International Journal of Applied Engineering Research ISSN 0973-4562 Volume 10, Number 14 (2015) pp 34464-34467

Scalable and Reconfigurable Kernel Configurations For Embedded System

R.Elavarasi[1], Dr. T.Jayasingh[2], Dr. S.Ravi[3] and Dr.M.Anand[4]
[1]Ph.D Scholar, ECE Department, Dr. M.G.R. Educational and Research Institute, Chennai
Email:elammece@gmail.com
[2]Supervisor, ECE Department, Dr. M.G.R. Educational and Research Institute, Chennai
[3]Professor&Head, ECE Department, Dr. M.G.R. Educational and Research Institute, Chennai
[4]Professor, ECE Department, Dr. M.G.R. Educational and Research Institute, Chennai

Abstract- The aim of this paper is to implement a basic core with are configurable user space and device drivers. Reconfigurable architectures will perform computational tasks optimally with unique capabilities and guarantees the performance and energy efficiency in hardware. To achieve scalability, the reconfigurable core is a better way without re fabricating integrated circuits. Their functionality can be upgraded or modified and specialized to the particular instance of a task. In this paper, the field of reconfigurable computing, compute models, runtime configuration and applications in the context of configurable kernels for embedded targets are surveyed. It provides the benefits of scheduling in a reconfigurable environment as more complex applications can be mapped for a given size and more efficient schedulers can be achieved.

Keywords- busy box, kernel configuration, device drivers, S3C2440 ARM core

1. Introduction

The aim of this paper is to implement a basic core with a reconfigurable user space and device drivers. This approach is tested on ARM core and is applicable to other cores as well. Implementation of reconfigurability is done using two different methods and includes

1. User space and
2. Kernel level dynamism

The tradeoff in the level of reconfiguration is

i. Total memory size of the image file representing the file space and kernel in hardware
ii. Scalability

User space dynamism

A modern computer operating system usually segregates virtual memory into kernel space and user space. Primarily, this separation serves to protect data and functionality from faults (by improving fault tolerance) and malicious behavior. The term user space refers to all code which runs outside the operating system's kernel. User space usually refers to the various programs and libraries that the operating system uses to interact with the kernel: software that performs input/output, manipulates file system objects, application software etc. Each user space process normally runs in its own virtual memory space, and, unless explicitly allowed, cannot access the memory of other processes. This is the basis for memory protection in today's mainstream operating systems, and a building block for privilege separation.

Kernel level Dynamism

In computing, the kernel is a computer program that manages I/O requests from software, and translates them into data processing instructions for the central processing unit and other electronic components of a computer as shown in Fig 1. The kernel is a fundamental part of a modern computers system. The critical code of the kernel is usually loaded into a protected area of memory which prevents it from being overwritten by other, less frequently used parts of the operation system of by applications. The kernel performs its tasks such as writing text in a text editor or running programs in a GUI, is done in user space. This separation is made in order to prevent user data and kernel data from interfering with each other and thereby diminishing performance or causing the system to become unstable.

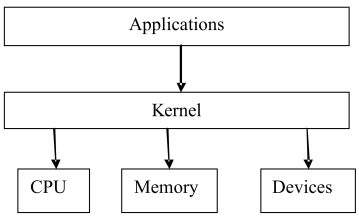

Fig 1. Functions of Kernel

Post booting target setup involves the following: The relevant commands are also listed:

To specify the environment setup on the target

1. setenvbootcmd 'nand read 0x30200000 0x500000 0x300000; bootm 0x30200000'
2. saveenv

To create tftp server so as to transfer the image file representing the configuration of userspace and kernel into the target. Alternate loading schemes include (i)x-modem (ii)serial port, etc.

3. tftp 0x30200000 zImage.img
4. nand erase 0x500000 0x300000
5. nand write 0x30200000 0x500000 0x300000

To specify the path where the network file system i.e. nfs exists, its ip address, target hardware ip address and subnet/gateway details.

6. setenvbootargsinit=/linuxrc root=/dev/nfsnfsroot=192.168.0.245:/home/ravi/nfs/rootfsip=192.168.0.235:192.168.0.245:192.168.0.201:255.255.255.0 console=ttySAC0 115200 display=lcd480
7. saveenv

To summarize, the sequence includes, building the core utilities, Configuring the bmenuconfig, Using make command and customizing bmenuconfig, Configuring the kmenuconfig, Using make command and customizing kmenuconfig, creating z image, setting up minicom/tftp (or) z-modem and transferring image to target board

