the application and device drivers that manages access to various peripherals on the system [1].

Device drivers are an important component in the embedded system software as this layer controls the interaction between the embedded hardware and software [2]. Device driver development is a deeply involved task in an embedded system design as it requires knowledge of the hardware interfacing, memory map, details of the registers of the hardware and its configuration.

This paper, deals with the design, implementation and testing of device driver for a DUART TL16C2550 peripheral on a MDROADM board. The device driver for the DUART TL16C2550 is designed based on Linux kernel V2.6.29.6 and is designed based on the customized interfacing with the MPC8308 PowerQUICC II Pro Processor. The paper is organized as follows. In Section 2, similar type of work carried out in the industry / academy is discussed. Section 3 discusses the type of the embedded system and the goal for this project. In Section 4, the design and implementation of the DUART device driver is discussed. In Section 5, the results obtained from the work are discussed. Section 6, gives the conclusion of the work.

2. RELATED WORK

Customized embedded system based on Linux is popular in the industry and academy.

Linux has become a very popular operating system in the embedded domain, the reasons being - No licensing fee, open source, Reliability, Scalability, Large programmer base, Support, Portability [3]. The source code of the kernel is open source; hence developers can customize the kernel based on their requirements for the target system. Linux is modular and scalable. This means that the kernel can be recompiled to suit to the processor and the devices.

Embedded Linux based system has been used for commercial digital TV system [4]. The advantages of using Embedded Linux in the work are that Linux is open source program, increases cost effectiveness and allows reusability of device drivers and application programs. The role of the Linux kernel for digital TV system is to provide effective resource management in order to support a multi-programming environment. The embedded Linux kernel is modified to suit the system and device drivers are developed for the entire system control.

In Linux, the device driver can be broadly classified as [5]

1. Character driver – for accessing sequential access devices

2. Block driver – for accessing random access devices
3. Network driver – for interacting with the network protocol stack.

A work on network device driver has been done for CS8900A, a 16-bit Ethernet controller for a Linux operating system based on ARM920T processor and implemented on the S3C2410-S development platform [6]. The authors have discussed implementation principle of embedded Linux network drivers and discussed the design for the various driver functions and sections of the code implemented.

3. BACKGROUND AND GOALS

The system referred in this paper is a Multi-Directional Reconfigurable Optical Add/Drop Multiplexer (MDROADM) card that is used for adding and dropping off wavelengths from the optical network. The wavelength that the card can selectively add or drop from the network can be re-configured based on the prevailing network conditions. MDROADM card contains the hardware and software applications that can switch the wavelength based on the runtime configuration. It also contains devices to monitor the network conditions. Several such MDROADM card are positioned in the optical network node and are controller by a master controller. The MDROADM card communicates to the master controller, via a serial bus as shown in Fig-1.

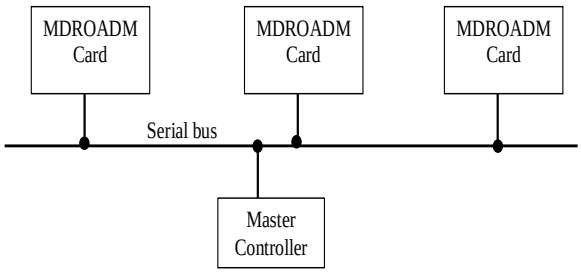

Fig-1: ROADM communication with master controller [7]

The MDROADM card is equipped with a 32 bit MPC8308 PowerQUICC II Pro Processor as the main processor and other components like the wavelength selective switch and optical channel monitor that are interfaced with the main processor. The card also contains a TL16C2550 DUART chip interfaced with the MPC8308 PowerQUICC II Pro Processor to extend the serial interfaces available on the MDROADM card. The MDROADM card communicates with the serial bus and hence the proper functioning of the TL16C2550 DUART chip is critical for the overall functioning of the MDROADM card.

