Designing and Mapping of a Turbo Decoder for 3G Mobile Systems Using Dynamically Reconfigurable Architecture

Mingwei Ding, Ahmad Alsolaim, and Janusz Starzyk
Electrical Engineering and Computer Science
Ohio University, USA
{imding|alsolaim|starzyk}@bobcat.ent.ohiou.edu

Abstract
Turbo codes enable high quality communication links by offering exceptional error correction capabilities. Turbo coding is proposed in the coming Universal Mobile Telecommunication Systems (UMTS) standard by the 3GPP for high data rates channels. Direct realization of the Turbo decoder algorithm in hardware encounters many challenges, such as algorithm complexity, large silicon area, high power consumption and low throughput. Reconfigurable computing is emerging as a favorable solution that provides flexibility and power savings without noticeable sacrifice in performance. As a possible application, a Turbo decoder compatible with 3GPP standard is designed and mapped onto new dynamically reconfigurable architecture that is specifically designed for the 3rd generation and future wireless mobile terminals. The implementation exploits the parallelism in the Turbo decoder algorithm and shows the possibilities of the architecture to be dynamically reconfigured in time and space. Simulation results are given.

Key Words: Turbo decoding, UMTS, Reconfigurable computing, Dynamic reconfiguration.

1. Introduction

Since its introduction in 1993 by Berrou et al. [1], Turbo code has generated a significant interest in research and industry for its outstanding error correction capability that is close to the Shannon limit. Turbo codes are currently being proposed in many application standards, including the UMTS standard for third generation wireless mobile system developed by 3GPP [2][3]. The standard calls for a flexible implementation of a Turbo coder/decoder with the number of bits/frame varying from 40 to 5114. Turbo codes proposed in the 3GPP UMTS standard are Parallel Concatenated Convolutional Codes (PCCC). In the literature there are many implementation attempts [4][5][6][7]aiming to address both power and adaptability of Turbo decoder.

Dynamically Reconfigurable Computing (DRC) offers a suitable implementation platform for the mobile wireless terminal in general and for Turbo decoder in particular. By dynamically swapping different blocks of a Turbo decoder in time, smaller area and significant power saving can be gained. This is an alternative to power saving techniques where unused parts of a system are powered down. Those powered down subsystems still occupy a valuable area which in DRC architecture can be reused to run other applications. We have developed a new Dynamically Reconfigurable Architecture for wireless communication called DRAW. The DRAW architecture is derived from our prior work on a similar DRC architecture called DReAM [8][9].

This paper is organized as follows. In section 2, it outlines a general description of the Turbo codes specified in the UMTS standards. Section 3 briefly describes our DRAW architecture including its distinctive features which makes it a suitable implementation platform for the targeted application. In addition a general description of the advantages of DRC platforms in the mobile wireless terminals is given. A detailed description of the implementation issues and challenges is given in section 4. The last section explains our implementation methods in DRAW and the possible processing rate achievable using DRAW. Then a conclusion section is given to summarize our findings.

2. Dynamically Reconfigurable Architecture for Wireless Applications (DRAW)

In the area of system design, there is a growing gap between what is available in terms of silicon processing power and the design productivity. To address this problem, a new processing model must be used to increase the design productivity. 3G wireless mobile applications require a flexible implementation solution, small area, low power consumption and low cost. Design and hardware reuse must be exploited to speed up the design process and to reduce the area requirements, which will reduce the coast of the solution. Flexibility is a major requirement for any proposed solution to survive the fast changing standards, and the introduction of many new services.