2. Literature Review

Alisson V.Brito, Matthias Kuehnle(2007) proposed a partial and dynamic reconfigurable processor developed using system c kernel. The modeling and simulation is done at the higher level hardware and embedded software. A novel reconfigurable architecture is developed by jianfeng, lioboliu(2014) to execute control intensive kernels efficiently. The Architecture applies three key designer methods. Mapping the applications on to reconfigurable arrays is a challenging task due to their parallel execution and sparse interconnection topology. Giovanni ansaloni and Laurapozzi(2011) presented a scheduling frame work that is able to efficiently map operations on reconfigurable arrays. Anupam Chattopadhaya, Xialin Chen (2008) presented Coarse – Grained Reconfigurable Architecture (CGRA), a strongly emerging class, is currently receiving due attention for offering excellent performance as well as flexibility post fabrication. Elena Suvorova, Nadezhda Matveeva presented a paper on methods of dynamically reconfigurable multi core system on chip (SOC) Additionally the proposed method is developed used FPGA and ASIC technologies. Mechanism for system level modeling of the dynamically reconfigurable Networks on Chip (NOC) implemented on the ASIC technology is also discussed.

Yi Wang, Jussipekka Leiwo and Thambipillai Srikanthan proposed an algorithm ported to FPGA ranging from 128 to 2048 for a bit width. The result shows bit width 2048 with k=1 significant speed up in the computation can be realized. It is a promising way to improve performance significantly by adding reconfigurable processing unit to a general purpose processors. Like Yan, Binbin Wu, Yuan Wen (2010) proposed a Reconfigurable multi core architecture combining general multi core and reconfigurable logic.

3. Fixed Core Configuration

Busy Box is software that provides several stripped-down Unix tools in a single executable file. It runs in a variety of POSIX environment such as Linux, Android and others, such as proprietary kernel, although many of the tools it provides are designed to work with interfaces provided by the Linux kernel. It was specifically created for embedded operating systems with very limited resources. Busy Box can be customized to provide a subset of large utilities. It can provide most of the utilities specified in the Single Unix Specification (SUS) in addition to many others that a user would expect to see on a Linux system. Typical computer programs have a separate binary (executable) file for each application. Busy Box is a single binary, which is a conglomerate of many applications, each of which can be accessed by calling the single BusyBox binary with various names (supported by having a symbolic link or hard link for each different name) in a specific manner (Ref Figure 2.).This busy box includes multiple options for both user space and Kernel (Device Drivers).

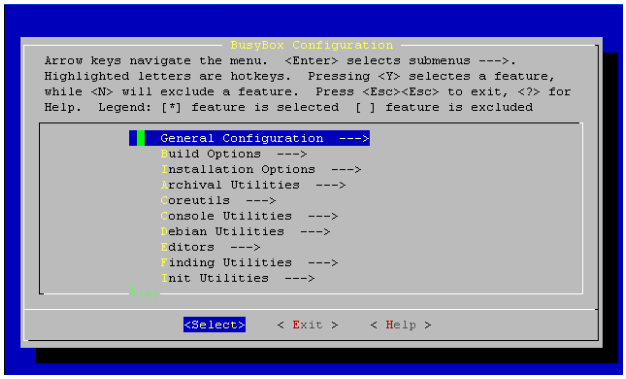

Fig 2.BusyBox configuration

4. Dynamic Device Drivers in Linux

The devices in LINIX fall in two categories- **Character devices** and **Block devices**. Character devices can be compared to normal files in that we can read/write arbitrary bytes at a time (although for most part, seeking is not supported).They work with a stream of bytes. Block devices, on the other hand, operate on blocks of data, not arbitrary bytes. Usual block size is 512 bytes or larger powers of two. However, block devices can be accessed the same was as character devices, the driver does the block management. Another category is networking devices. The interface provided by these drivers in entirely different from that of char/block devices. In LINUX all the devices are represented as files. Both character devices and block devices are represented by respective files in the /dev directory. It is possible to read and write into the device by manipulating those file using standard system calls like open, read, write, close,etc. Example: To directly see the contents of the RAM by reading /dev/mem.

Every device file represented in this manner is associated with the device driver of that device which is actually responsible for interacting with the device on behalf of the user request. Thus, when accessing a device file, the request is forwarded to the respective device driver which does the processing and returns the result. While reading device files, a particular function in the device driver registered for the file is invoked and the respective data is returned.

5. To Identify Specific Drivers

Each device and its device file has associated with it, a unique **Major number** and a **Minor number**. No two devices have the same major number. When a device file is opened, Linux examines its major number and forwards the call to the driver registered for that device. Subsequent calls for read/write/close too are processed by the same driver. As far as kernel is concerned, only major number is important. Minor number is used to identify the specific device instance if the driver controls more than one device of a type. The major & minor number of devices, can be identified using the **ls – l** command from /dev directory.