The TL16C2550 DUART chip is interfaced with the MPC8308 PowerQUICC II Pro Processor via the enhanced Local Bus Controller (eLBC), Integrated Programmable Interrupt Controller (IPIC) and General Purpose Input/Output (GPIO) lines of the processor as shown in Fig-2.

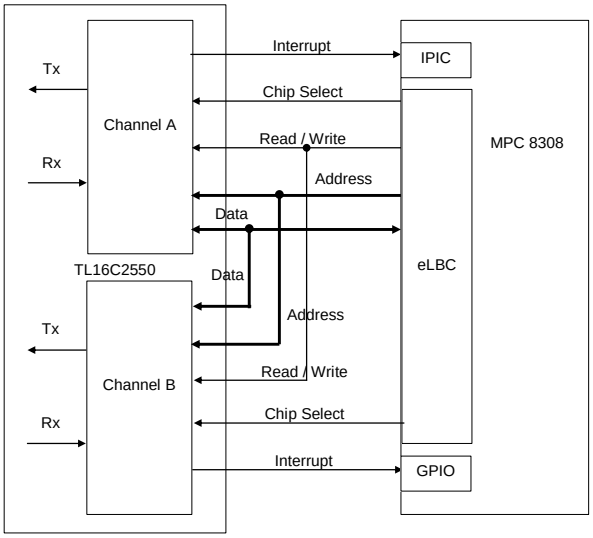

Fig-2: MPC8308 and TL16C2550 interfacing

The MPC8308 PowerQUICC II Pro Processor runs a Linux operating system and the basic configuration of the Linux operating system does not contain device drivers for TL16C2550 DUART chip based on the customized interfacing with the MPC8308 PowerQUICC II Pro Processor. Hence for the applications running on the MDROADM card to communicate with the master controller, a device driver has to be written for TL16C2550 DUART chip based on the customized interfacing.

4. DESIGN AND IMPLEMENTATION OF DUART DEVICE DRIVER

The DUART is a serial device and comes under the category of character drivers. In Linux kernel, there is separate layer called the TTY layer, which is a type of character driver. The TTY layer provides access to serial devices and the DUART device driver hooks to the TTY layer to register with the kernel. The TTY layer is organized into [8]

1. TTY core – this layer controls the interaction with the user space
2. TTY line discipline – This layer formats the data received from the user space and hardware in a specific manner
3. TTY driver - this layer interacts with the low level driver.

The serial core layer provides various interfaces to register the DUART device with the TTY driver layer. The DUART device driver needs to register its functions with the serial core layer so that various operations on the DUART can be performed. The following functions are implemented in the DUART device driver.

4.1 Initialization Function

The kernel calls the initialization function of the DUART device driver to install the driver.

In the initialization, function, the registers of the MPC8308 PowerQUICC II Pro Processor [9] are configured. The registers of the Local Access Windows and eLBC subsystem are configured to enable the access to the memory mapped registers of the TL16C2550 DUART chip. The System Configuration registers are configured to set the pin function of the multi-function pins. The GPIO and IPIC registers are configured to enable the MPC8308 PowerQUICC II Pro Processor to recognize the interrupt of the TL16C2550 DUART chip.

After the register configuration, the DUART device driver registers with the kernel via *uart_register_driver(struct uart_driver *)* interface. The data structure "*struct uart_driver*" contains information about the name, major and minor numbers, and number of ports of this driver. After the device driver is registered, a request to allocate the memory regions, required for communicating with the DUART, is made to the kernel using *request_mem_region(unsigned long start_address, unsigned long range, char *device_name)* and the memory regions are remapped using *ioremap(unsigned long start_address, unsigned long range)*, for accessing the region by the device driver. After the remapping, DUART device driver registers each individual port that it supports by calling the function *uart_add_one_port(struct uart_driver *, struct uart_port *).* The data structure "*struct uart_port*" contains all the configuration data of each of the channel of the DUART.

