A FLEXIBLE CHIP RATE PROCESSOR FOR CDMA RAKE RECEIVER

Zhuan Ye (Motorola Labs, Schaumburg, IL, USA; zhuan.ye@motorola.com)
Yun Kim (Motorola Labs, Schaumburg, IL, USA; yun.kim@motorola.com)
Anthony Schooler (Motorola Labs, Schaumburg, IL, USA; anthony.schooler@motorola.com)

ABSTRACT

This paper presents a chip rate processor suitable for software implementation of rake receiver for CDMA. The implementation is based on a novel code generator, a parallel correlator, and the traditional processor architecture. The code generator is capable of generating the dispreading codes for various CDMA systems. The correlator unit performs parallel correlation between the received samples and the generated codes. The processor allows the algorithms of rake receiver for different CDMA standards to be realized using different software on the same hardware. A scalable software approach to implement rake receiver algorithms on this processor is also presented. The processor is implemented using synthesizable Verilog HDL description.

1. INTRODUCTION

The software definable radio (SDR) advocates more flexible or "soft" approaches to implement key physical layer signal processing algorithms for radios. A good working definition of software radio is a radio that is substantially defined in software and whose physical layer behavior can be significantly altered through changes to its software [1]. One particular radio standard, CDMA based standard, has gained a lot of popularity since its initial commercialization in the 1990's. Currently there exist many standards within CDMA framework, for example, IS-95, CDMA2000, and WCDMA. It will be desirable that multimode operations for these CDMA standards can be achieved by software change without relying on multiple sets of hardware.

The availability of high speed digital signal processors (DSP) has made it possible to implement more and more of radio's physical layer processing in software. But the chip rate processing for rake receiver, which is an integral part in CDMA baseband processing, is still most likely carried out by dedicated hardware or ASIC. These ASICs are generally not designed to be flexible enough to perform multimode operations. Flexible architectures of rake receiver have been discussed in recent publications [2] and [3], but these architectures still lack the real software upgrade-ability that defines software radio.

Rake receiver is used in CDMA baseband processing to take advantage of multipath diversity. This is one of the most important capacity improvement features of CDMA systems [2]. This paper presents a novel approach to implement rake receiver algorithms for different CDMA standards on a common processor, chip rate processor (CRP). The CRP performs parallel generation of the despreading codes, and parallel correlation between the received samples and the generated codes. The flexible code generator can generate the despreading codes for various known CDMA systems. The algorithms of rake receiver can be implemented on the CRP using modular software architecture.

This paper is organized as follows. First the functionalities of rake receiver are introduced in general. Then the hardware architecture of the CRP is discussed in more detail. In the next section, the software architecture for an IS-95/CDMA2000 system is discussed. Finally, simulation results are presented and conclusions are drawn.

2. RAKE RECEIVER

Due to multipath propagation, the transmitted signal is replicated several times when arriving at the receiver in wireless systems. Each path arrives with different amplitude and time delay. A rake receiver separates the different multipath components and forms the weighted, phase adjusted, and delay adjusted sum of these multipath components [4]. A functional block diagram of rake receiver with three rake branches is depicted in Figure 1. The number of rake branches determines the number of multipath components that can be exploited to combine their energy.

Typically, rake branches are assigned to distinct multipath components according to multipath search results. These components have different energy, carrier phase, and are delayed in time relative to one another. The processing involved in a rake branch can be divided into two categories: chip rate processing and symbol rate processing. The chip rate processing block performs the front-end despreading and correlation functions. The input to this block is the digitized baseband complex signal. First, the rake branch generates the appropriate despreading sequences needed to remove the spreading sequences imposed at the transmitter side. Then the despreaded pilot samples and data samples are accumulated over certain period to produce pilot symbol and data symbol correlator outputs. These outputs are further used by symbol rate processing blocks to perform the other essential functions of rake receiver: channel estimation, frequency offset estimation, data symbol recovery, etc. Data symbol recovery process employs coherent detection to calculate weighted and delay adjusted sum from several multipath components, also known as Maximum Ratio Combining (MRC). MRC requires an estimate of the transmission channel of each multipath. This estimate is much more precise if performed on an unmodulated signal, or the pilot signal in a typical CDMA system. That is why rake branch involves correlations for both pilot channel and data channels. More correlations are performed on the pilot channel as well in order to track the timing difference between receiver and transmitter. The de-skew processing adjusts the delay among different multipath components before the MRC is performed.

