µC/OS-II Real Time Kernel Port for Cirrus Logic EP93xx Platform

Eugen DODIU[1], Adrian GRAUR[2], Cristina N. GAITAN[3], Vasile G. GAITAN[4], Adrian M. GAITAN[5]
Stefan cel Mare University of Suceava
13,University Street, RO-720229 Suceava
[1]*edodiu@usv.ro*, [2]*adriang@eed.usv.ro*, [3]*gaitan@eed.usv.ro*,
[4]*cristinag@eed.usv.ro*, [5]*agaitan@stud.usv.ro*

***Abstract* — Real-time systems are a key element for applications where deadlines must be satisfied. The absence of a time constraint in a hard real-time system can cause severe material damage or even life threatening scenarios. This is why the system designer's task is to make proper selection of an embedded operating system that can meet these demands.**

***Index Terms* — µC/OS-II, hard real-time/soft real-time, embedded system, EP9302, real-time scheduling, operating system**

I. INTRODUCTION

This article presents the practical steps toward running Micriµm's µC/OS-II real time kernel on a Cirrus Logic EP9302 processor.

Real-time systems are a bit different from the classical ones because they have to offer a certain response within a specified time period. In the case of real-time systems, correct execution of tasks will depend not only on the correctness of the results, but also on the time instance they are started. Unlike soft real-time where deadline miss is not a major problem, missing a time constraint in a hard real-time system can lead to physical damage [1][1][2][2].

All these problems are satisfied with µC/OS-II as we could see on other tested hardware architectures. Without any doubt, µC/OS-II is a very powerful product, since it was certified by the Federal Aviation Administration to meet the requirements of the RTCA DO-178B standard for software used in avionics equipment. Our previous testing of this operating system on some embedded architectures (ARM 7, HCS12) here at Stefan cel Mare University of Suceava, proved that µC/OS-II is the best choice for real-time applications. This OS was born back in 1992, when engineer Jean Labrosse, future founder of Micriµm Technologies Corporation, began working on a real-time operating system that was needed to satisfy strict timing requirements of one of his projects. The latest stable version is 286 and can be found on Micriµm's website along with the latest version, µC/OS-III.

Applications such as cameras, avionics, high-end audio equipment, engine controls, medical equipment, industrial machines, have been using µC/OS-II for a long time now with great success [2][2][1].

Latest distributions allow integration with other software packages such as µC/TCP-IP, µC/GUI, µC/File System, µC/USB, µC/CAN, µC/Modbus, µC/Bluetooth, for obtaining greater scalability and performance [2][2][5].

It is important to mention just a few characteristics of this real time kernel to observe its key features:

- ROM-able. This OS was designed for embedded systems and if using adequate development tools it can be embedded as a part of the final product.
- Scalable. The total amount of memory or the memory footprint can be modified from a configuration file accordingly to the hardware restraints.
- Portable. The way the software structure is built allows easy porting to other architectures.
- Preemptive. This means that µC/OS-II will always run the highest priority task that is ready to be executed.
- Deterministic. User knows how much CPU time is spent for µC/OS-II system function execution.
- Interrupt Management. The kernel can manage interrupts with up to 255 levels deep.
- Tasks Stacks. This feature allows using separate stacks with different sizes for each task thus permitting better footprint management.

All the above characteristics conclude that µC/OS-II is a well built, robust and reliable operating system that can be used in real-time embedded systems.

II. HARDWARE ARCHITECTURE

The test system uses Technologic System's TS7300 development board [3][3]. This board was initially sold with Linux preinstalled, but we decided to use it for µC/OS-II porting and testing, since the hardware platform allowed this software change.

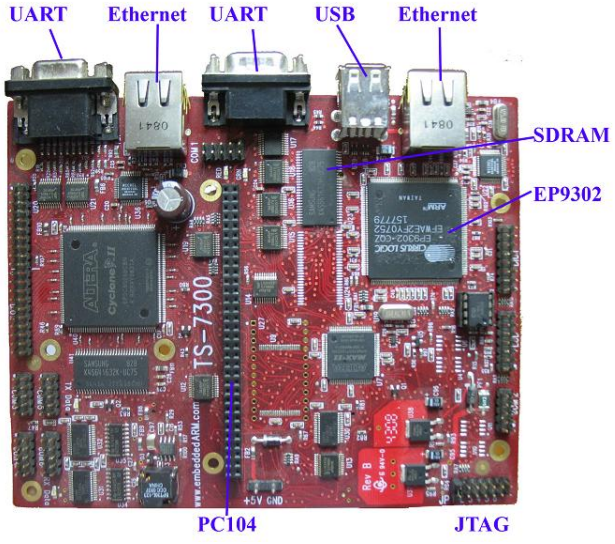

Figure 1. Hardware organization of the test board [3][3].

