NEW DIGITAL CODING ALGORITHMS UNDER MORPHOSYS

Hassan Diab[1] and Issam Damaj[1]

1:Department of Electrical and Computer Eng'g, Faculty of Engineering and Architecture American University of Beirut

P.O.Box 11-0236, Beirut, Lebanon
diab@aub.edu.lb; issamwd@ieee.org

ABSTRACT

At one extreme of the computing spectrum, we have general-purpose processors that are programmed entirely through software. At the other extreme are application-specific ICs (ASICS) that are custom designed for particular applications. General-purpose processors are designed to execute any application. On the other hand, ASICS are custom hardware circuits. They provide the precise function needed for a specific task. Combining the flexibility of general-purpose processors and the high performance of ASICS would lead to the desired goal. Consequently, this led to the introduction of reconfigurable computing (RC). The MorphoSys is one example of an RC system, which combines a reconfigurable array of processor cells with a RISC processor core and a high bandwidth memory interface unit. This paper introduces two new coding algorithms using reconfigurable computing (RC) and specifically chooses one of the prototypes in this field, MorphoSys (M1) [Bagherzadeh, 1998]. A performance analysis study of the M1 RC is also presented to evaluate the execution efficiency of the suggested algorithms on the M1 system. The mapped algorithms deal namely with checksum coding, and linear sequential coding. Examples (64-input bytes vector on the 8x8 RC array M1) were run, to validate our results, using the MorphoSys mULATE program, which simulates MorphoSys operations.

Keywords: *Digital Coding, Checksum, Linear Sequential Circuits, Reconfigurable Computers, MorphoSys, Reconfigurable Cells, Reconfigurable Array.*

الملخص

وحدات المعالجة متعددة الأنواع، هنالك وحدات المعالجة المتعددة الأهداف التي تبرمج كليةً بواسطة البرامج. وهناك ايضاً وحدات معالجة متخصصة (ASICs) ، وهي تعالج تطبيقات محددة. لقد صممت وحدات المعالجة المتعددة الاهداف من أجل تنفيذ أي برنامج. على الصعيد الآخر, فإن وحدات المعالجة المتخصصة هي دوائر الكترونية لعمليات محدده. دمج الإيجابيات في كلا النوعين (المرونة، والاداء العالي) تؤدي إلى معالجات ذات مقدرة متميزة، بالتالي هذا الدمج أدى الى اصدار معالجات (Reconfigurable). موروفوسيس (M1) هو إحدى معالجات (Reconfigurable). يقدم هذا البحث طرق جديدة (خوارزميات) للتشفير الرقمي. الخوارزميات المطروحه هي طريقة التشفير(Checksum) وطريقة التشفير (Linear Sequential Circuits). إختبارات عدديّه أجريت بإستخدام مصفوفات تتألف من ٦٤-بايت. بالإضافة إلى دراسة تحليله اجريت للخوارزميات المطروحة. الدراسة اجريت باستخدام برنامج المحاكات المخصص للمعالج موفوسيس (mULATE).

1. INTRODUCTION

Reconfigurable computing (RC) is becoming more popular and increasing research efforts are being invested in it. It employs reconfigurable hardware and programmable processors. The user designs the program in a way where the workload is divided between the general-purpose processor and the reconfigurable device. The use of RC opens the way for an increased speed over general-purpose processors and a wider functionality than application specific integrated circuits (ASICs). It is a good solution for applications requiring a wide range of functionality and speed at the same time.

Reconfigurable computers (RCs) offer the potential to greatly accelerate the execution of a wide variety of applications. Its key feature is the ability to perform computations in hardware to increase performance, while retaining much of the flexibility of a software solution. Reconfigurable computing systems are being investigated in many areas like: computer graphics, digital signal processing, digital image processing, communications security, DNA matching, and molecular modelling. For the M1 RC-system, the targeted areas of application are: computer graphics, digital signal processing, information coding, cryptography, and more [Eliseu, 2000] [Bagherzadeh, 1998] [Damaj, 2001] [Abdennour, 2000] [Maestre, 1999] [Bagherzadeh, 1999].

