Reconfigurable Pipelined Coprocessor for Multi-mode Communication Application

Liang Tang[1] Jude Angelo Ambrose[1] Sri Parameswaran[1]

[1] University of New South Wales, Australia
`{liangt, ajangelo, sridevan}@cse.unsw.edu.au`

Technical Report
UNSW-CSE-TR-201222
July 2012

Abstract

The need to integrate multiple wireless communication protocols into a single low-cost, low-power hardware platform is prompted by the increasing number of emerging communication protocols and applications. This paper presents an efficient design methodology for integrating multiple wireless communication baseband protocols in a pipelined coprocessor which can be programmed to support various baseband protocols. This coprocessor can dynamically select the suitable pipeline stages for each baseband protocol. Moreover, each carefully designed stage is able to perform a certain signal processing function in reconfigurable fashion. The proposed method is flexible (compared to ASICs) and suitable for mobile application (compared to FPGAs). The area size of the coprocessor is smaller than an ASIC or FPGA implementation of multiple individual protocols, while the overheads of timing delay (40% worse than ASICs and 30% better than FPGA) and power consumption (6X worse than ASICs, 100X better than FPGA on average) are kept within reasonable levels. Moreover, fast protocol switching is supported. Wireless LAN (WLAN) 802.11a, WLAN 802.11b and Ultra Wide Band (UWB) transmission circuits are developed and mapped to the pipelined coprocessor to prove the efficacy of our proposal.

1 INTRODUCTION

Numerous wireless communication protocols have been proposed recently, each targeting a different application domain. Some protocols are mainly used for wide area communication, such as WCDMA and GSM. Some protocols are suitable for high speed, medium range area, such as WLAN. Some protocols are for high speed, short distance, such as UWB. And some are designed for low power consumption protocols, such as ZigBee. To meet stringent market demands, modern mobile devices have to combine a number of these communication protocols. It is even predicted that multi-mode communication will be the norm in [23]. Small area and low power consumption are critical for the success of a mobile product, thus ASIC chips are preferred to integrate multiple communication protocols for mass production of mobile devices. Current state-of-the-art communication solutions for mobile devices are able to integrate several protocols on a single ASIC chip. For example, TI's latest WiLink 7.0 chip includes WLAN, GPS, Bluetooth and FM communication protocols [4]. Although all these protocols are included in a single chip, each of them is still implemented as individual hardware blocks, and some components sit idly as only one of the protocols will be active at any one time. Thus, sharing amongst units can greatly reduce area consumption. One example is the multiply accumulate (MAC) circuits from both UWB and WLAN are similar and can be shared, another example is the shift registers in convolutional encoders of UWB and 802.11a which are identical and can be reused.

In the future, wireless terminals are expected to be programmed to support hitherto unknown, upcoming protocols allowing for longer lifetime of a single design. This flexible programmability will enable a single terminal to support various communication protocols (time multiplexed), provided the computation meets the required throughput. This unified single platform can be mass manufactured to reduce Non-Recurring Engineering (NRE) cost, compared to ASICs which contain only a limited number of specification-ready protocols.

Software Defined Radios (SDRs) have been proposed by researchers to meet this technology trend, and various approaches have been used for SDR implementations. Some of these platforms, such as FPGA, DSP or processor array [20, 12, 17, 11, 18], are flexible, but consume a significant amount of power, have large footprints, and are infeasible for mobile terminals. Typically such solutions are useful for base stations. Cost effective solutions with such flexible circuits for mobile terminals usually result in inadequate performance. For example, it is difficult to achieve the 480 Mbps data rate [5] required for UWB using FPGAs, DSPs or processor arrays in mobile terminals. Some of the newer protocols reach 1Gbps data rate, and certainly require the use of ASICs. One example of such a system is the WLAN 802.11ac [6]. Some other approaches, such as datapath merging [26], are area effective, but only a limited number of specification-ready protocols can be supported and future protocols cannot be implemented.

To overcome the limitations of these SDR approaches, a novel, two-level, reconfigurable, pipelined coprocessor architecture is proposed in this paper and illustrated in Figure 1.1. This reconfigurable coprocessor is designed by ASIC technology, instead of FPGA. Thus the high performance is achieved and this coprocessor is suitable for mobile application. Each stage in the pipeline fulfills a certain signal processing function in the communication protocol, such as convolutional encoding, scrambling, etc. The first level architecture controls the inter stage reconcilability. Each pipeline stage can be dynamically enabled and disabled by setting parameters based on the active protocol. Moreover, a general purpose processor (GPP) can also be involved in the coprocessor pipeline to replace/add functionalities of a certain stage through the presented *On-Call bus*. The second level controls the intra-stage reconfigurability. All of these stages are carefully designed and they can be reconfigured to adapt to various protocols. For example, various convolutional encoders are supported in the convolutional encoder stage, such as the ones in 802.11a, UWB, WiMAX. The detail architecture of the flexible convolutional encoder stage is given in section 3.2.

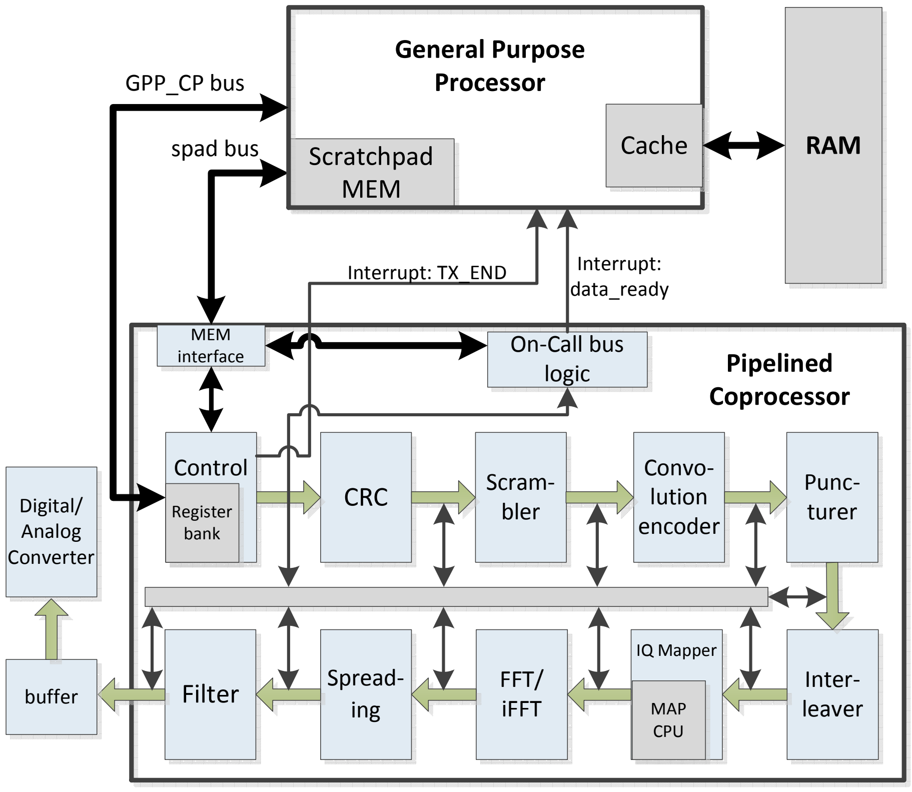

Figure 1.1: Architecture of pipelined coprocessor

Motivation

We are motivated by the need for a methodology to design a reconfigurable baseband circuit which is low in cost and power, has a short timing delay for use in a mobile terminal. Previous methods which are FPGA based, DSP based, or processor array based are not suitable for mobile terminals, and the datapath merging based method cannot provide enough flexibility.