4.2 Device Operation Functions

The DUART device driver provides the following functions which are invoked by the kernel for the various device operations.

1) startup Function: In the startup function, the mapping for the GPIO interrupt is created and an interrupt handler is registered with the kernel using *request_irq(unsigned int irq, irqreturn_t (*handler)(int irq, void * dev_id), unsigned long flags, const char * device _name, void *dev_id).*

After registering the interrupt handler, the FIFO and the interrupts are cleared, the Line Control register is set with the default parameters of Word Length 8 bits, no parity and 1 stop bit, the interrupt is enabled in the DUART channel and the generation of interrupts for reception of characters is enabled[10].

2) setterminos Function: The settermios function set the DUART channel parameters set by the user. The parameters of DUART channel include the baud rate, word length, whether parity has to be enabled or not and if enabled if it has to be set to even or odd parity and the number of stop bits.

The FIFO is enabled and the various masks for the DUART channel is set based on the termios structure set by the user space.

3) type Function: The type function returns the name of the DUART.

4) shutdown Function: The shutdown function disables the interrupt and the break condition in the DUART channel and deregisters the interrupt handler using *free_irq(unsigned int irq, void *dev_id).*

5) breakctl Function: The breakctl function sets the break condition as requested by the user space.

6) stoprx Function: The stoprx function disables the generation of reception interrupts in the DUART channel.

7) starttx Function: The start function enables the generation of interrupts when the Transmit Holding Register is empty.

8) stoptx Function: The stoptx function disables the generation of interrupts when the Transmit Holding Register is empty.

9) txempty Function: The txempty function checks whether the Transmit Holding Register is empty.

10) interrupt Handler: The interrupt handler handles the interrupt generated by the DUART channel.

If the transmit holding registers empty interrupt is generated, the driver checks if there is data from the user space to be sent out. If the data is available to be transmitted, the data is transferred to the DUART channel FIFO. This process continues till all the data from the user space is transmitted. Once the transmission is complete, the generation of interrupts when the Transmit Holding Register is empty is disabled.

If the receive line interrupt is raised, the error is intimated to the TTY layer via the TTY flip buffer.

If the data ready interrupt is raised, the data is pushed to the TTY flip buffer and is intimated to the TTY layer via the TTY buffer.

4.3 Exit Function

The kernel calls the exit function of the DUART device driver to uninstall the driver.

In the exit function, DUART device driver deregisters each individual port by calling the function *uart_remove_one_port(struct uart_driver *, struct uart_port *).* The memory regions are unmapped using *iounmap(unsigned long mapped_address).* The memory regions allocated for communicating with the DUART is released using *release_mem_region(unsigned long start_address, unsigned long range)* and the DUART device driver deregisters with the kernel via *uart_unregister_driver(struct uart_driver *)* interface. The registers of the MPC8308 PowerQUICC II Pro Processor are re-configured to disable the GPIO interrupts.

5. TESTING AND RESULTS

The device driver is tested using the setup shown in Fig-3.

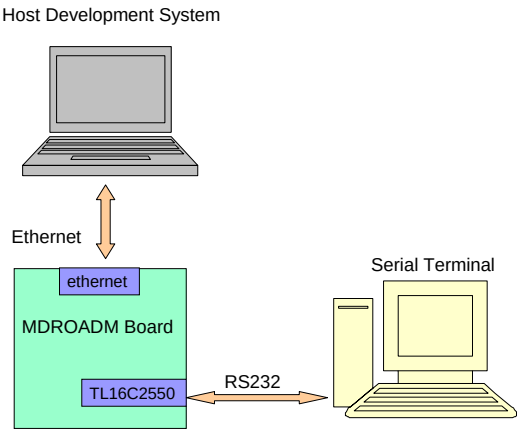

Fig-3: Development and Testing Environment

There are two connections to be done with the MDROADM board. These are

1) An RS-232 connection via serial cable between the TL16C2550 DUART chip and the serial terminal system
2) An Ethernet connection via cross cable between the on-chip Ethernet controller on the MPC8308 processor and host development system.