Ex:
crw-rw-rw- 1 root root 1, 8
Here c represents that it is a character device. 1, 8 represent major and minor number.
1 is the major number and 8 is the minor number.

6. Configurable Device Drivers

The syntax of register_chrdev is **intregister_chrdev (unsigned in t major, const char *name, structfile_operations *fops)**. Driver is unregistered by calling the unregister_chrdev function. Since device driver is a kernel module, it should implement init_module and cleanup_module functions. The register_chrdev call is done in the init_module function andunregister_chrdev call is done in the cleanup_module function. The register_chrdev call returns a non-negative number on success. If we specify the Major number as 0, the kernel returns a Major number unique at that instant which can be used to create a device file. A device file can be created either before the driver is loaded if we know the major and minor number beforehand or it can be created later after letting the driver specify a major number.

7. Configuring kernel functions

Apart from those, the driver must also define certain **callback functions** that would be invoked when file operations are done on the device file. It must define functions that would be invoked by the kernel when a process uses open, read, write, close system calls on the file. Every driver must implement functions for processing these requests. When register_chrdev call is done, the fourth argument is a structure that contains the addresses of these callback functions, callbacks for open, read, write, close system calls. The structure is of the type file_operations and has 4 main fields that should be set **read, write, open and release**. Each field must be assigned an address of a function that would be called when open, read, write, close system calls are called respectively. For egIn a Camera driver, the following operations defined in file_operations structure.static struct file_operations cam_ops =

```
{
        .open = v4l_cam_open,
        .release = v4l_cam_release,
        .read = v4l_cam_read,
        .ioctl = v4l_cam_ioctl,
        .poll = v4l_cam_poll,
        .mmap = v4l_cam_mmap,
};
```

v4l_cam_open function is called when the device is opened using the linux open system call.v4l_cam_read function is executed to read data from the camera.v4l_cam_ioctl function is executed ,to do some ioctl operation on the device file (camera). IOCTL is Input output control and is used to change various parameters of the device file. For Example: To get the capture size or to get or set the color palette we use this IOCTL call is used. Similarly other functions in the file_operations structure translate to different OS File operations. It is necessary in the driver file to define module_exit(s3c2440_camif_exit);module_init(s3c2440_camif_initialize);Module_Exit has the function that will be called when the driver is unloaded and module_init has the function that will be called when the driver is loaded.

Need for reconfigurable user space / Kernel

Advantage of lesser options in busy box .
Let' M' be the scalable options and 'N' be the total possible
Three metrics memory, resources available and scalability are studied.
The effect of M on the optimizing metrics is shown in **Figure 3a, 3b and 3c**

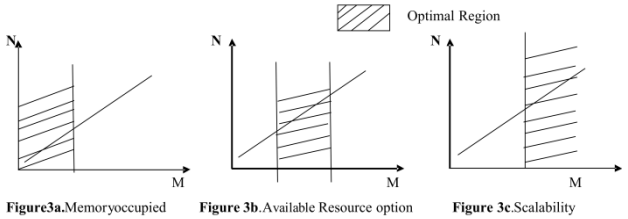

Figure3a. Memoryoccupied **Figure 3b**.Available Resource option **Figure 3c**.Scalability

From figures 3a to 3c, it can be inferred that the desired threshold level i.e. Region of operation is variable. In order to simulate dynamic and partially reconfigurable systems, the simulator must perform some specialized operations. A partially reconfiguration simulator should perform basic operations such insertion and removal of modules into and from running systems. These operations are the basis for all possible operations to perform dynamic and partial reconfiguration.

8. Meta Data Handling

Meta data includes size, time of inclusion, time of modification etc. Metadata is data about a data. There are two types of metadata: structural metadata and descriptive metadata. Structural metadata is data about the containers of data. Descriptive metadata uses individual instances of application data or the data content.

Metadata was traditionally used in the card catalogs of libraries. As information has become increasingly digital, metadata is also used to describe digital data using metadata standards specific to a particular discipline. By describing the contents and context of data files, the usefulness of the original data/files is greatly increased.
Metadata is defined as the data providing information about one or more aspects of the data, such as:

- Means of creation of the data
- Purpose of the data
- Time and date of creation
- Creator or author of the data
- Location on a computer network where the data was created
- Standards used

For example, a digital image may include metadata that describe how large the picture is, the color depth, the image resolution, when the image was created, and other data. A text document's metadata may contain information about how long the document is, author affiliation, birth date of document, a short summary of the documented etc.,

9. Conclusion

Reconfigurable kernels and user space with macro like enabling/disabling options has become an alternative to fixed microprocessors with increasing performances. If customization and configuration support is dynamic, then the RTOS – reconfigurable architecture is possible. Complex applications are still needed to understand the strength and weakness of the Reconfigurable system. In future it is possible to implement an efficient reconfigurable RTOS's. Then self – adaptation to evolving requirements of applications during their life time is possible. Also it would be possible to analyze the system resources and dynamic behavior at run time, enabling faster and more intelligent decisions. These features will be used by different dynamically reconfigurable platforms.