Fig. 1 presents the hardware architecture of the TS-7300 test board [3][3][3]:

- EP9302 Cirrus Logic Processor (ARM920T @ 200MHz)
- 32 Mbytes Samsung SDRAM K4S561632H
- Peripherials: IO ports, USB port, JTAG, CAN, USART, power connectors
- Real time clock
- Altera CycloneII FPGA for application development
- Altera MAX II CPLD companion chip
- SD card storage
- PC104 expansion slot
- RS232 drivers
- Serial FLASH memory for Linux bootloader.

Featuring a five stage pipeline consisting of fetch, decode, execute, memory and write stages, the ARM920T 32 bit architecture, delivers impressive performance with power consumption under 2 Watts [3]. Trial results showed that this ARM920T processor can produce impressive throughput with over 309 Mbytes/s at 200MHz core clock using block transfer *STMIA* instructions. ARM920T has the following characteristics: ARM and Thumb instructions, 32 bit Advanced Micro-Controller Bus Architecture, 16 Kbyte instruction and 16 Kbyte data cache, MMU for operating systems, TLB with 64 entries for data and instructions, programmable page sizes, independent lockdown for TLB entries. Activating all these characteristics can lead to very good raw performance of the controller.

There are a few integrated circuits that are not directly addressable by the EP9302 processor. In most of the cases this is done using the Altera MAX II glue-logic companion chip. The documentation of the board shows the mapped addresses in the ARM9 physical address space. SD cards are also accessed via this companion chip.

III. PROGRAMMING AND BOOTING PROCESS

Cirrus Logic designed the EP9302 processor with multiple ways of booting. The JTAG connector has a dual function: it controls both the programming via JTAG for the CPLD devices and the boot mode of the EP9302 processor. This connector cannot be used for EP9302 JTAG programming since the JTAG pins of this device are not welded to it. If someone decides to perform a JTAG programming of the EP9302 processor on this test board there are a few operations that must completed. The tester must weld the JTAG_DIN (pin 78), JTAG_DOUT (pin 79), JTAG_CLK (pin 77), JTAG_TMS (pin 80) to the JTAG connector. Since the processor has no on-chip flash the user has to make an initialization file of the SDRAM chip that will be executed prior to any application code. Only after the on-board Samsung memory will be initialized, the application code will be downloaded and executed from the SDRAM. This sequence works fine with IAR Embedded Workbench version 5.40.1 and JLink programmer with hardware version 5.3.

As mentioned before the JTAG connector controls the booting method of the processor (Fig.2).

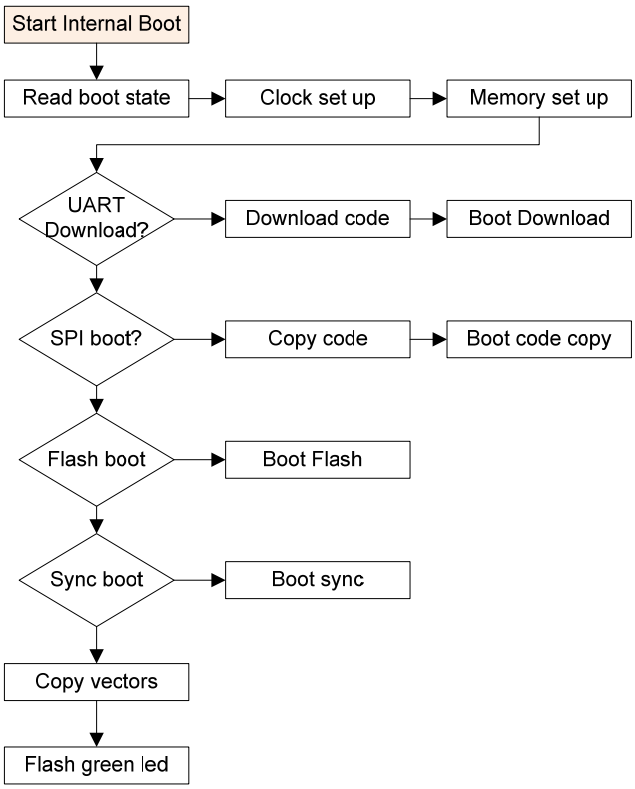

Figure 2. Logical evolution of the booting algorithm. [6] [6][6]