Outline

The rest of this paper is organized as follows. In section 2, the current research on SDR is reviewed. In section 3, the reconfigurable pipelined coprocessor methodology is proposed. Section 4 and Section 5 show the experimental setup and results. The conclusion is given in Section 6.

2 RELATED WORK

Reconfigurable hardware, which is FPGA, has been studied for SDR implementations [10] [9] [8]. Since FPGAs' cost, power consumption, and timing delays are considerably high when compared to ASIC designs [15], they cannot be used in mobile devices. General purpose DSP solutions have been proposed for SDR by [13]. Additionally, communication signal processing oriented DSP solutions, such as SODA from Lin et al. [17], have been proposed to improve the calculation ability of the general purpose DSP. SODA can meet the processing requirements of WCDMA and 802.11a. However, modern high speed standards, such as 480Mbps Ultra-Wideband (UWB) or 1Gbps 802.11ac, are not supported, as the timing constraints are violated. Due to the complexity of the modern wireless communication signal processing requirements, a single (or a limited number of) CPU/DSP may not be enough to complete the signal coding/decoding within the required real time constraints. Thus, processor arrays were proposed as an implementation medium for SDR baseband circuits [11]. However, this is a costly and power hungry solution and it is only suitable for base stations instead of hand held devices [22].

Datapath merging solutions search the shareable components between individual communication circuits, and generate MUXes to be inserted into generic circuit [19], [26]. Although this method has small area and timing overhead, the generated circuit is fixed to the predefined individual circuits and future protocols cannot be supported.

There are proposals targeting reconfigurable individual communication function blocks, such as reconfigurable convolutional encoder [14], reconfigurable interleaver [24, 27], and reconfigurable IQ mapper [25], etc. These manually designed reconfigurable function blocks achieve high performance when implemented in ASIC while having similar programmability to FPGAs and processors. However, a methodology for the design of whole baseband by reconfigurable functional blocks is not proposed as yet and some reconfigurable functional blocks necessary for a whole system are yet to be created, such as the reconfigurable scrambler, reconfigurable CRC, etc.

We propose a coprocessor based methodology for the entire baseband by the use of reconfigurable communication functional blocks. The reconfigurable convolutional encoder in [16] eliminated the common sub-expression and reduced the calculation complexity of DSP. However, according to Table 3 in [16], 382 clock cycles are needed to transmit 216 bits in 802.11a 54Mbps data rate. This large clock cycle/data bits ratio requires high clock frequency to maintain 54Mbps data rate, which results in high power consumption. We designed a reconfigurable Shifter-XOR based circuit to process arbitrary polynomial convolutional encoder in parallel fashion. For the same 216 transmission bits, only 108 cycles are needed in our proposal. The reconfigurable interleavers in [24, 27] are only suitable for WLAN. We extend it to a general form so that both WLAN and UWB can be supported. The reconfigurable IQ mapper in [25] can only support BPSK, QPSK, and QAM modulation mapping. We designed a tiny processor embedded in IQ mapper which can process any derivation based on BPSK, QPSK, QAM based modulation mapping, including π/4QPSK, DQPSK, Complementary Code Keying (CCK) [7], and Dual-Carrier Modulation (DCM) [5]. We also designed the reconfigurable blocks which are yet to appear in any research literature, such as reconfigurable scrambler, reconfigurable CRC and reconfigurable puncturer. The coprocessor is controlled by a GPP which gives parameters for each communication protocol. Moreover, a special mechanism is created for the GPP to handle certain digital signal processing, if the current reconfigurable blocks cannot meet the requirement of the future specification (i.e. a new protocol).

Our Contribution

- Novel pipelined coprocessor is developed to construct reconfigurable communication baseband circuit by parameter setting from GPP.
- Novel interface between GPP and coprocessor is proposed to use GPP's computing power when certain baseband signal processing is not supported in coprocessor.
- Multiple reconfigurable communication function blocks are developed to build the coprocessor, including scrambler, convolutional encoder, puncturer, interleaver and IQ mapper, etc.

3 METHODOLOGY

The proposed methodology provides a reconfigurable hardware platform for various wireless communication baseband protocols. We studied numerous communication baseband protocols, including 802.11a, 802.11b, 802.11n, UWB, Bluetooth, WiMAX, DVB, ISDB and DTMB, and found that all of these protocols can be represented by a single pipelined structure, the transmission part of these protocols is illustrated in Figure 3.1. Various communication protocols are used in different application areas, with differing noise tolerance levels and data rate requirements. However, based on the wireless communication theory [21], this pipelined structure is necessary and only the implementations of certain functional blocks are changed to adapt to different protocols.

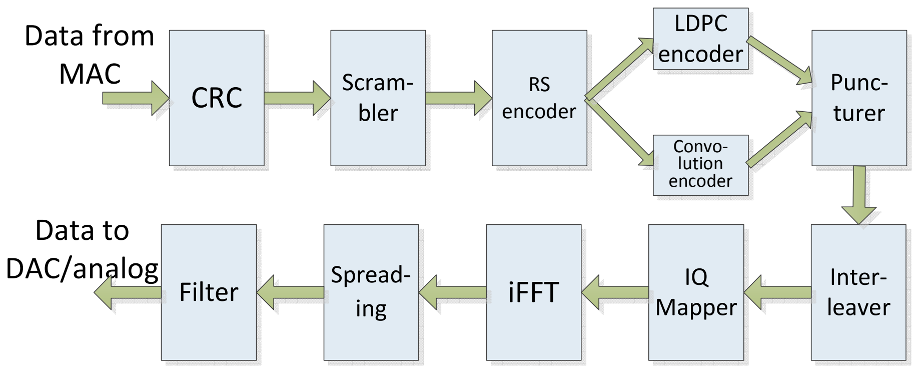

Figure 3.1: General pipeline structure of wireless communication baseband

To effectively design a reconfigurable platform, our design is based on the general form of pipeline communication functional structure shown in Figure 3.1. The architecture of the proposed pipelined coprocessor and interface with GPP is shown in Figure 1.1. There are 10 pipelined stages in the coprocessor. Each baseband function block in Figure 3.1 can be mapped to a corresponding stage in the coprocessor except the Reed-Solomon encoder (RS encoder) and LDPC encoder which are not supported in our current coprocessor. The Control stage is provided before the CRC, and the coprocessor/GPP interface is supported at this stage. The coprocessor's output, which is from the Filter stage, is fed into a digital/analog converter through a buffer. Thus, the whole baseband transmission chain is covered by the coprocessor. There is a block called *On-Call bus logic* which provides signal processing capability on the GPP when this signal processing is not supported on the coprocessor. This block will be detailed in section 3.1. The *GPP_CP bus* is used to transfer commands from GPP to coprocessor, and status from coprocessor to GPP. There are three important GPP commands: *Load_Param* instructs the coprocessor to load parameters; *TX_Start* initiates loading of payload data to the coprocessor and activates the coprocessor pipeline for transmission; and *SPD_Ready* command is used for *On-Call bus* operations. The payload data between GPP and coprocessor are transferred by *spad bus* which connects the scratchpad memory in GPP and the memory interface module in coprocessor. Two interrupts are provided by the coprocessor for efficient GPP response. One interrupt is *TX_END* which indicates whether the current packet frame transmission is finished, another interrupt is *data_ready* which is for *On-Call bus* operation again.