The host development system is used for developing the device driver code and downloading the device driver binary to the MDROADM board via the Ethernet connection.

The device driver code is implemented in C language, as per the design, in the host development system. The code is compiled in the host development system using the Linux target image builder (LTIB). On successful compilation, a .ko file is generated. This binary file of the device driver is loaded onto the MDROADM board. The driver is installed in the MDROADM board.

A test application is written and loaded onto the MDROADM board to test the implemented device driver. The application opens the device node of the TL16C2550 DUART, sets the communication parameters, receives some test data transmitted from the serial terminal system through the TL16C2550 DUART and also transmits data to the serial terminal system via the TL16C2550 DUART. The test application sends a data stream 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7, 0xF8, 0xF9, 0xFA, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7, 0xF8, 0xF9, 0xFA, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5.

Fig-4 and Fig-5 shows the snap shots of the serial terminal application that captures the data transmitted from the application via the TL16C2550 DUART. The DUART parameter is set at 9600 bit/sec, Word Length 8, no parity, 1 stop bit. The 8 bits of the data is received at the terminal. Hence the received data is - 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7, 0xF8, 0xF9, 0xFA, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7, 0xF8, 0xF9, 0xFA, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5 – as expected.

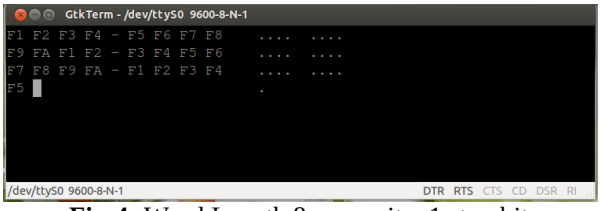

Fig-4: Word Length 8, no parity, 1 stop bit

When the Word Length is set to 6, the last 6 bits starting from the LSB is transmitted as shown in Fig-5.

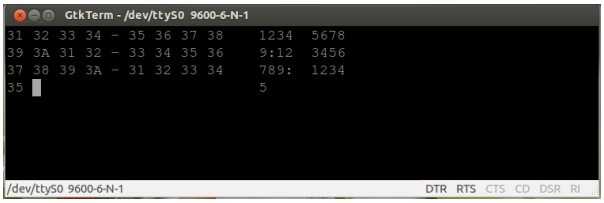

Fig-5: Word Length 6, no parity, 1 stop bit

As only the last 6 bits are transmitted the received characters are in the series of 0x3x. Hence the received data is - 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x3A, 0x31, 0x32, 0x33, 0x34, 0x35 – as expected.

Table -1 shows the results of the tests conducted with non-matching parameter at the serial terminal and TL16C2550 DUART. TL16C2550 DUART is set with the parameter – 9600 bit/sec, Word Length 8, Even Parity, 2 stop bit.

Table -1: Non matching parameter test results

Sl. No.	Parameter set at Serial Terminal				Result
	Number of Data Bits	Parity Enabled	Parity Type	Number of Stop bits	
1	8	No	-	1	Framing Error
2	8	No	-	2	Correct Data
3	8	Yes	Even	1	Correct Data
4	8	Yes	Even	2	Correct Data
5	8	Yes	Odd	1	Parity Error
6	8	Yes	Odd	2	Parity error

A string “ab” is sent from the serial terminal. The Table -1 list the error detected for the first character. The expected parity bit for character “a” will evaluate to ‘1’.

In case 1, the stop bit of the first character is treated as parity bit and the start bit of the next character is treated as stop bit. Hence, stop bit is corrupted and framing error is detected. In case 2, the stop bit of the first character is

treated as parity bit and the second stop bit of the first character is treated as the stop bit. When stop bits are configured as 2 in the receiver side, only the first received stop bit is validated. Hence in this case, the stop bit is treated as valid and the data received is interpreted as correct. In the case 3, the second stop bit of the first character is treated as the stop bit and hence no error is detected.