Our intention was not to modify the contents of the 25160 EEPROM SPI memory that boots Linux from the compact flash. This is why we decided to use the UART programming method. The absence of the EP9302 on-chip flash memory implies reprogramming of the device every time a power off-on cycle is completed. Jumper 1 from the JTAG connector selects the loading via UART or SPI. In our case, the presence of a jumper on the first pair of pins specifies that a serial download will be initiated. A number of 2048 characters are expected to be received via the UART serial port by the on chip boot ROM. These characters will be placed in the receive Ethernet buffer that starts at address 0x80014000.

Since the 2K program received in buffer space is not enough to load and run the µC/OS-II kernel and application tasks, we had to build our own boot loader whose only basic function is the loading of the binary file in the SDRAM. The Boot ROM utility leaves the UART 1 serial port configured at 9600 baud, 1 bit stop, no parity, 8 data bits and opened for incoming data. After receiving the "< " the second bootloader is sent to the system using a simple terminal application that has the possibility of sending data files. Even if the effective code of the second loader is less then 2K, IAR Embedded Workbench will fill the remaining space till 0xFFFF with 0x00. When the second bootloader starts, the first action is to disable the external watchdog timer that is built using the MAXII CPLD. This is done by first writing the feed register at address 0x23C00000 and then the configuration register at adress 0x23800000. If these actions are not performed, the board will reset peridocally at 8 seconds. Since we want to use the full computing power of the ARM core, the caches are enabled by writing to CP15 registers with dedicated MRC and MCR instructions. This task is fulfilled when boosting the speed of the core up to 200MHz using the following PLL parameters:

PLL1_X1FBD = 21, PLL1_X2FBD = 31, PLL1_X2IPD = 24, P2L1_PS = 1. This gives an output frequency of 199.987200 MHz. The HCLK is POUT/2 and PCLK is HCLK/2. Writing this configuration word in the apropriate PLL config register folowed by 5 NOPs will restart the core at its desired frequecy. The user can check if the PLL is locked by reading a dedicated register. Next we decided to raise the speed of the UART to 115200 for obtaining better programing times. By doing so, a 64K image files is downloaded in memory in less then 6 seconds. Before jumping to the reception loop the SAMSUNG SDRAM memory initializatin is performed. This is done in the following steps:

1 -insert 200us start-up delay

2 -load EP9302 memory controller config register with correct values: RAS-TO-CAS latency 3, CAS delay 3, 4 banks , 16 bit data width

3 - 200us delay is inserted

4 -GLConfig issue NOP commands

5 - 200us delay is inserted

6 - GLConfig – Precharge all comand

7 - Errata for E2 revision requires reading any adress of all 4 banks of the memory in order to make the precharge all command work. This is done by reading from adress 0x00000000,0x00200000,0x00400000 and 0x00600000.

8 -Refresh register timer is loaded with value 0xA

9 -20us delay is inserted

10 –GLConfig Mode register select

11 –*Write* configuration word with CAS = 2 (010), BL = 4 (010) offset 9 by *reading* from adress 0x00006600.

12 - Go to normal mode by writing the configuration register.

The reception is done by polling the FIFO full status flag. When the 16 byte depth FIFO is full, the processor copies all the data from the receive stack into the memory respecting the little endian organisation. When a 64K boundary is reached a jump to adress 0x00000000 is done using a branch instruction. We had a few problems since the memory chip U23 doesn't have the correct notation on the board. Instead of being mapped to domain 3 as the electrical schematic tells, our memory chip is adressed with SD_CS3 and is mapped to domain 1 that starts at adress 0x00000000. It is probably that notations are swapped, but this is a minor issue.