Reconfiguration of the pipelined coprocessor is located at two levels. First, since not all of the functional blocks are necessary for all protocols (for example, an FFT module is not needed in 802.11b), inter-stage reconfiguration is provided such that stages can be disabled and enabled. In addition, a GPP interface is provided by the *On-Call bus* to utilize the computation power from the GPP. Secondly, all stages in the coprocessor, which correspond to functional blocks in protocols, are carefully designed to support various protocols by setting parameters within it. We call this Intra-Stage reconfigurability.

3.1 Inter-Stage Reconfiguration Level

All functional blocks are implemented as stages in the pipelined coprocessor to accelerate the communication throughput. However, as stated previously, only a subset of functional blocks are needed for each protocol (e.g., convolutional encoder is essential in 802.11a and UWB, but it is not used in 802.11b; spreading is a key component in 802.11b, though not used in 802.11a and UWB). Each stage in the pipeline can be disabled and bypassed by setting parameters.

Besides the main datapath for payload packet data (signal *spad_bus* in Figure 1.1), an extra data processing interface, named *On-Call bus*, is provided between the coprocessor and GPP to allow more flexible signal processing. If a future protocol is created with a specific baseband function block, which is out of the range of the current pipeline stage, this specific function block can be implemented in the GPP by the use of software. For example, an RS encoder block is used for the UWB packet header and this block should be located between the scrambler and convolutional encoder. However, the RS encoder hasn't been implemented in our system yet. To support UWB packet header transmission, the RS encoding signal processing can be executed on the GPP. The *On-Call bus* exchanges data between GPP and coprocessor through scratchpad memory, and its operation is described here. First, because scratchpad memory is used to exchange data between coprocessor and GPP, the parameters of transferring data in the scratchpad memory, such as size and starting address of the transferring data block, need to be set by the GPP to the coprocessor before baseband frame transmission. Second, GPP needs to set the access point which is the location for missing signal processing capability in pipelined coprocessor to fetch data for processing on the GPP (for example, the access point is located on the output of the scrambler stage in the RS encoder missing scenario). The coprocessor needs this information to select the data to/from specific stage via MUXes. Third, the output of the stage before the access point will be selected by the coprocessor and stored in scratchpad memory and an interrupt signal (*data_ready*) will be asserted to inform the GPP when the space in the scratchpad memory is full (or the coprocessor dumps all processed data at the completion of the last data block of frame). Then, the GPP will perform the corresponding signal processing and a special coprocessor command, called *SPD_Ready*, will be issued to inform the coprocessor via a coprocessor instruction when the GPP has finished signal processing. Finally, once the coprocessor receives an *SPD_Ready* command, the GPP processed data are fed into the pipeline stage, just after the access point, to ensure the completeness of the pipeline. In the missing RS encoder scenario, the coprocessor feeds data from the scratchpad memory to the input of the convolutional encoder. By this *On-Call bus* implementation, only one extra physical signal, *data_ready* interrupt, is added for GPP and coprocessor communication, and the impact on the GPP modification is minimized. To improve throughput, three same size memory blocks are used by the GPP in scratchpad memory for *On-Call bus*: one block of memory is for coprocessor data writing; one block is for GPP data writing; and, the last one is for coprocessor data reading.

3.2 Intra-Stage Reconfiguration Level

For effective multi-mode baseband methodology, reconfigurable baseband functional blocks are created and implemented in each stage of the pipelined coprocessor. The whole SDR baseband transmission path has been established in this paper, including reconfigurable convolutional encoder, reconfigurable interleaver, reconfigurable IQ mapper, reconfigurable scrambler, etc.

Due to limitation of space, only reconfigurable convolutional encoder, interleaver and IQ mapper are described here.

Reconfigurable convolutional encoder

A convolutional encoder is used for error correction and is commonly used in wireless systems. The convolutional encoder can be implemented by a series of concatenated shift registers, and certain registers' outputs are wired to XOR cells to generate encoded data. All convolutional encoders share the same structure, but differences are: number of shift registers, the different shift register whose output should be wired to XOR, and the number of output ports (branch). These parameters can be defined as a polynomial, for example, the polynomials of 802.11a convolutional encoder can be expressed as $g0=133_8$, $g1=171_8$ [7], and the polynomials of UWB convolutional encoder are $g0=133_8$, $g1=165_8$, $g2=171_8$ [5], each polynomial defines the implementation of one output branch. Figure 3.2 (a) and (b) gives the block diagram of these two convolutional encoders in serial implementation. To improve throughput, it is a common practice to implement convolutional encoder in parallel fashion, usually in 8-bit width. The parallel circuit of one output branch of 802.11a is illustrated in figure 3.2 (c). The 8-bit input data will be combined with the register output, which is the buffered input data with one clock cycle delay, to a 16-bit bus connecting to the shifters. The final result can be derived by the shifters' output and XORs. The parallel circuits for UWB branches have the similar structure, the only difference being the configuration of shifter logics.

We studied the structure of parallel processing implementations of convolutional encoders and found the XORs and shifters compose an *XOR matrix* which is unique for each protocol. However, the inputs for each tier of XOR operations in the *XOR matrix* are merely a shifted version of the buffered input bytes. Thus, this *XOR matrix* can be split into multiple stages to be reconfigurable efficiently. An eight-bit register

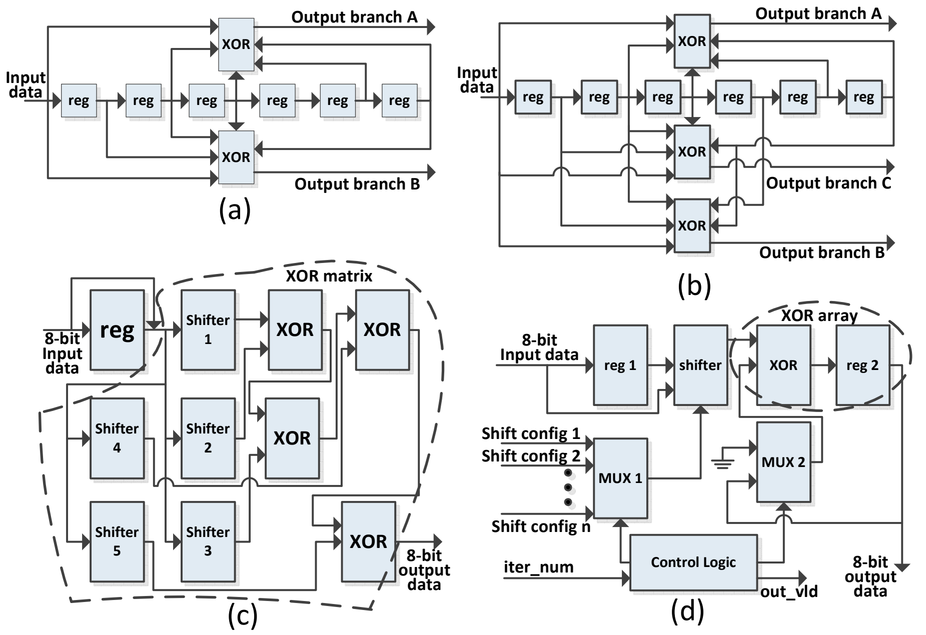

Figure 3.2: 802.11a and UWB convolutional encoder implementation (a) 802.11a serial encoder (b) UWB serial encoder (c) one output branch of 802.11a parallel encoder (d) one output branch of reconfigurable encoder

(*reg 2* in Figure 3.2 (d)) is added after the eight-bit XOR operator, to hold the XOR result. The combination of this register and XOR is called *XOR array* which has two inputs: one is *MUX 2* controlled feedback from the *reg 2* which is in the *XOR array*; another is from the barrel shifter circuit which can generate any shifted version of the buffered input bytes according to the predefined parameters selected by *MUX 1*. Note that the operation of *XOR array* will be transparent for the input data during the first iteration controlled by *MUX 2*, as one of the *XOR array*'s inputs will be zero due to the selection of *MUX 2* and the output of the XOR will be identical to the shifter circuit output, which is another input of the *XOR array*. The *Control Logic* selects the MUXes and manages the number of iterations according to the input *iter_num* signal. The detailed reconfigurable *XOR matrix* implementation by shifter and *XOR array* is shown in the Figure 3.2 (d).