2. MORPHOSYS DESIGN

One of the emerging RC systems includes the MorphoSys designed and implemented at the University of California, Irvine. It has the block diagram shown in Figure 1 [Bagherzadeh, 1998]. It is composed of: 1) an array of reconfigurable cells called the RC array, 2) its configuration data memory called context memory, 3) a control processor (TinyRISC), 4) a data buffer called the frame buffer, and 5) a DMA controller.

A program runs on MorphoSys in the following manner: General-purpose operations are handled by the TinyRISC processor, while operations that have a certain degree of parallelism, regularity, or intensive computations are mapped to the RC array. The TinyRISC processor controls, through the DMA controller, the loading of the context words to context memory. These context words define the function and connectivity of the cells in the RC array. The processor also initiates the loading of application data, such as image frames, from main memory to the frame buffer. This is also done through the DMA controller. Now that both configuration and application data is ready, the TinyRISC processor instructs the RC array to start execution. The RC array performs the needed operation on the application data and writes it back to the frame buffer. The RC array loads new application data from the frame buffer and possibly new configuration data from context memory. Since the frame buffer is divided into two sets, new application data can be loaded into it without interrupting the operation of the RC array. Configuration data is also loaded into context memory.

Top

The MorphoSys emulator is written in C++. The graphical user interface is written using Gtk. Gtk is available in most UNIX platforms. The program is distributed with its source code as a zipped tar file. This emulator is called mULATE it's a very efficient tool for testing and analyzing algorithms [Damaj, 2001].

3. RECONFIGURABLE DEVICE

As stated earlier, the reconfigurable device in MorphoSys is the RC array divided into four quadrants. The interconnection network is built on three hierarchical levels. The first is a nearest neighbour layer that connects the RCs in a 2-D mesh. The second is an intra-quadrant connection that connects a specific RC to any other RC in its row or column in the same quadrant. The third is an inter-quadrant connection that carries data from any one cell (out of four) in a row (or column) of a quadrant to other cells in an adjacent quadrant but in the same row (or column). The context words present on context memory configure the function of the RCs as well as the interconnection, thus specifying where their input is from and where their output will be written [Eliseu, 2000].

4. CODING APPLICATIONS

With the presence of high speed data communications devices, the need for more processing power is essential. With the emergence of extremely powerful reconfigurable systems, the use of such systems would grant the ever increasing demand on high coding speed. The coding techniques presented here are the checksum coding algorithm, and an example on mapping linear sequential circuits onto the RC-array. Other coding algorithms like cyclic coding, and turbo coding along with the addressed algorithms would give a complete view about the use of the M1 system in digital coding.

4.1. Checksum Coding Technique

Checksum codes are generated by appending n-bit words, called the checksum words, to a block of m-(n-bit) data words, formed as the sum of the m words using modulo 2 addition. On one hand, encoding using the checksum coding technique means to use an n-bit arithmetic adder to add the m data words, with any carry beyond the nth bit being discarded. The sum is then appended to the data block as the n-bit checksum. On the other hand, decoding means to use the same n-bit arithmetic adder to add the m data words and XOR the sum with the appended checksum, if the result is zero, then there is no error. In this paper we introduce two parallel algorithms to perform the encoding and the decoding and their mapping onto the M1 reconfigurable cells array.

Top

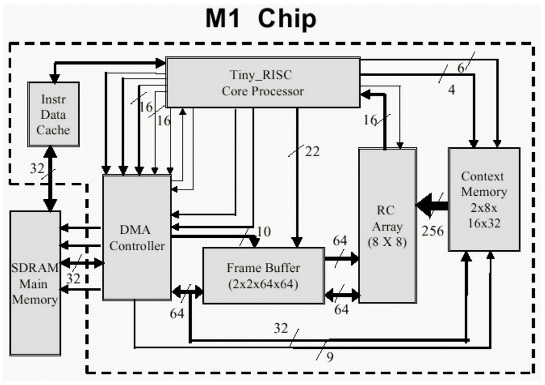

Figure 1. MorphoSys Block Diagram.

