DESIGN OF AN SDR BASED RAKE RECEIVER

Craig Dolwin (Toshiba Research Europe Limited, Bristol, UK;
craig.dolwin@toshiba-trel.com)

ABSTRACT

This paper presents a design study for a flexible rake receiver using techniques and concepts suitable for a Software Defined Radio (SDR). This specific design combines software and hardware accelerators to create a basic rake receiver that could potentially support multiple standards (e.g. WCDMA, cdma2000) as well as allowing high level configuration software to trade resources to match a dynamically changing propagation channel.

By encapsulating both software and hardware functions using Real Time Operating System (RTOS) tasks and channels as well as developing a common message based control interface we have demonstrated how components in SDR can be controlled and linked independent of the underlying physical architecture. Furthermore to evaluate the effect on power consumption we have analysed the increase in overhead due to the additional control code

1. INTRODUCTION

1.1. Motivation

In the last fifteen years the mobile phone has continuously evolved. This evolution has been driven by a demand for improved spectral efficiency, higher quality audio, extra services and increased flexibility. Each time the wireless terminal went through another stage of evolution the hardware and software associated with the signal processing stage had to undergo a substantial redesign [1]. This continuous redesign of the signal processing stages is particularly difficult due to its direct impact on power consumption and tight timing constraints. This has resulted in a slow and expensive development cycle. To address this problem researchers are currently looking at software and hardware architectures that support dynamic reconfiguration to provide multiple implementations [2]. This type of system is known as Software Defined Radio (SDR). The primary issue that we must address when proposing an architecture for SDR is the power consumption. It is clear that for SDR to be successful it must operate at equivalent or lower power consumption levels when compared against its non-reconfigurable predecessor.

1.2. Background

This paper describes the real time implementation of a flexible rake receiver. The rake receiver is a key component in a Code Division Multiple Access (CDMA) receiver and has traditionally been implemented in hardware. An all software solution is possible [3][4][5] and potentially offers the maximum flexibility but requires the processor and data bus to operate at higher clock frequencies than a hardware solution that can use multiple processing elements and interconnects. A circuit running at higher clock frequency will have to operate at a higher supply voltage and will therefore have higher power consumption [6]. In addition the amount of MIPS required to implement all the processing at the IQ sample level is well beyond current technology [7]. So in this design we have used a combination of software and hardware to try and achieve the flexibility of software while maintaining the low power consumption of a hardware solution. The rake receiver was chosen because it was seen as a complex real-time problem that could benefit from the extra flexibility offered by applying the ideas and concepts associated with SDR. By analysing a system that has complex real-time interactions between hardware and software modules we hope to gain significant insight into future SDR design issues.

1.3. Software Defined Radio

At its most flexible, Software Defined Radio will allow a processing module to be downloaded into the terminal via the wireless link and then integrated into the rest of the system. Typically these modules might be a complete receiver chain or elements of the transceiver (e.g. an improved channel decoder). As might be expected these modules will be made up of both software and hardware. In a system that supports reconfigurable logic the hardware configuration may also be downloaded.

One of the key challenges for implementing SDR will be how to integrate a new module into the rest of the system. It is made especially difficult because it cannot be assumed that the combination of modules can have been anticipated. This problem is similar to that of applications running on a Personal Computer except in the case of SDR the performance of each module must be guaranteed. This