The designs of the reconfigurable scrambler and CRC are similar to the method used for reconfigurable convolutional encoder. Due to limitation of space, they are not discussed in this paper.

Reconfigurable interleaver

An interleaver reorders the transmission bits to distribute the burst errors on multiple symbols, so that the error rate per symbol is reduced and thus it is more likely that all symbols can be decoded successfully by an error correction circuit. In different protocols, the interleaver reordering patterns are different. For example, the interleaver in 802.11a can be expressed as two permutations in Equation 3.1 and Equation 3.2 [7]. The index of the bit before the first permutation shall be denoted by k; i shall be the index after the first permutation; j shall be the index after the second permutation. After the interleaving operation, the bit address k will be reordered to address j. Note that N_{CBPS} is the number of coded bits per OFDM symbol and N_{BPSC} is the number of bits in each OFDM subcarrier [7]. Both these two parameters vary between different data rate in 802.11a and different interleaving patterns are generated. The first 32-write addresses of 802.11a interleaving is given in table II of [27].

$$i = \frac{N_{CBPS}}{16} \times (k \, mod \, 16) + \lfloor \frac{k}{16} \rfloor \tag{3.1}$$
$$k = 0, 1, ..., N_{CBPS} - 1$$
$$j = s \times \lfloor \frac{i}{s} \rfloor + (i + N_{CBPS} - \lfloor (\frac{16 * i}{N_{CBPS}}) \rfloor) \, mod \, s \tag{3.2}$$
$$j = 0, 1, ..., N_{CBPS} - 1, \; s = \max (\frac{N_{BPSC}}{2}, 1)$$

The execution of these equations without optimization is computation intensive as there are modular and division operations associated with each data bit. Upadhyaya and Sanyal proposed reconfigurable circuits for different interleaver patterns in 802.11a and the interleaving circuit is significantly simplified [24, 27]. However, only the two-permutation 802.11a interleaver is supported and the three-permutation interleaver, such as interleaver in UWB, is not supported. Furthermore, only serial bit implementation is proposed. Thus higher clock frequency is needed to support high data rate, which results in high power consumption.

The interleaver in UWB has the similar equations to Equation 3.1 and Equation 3.2 with different parameters. However, there is a third permutation in UWB for cyclic shifting, which is not supported by the proposal in [27].

We proposed a more flexible interleaver based on the proposal from [27] and is shown in Figure 3.3. Two memory blocks are used to eliminate data halting, one memory is used for data writing and another for reading, and their role will be alternated when all the data in each interleaving block has been stored in the writing memory. The writing address generation circuits are flexible by setting parameters, thus supporting different interleaving patterns.

The *adder 1* in Figure 3.3 is proposed by Upadhyaya in Figure 2 of [27]. To support the third permutation in UWB, an extra adder, *adder 2*, is inserted to provide cyclic shifting. This differs from the circuit in Figure 2 of [27] meaning the interleaver in UWB can be supported. All input parameters can be set by the GPP in our methodology to achieve a fully programmable interleaver, in contrast, these parameters were fixed in the reference above and limited the application domain to WLAN only. Moreover, instead of sequential memory reading in [27], the reading address generation circuit which is similar to our writing address generation circuit is also implemented. Thus the more complicated interleaving equations can be supported.

To improve throughput, two-bit parallel processing is supported. The parallel reconfigurable interleaving circuit is created according to [27] with another extra adder (*adder 3*) to generate the writing address for bit 1. The bit 1 address generation circuit is based on the calculated bit 0 address in Figure 3.3. The bit 1 address generation circuit is similar to the bit 0 address generation, except the register is not presented and cyclic shift is not added. Thus, addresses for bit 0 and bit 1 are generated in the same clock cycle in parallel.

Reconfigurable IQ mapper

IQ mapper is used to map data bits to certain amplitude and phase for analog transmissions, and it is an indispensable function block in all wireless baseband circuits. The IQ

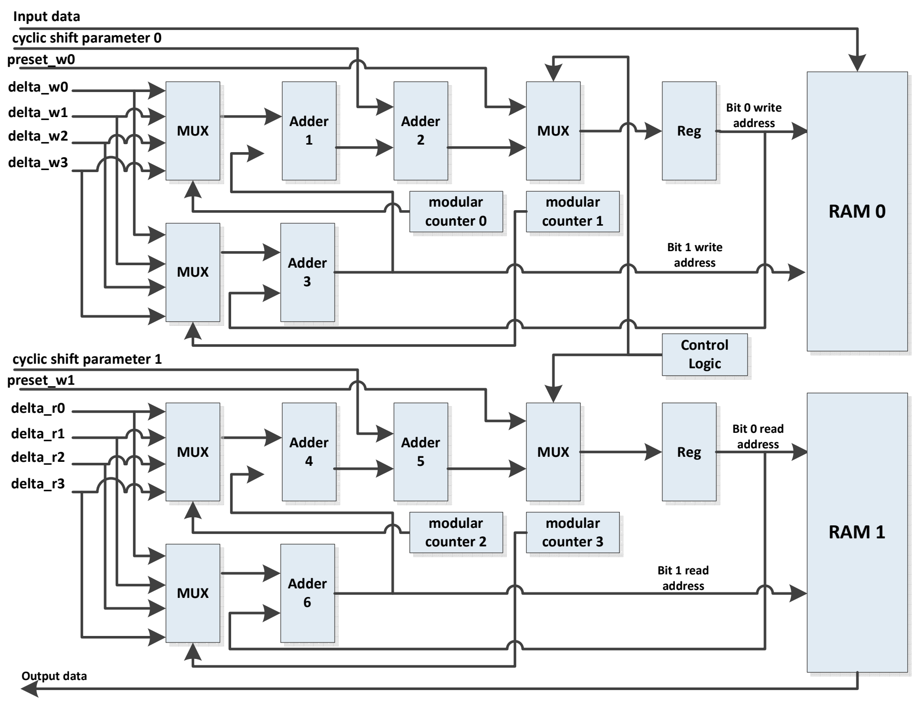

Figure 3.3: Reconfigurable interleaver block diagram

mapping schemes are mostly BPSK, QPSK, 16-QAM, 64-QAM and their derivatives. As mentioned in [25], the reconfigurability of the IQ mapper is hardly exploited in literature. The reconfigurable IQ mapper in [25] supports BPSK, QPSK, 16-QAM and 64-QAM mapping. However, other reconfigurable derivations based on these four basic mapping are not explored. Instead of normal BPSK and QPSK, differential BPSK (DBPSK) and differential QPSK (DQPSK) are used in 802.11b for data rates of 1Mbps and 2Mbps. Complementary Code Keying (CCK) mapping is used in 802.11b for data rates of 5.5Mbps and 11Mbps. In UWB, a special 16-QAM, called dual-carrier modulation (DCM), is used instead of normal 16-QAM. All these special mapping formats are not supported by [25].