Table 1 shows the partial mapping of the Samsung K4S561632H memory in domain 1.

TABLE I. ADRESS MAPING FOR SAMSUNG SDRAM

Bank no.	Adress range
1	0x0000_0000 - 0x003F_FFFF
1	0x0100_0000 - 0x013F_FFFF
2	0x0400_0000 - 0x043F_FFFF
2	0x0500_0000 - 0x053F_FFFF
3	0x0800_0000 - 0x083F_FFFF
3	0x0900_0000 - 0x093F_FFFF
4	0x0C00_0000- 0x0C3F_FFFF
4	0x0D00_0000 - 0x0D3F_FFFF

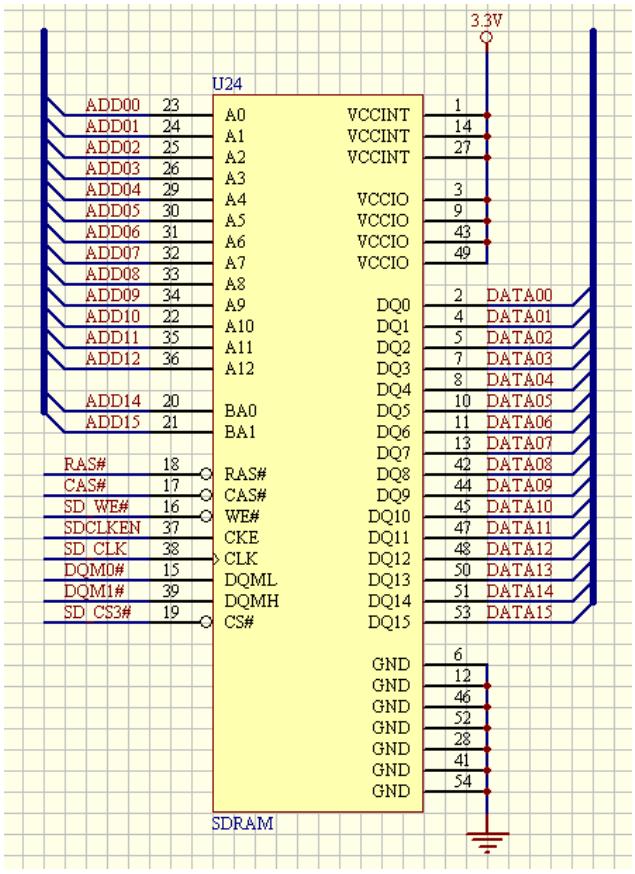

Figure 3. Partial mapping of the Samsung K4S561632H SDRAM memory.

If the first memory bank is used alone, the hardware allows booting images of up to 4 Mbytes footprint. As shown in Table 1, the SDRAM memory domain is not a contiguous space and this depends on the way the ARM core addresses the external devices. In a Samsung K4S561632H SDRAM, memory the organization is as follows: 13 address lines are used for ROW addressing, 9 address lines are used for column addressing and 2 lines for bank selection. EP9302 has the following internal address decoding scheme: A1-A8, A24, A25 for column selection, A9-A22 row selection, A26, A27 bank selection[6]. For this hardware configuration the following address decoding is used: A1-A8, A24 for column selection, A9-A21 row selection, A26, A27 for bank selection. EP9302 memory controller allows swapping of bank selection lines A26 and A27 with A21 and A22. By doing so, the address space will only be divided in 4 banks, as can be seen in Table 2. Each bank has now 8 Mbytes in size and the address space is less divided. Fig. 4 shows the internal structure of the K4S561632H-TC/L75 Samsung memory, including row, column and bank addressing logic.

TABLE II. ADRESS MAPING WITH SROMLL BIT SET TO 1.