Data rates involved in symbol rate processing are lower compared to that involved in chip rate processing. In addition, the symbol rate processing also requires a fair amount of mathematical functions such as conjugate multiplication, multiply and accumulate, etc. Therefore the symbol rate processing is more suitable to be carried out on DSPs. On the other hand, the chip rate processing involves relatively simpler, higher data rate, and shorter word-length processing tasks, thus making it more desirable to be implemented on ASICs. These ASICs are less flexible although they traditionally have higher performance and less power consumption. Some vendors have started to emphasize the flexibility of their solutions to support multimode operations of radio. The CRP is also intended to be an evolving SDR replacement of dedicated ASIC to perform chip rate processing.

3. HARDWARE ARCHITECTURE

The CRP is a standalone processor consisting of an instruction-decode unit, a branching unit, and three processing units. In addition, the processor has three memory instances, one for the instructions (instruction memory), one for the operands (operand memory), and another for the input data samples to the CRP (input memory). A memory control unit generates and arbitrates addresses and control signals to all of these memory instances. The three processing units in the CRP are: the code generator unit, the correlator unit, and the general-purpose ALU unit. The centralized register file provides operands to the three processing units, while the operand memory serves as storage for various purposes. Figure 2 illustrates the major blocks of the CRP and the inter connections among these blocks.

The code generator unit is designed to generate the despreading codes used by a variety of CDMA standards. It is capable of generating the pseudo-random noise (PN) sequences for CDMA2000, WCDMA, GPS, and several other periodic codes. The correlator unit is used to correlate the input data with the despreading codes and provide a complex correlation sum. The correlation sum can then be accumulated with a previous sum to calculate the correlator output in the rake receiver. The general-purpose ALU unit is responsible for performing miscellaneous arithmetic operations such as addition, subtraction, multiplication, etc. The general-purpose ALU will also set the flags that control the subsequent conditional branching instructions, which allows complex program flow. Most of the datapaths in the CRP are complex, reflecting the I/Q inputs used in baseband digital receiver. However, some of the operations in the general-purpose ALU unit can revert back to a non-complex mode when needed.

The register file provides 40-bit complex operands and a storage place for the three processing units of the CRP. It is composed of two read ports and one write port to accommodate single cycle operation. Movement of data between the register file and the operand memory is achieved through the memory controller unit. The input memory is a single write, single read memory composed of 4 banks of memory. Only the analog to digital converters (ADC) can write into the input memory, and the CRP can then control the read behavior of the input memory as dictated by the programming. The operand memory is a single port memory primarily used as long-term storage of variables. The instruction memory is for the storage of the instructions of the CRP. Once loaded, the instructions are fetched in sequence and decoded by the instruction decode unit for appropriate interpretation.

The CRP is capable of performing branches using its own branch unit. The branching operations can be either unconditional branches or conditional branches. The assembly code for the CRP supports programmer hints for predictive branching in order to reduce the number of stalled cycles. Hardware loops are supported for zero overhead loop structures in the CRP. Furthermore, up to 4 loops can be nested creating a potential for very complex program flow.

The flexibility of the CRP lies mostly in the code generator unit and the correlator unit.

3.1 Code Generator Unit

The code generator is capable of generating 4 consecutive bits of various spreading codes in one clock cycle. In particular, the unit is optimized to generate codes that are specified using linear feedback shift registers (LFSR) as required by modern CDMA standards; however, the unit is not limited to such codes. Figure 3 is a coarse block diagram of the 20-bit code generator in the code generator unit. The code generator is actually composed of two such processing units, one for the imaginary part and the other for the real part. The description of the code generator from hereon will focus on the operation of this 20-bit code generator.