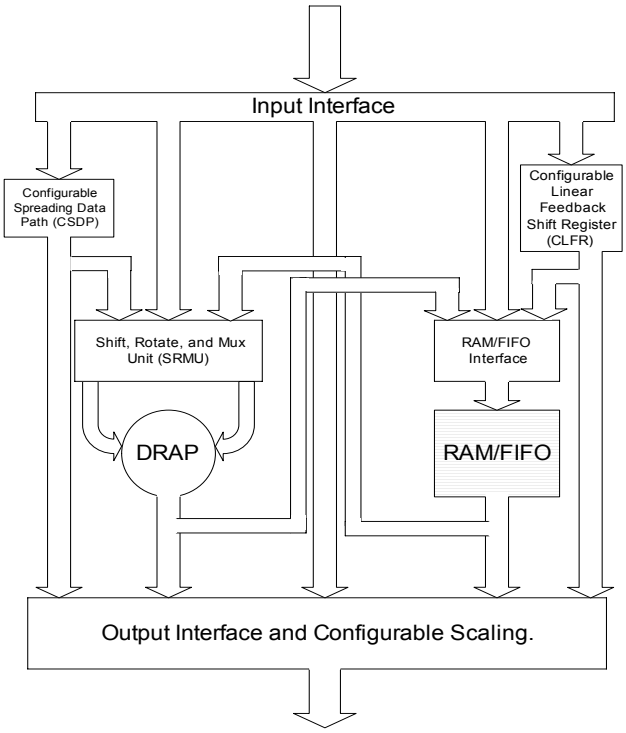

Figure 1: A block diagram of the DRPU.

Based on our previous work on dynamically reconfigurable architecture called DReAM [8][9], we designed a new optimized version called DRAW. This architecture is specially designed for future wireless mobile terminal. The DRAW architecture is optimized in many aspects. The most noticeable (and directly related to this work) is that the Dynamically Reconfigurable Processing Unit (DRPU) is smaller yet more powerful than its predecessor. As a result more DRPU units can be built in the SOC architecture, which in turn provides more powerful and flexible processing resources. DRAW also supports IP-based mapping by providing a regular hardware architecture, and fast (in nano seconds) dynamic reconfiguration.

Figure 1 shows the block diagram of the DRPU. The input interface handles selection and routing of the incoming data to the inside of the DRPU. The output interface provides a routing of the processed data and their normalization. Normalization by power of two and modular normalization are both supported. The Configurable Spreading Data Path (CSDP) and the Configurable Liner Feedback Shift Register (CLFR) are designed to handle the spreading/dispreading and generation of codes. The reason to have a special unit for these two important tasks is that they are one bit processing type operations, and it would be a waste of resources to handle them with the 16-bit-based Dynamically Reconfigurable Arithmetic Processing (DRAP) unit. A special storage unit is also available in the DRPU that can be configured into FIFO or RAM functionality. In addition, an interface module is shown in Figure 1 for the DRAP and the RAM/FIFO units. The DRAP interface provides a routing selections and shifting/rotating operation. The RAM/FIFO interface provides the addressing mechanism for the RAM/FIFO unit. The DRAP implements all the arithmetic and bitwise operations shown in Table 1. As can be seen from this table, the DRAP can directly implement many vital functions for base band processing of the 3G and future wireless mobile terminals. In Turbo decoder, a chosen example application for DRAW, the addition and the MAX operations are the major functions used. Other functions are useful in many other applications like Rake receiver and spreading units.

Table 1: A list of operations supported by the DRAP unit.

#	Operation	Description
1	MUL	2's complement Multiplication
2	ADD	2's complement addition
3	SUB	2's complement subtraction
4	SHIFT	Logic & Arithmetic shift

5	ROTATE	Rotate
6	AND	Bit-wise and
7	NAND	Bit-wise nand
8	OR	Bit-wise or
9	NOR	Bit-wise nor
10	XOR	Bit-wise xor
11	XNOR	Bit-wise xnor
12	NOT	Bit-wise not
13	MIN and MAX	Minimum, Maximum

3. Turbo Decoding Algorithm

There are two types of procedures to generate a concatenated Convolutional coding. The two types are the Serial Concatenated Convolutional Coding (SCCC) and the Parallel Convolutional Coding (PCCC). The PCCC scheme is the one proposed in the 3GPP standard [2]. This type of Turbo codes can be decoded in an iterative manner [10][11]. The encoding of the data bits is performed by using two Recursive Systematic Coders (RSC) concatenated in a parallel, as shown in Figure 2. This encoder is rate 1/3 encoder, which means that for every input bit, three output bits are generated.

The three output signals generated by the encoder are called, systematic (original signal), and redundant signals (Parity bits P1, and P2). The input to the second RSC2 is first interleaved using a random-like interleaving process. The encoder operates on a block of bits, which is between 40 to 5114 bits [1] long [2][3]. Note that puncturing is omitted in this paper for the sake of simplicity.