The case 4, in Table -1, has matching parameters and hence the data received is proper. The case 5, 6, there is a mismatch in parity and parity error is detected.

6. CONCLUSIONS

In this paper, a device driver for TL16C2550 DUART chip that is interfaced with the MPC8308 processor via eLBC, IPIC and GPIO lines is designed and implemented. The device driver is able to successfully register with Linux kernel and the DUART device is made visible to the user space application. Also the device driver is verified to be able to read and write to the various registers properly. The device driver is able to set the parameters of the TL16C2550 DUART – baud rate, word bits, parity and number of stop bits – properly and hence enable the DUART for transmitting and receiving the data properly. The TL16C2550 DUART is able to detect the errors when non-matching parameter is set in its transmitter/ receiver counterpart. The appropriate error flags are set in the TTY layer for the detected error. Hence, it is ensured that the communication link between the MDROADM and the master controller is reliable.

ACKNOWLEDGEMENTS

The authors would like to thank Mr. Shaik Saifulla (UTL Technologies Ltd.) and Dr. Siva Yellampalli (Dept. of PG Studies, VTU Extension Centre, UTL Technologies Ltd.) for their guidance and feedback to the work.

REFERENCES

[1] Noergaard. Tammy, *Embedded Systems Architecture - A Comprehensive Guide for Engineers and Programmers*, Har/Cdr edition, Elsevier Inc, 2005

[2] Shibu KV, *Introduction to Embedded Systems*, 1st Ed, Tata McGraw Hill, 2011.

[3] A. Lennon, "Embedding Linux", *IEE Review*, vol. 47, no. 3, pp.33 -37, May 2001.

[4] Sang-Pil Moon, Joo-Won Kim, Kuk-Ho Bae, Jae-Cheon Lee and Dae-Wha Seo, "Embedded Linux Implementation on a Commercial Digital TV System", *IEEE Transactions on Consumer Electronics*, Vol. 49, No. 4, pp. 1402 – 1407, November 2003.

[5] Jonathan Korbet et al., *Linux Device Drivers*, 3 rd ed., O'Reilly 7 Media Inc., 2005.

[6] Yongxiang Guo, Wu Deng, "Design of Network device driver in embedded Linux", *IEEE International Conference on Computer Application and System Modeling*, ICCASM 2010, pp.V12-445 - V12-448.

[7] Luis Velasco, Salvatore Spadaro, Jaume Comellas, Gabriel Junyent, "ROADM design for OMS-DPRing protection in GMPLS-based optical networks", *IEEE 6th International Workshop on Design and Reliable Communication Networks, 2007,*. DRCN 2007, pp.1 - 7.

[8] P. Raghavan , Amol Lad, Sriram Neelakandan, *Embedded Linux system design and development*, Auerbach Publications, 2006.

[9] Freescale Semiconductor Inc., *MPC8308 PowerQUICC II Pro Processor Reference Manual*, Rev 1, 2013.

[10] Texas Instruments Inc, *TL16C2550 data sheet*, 2012.

BIOGRAPHIES

Risma Rajan is pursuing her M. Tech. in VLSI Design and Embedded Systems at VTU Extension Centre, UTL Technologies Limited, Bangalore. She has around 8.5 years of experience in embedded domain and is currently working at Tata Consultancy Services Limited, Bangalore.

Dr. V. Venkateswarlu received his B.E. degree in Electrical & Communication Engineering, M.E. degree in Electronics and Ph.D. degree in Semiconductor Devices from the Indian Institute of Science, Bangalore. He has around 30 years of industry experience and is currently the Principal and Head of Department at VTU Extension Centre, UTL Technologies Limited, Bangalore.