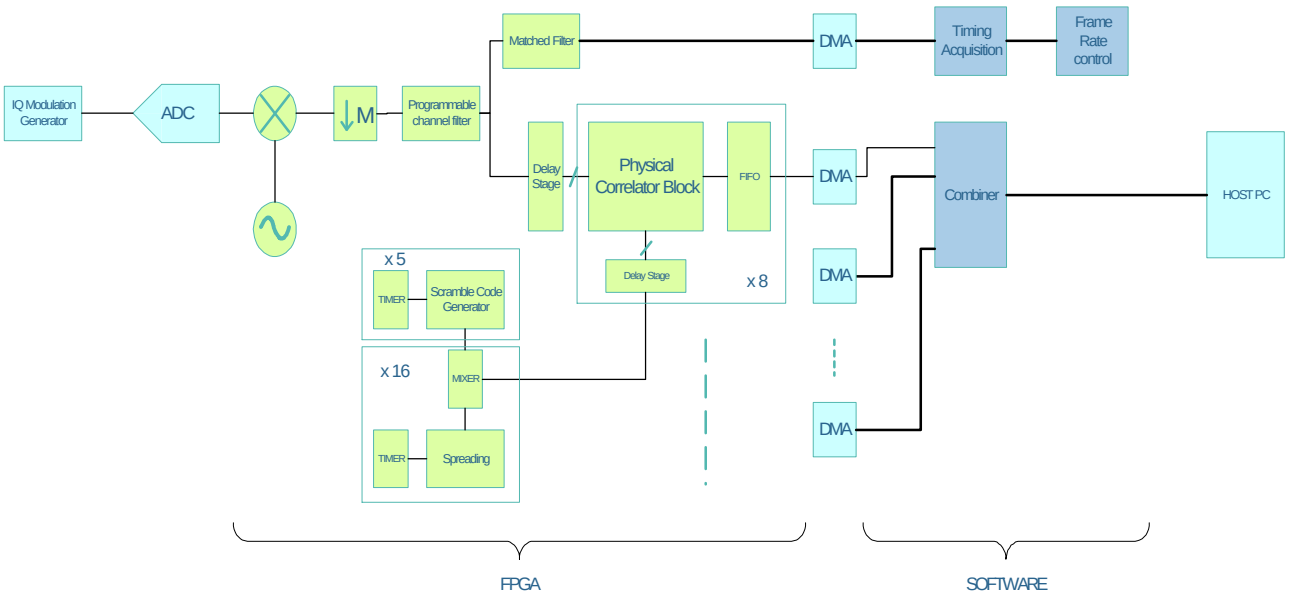

Figure1 Simplified block diagram of the rake receiver

timing deadlines [8] otherwise data may be lost or the complete system may crash. In addition, and equally importantly, the implementation must also be very power efficient.

2. OVERVIEW

By using an object orientated approach to analyse the rake receiver we isolated a number of objects that had clear boundaries and shared very little common data. The processing requirements for each of these objects were then analysed to see if they would be best implemented in hardware or software. This analysis required a trade-off between flexibility and power consumption so this involved a certain amount of speculative design work to evaluate the difference in complexity between the hardware and software solution. The result of this work was to split the system so all the chip level processing was done in hardware and the symbol level processing was done in software. In practice this meant that the software processed blocks of symbols rather than symbol by symbol as this reduced the overhead due to interrupt service routines.

The hardware is split into the following modules (Figure 1)

- Scrambling code generator
- Spreading code generator
- Partial discriminator
- Matched filter
- Signal Conditioning

To allow maximum flexibility, each module is connected via programmable switches.

An important aspect of SDR is the encapsulation of real-time functions so they are isolated from the rest of the system. In a single threaded system this is relatively easily achieved by using function calls. The detail of how that function is implemented is hidden by a standard interface but such a call will only return when the operation has been completed. So if the function was implemented using a hardware accelerator the processor may sit idle waiting for its completion. By allowing the system to support multiple threads and developing a message-based command interface we allow the processor to execute other threads while it waits for hardware to complete. We then use services such as pipes or channels to route data between tasks see Figure 2 and [9].

It is anticipated that future systems [10] will include a combination of multiple processors and hardware accelerators so it is important that the control messages, mailboxes and data channels work transparently across different processors as well as between threads on a single processor. This feature should be supported by the RTOS.

3. DESCRIPTION

In Figure 1 we show a block diagram of the complete system but to simplify the diagram we have not shown the multiplexing of signals between each set of blocks.

Within the rake receiver hardware there are 5 scramble code generators, 16 Spreading code generators and 8 physical correlators. The scramble code generators can produce an arbitrary Pseudo Noise (PN) sequence compatible with WCDMA, cdma2000 and any other