References

[1] Alisson V. Brito, Matthias Kuehnle, "A General Purpose Partially Reconfigurable Processor Simulator (PReProS)" proceedings on the conference 2007.

[2] Jianfeng Zhu, Leibo Liu, Shouyi Yin, Xiao Yang, and Shaojun Wei "A Hybrid Reconfigurable Architecture and Design Methods Aiming at Control-Intensive Kernels IEEE Transactions On Very Large Scale Integration (Vlsi) Systems" 2014.

[3] DevisTuia, Gustavo Camps-Valls, GionaMatasci, and Mikhail Kanevski "Learning Relevant Image Features With Multiple-Kernel Classification" IEEE Transactions On Geoscience And Remote Sensing, vol. 48, no. 10, october 2010.

[4] Giovanni Ansaloniand Laura Pozzi, "Slack-aware scheduling on Coarse Grained Reconfigurable Arrays" proceedings on the conference 2011

[5] AnupamChattopadhyay, Xiaolin Chen, Harold Ishebabi, Rainer Leupers, GerdAscheid, Heinrich Meyr Integrated Signal Processing Systems, RWTH Aachen University 52056 Aachen, Germany, "High-level Modelling and Exploration of Coarse-grained Re-configurable Architectures" in 2008

[6] Elena Suvorova, Nadezhda Matveeva, Alexey Rabin, ValentinRozanov Saint-Petersburg State University of Aerospace Instrumentation Saint-Petersburg, Russian Federation System Level Modeling of Dynamic, "Reconfigurable System-on-Chip" proceeding on the conference of fruct association.

[7] Yi Wang, JussipekkaLeiwo and Thambipillai Srikanthan Center for High Performance Embedded Systems School of Computer Engineering Nanyang Technological University Singapore, "Efficient High Radix Modular Multiplication for High-speed Computing in Re-configurable Hardware" 2005.

[8] Like Yan, Binbin Wu and Yuan Wen, College of Computer Science,Zhejiang University

Hangzhou, PRC,"A reconfigurable processor architecture combining multi-core and reconfigurable processing unit" IEEE International Conference on Computer and Information Technology 2010.

[9] Jianfeng Zhu, Leibo Liu, *Member, IEEE*, Shouyi Yin, *Member, IEEE, "*A Hybrid Reconfigurable Architecture and Design Methods Aiming at Control-Intensive Kernels", IEEE Transactions On Very Large Scale Integration (VLSI) Systems 2010.

[10] L. Liu et al., "An energy-efficient coarse-grained dynamically reconfigurable fabric for multiple-standard video decoding applications," in Proc. IEEE Custom Integr. Circuits Conf. Sep. 2013,

[11] C. Mei et al., "Exploration of full HD media decoding on a software defined radio baseband processor," IEEE Trans. Signal Process. Sep. 2013.

[12] B. Scholkopf and A. J. Smola, Learning With Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. Cambridge, MA, USA: MIT Press, 2001.

[13] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least angle regression," Ann. Statist., vol. 32, no. 2, pp. 407–499, 2004.

[14] Y. Chen, N. M. Nasrabadi, and T. D. Tran, "Hyperspectral image classification via kernel sparse representation," IEEE Trans. Geosci. Remote Sens., vol. 51, no. 1, pp. 217–231, Jan. 2013.

[15] G. R. G. Lanckriet, N. Cristianini, P. L. Bartlett, L. E. Ghaoui, and M. I. Jordan, "Learning the kernel matrix with semidefinite program-ming,"J. Mach. Learn. Res., vol. 5, pp. 27–72, Jan. 2004

[16] M. Lu, C. L. P. Chen, J. Huo, and X. Wang, "Optimization of combined kernel function for SVM based on large margin learning theory," inProc. IEEE Int. Conf. Syst., Man Cybern., 2008, pp. 353–358.

[17] S. Chen, X. Hong, and C. J. Harris, "Probability density estimation with tunable kernels using orthogonal forward regression," IEEE Trans. Syst., Man, Cybern. B, Cybern., vol. 40, no. 4, pp. 1101–1114, Aug. 2009.