We studied the IQ mapper of various protocols and found that there are potential calculations associated to QPSK modulation mapping, such as CCK modulation in 802.11b [7]. To effectively support various calculations, a tiny two-bit data bus pipelined processor, called MAP CPU, is created. Various IQ mapping schemes, including BPSK, DBPSK, QPSK, DQPSK, Offset QPSK, π/4-QPSK, CCK, 8PSK, 16-QAM, 64-QAM and DCM, are supported in our methodology due to the highly programmable MAP CPU. The MAP CPU has five pipelined stages listed in Table 3.1. And there are only six types of instructions (each instruction: 12-bit) for this CPU (given in Table 3.2), thus the instruction decoding circuit can be simplified. The block diagram of MAP CPU is illustrated in Figure 3.4.

The first stage, *IF*, fetches instructions from the instruction memory whose size is only 16*12 bit, i.e. the maximum number of instructions is 16. The PC generation circuit in IF stage is different from a traditional processor: when the PC reaches a predefined thresholds (*num_inst*) which is set by *static control signals* (from MAP

Table 3.1: MAP CPU Stage Description

Stage	Description
IF	Instruction fetch.
ID	Instruction decoding.
EX	Calculation execution.
PG	Phase generation for BPSK and QPSK.
IQ	Final IQ signal generation.

Table 3.2: MAP CPU Instruction Description

Instruction	Description
MOV	Move data from source register or immediate value to destination register.
LSADD	Combined left shifting and addition operation.
MADD	Perform multiple addition by at most four source registers.
PSK	Perform BPSK/QPSK phase change and give the final phase.
BSEL	Select specified input bits and store them in register file.
NOP	Nop instruction.

CPU input port), the PC will be reset to zero to fetch the instruction from the first address again. When the number of PC reset times reaches another predefined value (*blk_size*) set by *static control signals*, the PC will be stopped and a NOP instruction will be issued from *IF* stage, thus the MAP CPU will stop running. The design of PC generation is the optimization for baseband packet IQ mapping. The traditional jump and condition checking instructions are not used here, and this is one of the factors used to reduce the complexity of the MAP CPU. The second stage, *ID*, decodes the instruction via its *Control* submodule and prepare the packet bits for IQ mapping by *Bits Select* submodule. Various control signals will be generated in the *Control* submodule by analyzing the input instruction. These control signals will be transferred to all the following stages. The *Bits Select* submodule selects correct processing packet bits from the *packet data bus*, according to the address (*bit_addr*) provided by the *IF* stage. A tiny *register file* (*RF*) is constructed in this stage. Each register in *RF* is two bits widd, totally there are eight registers, so the size of *RF* is only 8*2 bits. The input of *RF* can be either from the *in_dibit* signal from the *Bits Select* submodule, or from the *rf_wr_data* which is the result of the PG stage. The *RF*'s output, *rf_rd_data* signal, including four registers' read out results, is fed to the EX stage via a MUX. The *ALU* has four inputs in the *EX* stage and can do three operations only: *LSADD*, *MADD* and *NOP*. The definition of these three operations can be found in Table 3.2. The output of the *ALU* is used in the *PG* stage for phase generation or just passing through. Finally, the generated phase will be mapped to *I/Q* signals in the *IQ* stage.

Different with traditional processor, the MAP CPU is not only controlled by instructions, the input *static control signals* provides many control parameters from GPP. The parameters from *static control signals* will not be changed during each data block processing, such as the *num_inst* and *blk_size* in the IF stage. Many other parameters from *static control signals* are not illustrated on Figure 3.4. In contrast to *static control signals*, the *dynamic control signals* includes a bunch of control signals generated by the *Control* submodule in the *ID* stage and these control signals will be changed instruction by instruction. The MAP CPU is controlled by *dynamic control signals* in

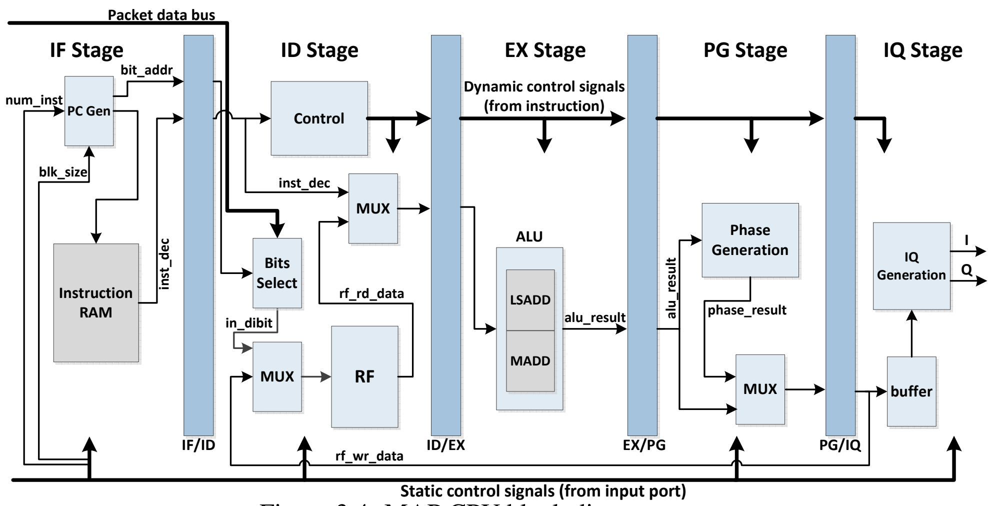

Figure 3.4: MAP CPU block diagram

various stages, such as the MUX before the *RF* in *ID* stage will select *in_dibit* only when the instruction is *BSEL*. All of the control signals belonging to *dynamic control signals* are not shown in Figure 3.4 due to limitation of space.

Another difference between the traditional processor and MAP CPU is the data memory accessing method. As the processing data are the result of the stage before the IQ Mapper in our pipelined coprocessor, instead of accessing data RAM, a buffer (which is outside of MAP CPU) is created to hold a block of processing data and feeds these data into MAP CPU via *packet data bus*. Then the *Bits Select* submodule in *ID* can correctly select the processing bits with the activation of the *BSEL* instruction.

After all these optimizations on the MAP CPU, it can support all IQ mapping methods discussed in this section maintaining the throughputs defined by the individual protocol specifications. The synthesized size of MAP CPU is only 2K Gates (excluding the instruction RAM) and very suitable for mobile application.

3.3 Programming of the coprocessor & mode switching

To program the coprocessor, all parameters for functional blocks need be transferred from the GPP to the coprocessor before the packet data transmission in each protocol. The coprocessor stores all parameters in a register bank which is located in the control stage. During communication mode switching, the parameters for new protocols need to be set from the GPP again. Based on our current coprocessor design, there are about 600 bits for parameters. It can take more than 20 clock cycles in our system for these parameters to be transferred between GPP scratchpad memory and coprocessor. Mode switching period can be further shrunk if the register bank in the control sub module of the coprocessor can be doubled and then two sets of parameters from both protocols can be stored simultaneously. Thus the turn-around time of mode switching is only determined by stage flushing time.