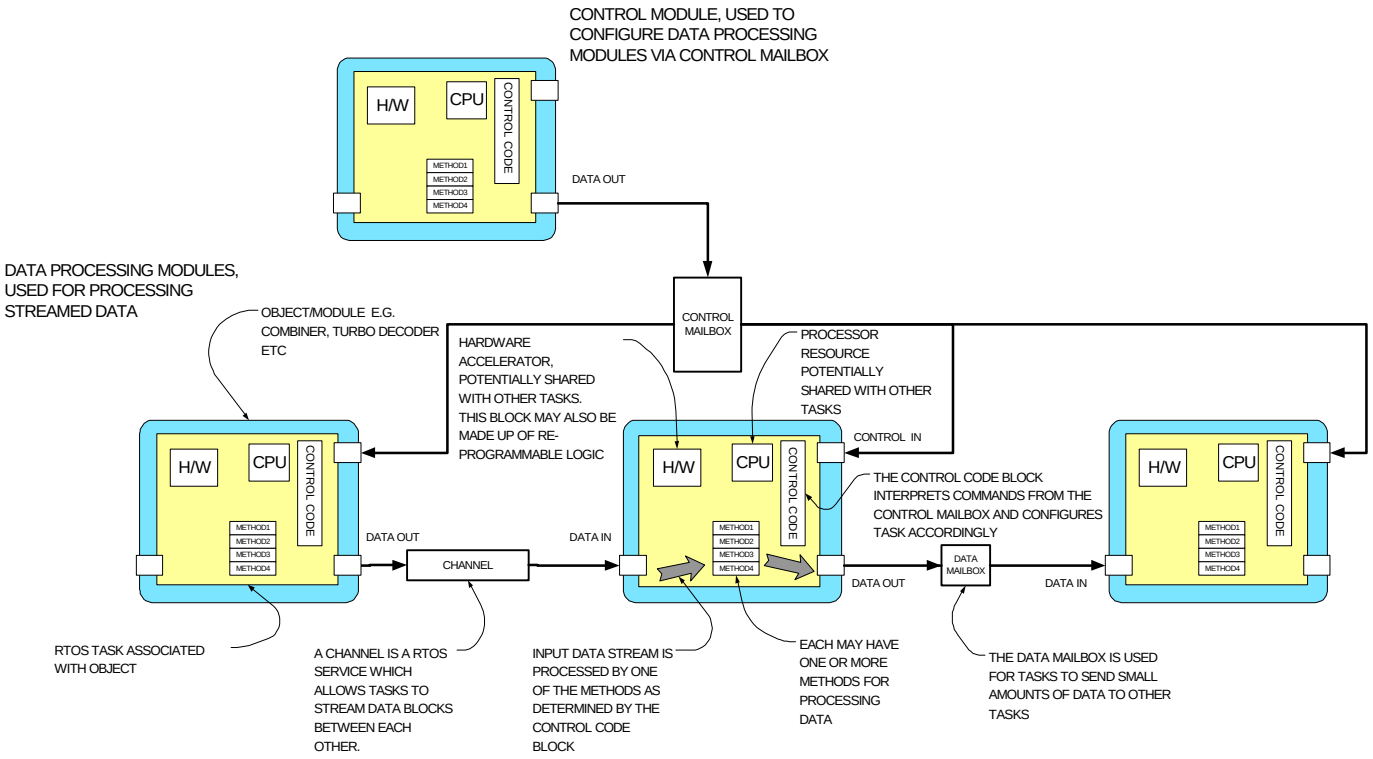

Figure 2, Diagram showing encapsulation of data processing modules

similar system. In addition the absolute timing of each PN sequence can be set to an accuracy of one chip.

The spreading code generator includes a Walsh code generator and mixing logic to combine the Walsh code and scramble code. Each spreading code generator can select any of the scramble code generators as its input.

The physical correlator block operates at 16 x chip rate and implements 16 partial logical correlators. A partial correlation is the calculation of either the real or imaginary component of a complex correlation. A logical correlator can select one of two spreading code generators as its input and can be configured to take IQ samples at a timing offset of $1/8^{th}$ of the chip period. In this implementation a rake finger is made up of a single data discriminator. A data discriminator is physically implemented by using two logical correlators configured to calculate the imaginary and real component of the correlation between the selected PN sequence and the input IQ signal. This means that a single physical correlator can implement 8 rake fingers and so the complete system could implement 64 fingers**.**

The matched filter is designed to detect the primary Synchronisation Channel (SCH) burst in the WCDMA synchronisation channel [11]. The timing acquisition module uses the output from the matched filter to determine the timing of the scramble code and spreading code generators [12]. In many other implementations the Early-Late Delay Locked Loop (DLL) method is used to track paths [13] but the matched filter approach is used because it was better suited for implementation on this platform. In addition it also allows us to easily upgrade the system to support adaptive algorithms. It is envisaged that these adaptive algorithms will increase or reduce the number of fingers depending on the type of channel environment the terminal is working in [14].

The output from the timing acquisition module is used by the frame rate control task to determine when the hardware is reprogrammed. This precise timing is important to ensure that changes in the PN phases are gradual and do not cause glitches at the output of the combiner. The combiner takes the complex output from each selected rake finger and a weighting coefficient from a channel estimator (not implemented in this design) to calculate a soft symbol value. The output from the combiner can then be streamed via a RTOS channel to other modules in the receive path or on to a file on the host PC.