Simply, adding in parallel two vectors of bytes performs the encoding part. Generally, a one-dimensional vector has the form: $M^T = [M_0 M_1 M_2 .....M_n]$, which includes n-elements. Mapping an algorithm for checksum is done by first storing the encoded and the coding vector both in the Frame Buffer set “0” and set “1”. Then we can exploit the properties of the interconnection, where some contents of Frame Buffer set “0” are added to some contents of Frame Buffer set “1” and the result would be in columns 0-7 of the RC-array. Figure 2 shows the final output in the RC-array after running the algorithm of encoding a 64-element vector. The desired function of the interconnection is: Out = U + V; the context word of this operation is loaded through the context memory.

The decoder part contains the same encoding algorithm but with an addition of XORing the result with the received checksum, if the result is all zeros then there are no errors in the received bytes. This is done by XORing the two vectors of Bytes available in the Frame Buffer the one resultant from the addition and the one stored for the checking. The code and explanation for the decoding algorithm is shown in tables 1 and 2, for the M1 and some Intel MPUs.

Columns Rows	C_0	…	C_7
R_0	U_0+V_0	.	.
R_1	U_1+V_1	.	.
R_2	U_2+V_2	.	.
R_3	U_3+V_3	.	.
R_4	U_4+V_4	.	.
R_5	U_5+V_5	.	.
R_6	U_6+V_6	.	.
R_7	U_7+V_7	.	$U_{63}+V_{63}$

Figure 2. RC array contents after encoding.

Table 1. The TinyRISC code for the decoding algorithm.

0:	ldui	r1, 0x1;	R1 ← 10000_{hex}. vector U is stored.
1:	ldfb	r1, 0, 0, 16 ;	FB ← 16 x 32 bits at set 0, bank A, address 0.
2:	add	r0, r0, r0;	No-operation.
.	.	.	.
33:	ldui	r1, 0x2;	R1 ← 20000_{hex}. This is where vector V is stored.
34:	ldfb	r1, 1, 0, 16;	FB ← 16 x 32 bits at set 0, bank B, address 0.
35:	add	r0, r0, r0;	NOP
.	.	.	.
66:	ldui	r3, 0x3;	R3 ← 30000_{hex}. This is where the context word is stored in main memory. To perform the addition part of the routine.
67:	ldctxt	r3, 0, 0, 0, 1;	Load one context word from main memory starting at the address stored in register 3 into plane 0, block 0 and starting at word 0.
68:	add	r0, r0, r0;	NOP
.	.	.	.
71:	ldui	r4, 0x0;	R4 ← 00000_{hex}.
72:	dbcdc	r4, 0, 0, 0, 0, 0, 0;	Double bank column broadcast. It sends data from both banks address 0 in the frame buffer and broadcasts the context words column-wise. It triggers the RC array to start execution of column 0 by the context word of address 0 in the column block of context memory operating on data in set 0. Bank A starting at 0x0. Bank B starting at (0x0 + 0).
73:	ldli	r4, 0x4	R4 ← 4_{hex}
74:	dbcdc	r4, 0, 0, 1, 0, 0, 0x40;	It sends data from both banks address 40_{hex} in the frame buffer. Bank A starting at 0x40. Bank B starting at (0x4 + 0x0 = 0x40).
75:	ldli	r4, 0x8	R4 ← 8_{hex}
76:	dbcdc	r4, 0, 0, 2, 0, 0, 0x80;	It sends data from both banks.
77:	ldli	r4, 0xC	R4 ← C_{hex}
78:	dbcdc	r4, 0, 0, 3, 0, 0, 0xC0;	It sends data from both banks address $C0_{hex}$ in the frame buffer. Bank A starting at 0xC0. Bank B starting at (0xC + 0x0 = 0xC0).
79:	ldli	r4, 0x10	R4 ← 10
80:	dbcdc	r4, 0, 0, 4, 0, 0, 0x100;	It sends data from both banks address 100_{hex} in the frame buffer. Bank A starting at 0x100. Bank B starting at (0x10 + 0x0 = 0x100).
81:	ldli	r4, 0x14	R4 ← 14_{hex}
82:	dbcdc	r4, 0, 0, 5, 0, 0, 0x140;	It sends data from both banks address 140_{hex} in the frame buffer. Bank A starting at 0x140. Bank B starting at (0x14 + 0x0 = 0x140).