A communication flowchart between the GPP and the coprocessor is shown in Figure 3.5. At first, the GPP selects the active protocol and stores the parameters in the memory location 1 and sends *Load_Param* command to the coprocessor via *GPP_CP bus* if these parameters haven't been loaded to the coprocessor, i.e. switching to new protocol. Once the coprocessor receives *Load_Param* command, it will load the parameters from the memory location 1 to the register bank and set the pipeline accordingly. Secondly, the GPP stores the transmission packet data in the memory location 2 and issues *TX_Start* command. When the coprocessor receives this command, it will exit from its low power idle state and load a block of packet data from the memory location 2 to the pipeline stage for processing. If there are unprocessed data in the memory location 2, the coprocessor will repeatedly load and process until all packet data are processed. During the coprocessor pipeline processing, if the GPP's computing power is needed for certain signal processing, *On-Call bus* activities will be initiated between the GPP and the coprocessor and the detail can be found in section 3.1. The memory location 3 is reserved for *On-Call bus* and three same size memory blocks are included as discussed in section 3.1. Finally, the *TX_END* interrupt will be asserted by the coprocessor if all packet data are processed. Once the GPP detects this interrupt, the GPP finishes the current packet transmission and can send new packet or switch to new protocol by setting the new parameters to the coprocessor. Please note that all memories here are in the scratchpad memory of the GPP.

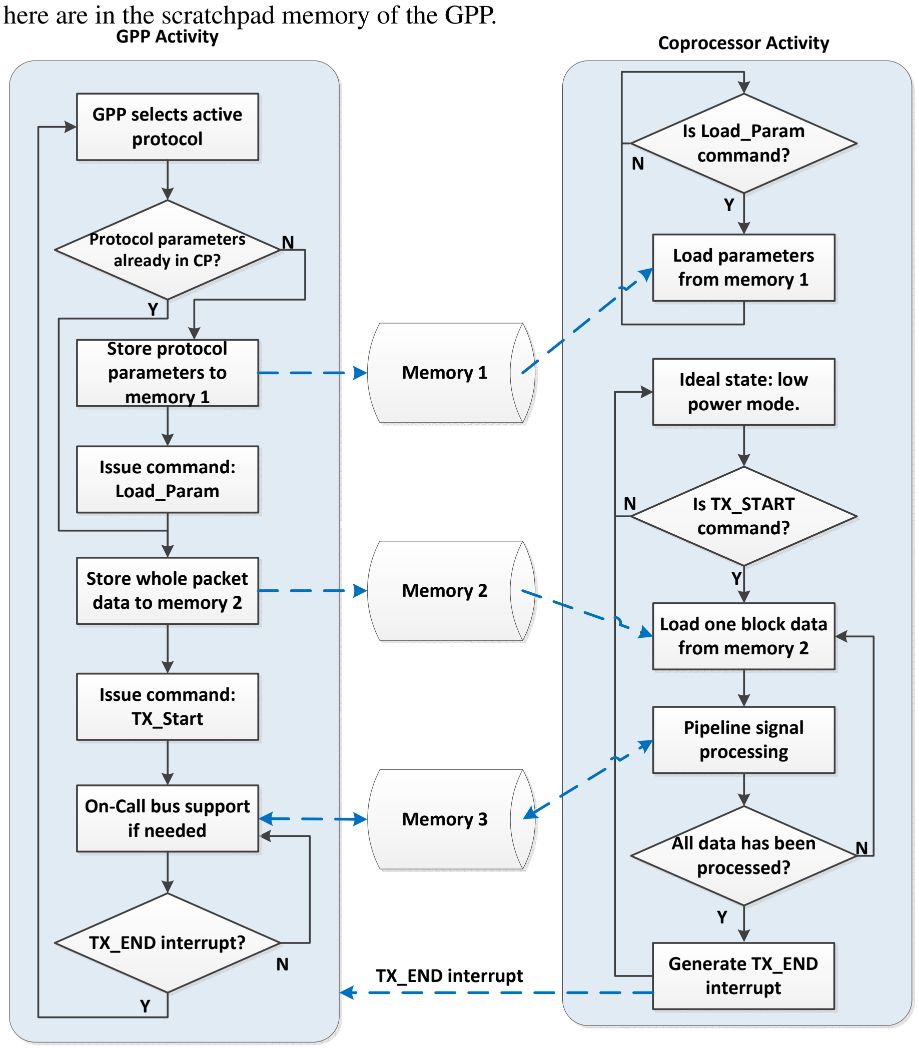
GPP Activity
Coprocessor Activity
GPP selects active protocol
Protocol parameters already in CP?
N
Y
Store protocol parameters to memory 1
Issue command: Load_Param
Store whole packet data to memory 2
Issue command: TX_Start
On-Call bus support if needed
TX_END interrupt?
N
Y
Memory 1
Memory 2
Memory 3
TX_END interrupt
Is Load_Param command?
N
Y
Load parameters from memory 1
Ideal state: low power mode.
Is TX_START command?
N
Y
Load one block data from memory 2
Pipeline signal processing
All data has been processed?
N
Y
Generate TX_END interrupt

Figure 3.5: GPP and coprocessor communication flowchart

4 EXPERIMENTAL SETUP

To evaluate the proposed methodology, the GPP and coprocessor circuits were built. By simulating the different instructions in the GPP, the whole baseband packet payload transmission chain from WLAN [7] (including 802.11a and 802.11b) and UWB [5] were shown to be supported.

Wireless LAN (802.11 protocol), one of the most popular wireless protocols, is selected for evaluation. Modulations in the 802.11 family use Orthogonal Frequency Division Multiplexing (OFDM) in 802.11a and Direct Sequence Spread Spectrum (DSSS) in 802.11b. Thus both 802.11a and 802.11b protocols were chosen. UWB, a high speed wireless communication protocol, which can support 480Mbps data rates, was also selected.

High level block diagrams of 802.11a, 802.11b and UWB are shown in Figure 4.1 according to [7, 5], and all function blocks in these three protocols are mapped into pipelined stages in our coprocessor. The performance of the coprocessor is compared to ASIC implementation of individual protocols, FPGA and the work in [26] which is the only whole transmission chain for multi-mode protocols. Since the CRC, iFFT and Filter blocks are excluded in [26], these two blocks are also removed in our testbench for fair comparison.

We developed the baseband packet payload transmission circuits of these three protocols in RTL, the GPP was developed using the ASIPMeister tools suite [1]. After that the reconfigurable pipelined coprocessor was designed to support 802.11a, 802.11b and UWB. Note that all data rates in these protocols are supported, including data rate 6, 9, 12, 18, 24, 36, 48 and 54Mbps for 802.11a, 1, 2, 5.5 and 11Mbps for 802.11b, 53.3, 80, 106.7, 160, 200, 320, 400 and 480Mbps for UWB. Since 8-bit parallel processing was supported, the highest frequency at which the coprocessor operates is at 240MHz for 480Mbps in UWB.