General Turbo decoder structure is shown in Figure 3. The Turbo decoder consists of two Soft-Input Soft-Output (SISO) decoders. An SISO decoder is capable of computing the *a posteriori* likelihood of the constituent codes. The original Maximum *a posteriori* (MAP) algorithm is very expensive to implement, due to the required multiplication and exponential operations [12][13]. Therefore, we choose Max-Log-MAP algorithm [10] as the algorithm of implementation, and we use sliding window technique to reduce the storage requirements.

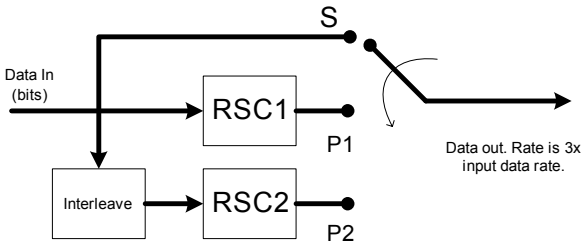

Figure 2: PCCC encoder proposed by 3GPP

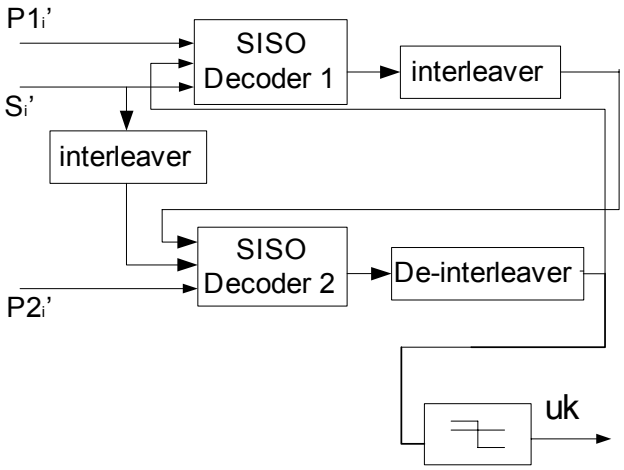

Figure 3: Turbo Decoder structure

According to BCJR algorithm [1], the calculation of *a- posterior* probability p can be decomposed to calculation of α , β and γ parameters, as follows:

$$p(s',s,y) = \alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s).$$

Where s and s' are the encoder states at time k , k-1, respectively, and

$$\alpha_k(s) = \sum \alpha_{k-1}(s')\gamma_k(s',s) \quad (1)$$

$$\beta_{k-1}(s) = \sum \beta_k(s')\gamma_k(s',s) \quad (2)$$

In Max-Log-MAP algorithm, calculation of α , β and γ is simplified to performing a sequence of addition, subtraction and maximum operations. Figure 4 shows the datapath for the calculation of

[1] The Turbo encoder operates on bits, while the decoder operates on symbols. We use the term "bit" at the decoder to correspond to the output bit after completing all iterations, while the internal processing is performed on 8-bits quantized symbols.

α parameter for state m at time k. β has the same datapath as α.

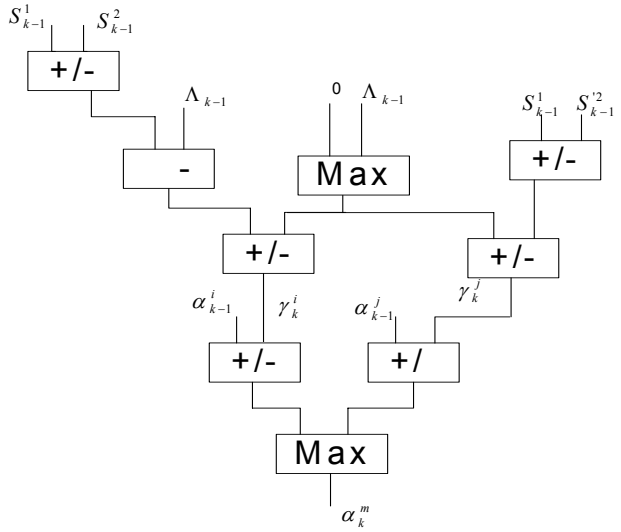

Figure 4: Datapath for α calculation.