Figure 2 shows how RTOS services were used to encapsulate the major functional modules within the system. A function may be implemented in a hardware accelerator or as a function call in the processor but by using a RTOS task as a wrapper for the function we can force a common interface to the rest of the system. The wrapper must supply two types of interfaces, control and data. The control interface is implemented using a mailbox and a messaging scheme. A wrapper task will poll the

mailbox after each operation to see if it has a control message. If the task is idle it will wait until a control message is sent. On receiving a message it will then configure the associated hardware or call the appropriate software function to execute the operation requested by the message. A task maybe instructed to repeat an operation until told to stop or may only execute the operation a fixed number of times.

The data interface is either hardwired as in the case between two hardware modules or can be implemented as a RTOS channel. Using a channel gives the flexibility to locate the modules on different processors without modifying the wrapper or data processing function.

In this design the DSP has to support 10 threads or tasks. To determine which task has access to the processor the RTOS will choose the task with the highest priority from a list of all pending tasks. The setting of individual task priorities is done at design time and is static. Many methods are available for assigning priorities to tasks [15][16][8] but most of these algorithms will only guarantee that a set of timing deadlines will be achieved when certain constraints are met. In this design we assigned higher priorities to task that repeat most frequently. This is known as rate monotonic scheduling [15]. This scheduling scheme requires that all tasks are independent, support pre-emption and are periodic. In addition to assigning priorities to the tasks we also had to prioritise access to the data bus. Without this lower priority tasks would block higher priority tasks by initiating data transfers that could not be pre-empted. Prioritising access to the data bus is achieved by assigning different Direct Memory Access (DMA) channels depending on the priority of the task requesting the data transfer. Each DMA channel in the DSP has a different priority so higher priority DMA transfers will pause lower priority transfers.

3.1. Cycle count

In this design the majority of the data processing is located in the hardware but an extra overhead in software is required to program the hardware. This overhead is the price we pay for being able to reconfigure the system to meet the demands of Software Defined Radio. The control code overhead will result in an increase in power consumption when compared to a non-configurable system.

In Figure 3 we show how the ratio of control code cycles to non-idle cycles in the DSP decreases as the data rate increases.

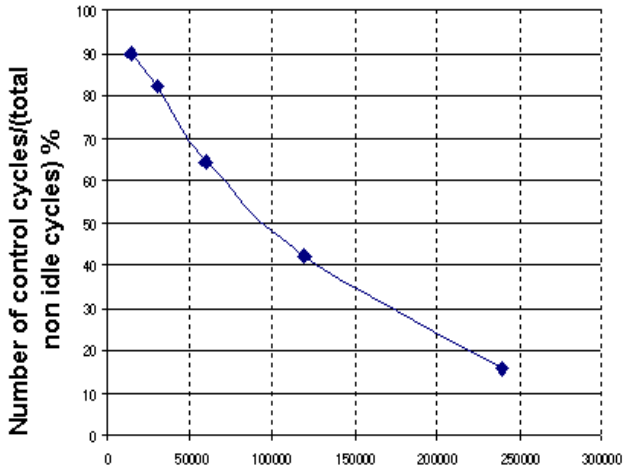

Figure 3 Control code overhead versus data rate

4. OBSERVATIONS

Many of the issues encountered during this design exercise have been due to partitioning the rake receiver into a set of general-purpose correlators and then trying to save silicon by time multiplexing them on to one physical unit. This required significant amounts of additional synchronisation code to ensure that when changes in the PN phase were implemented the effect of reprogramming each hardware block did not cause a glitch in the processing of the data.

In this implementation we used a simple scheduling scheme to determine which tasks would have access to the DSP. The slack in the DSP made it possible to guarantee that all tasks would meet their deadlines but it is clear that as the system became more complicated and the available slack in the DSP was reduced the task of the scheduler would become significantly more difficult.

The memory requirements for our system are large (>146Kbytes) most of this memory is associated with the control code. Further optimization of the design could make substantial savings, but it is clear that a software controlled reconfigurable platforms will need much larger amounts of control code and data memory when compared with a specialized non-configurable solution.