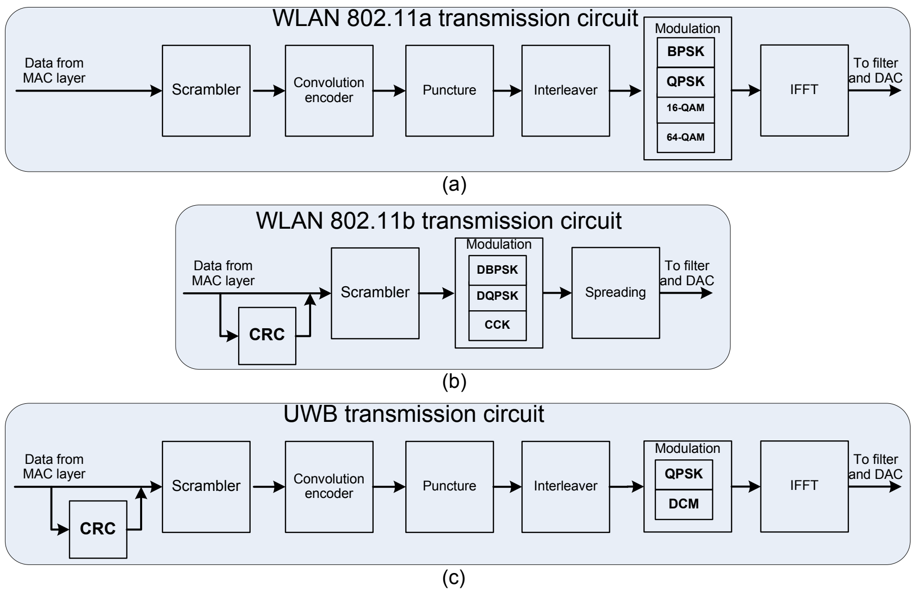

Figure 4.1: 802.11a/802.11b/UWB transmission chain block diagram

The test vectors for the system are generated by Matlab [2] at first. Then, by feeding these test vectors to the individual circuits in RTL simulation tool (ModelSim [3] in our system), the functionality of individual circuits are verified. After that the GPP instructions are manually coded to control the coprocessor. Finally the test vectors generated by Matlab are pumped into the coprocessor via the GPP. The results from Matlab, individual circuit and coprocessor are identical, thus verifying the correctness of coprocessor design. Area and timing results are generated by Synopsys Design Compiler with TSMC 65nm technology, and power consumption is provided by Synopsys Prime Time and Power Compiler. Xilinx Virtex5 XC5VLX50 is selected for FPGA implementation as it is also fabricated by 65nm process, thus providing a fair comparison. The area, timing and power results of the FPGA are from Xilinx ISE and XPower tools.

5 RESULTS and ANALYSIS

5.1 Analysis of coprocessor hardware area & timing

As stated previously, 802.11a, 802.11b and UWB protocols are implemented and mapped to our coprocessor. We compared our result individual implementations of with the three protocols on ASIC and FPGA. Moreover, the datapath merging results from [26] are also compared. The results prove that the area consumption of the coprocessor is better than combination of ASIC individual circuits and datapath merging. The overhead of timing delay is not significant compared to individual ASICs and the datapath merging method.

The circuit area and timing delay results are summarized in Table 5.1. The first column shows the classification of test scenarios. These are: purely individual ASIC implementation; ASIC datapath merging; our coprocessor method; and FPGA implementation. The second column details individual protocols in the ASIC case, and the protocol merging pairs by datapath merging. Note that the merging of only two protocols are supported in [26]. The test scenarios are defined by these first two columns. The third column lists the logic area occupied and the fourth column gives the required memory. The memory is not used for ASIC individual protocol implementation and datapath merging, however, three kinds of memories are used in the coprocessor. First, a 128*8 bits memory is used to store the payload data acting as the interface between the main memory of GPP and coprocessor pipeline stages. Second, two 1200*1 bits memories are used in reconfigurable interleaver block. This is due to the fact that the interleaver block size for 480Mbps data rate in UWB is 1200 bits. Third, there is a 16*12 bits instruction memory in the MAP CPU. In the FPGA scenario, bitstreams of different protocols need to be stored in non-volatile memory (such as flash memory) and only the active protocol's bitstream will be loaded from non-volatile memory to FPGA. To store 802.11a, 802.11b and UWB baseband payload transmission bitstreams, 37.7M bits storage spaces are needed for the Xilinx Virtex-5 XC5VLX50 FPGA. The last column gives the timing delay in each of the scenarios.

From Table 5.1, it can be seen that the logic area of the coprocessor is very small, even smaller than the individual UWB ASIC implementation. The reason is the difference in implementation of the interleaver. In UWB, 1200 registers are used to reorder input bits, which is equivalent to about 6K gates in area consumption. Instead of using registers, two 1200*1 memory blocks are used to improve reconfigurability of the interleaver in the coprocessor, thus the logic area size of the coprocessor can be small. However, if the area contributed by memory is combined with the logic area, the co-

Table 5.1: Area and delay result comparison

Test Case		**Area (k gates)**	**Memory (bits)**	**Delay (ns)**
ASIC	11a	5.7	0	2.4
	11b	3.9	0	1.9
	uwb	12.0	0	2.4
Datapath	11a/11b	5.8	0	2.4
	11a/uwb	12.2	0	2.4
	11b/uwb	12.2	0	2.4
Coprocessor		11.9	3.6K	3.4
FPGA		NA	37.7M	4.5

processor's size will be larger than the size of the UWB circuit. To fairly compare the area consumption between the coprocessor and the ASIC implementations, the sum of the individual ASIC circuits should be considered for multi-mode protocol, instead of single protocol. In this context, the area consumption of the ASIC implementation is 21.6K gates which is the sum of the size of three individual ASIC circuits, which is nearly double the size of the coprocessor. The area of the FPGA implementation is reported by Xilinx ISE and cannot be converted to a number of gates. According ISE, 347 registers, 120 LUTs and 89 slices are consumed by the FPGA implementation.

According to Table 5.1, the timing delay of the coprocessor is about 40% worse than ASIC and datapath merging approaches and about 30% better than FPGA. The main reason for worse timing delay compared to ASIC/datapath is in the reconfigurable interleaver. To support parallel processing, as showing in Figure 3.3, the address of bit 1 has to be generated based on the address of bit 0 within one clock cycle. The bit 1 address generation has the longest datapath, which contributes to the worst timing delay in the coprocessor.

In conclusion, the area consumption of the coprocessor is the smallest among the ASIC, datapath and FPGA approaches. The timing delay of the coprocessor is worse than ASIC and datapath methods, however, the overhead is small. The benefit of the coprocessor is in the ability to be reconfigured. Comparing to the fully reconfigurable FPGA, the coprocessor outperforms FPGA implementations.

5.2 Analysis of power consumption

Low power consumption is a critical factor in mobile devices, the power consumption of the coprocessor has been reduced due to two reasons. First, the logic size of the coprocessor is small as indicated in Table 5.1, thus the static power consumption can be reduced compared to other reconfigurable designs. Second, because parallel implementation is supported by the coprocessor, the clock frequency can be reduced while maintaining the required throughput from different protocols. As clock frequency is one of the main factors of dynamic power, low clock frequency results in low dynamic power consumption.

Table 5.2 gives power consumption comparison between the coprocessor, ASIC and FPGA. The first two columns define the test scenario, i.e. the protocol and data rate. The third column gives the coprocessor working frequency for each scenario. Column four to column six give the power consumption of the coprocessor, individual ASIC and FPGA implementation. Please note that the power consumptions of the ASIC and

FPGA implementation are not available for some test scenario because of the limitation of individual protocol hardware design in our testbench.

Compared to the ASIC, pipelined coprocessor consumes 6X higher power than single protocol ASIC implementation on average as the lower clock frequencies are used in the corresponding ASIC scenarios. For 802.11a and UWB scenarios, four clock cycles are used by the reconfigurable convolutional encoder for every eight parallel input bits. This constrains the clock frequency if the specification defined data rate should be supported. Compared to the ASIC, only one cycle is used to perform convolutional encoding by fixed XOR matrix, thus the power consumption is low in ASIC. However, this comparison is not fair as we are comparing a reconfigurable coprocessor to a single protocol ASIC circuit. For a multi-mode communication circuit implemented in ASIC, the power consumption will be increased if power gating is not used. Power gating can reduce the power consumption of a hibernated protocol, however, the protocol switching time will be longer as the hibernated circuit needs time to be woken up.