The code generator employs Fibonacci configuration of the LFSR to generate PN sequences [5]. In this paper, the contents of the shift register are referred as the state, and the feedback tap weights are referred as the mask. The state and the mask are the two fundamental input operands to the code generator. The two values are bitwise ANDed by 20 AND gates, and XOR reduced to a single bit. In a typical single bit PN generator, this resulting bit will be shifted into the LSB of the LFSR. However in the CRP's code generator, a temporary "next state" is created using the shifting operation. The new resulting state value is once again bitwise ANDed with the original mask and XOR reduced to create another resulting bit. The process is repeated two more times until 4 consecutive PN bits are generated. The 4 generated bits are presented to the holding register called PNflags to be used by the correlator unit. Meanwhile, the updated state, with the 4 new values shifted in the LSBs of the state are written back into the register to be used in future PN generation routines.

Additional hardware blocks are also included in the code generation unit for more flexibility. The first block of such hardware is a 20-bit shift register. The shift register is used to build a complete 20-bit state value. The new state value is assembled 4-bits per clock cycle into the shift register. When the 20-bit state is complete, the result can then be stored in the register file. This is a convenient means of combining the results from the 5 previous code generations into a single 20-bit result. The mechanism is useful when generating a future state value for a quick jump in PN sequence cycle (e.g. jumping ahead by 20,000 PN bits).

The second additional hardware is a 4-bit accumulator. The accumulator is used to modify the current LFSR code by other LFSR codes or a constant. The accumulator provides a convenient temporary 4-bit storage where a code can be XORed with other codes. For instance, generation of Gold codes can be accomplished using the accumulator. The first set of LFSR PNs can be stored in the 4-bit accumulator. Then, the second set of LFSR PNs can be generated and XORed with the 4-bit accumulator resulting in 4 consecutive bits of Gold code.

Another additional hardware is the 4-bit carry register. Basically, the carry register provides storage for 4 MSBs of the previous state (operand). At the output stage of the code generator unit, where the state value is written back to the register file, a multiplexer selects either the carry register or the generated PN code to be fed into the 4 LSBs of the outgoing state. This is the mechanism by which the state information from previous operation can be passed on to the next operation. Such operation is useful when dealing with LFSR of greater than 20-bits in length. Theoretically, the code generator is capable of generating any length LFSR by computing 20-bit segments of the LFSR per clock cycle; however, generation of codes specified by longer LFSR would consume more cycles.

Walsh code generation is also possible on the code generation unit of the CRP. Walsh code generation is not a complex operation, therefore both the real and imaginary code generators can be used to generate two sets of independent Walsh codes. Generation of Walsh code begins with the column address and row address as operands to the code generator unit. It requires some extra decode logic besides sharing the AND and XOR gates used in the PN generation. The resulting 4 bits can then be interpreted as the 4-bit values of the Walsh code at the specified row and column. In addition, OVSF codes can be generated in the same fashion as well.

3.2 Correlator Unit

The correlator unit performs a complex-conjugate multiplication of 4 complex input samples and the 4 complex despreading code samples generated by the code generator unit. Besides the data samples and the despreading codes, the correlator has one more input operand. This input operand is a complex value, supplied by the register file, to which the complex correlation value is coherently added. The resulting complex sum is written back to the register file. In cases where it is desirable to correlate less than 4 samples per cycle, a multiplexer gives an option of correlating a single data point. Such option is useful for correlating data samples of length that is a non-integer multiples of 4 (i.e. Barker series of 802.11b).

The correlator is supported by a flexible input memory to simultaneously supply the correlator with 4 data samples per clock cycle. The input memory is composed of 4 separate banks of 16-bit wide and 2K words deep memory. The 4 separate banks allow 4 data samples to be read out simultaneously, one from each bank per clock cycle. To support such operation, it is important to deposit the incoming data into the 4 separate banks such that each desired samples (samples separated by interval of one chip, used in correlation) reside in different memory banks. The memory controller unit simply increments the write address of the input memory by one every time a new sample is received from the ADC. The intelligence lies in the interpretation of the write address such that the input samples will be deposited in the appropriate banks.

For instance, if the samples are sampled at twice the chip rate, the second and the third LSBs will be used to select the bank of the memory. The remaining bits of the address will be used as the new effective address into the selected memory bank. For example, binary address of 000001 will be mapped to physical address 1 of memory bank 0, while binary address of 000010 will be mapped to physical address 0 of bank 1. The purpose of this address remapping is to deposit samples separated by one chip interval into different banks of the memory, so that they can be accessed simultaneously. This memory access mechanism allows the CRP to perform 4-chip correlations in one clock cycle. There is a crossbar network dedicated to rotating the data samples from the 4 banks to a correct order before relaying them to the correlator unit. The CRP currently supports oversampling factors of ×2, ×4, ×8, ×16, and ×32.