Top

83:	ldli	r4, 0x18	R4 ← 18_{hex}
84:	dbcdc	r4, 0, 0, 6, 0, 0, 0x180;	It sends data from both banks.
85:	ldli	r4, 0x1C	R4 ← $1C_{hex}$
86:	dbcdc	r4, 0, 0, 7, 0, 0, 0x1C0;	It sends data from both banks.
87:	wfbi	0, 0, 0, 1, 0x0;	Write data back to the frame buffer from the output registers of column 0 into set 0, address 0, FB A.
88:	wfbi	1, 0, 0, 0, 0x40;	of column 1 into set 0, address 64.
89:	wfbi	2, 0, 0, 0, 0x80;	of column 2 into set 0, address 128.
90:	wfbi	3, 0, 0, 0, 0xC0;	of column 3 into set 0, address 192.
91:	wfbi	4, 0, 0, 0, 0x100;	of column 4 into set 0, address 256.
92:	wfbi	5, 0, 0, 0, 0x140;	of column 5 into set 0, address 320.
93:	wfbi	6, 0, 0, 0, 0x180;	of column 6 into set 0, address 384.
94:	wfbi	7, 0, 0, 0, 0x1C0;	of column 7 into set 0, address 448.
95:	ldui	r1, 0x1;	R1 ← 10000_{hex}. vector U is stored.
96:	ldfb	r1, 1, 0, 16 ;	FB ← 16 x 32 bits at set 0, bank B, address 0. Where the checksum bytes are stored.
97:	add	r0, r0, r0;	No-operation.
.	.	.	.
128:	ldui	r3, 0x3;	R3 ← 30000_{hex}. This is where the context word is stored in mem.
129:	ldctxt	r3, 0, 0, 0, 1;	Load one context word from main memory starting at the address stored in register 3 into plane 0, block 0 and starting at word 0.
129:	add	r0, r0, r0;	NOP
.	.	.	.
132:	ldui	r4, 0x0;	R4 ← 00000_{hex}.
133:	dbcdc	r4, 0, 0, 0, 0, 0, 0;	Double bank column broadcast. The XOR operation.
134:	ldli	r4, 0x4	R4 ← 4_{hex}
135:	dbcdc	r4, 0, 0, 1, 0, 0, 0x40;	Double bank column broadcast. The XOR operation.
136:	ldli	r4, 0x8	R4 ← 8_{hex}
137:	dbcdc	r4, 0, 0, 2, 0, 0, 0x80;	Double bank column broadcast. The XOR operation.
138:	ldli	r4, 0xC	R4 ← C_{hex}
139:	dbcdc	r4, 0, 0, 3, 0, 0, 0xC0;	Double bank column broadcast. The XOR operation.
140:	ldli	r4, 0x10	R4 ← 10
141:	dbcdc	r4, 0, 0, 4, 0, 0, 0x100;	Double bank column broadcast. The XOR operation.
142:	ldli	r4, 0x14	R4 ← 14_{hex}
143:	dbcdc	r4, 0, 0, 5, 0, 0, 0x140;	Double bank column broadcast. The XOR operation.
144:	ldli	r4, 0x18	R4 ← 18_{hex}
145:	dbcdc	r4, 0, 0, 6, 0, 0, 0x180;	Double bank column broadcast. The XOR operation.
146:	ldli	r4, 0x1C	R4 ← $1C_{hex}$
147:	dbcdc	r4, 0, 0, 7, 0, 0, 0x1C0;	Double bank column broadcast. The XOR operation.