When the results of the coprocessor and FPGA are compared together, the pipelined coprocessor outperforms FPGA in the order of magnitude of 100. This result proves the coprocessor methodology is more suitable for mobile devices than FPGA. Moreover, the size of an FPGA is usually fixed from its vendor, such as Xilinx and Altera, hence its area and static power consumption are not able to be reduced even if only a small portion of the FPGA resources are used.

Note that all memory blocks are excluded for this comparison as we lack a consistent memory modeling tool. And the reading of FPGA power consumptions includes the dynamic power for different test scenario and static power for the whole FPGA chip.

5.3 Discussion

The wireless communication baseband tailored reconfigurable
method is provided in this paper. The proposed methodology has greater flexibility than ASICs, with near ASIC performance. It is less flexible than DSP and FPGA, which can support any protocol as long as the computation resources are sufficient. However, since most wireless protocols are similar, the architecture provided in this paper will form an ideal platform for flexible baseband communication chips.

There are limitations in our proposal. Only the baseband transmission path is studied. The receiving path usually contributes more to area and power consumption in the baseband. Our next step is to extend our methodology to the receiving path. The power consumption of the pipelined coprocessor is high comparing with single protocol ASIC implementation. By improving the parallelism of data processing in the pipelined coprocessor, the working frequency can be reduced, thus the power consumption of coprocessor can be reduced. This high parallelism coprocessor is under development.

6 CONCLUSION

In this paper, a two-level reconfigurable pipelined coprocessor is presented to build a reconfigurable baseband circuit in the context of the SDR. We state that this is a highly efficient and flexible solution for multi-mode baseband integration. This methodology

Table 5.2: Power consumption comparison

Protocol	Data rate (Mbps)	CP working Freq (MHz)	Power consumption (mW)		
			CP	ASIC	FPGA
802.11a	6	6	0.73	0.21	530
	9	9	0.92	0.21	531
	12	6	0.73	0.21	530
	18	9	0.92	0.21	531
	24	12	1.11	NA	NA
	36	18	1.48	NA	NA
	48	24	1.86	NA	NA
	54	27	2.05	NA	NA
802.11b	1	11	1.04	0.10	532
	2	11	1.04	0.10	532
	5.5	30.25	2.25	NA	NA
	11	16.5	1.39	NA	NA
UWB	53.3	26.65	2.03	0.66	540
	80	40	2.87	0.68	546
	106.7	53.35	3.71	0.69	552
	160	80	5.38	0.71	563
	200	100	6.64	0.73	572
	320	160	10.40	NA	NA
	400	200	12.90	NA	NA
	480	240	15.50	NA	NA

can save area when integrating multiple protocols without worsening the circuit timing and power performance, while allowing fast mode switching.

Bibliography

[1] Asip solutions asipmeister. Available at: http://www.asip-solutions.com/en/asip_meister.html/.

[2] Mathworks matlab. Available at: http://www.mathworks.com/.

[3] Mentor graphics modelsim. Available at: http://model.com/.

[4] Ti wilink single-chip wlan, gps, bluetooth and fm solution. Available at: www.ti.com/wilink7-pb.

[5] Multiband ofdm physical layer specification, 2005. Available at: http://www.wimedia.org/.

[6] Official ieee 802.11 working group project timelines, 2011. Available at: http://www.ieee802.org/11/Reports/802.11_Timelines.htm.

[7] Wireless lan medium access control (mac) and physical layer (phy) specifications, 2011. Available at: http://standards.ieee.org/.

[8] Paul Coulton and Dylan Carline. An sdr inspired design for the fpga implementation of 802.11a baseband system. In *IEEE Symposium on Consumer Electronics*, 2004.

[9] Mark Cummings and Shinichiro Haruyama. Fpga in the software radio. *IEEE Communications Magazine*, 37:108–112, 1999.

[10] Chris Dick and Fred Harris. Configurable logic for digital communications it's about time. In *Signals, Systems, and Computers, Conference*, 1999.

[11] Andrew Duller, Daniel Towner, Gajinder Panesar, Alan Gray, and Will Robbins. picoarray technology: The tool's story. In *DATE*, 2005.

[12] N.Vasudevan Himanshu Shekhar, C.B.Mahto. Fpga implementation of tunable fft for sdr receiver. In *International Journal of Computer Science and Network Security*, pages 186–190, 2009.

[13] H.R. Karimi, N.W. Anderson, and P. McAndrew. Digital signal processing aspects of software definable radios. In *IEE Colloquium on Adaptable and Multistandard Mobile Radio Terminals*, 1998.

[14] B. Krill and A. Amira. Efficient reconfigurable architectures of generic cyclic convolution. In *B. Krill, A. Amira*, 2011.

[15] Ian Kuon and Jonathan Rose. Measuring the gap between fpgas and asics. In *FPGA '06: Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, pages 21–30, New York, NY, USA, 2006. ACM.

[16] Jui-Chieh Lin, Chu Yu, Mao-Hsu Yen, Pao-Ann Hsiung, Sao-Jie Chen, and Yu Hen Hu. Parallel implementation of convolution encoder for software defined radio on dsp architecture. In *ICSAMOS*, pages 180–186, 2009.

[17] Yuan Lin, Hyunseok Lee, Mark Woh, Yoav Harel, Scott Mahlke, Trevor Mudge, Chaitali Chakrabarti, and Krisztian Flautner. Soda: A high-performance dsp architecture for software-defined radio. *IEEE Micro*, 27:114–123, 2007.

[18] M.I.Taj, O.Hammami, and M.Akil. Sdr waveform components implementation on single fpga multiprocessor platform. In *ICECS*, pages 790 – 793, Dec. 2010.

[19] Nahri Moreano, Edson Borin, Cid De Souza, and Guido Araujo. Efficient datapath merging for partially reconfigurable architectures. In *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, pages 969–980, 2005.

[20] Ladimer S. Nagurney. Software defined radio in the electrical and computer engineering curriculum. In *Proceedings of the 39th IEEE international conference on Frontiers in education conference*, FIE'09, 2009.

[21] J.G. Proakis. *Digital communications*. McGraw-Hill series in electrical and computer engineering. McGraw-Hill, 2001.

[22] D. Pulley and R. Baines. Software defined baseband processing for 3g base stations. In *4th International Conference on 3G Mobile Communication Technologie*, 2003.

[23] Daniel A. Reed, James R. Larus, and Dennis Gannon. Imagining the future: Thoughts on computing. *IEEE Computer*, 45(1):25–30, 2012.

[24] Carlos R. Sanchez-Ortiz, R. Parra-Michel, and M.E. Guzman-Renteria. Design and implementation of a multi-standard interleaver for 802.11a, 802.11n, 802.16e & dvb standards. *Reconfigurable Computing and FPGAs, International Conference on*, 0:379–384, 2008.

[25] K.G. Smitha, A. P. Vinod, and R. Mahesh. Reconfigurable area and power efficient i-q mapper for adaptive modulation. In *International Midwest Symposium on Circuits and Systems*, 2011.

[26] Liang Tang, Jorgen Peddersen, and Sri Parameswaran. A rapid methodology for multimode communication circuit generation. In *VLSI Design*, 2012.

[27] Bijoy Kumar Upadhyaya and Salil Kumar Sanyal. Design of a novel fsm based reconfigurable multimode interleaver for wlan application. In *International Conference on Devices and Communications*, 2011.