The extra control code used in this design will increase the power consumption of the baseband. The power consumption due to the control code will increase with data rate and receiver complexity. However because the combiner is also done in software the ratio of control code power dissipation to total processor dissipation will decrease as the data processing becomes the larger element. This suggests that a processor based SDR solution looks more attractive at higher rather than lower data rates.

5. CONCLUSION

In this paper we have described the philosophy, principles and design details for an SDR based rake receiver. This work has focused on evaluating the consequences of splitting the rake receiver into hardware and software modules while still achieving levels of flexibility more normally associated with a pure software solution.

The amount of control code required in this system is clearly a limiting factor when we try to reduce power consumption. This is especially true for low data rates when the power consumption for the data processing is low and we would therefore expect a proportionally low value for the control code. Ultimately the power consumed by the control code is limited by the architecture of the processor it is executing on. DSPs are designed for data processing and have long instruction pipelines that are inefficient when executing unpredictable branches. It would therefore seem sensible to locate the control code in a dedicated processor and the data processing in logic or a DSP.

In this design study we have found that by attempting to reduce the silicon area we increased the complexity of the control code and hence, potentially, increased the power consumption. This suggests that the optimum architecture for a SDR terminal will require a tradeoff between power consumption and silicon area.

One of the issues that we only briefly addressed in this study is the problem of scheduling tasks and the resources shared by these tasks (e.g. the data bus). It is clear that in an SDR terminal, where the number and type of tasks will not be known at design time, it will be difficult to devise a generic scheduling scheme that will guarantee that tasks will always meet their deadlines.

The component that limited the data throughput in our system was the data bus and while its performance was improved by using priority driven DMA schemes it was clear that by restricting all data movements so they only pass through a single resource we will encounter significant problem in developing power efficient SDR solutions.

6. REFERENCES

[1] H. Blume, Hubert, H. T. Feldkamper, T. G.Noll, Model based exploration of the design space for heterogeneous systems on chip, ASAP' 02, p29, July 17-19 2002

[2] Ralf E. Schuh, Peter Eneroth, Peter Karlsson, Multi-Standard Mobile Terminlas, Telia Research AB, Ralph.x.Schuh@telia.se

[3] Texas Instruments, Implementation of a WCDMA Rake Receiver on a TMS320C62x. DSP Device, Application report SPRA680, July 2000

[4] N. Zhang, C. Teuscher, H. Lee, B. Brodersen, Architectural Implementation Issues in a Wideband Receiver Using Multiuser Detection, University of California, Berkeley.

[5] Sridhar Rajagopal etal, Implementation of Channel Estimation and Multiuser Detection Algorithms for WCDMA on Digital Signal Processors, Rice University, Houston

[6] A.P. Chandrakasan, R.W. Brodersen, Minimising power consumption in digital CMOS circuits, Proceedings of the IEEE, Vol 83, No. 4, April 1995

[7] Hausner J, Integrated Circuits for Next Generation Wireless Systems, Proc of ESSCIRC 2001

[8] Jackie Silcock, Swamy Kutti, Taxonomy of Real-Time Scheduling, Deakin University, Geelong, Australia

[9] Eric Verhulst, Communication as a backbone for a well balanced MP-SoC system design, MP-SoC Summer School, July 2002

[10] Amer Baghdadi etal, An Efficient Architecture Model for Systematic Design of Application-Specific Multiprocessor SoC, DATE2001 proceedings, page 55. March 2001

[11] 3GPP, Physical channels and mapping of transport channels onto physical channels (FDD), TS 25.211 V5.2.0 (2002-09).

[12] Benny N Vejlgaard, Preben Mogensen, Jasper Bailum Knudsen, Grouped Rake Finger Management Principle for Wideband CDMA, VTC, May, 1999

[13] J. S. Lee, L. E. Millar, CDMA Systems Engineering Handbook, Artech House Publishers, 1998

[14] H. El-Sallabi, H. Bertoni, P. Vainikainen: Channel characterization for CDMA rake receiver design for urban environment, Proc. of ICC 2002 - 2002 IEEE International Conference on Communications, New York, USA, 28 April – 2 May 2002, pp. 911–915

[15] J.A. Stankovic, M. Spuri, M. Di Natale, G. Buttazzo, Implications of Classical Scheduling Results For Real-Time Systems, *IEEE Computer*, Vol. 28, No. 6, pp. 16-25, June 1995

[16] Jia Xu, David Lorge Parnas, On Satisfying Timing Constraints in Hard-Real-Time Systems, IEEE Trans on Software Eng, Vol. 19, No.1, January 1993