3.3 Other Blocks

The last remaining processing unit of the CRP is a general-purpose ALU capable of 40-bit complex or real operations. Examples of the operations include additions, subtractions, multiplication, logic operations, shifts, and magnitude calculation. The general-purpose ALU is also responsible for setting the zero flag, carry flag, and the negative flag, according to the results of the operation. The flags are then be used by the branch unit to branch conditionally. The branch unit operates completely independently of the three processing units. Therefore it is possible to perform an operation and a branch simultaneously in a VLIW fashion.

4. SOFTWARE ARCHITECTURE: AN IS95/CDMA2000 EXAMPLE

This section will demonstrate the software implementation of rake receiver on the CRP targeting IS95/CDMA2000 as an example.

4.1 Development Tools

Several tools were developed to facilitate the software development on the CRP. The hardware and software co-development flow is illustrated in Figure 4. The tool programs are denoted in shaded boxes, including the commercial Verilog simulator. The software source code is developed using CRP's own assembly language. Next an in-house developed linter program will check for illegal code sequences, and an assembler tool will generate the machine code for simulation. The Verilog HDL simulator simulates the CRP hardware and software together, and produces results for post processing and statistical analysis.

4.2 Software Rake Receiver for IS95/CDMA2000

The IS95 standard and the CDMA2000 (single carrier) standard have a lot of similarities. One of the main differences between them is that CDMA2000 supports more than one spreading factor on the data channel. The intention is to demonstrate one piece of software that can support both systems.

The software implementation of rake receiver for IS95/CDMA2000 on the CRP contains the following modules: a simple round robin task scheduler for real time task scheduling; a message handler decodes the control messages from control processor; several pilot correlation tasks calculate pilot correlator results, clock recovery correlator results, and perform channel/parameter estimation; and several data correlation tasks calculate data correlator results and perform MRC. The number of rake branches utilized determines the number of pilot correlation tasks; the number of data channels allocated determines the number of data correlation tasks. This software example of rake receiver includes the implementation of both chip rate processing and symbol rate processing on the CRP, although the symbol rate processing is more suitable for DSP implementation.

4.3 Rake Receiver Implementation on the CRP

The fundamental operations involved in performing chip rate processing are, to generate despreading sequences and correlate them with the appropriate samples in the input buffer. The correlation results will then be accumulated for a certain period before used by symbol rate processing. This operation can be implemented using the basic routine shown in Figure 5. The first three instructions will set up the hardware loop. Inside the loop there are one multiple PN generation instruction (MPNgen), one multiple Walsh generation instruction (Mwalsh), one add instruction to set up next Walsh generation, and one parallel correlate and accumulate instruction (McorAcc). This basic routine varies slightly between pilot correlation tasks and data correlation tasks. In pilot correlation, there is no need to generate user specific Walsh codes. In data correlation, the same codes need to be correlated with different multipath components, therefore more than one correlate and accumulate operations are performed. Another more straightforward approach is to dedicate one task for each multipath component, while performing both the pilot correlation and data correlation for this multipath altogether. The drawback of this architecture is: if there are more than one code channel to be demodulated and they have different spreading factors, it is more difficult to utilize the hardware looping effectively.

It is assumed that multipath search results are already available to distinguish different multipath components. The task scheduler and the message handler are responsible for updating the correlation tasks. Each correlation task has, among other parameters, one parameter to indicate its time to execute. The real time information is derived from a special peripheral to the CRP: a sample counter tracking the number of ADC inputs dumped into the receive sample buffer. The control processor is responsible for interpreting the search results and extrapolating these key parameters to set up correlation tasks appropriately: the start pointer in the data buffer where the correlation starts; the PN generator state matching the start sample; the end pointer in the data buffer where the correlation ends; the number of user code channels and their Walsh indexes.