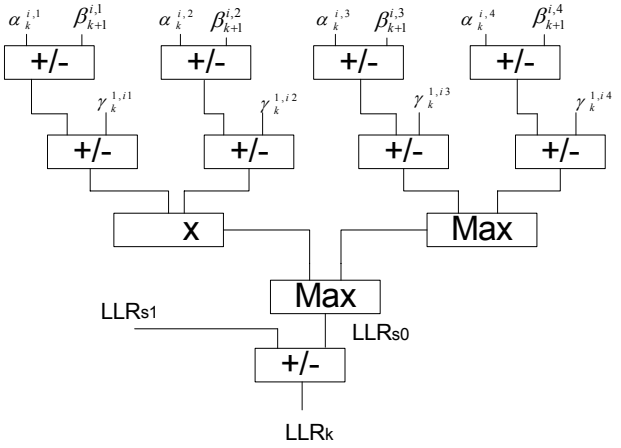

Figure 5: Datapath for LLR (4 states)

Similarly, the Log-Likelihood Ratio (LLR) can be calculated as

$$LLR_k = \underset{S1}{Max}[\alpha_k(s) + \gamma_k(s,s') + \beta_{k+1}(s)] - \underset{S0}{Max}[\alpha_k(s) + \gamma_k(s,s') + \beta_{k+1}(s)]$$

where S1 stands for all the states related to information bit 1, while S0 stands for all the states related to information bit 0. For simplicity, Figure 5 shows the datapath for LLR for only 4 states.

Figure 6 shows the sliding window technique, in which whole frame is divided into several sliding windows accompanied with training window, α is computed in forward recursion as usual, while β is computed from end of each training window instead of from end of frame backward to the head of each sliding window. In this way, the calculation of LLR can be performed as soon as β backward recursion is finished. Then move to the next sliding window performing the same operation. Thus output delay can be reduced to proportional to the size of sliding window N_s instead of frame length L.

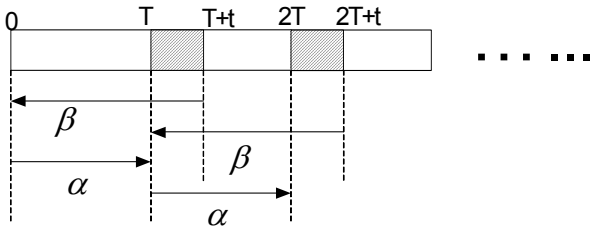

Figure 6: Sliding window scheme

Considering decoding delay for SISO module, there will be a slight difference between first iteration and sub-sequent iterations. For the first iteration, it will take about $k(2N_s+2N_t)$ cycles to get the first block of output, where k is the number of clock cycles needed to process every information bit, N_s is the sliding window size, and N_t is the training window size. In the subsequent iterations decoding delay can be reduced to $k(N_s+N_t)$ cycles, since data is already stored and ready.

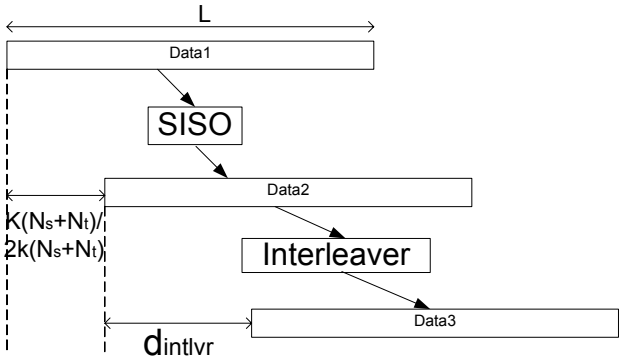

Figure 7: SISO decoding delay

As illustrated in Figure 7, decoding delay per iteration(in clock cycles) is

$$\text{Delay} = \begin{cases} k \cdot (2N_s + 2N_t) + d_{intlvr} & n = 1 \\ k \cdot (N_s + N_t) + d_{intlvr} & n > 1 \end{cases} \quad (3)$$

where n is the iteration number for whole decoder, d_{intlvr} is the processing delay for interleaver (de-interleaver) and is affected by the size of frame length L as shown in Figure 7. According to 3GPP

standard, the size of permutation matrix is decided by L[2]. Assuming the matrix has R rows and C columns, d_{intlvr} can be expressed as C(R-1)+1 cycles.