Bank no.	Adress range
1	0x0000_0000 - 0x007F_FFFF
2	0x0100_0000 - 0x017F_FFFF
3	0x0400_0000 - 0x047F_FFFF
4	0x0500_0000 - 0x057F_FFFF

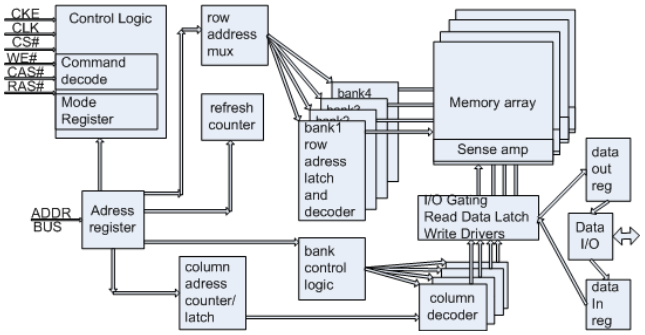

Figure 4. SDRAM Functional Block Diagram.

IV. SOFTWARE CONFIGURATION

The second bootloader starts downloading into the memory from address 0x00000000. The memory domain 1 to which U24 is attached, starts from this address. Since all available memory from the board is SDRAM, both code and data segments will be placed in RAM starting from address 0x00 as can be seen in Listing 1.

```
define symbol __ICFEDIT_intvec_start__ = 0x0;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x80;
define symbol __ICFEDIT_region_ROM_end__   = 0x7FFF;
define symbol __ICFEDIT_region_RAM_start__ = 0x8000;
define symbol __ICFEDIT_region_RAM_end__   = 0xFFFF;
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__   = 0x400;
define symbol __ICFEDIT_size_svcstack__ = 0x400;
define symbol __ICFEDIT_size_irqstack__ = 0x200;
define symbol __ICFEDIT_size_fiqstack__ = 0x200;
define symbol __ICFEDIT_size_undstack__ = 0x200;
define symbol __ICFEDIT_size_abtstack__ = 0x200;
define symbol __ICFEDIT_size_heap__     = 0x2000;
```

Listing 1. ROM, RAM and stack memory usage.

IAR Embedded Workbench version 5.40.1 and 286 µC/OS-II source files were used for testing. Execution of the µC/OS-II binary image starts with the exception handlers and stack initialization sequence that are coded in file *cstartup.s*. Control is next passed to the main function that contains the initialization of the system, *AppStartTask* creation function and the function that actually starts the system: *OSStart()*. Task switching is assured by configuring the timer 1 to generate periodic interrupts that launch the scheduler. We set it to trigger with 1KHz frequency. This value must be in correlation with the *OS_TICKS_PER_SEC* value used in the *os_cfg.h* file. The starting task is also responsible with interrupt configuration and system timer starting. Since we expect to have full performance from this system, the Vectored Interrupt Controller is set to execute the functions that are for exception handler processing.

V. CONCLUSION

The paper presents the basic steps that need to be performed for loading µC/OS-II on TS7300 development board. This includes PLL configuration, cache activation and writing a bootloader that copies the image sent serially to the SDRAM. For testing and development purpose the UART serial download has been used. Using this system in an application requires adding a bigger EEPROM memory that can host the entire application code. An alternative option can be the redesigning of hardware, so that stand-alone flash memories can be used.

Even though Technologic Systems TS7300 was initially developed for use with Linux, we found that its real time computing power comes to life when using true RT operating systems such as µC/OS-II. Practical results of task switching times, interrupt latencies and jitter will be presented in a future paper.

ACKNOWLEDGMENT

We would like to thank Mr. Jean Labrosse for helping us with porting µC/OS-II to Analog Devices ADuC7026 back in 2005 and for optimizing system performance and memory footprint for a few of our educational projects.

REFERENCES

[1] Cottet F., Delacroix J., Kaiser C., Mammeri Z., "Scheduling In Real-Time Systems, " John Wiley & Sons Ltd, England, 2002, pp 1-41.
[2] J. Labrosse, "MicroC/OS-II The Real Time Kernel ", 2nd Ed, CMP Books, 2002, pp. 20-150.
[3] Technologic Systems, TS7300 Manual Hardware and Software Revision 1.5 Jul 2008
[4] Technologic Systems, TS7300 Schematic 1 May 2006
[5] http://www.micrium.com
[6] Cirrus Logic, EP93xx User Guide, September 2007
[7] IAR Embedded Workbench, ARM IAR Assembler- Reference Guide
[8] IAR Embedded Workbench, IDE User Guide
[9] http://www.cirrus.com EP9302 Rev E2 Silicon
[10] Samsung Electronics, 256Mb H-die SDRAM Specification, Revision 1.0, October 2005