Figure 6 shows an example of the execution of three pilot correlation tasks and one data correlation task based on the search results. Since the software routine for each correlation task requires the overhead of parameter passing, it is desirable to execute these tasks at a reasonably long interval. Such batch processing allows the real time execution of correlation tasks. In this example, the interval between the consecutive executions of pilot correlation is set to be 64 chips. For data correlation with spreading factors fewer than 64 chips, the routine will calculate more than one correlation output in each task. In Figure 6 there are three multipaths with different delays. Once the search results are available, it should be easy to calculate the convenient time when the correlation tasks should start, i.e., t1, t2 and t3. These time instants correspond to the received samples from different multipath components that the correlation will use. This figure interchangeably uses sample buffer pointer and real time instants, since they essentially contain the same information. By adding the task execution interval to the start time, the "time to run" parameter of the correlation task for each multipath component, i.e., t4, t5 and t6, can also be extracted. Inside each correlation task, it compares the current time with its "time to run" parameter and determines whether the correlation should proceed. The start time of the data correlation tasks should coincide with the symbol boundary to assure the correct calculation of data correlation results. In this example, the start time of the pilot correlation tasks, for the sake of convenience, falls on the symbol boundary as well. There the "time to run" parameter of the data correlation task will be the same as the "time to run" parameter of the pilot correlation task for the latest arriving multipath component, i.e., t6, as shown in Figure 6. This implementation implicitly adjusts the delay among different multipath components at the chip rate input buffer, rather than at the symbol rate output buffer depicted in Figure 1.

The described implementation of rake receiver algorithms on the CRP is motivated by the modularity concerns about multicode reception. The pilot correlation tasks can be added for each active multipath component; while for each user code channel, one data correlation task will be allocated to it. Table 1 shows some statistics for these correlation routines and their equivalent processor loading assuming the CRP is running at 150 MHz. The equivalent million cycles per second (Mcps) number is calculated by using the IS95 chip rate of 1.2288 million chip per second and the correlation tasks are scheduled to execute every 64 chips. Most symbol rate processing operations shown in Figure 1 are also included in this statistics.

Task Name	Clock Cycles	Number of Tasks	Equivalent Mcps	Processor Loading
pilot correlation	208	6	24	16%
data correlation (SF=64)	444	1	8.52	5.7%
data correlation (SF=32)	578	1	11.10	7.4%
data correlation (SF=16)	846	1	16.24	10.8%

Table 1 Statistics of CRP's rake receiver implementation

5. CONCLUSIONS

This paper presented a flexible processor suitable of implementing rake receiver algorithms using software. All the multipath components share the centralized processing units instead of having their own functional blocks. The proposed code generator can support various CDMA standards. The parallel correlation unit and the flexible input queue increased the correlation throughput. The software architecture to implement the rake receiver on the CRP is presented as well; the modular architecture offers easy scalability. The processor loading numbers are also presented for the described IS95/CDMA2000 software example. The results demonstrate that the CRP architecture is suitable for rake receiver implementation using software-defined approaches.

6. REFERENCES

[1] J.H. Reed, *Software Radio: A Modern Approach to Radio Engineering*, Prentice Hall PTR, New Jersey, 2002

[2] L. Harju, M. Kuulusa, J. Nurimi, *"Flexible Implementation of A WCDMA Rake Receiver"*, in Proc. of SIPS 2002, pp.177-182, Oct. 2002

[3] S. Lee, J. Kim, *"VLSI Architecture of Rake Receivers for cdma2000 Systems"*, in Proc. of SIPS 2002, pp. 183-188, Oct. 2002

[4] A.J. Viterbi, *CDMA: Principles of Spread Spectrum Communication*, Addison-Wesley, Massachusetts, 1998

[5] R.L. Perterson, R.E. Ziemer, D.E. Borth, *Introduction to Spread Spectrum Communications*, Prentice Hall, New Jersey, 1995