The implementation of the Turbo decoder needs to be flexible, small in area, and low in power consumption. Adaptability is highly desired since in a typical baseband processing, Turbo coding is not always required. According to the 3GPP standards, Turbo coding is only used for high quality data links. This means that implementing the decoder in ASIC is a waste of area and power whenever the high quality link is not established. In addition at the time of the design of a Turbo decoder, a tradeoff between the number of iterations (processing efficiency) and the memory requirement (power and area) needs to be dealt with. This will lead to situations when the quality of the communication link changes, and a different design of the decoder would result in either less area or better throughput or both. In DRC we can design different decoders with different tradeoffs in mind, and as the condition of the communication links change, an upper layer in a wireless terminal would select the appropriate decoder in a way that balances these two implementation aspects.

4. Implementation Issues

The cost of implementing the above designed decoder into DRAW is as follows.

1 Memory requirement for one MAP

The storage for α is $S \cdot N_s$ bytes where S is the number of encoder states, which according to [2], will be 8. The storage for data is 3L bytes, where L is the frame length. In our design, one DRPU has 16 byte memory, so it takes $(8N_s+3L)/16$ DRPUs. Storage for interleaver can be estimated as L bytes.

2 Arithmetic requirement for one MAP

Area for α (β) computing is $(6+3)\times S$ =72 DRPUs and for LLR it is $2(3S-1)+1$ =47 DRPUs, so total area for one MAP module is 119 DRPUs. Consequently, the mapping of SISO module to our architecture will take an 8 × 15 array of DRPUs. From Figure 3, we can see, one input of the MAP module is the output of (de)interleaver, thus total required DRPUs for a Turbo decoder would not incur extra storage cost for (de)interleaver and it will be $2\text{Max}[(8N_s+3L)/16, 119]$.

3 Timing requirement for one MAP

Figure 4 shows that the calculation of α or β costs 5 clock cycles, and from Figure 5 we can figure out that the calculation of LLR for 8 states, whenever α , β , γ are ready, takes 6 clock cycles. So the computation cost for LLR is 16 cycles/iteration/bit. (Computing α , β in parallel can achieve faster speed at the cost of extra area, which is a possible option) Based on our analysis, the processing delay δ for decoder is

$$\begin{aligned}\delta &= 16(2N_s+2N_t)+2[C(R-1)+1] \\ &\quad +(n-1)\{16(N_s+N_t)+[C(R-1)+1]\} \\ &= (n+1)[16(N_s+N_t)+C(R-1)+1] \text{ (cycles)} \quad (4)\end{aligned}$$

Note that C(R-1)+1 is usually close or comparable to L. We have

$$\delta = (n+1)[16(N_s+N_t)+L] \quad (5)$$

This gives a good estimate of δ , and can be useful to guide for N_s, and N_t choice with respect to L. From the above discussion, we note the choice of n, N_s and N_t should consider all these three factors balancing the area and speed constraint. And we also can see that the throughput bottleneck will shift to (de)interleaver as L increases, and data storage will have more effect on the processing delay.

Our simulation shows the following results: Max-Log-MAP, r=1/3, S=8 states, frame length L=512, sliding window N_s=88, N_t =12, after n=3 iterations, bit error rate reaches a level of 10^{-5}.

Using design parameters mentioned above, total number of DRPUs required for Turbo decoder application is $2\text{Max}[(8N_s+3L)/16, 119]$=280. A 15×19 array of DRPUs including other auxiliary units (like configuration unit, switch box, etc.) can be fabricated on a single chip.

From (5), we can find that for frame length of 512 bits, the required clock rate is about 2.5MHz for 10 iterations. According to 3GPP standard, maximum frame length is 5114 bits, we can estimate the required clock rate will be 7.4MHz for 10 iterations, which is well within DRAW's capability.

The Data Flow Graph (DFG) of the Turbo decoder (for one iteration) can be seen in Figure 8. The diagram shows the way the DRAW architecture is configured to process different processing steps

of the Turbo decoder assuming a fixed number of iterations. The DRAW architecture provides three levels of configuration context. The configuration contexts are stored in close vicinity to the DRPU. A new configuration can be loaded into the DRPU in three clock cycles. That is approximately 20 ns using a clock rate of 150 MHz.