Top

4.2. Linear Sequential Circuits Coding Algorithms

In this section we'll introduce the mapping of another application using reconfigurable computing. The circuits considered here are finite state machines with a finite number of inputs and outputs. The inputs, outputs and state transition occur at discrete intervals of time. The elements used are adders (XOR) and the delays (D) to delay input words. A sequence of 0s and 1s can be expressed by a polynomial in which the 0s and 1s are coefficients of the powers of a dummy variable. Hence the sequence 11001 can be written as $1D^4 \oplus 1D^3 \oplus 0D^2 \oplus 0D^1 \oplus 1D^0$. This representation is the basis of the Feed forward Binary Circuits, which are very useful in coding techniques. An example of these circuits is the circuit of the form: $T(D) = (1 \oplus D^1 \oplus D^2 \oplus D^3)$, as shown in Figure 3.

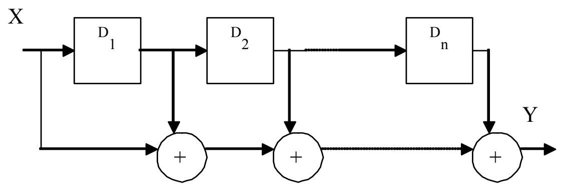

Figure 3. A General Representation.

This circuit is used to code any stream of input vector X and yields a set of outputs Y. Therefore, the input vector X is a vector of 0s and 1s and the output is the coded output Y vector, which is the result of multiplying the input polynomial (vector) X with the polynomial represented by T(D). This could be generalized to take the form:

$y_k = \sum_{j=0}^{N} \otimes x_{k-j} D^j$, or, finally $Y = (\sum_{j=0}^{N} \otimes D^j) X$

i.e. $Y = (D^0 \oplus D^1 \oplus D^2 \oplus \ldots \oplus D^N) X$

where, N is the number of stages of the circuit, and X is of the form $X = x_0 \ldots x_{k-1} x_k$ and xk is the first bit to enter the multiplier circuit. However, in this paper we will introduce the mapping for the reconfigurable parallel computation of this algorithm. The proposed mapping assumes column broadcast mode where all the cells in the same column perform the same function. The RC cells in each column are configured to perform:

Column 0: Out(t+1) = A x 1

Column 1: Out(t+1) = A ⊕ B, where B accesses its left.

Top

Column 2: Out(t+1) = A ⊕ B, where B accesses its left.

:

:

Column 7: Out(t+1) = A ⊕ B, where B accesses its left.

Where “A” represents the input, and “B” represents the interior stages taken each time from the left output. Where,”⊕” the EX-OR operation supported in the MorphoSys ALU. Due to the non-linearity in this application the first desired output appears starting from the third cycle. For a 3-delay multiplying circuit, the contents of the cells in column 2 at the end of cycle 3 are the first 3 elements in the output vector. This is also the case for the same RCs at the end of cycles 6 and 9 where they are the second and third 3-element sets in the output vector. Therefore, a write back to frame buffer of column 2 contents has to be made at the end of those specific cycles. Then, the process is repeated.

5. PERFORMANCE ANALYSIS

This performance is based on the execution speed of the algorithms. The MorphoSys system is considered to be operational at a frequency of 100 MHz.

5.1. Checksum Coding Technique

The algorithm in Table 1 takes 147 cycles in order to terminate. Thus the speed in Bytes-Per-Cycle of the algorithm of Table 1 is equal to 0.435 Bytes/Cycles i.e. 2.3 cycles for each byte to be decoded. Accordingly, the time for the algorithm to terminate is equal to 1.47 μsec. Comparisons with the same algorithm mapped onto some Intel microprocessing systems are shown in Table 2; the calculated speedup factor is considered to be the ratio of the M1 number of cycles over the other suggested systems.