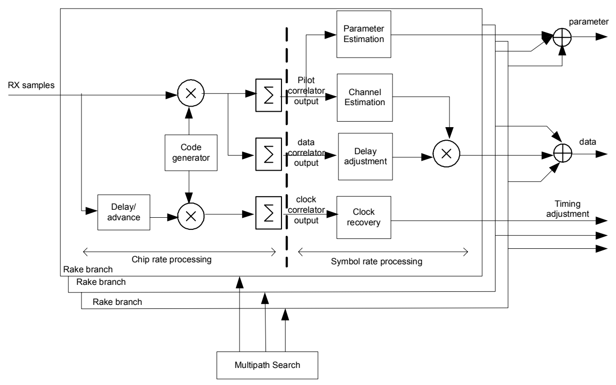


Figure 1 Rake receiver functional diagram

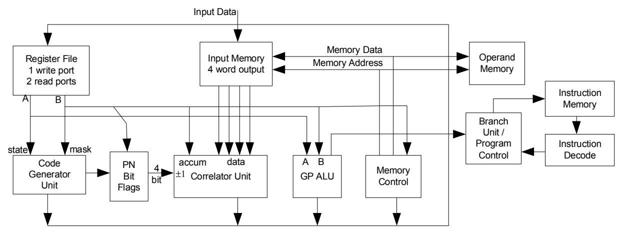


Figure 2 Architecture block diagram of CRP

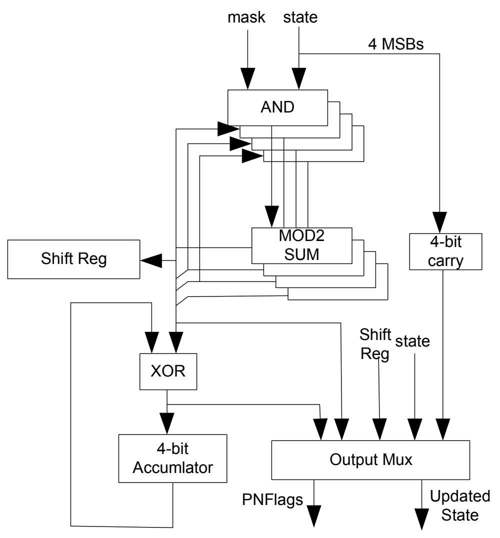


Figure 3 Code generation unit datapath

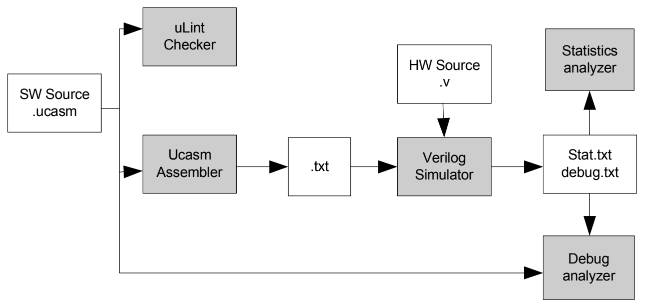
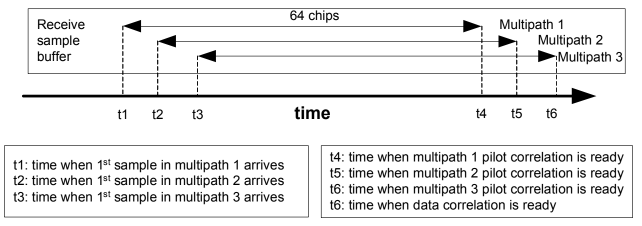


Figure 4 Hardware/software development tools of CRP

```
DoStart0 CorStart;
DoEnd0 CorEnd;
DoEn0 CorCnt;      -- setup loop
CorStart:
            MPNgen Pnmask, Pnstate, Pnstate;  -- generate 4 PN chips
            Mwalsh.c WalshIndex, WalshCol, Temp; -- generate 4 walsh bits
            Add WalshCol, #4, WalshCol; -- next walsh bits
            McorAcc A1, Cor, Cor; -- correlate and accumulate
CorEnd:
```

Figure 5 Example correlation routine implemented using CRP assembly code



t1: time when 1st sample in multipath 1 arrives
t2: time when 1st sample in multipath 2 arrives
t3: time when 1st sample in multipath 3 arrives

t4: time when multipath 1 pilot correlation is ready
t5: time when multipath 2 pilot correlation is ready
t6: time when multipath 3 pilot correlation is ready
t6: time when data correlation is ready

Figure 6 Real time execution of correlation tasks on CRP