5. Conclusion

Turbo decoder performance is very essential to the full implementation of 3G and future generation wireless mobile systems. A flexible and efficient implementation of the Turbo decoder that is compatible with 3GPP standard has been designed and implemented as an example application on a DRAW architecture. The proposed architecture is simulated using VHDL to verify the performance and functionality. A saving in area without degradation in performance is accomplished.

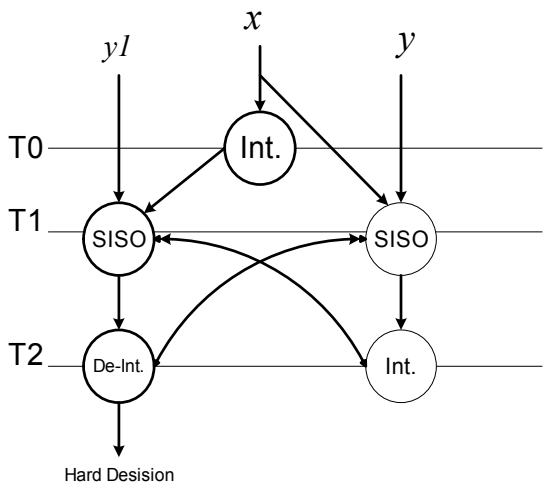

Figure 8: DFG of the Turbo decoder.

6. References

[1] C. Berrou, A. Glavieux, P. Thitimajshima, "Near Shannon limit error correcting coding and decoding: Turbo codes", Proc. 1993 Inter. Conf. Communication, pp. 1064-1070, May 1993.

[2] 3GPP TG 25.212 http://www.3gpp.org/.

[3] H. Michel, N. When, "Turbo-Decoder Quantization for UMTS," IEEE Communications Letters, Feb., 2001, Vol. 5, No. 2.

[4] W. Gross, P. Gulak, "Simplified MAP Algorithm Suitable for Implementation of Turbo Decoders", Electronics Letters, Vol. 34. No. 16. Aug. 1998. pp. 1577-1578

[5] T. Ngo, I. Verbauwhede "Fixed Point implementtation for Turbo codes," IEEE Micro 98-168 project report. Feb. 2000.

[6] F. Viglione, G. Masera, G. Piccinini, M. Ruo Roch, M. Zamboni, "A 50 Mbit/s Iterative Turbo-Decoder", Proc. DATE2000 Conf., Paris, Mar. 2000, pp. 176-180.

[7] G. Masera, G. Piccinini, M. Roch, M. Zamboni, "VLSI Architecture for Turbo Codes", IEEE Trans. On VLSI systems, Vol. 7, No. 3, Sep. 1999.

[8] A. Alsolaim, J. Becker, M. Glesner, J. Starzyk, "A Dynamically Reconfigurable System-on-Chip Architecture for Future Mobile Digital Signal Processing," Proc. European Signal Processing Conf. (EUSIPCO 2000), Tampere, Finland, Sep.4-8, 2000.

[9] A. Alsolaim, J. Becker, M. Glesner, J. Starzyk, "Architecture and Application of a Dynamically Reconfigurable Hardware Array for Future Mobile Communication Systems", Proc. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM´00), Napa, USA, Apr. 17-19, 2000".

[10] A. Chass and A. Gubeskys, "Efficient Software Implementation of Max-Log-MAP Turbo Decoding Algorithm on StarCore SC140 DSP", Motorola Semiconductor Israel, Ltd., ICSPAT 2000 Israel.

[11] G. Masera, M. Mazza, G. Piccinini, F. Viglione, M. Zamboni, "Low-Cost IP-blocks for UMTS Turbo decoders", Proc. ESSCIRC 2001, Villach, Austria, Sep. 2001.

[12] A. Worm, H. Lamm and N. When, "Design of Low-Power High-Speed Maximum a Posteriori Architectures" Proc. Design, Automation and Test in Europe (DATE) Conf. 2001, pp. 258-265, Munich, Germany, Mar. 2001.

[13] P. Robertson, P. Hoeher, E. Villeburn, "Optimal and suboptimal maximum a posterior algorithm suitable for Turbo decoding", European Trans. On Telecommunication, vol. 8, 1997. pp. 119-125.