5.2. Linear Sequential Circuits Coding Algorithms

For the case of a 3-delay multiplying circuit, the first output set is available from the third cycle, but after accounting the cycles needed for loading and storing data the calculations differ. In general it would take 9 cycles to code an 8-word code with a 3-delay multiplying circuit. The algorithm finishes coding 16 inputs (a total of 24 inputs of which the first 6 are zeros padding) in 32 cycles. The output vector is available in a file; this also takes additional cycles. Indeed, this algorithm takes 32 cycles to terminate, coding 16 bytes i.e. with a rate of 0.5 bytes/cycles, or 2 cycles per byte. With a total time equals to 0.32 μs. Findings are presented in Table 3.

Top

Table 2.Comparisons with other systems for the Checksum Coding/Decoding Algorithms.

Algorithm	System	N# of Cycles	Speedup	Time in Micro Sec.	Bits per Cycle	Mega Bits Per Second	Cycles per Bits
Encoding parallel algorithm of 64-Bytes	M1	96		0.96	5.33	533.33	0.19
	Pentium	580	6	0.96	0.88	117.43	1.13
	80486	769	8	4.36	0.67	66.58	1.50
Decoding parallel algorithm of 64-Bytes	M1	147		1.47	3.48	348.30	0.29
	Pentium	904	6.14	6.79	0.57	75.41	1.77
	80486	1156	7.86	11.56	0.44	44.29	2.26

Table 3.Findings for the Linear Sequential coding Algorithm.

Algorithm	N# of Cycles	Cycles / Byte	Bytes / Cycle	Total time in Micro	Speed in Mega bits
Linear Sequential Circuits Coding Algorithm under M1 for a 16-Byte Vector	32	2	0.5	0.32	400

6. CONCLUSION

New mapping algorithms are introduced dealing with coding operations and its performance analysis under MorphoSys is proposed. Results are compared with other processing systems. On one hand, the checksum coding algorithm is presented with its mapping onto the M1. Accordingly, a speed of 0.67 Bytes/Cycle for encoding was achieved, while a decoding speed of 0.43 Bytes/Cycle was obtained. On the other hand, a linear sequential coding algorithm was tested achieving a coding speed of 0.5 Bytes/Cycle. Future efforts could be invested in mapping other algorithms like the cyclic coding algorithms and turbo coding algorithms. In addition, comparisons could be made with results available on other parallel processors.

Top

REFERENCES

1. Abdennour E., H. Diab, and F. Kurdahi, 2000, "FIR Filter Mapping and Performance Analysis on MorphoSys," *Proceedings of the 7th IEEE International Conference on Electronics, Circuits and Systems*, Lebanon.
2. Bagherzadeh N., F. Kurdahi, H. Singh, G. Lu, M. Lee and E. Filho, 1998, "MorphoSys: A Reconfigurable Architecture for Multimedia Applications," *Proceedings of XI Brazilian Symposium on Integrated Circuit Design*, Rio De Janeiro.
3. Damaj I., 2001, "Performance Analysis of Linear Algebraic Functions using Reconfigurable Computing," Masters of Engineering thesis, The American University of Beirut, Lebanon.
4. Damaj I., H. Diab, 2001, "Performance Analysis Of Extended Vector-Scalar Operations Using Reconfigurable Computing," *Proceedings of the ACS International Conference of Computer Systems and Applications*, Beirut, Lebanon, p.270.
5. Eliseu M. C. Filho, 2000, "Design and Implementation of the MorphoSys Reconfigurable Computing,*" submitted to the Journal of VLSI and Signal Processing-Systems for Signal, Image and Video.*
6. Maestre R., F. Kurdahi, N. Bagherzadeh, H. Singh, R. Hermida, and N. Fernandez, 1999, "Kernel Scheduling in Reconfigurable Computing," *Proceedings of Design and Test in Europe* (DATE'99), Munich, Germany.
7. N. Bagherzadeh, F. Kurdahi, H. Singh, G. Lu, M. Lee, and E. Filho, 1999, "MorphoSys: A Parallel Reconfigurable System," *Proceedings of Euro-Par 99*, Toulouse France.

Top