A Modified Reconfigurable Data Path Processor

G. Ganesh, S. Whitaker and G. Maki [1]

NASA Space Engineering Research Center for VLSI System Design
University of Idaho, Moscow, Idaho 83843
Phone: 208-885-6500 Fax: 208-885-7579

Abstract- **High throughput is an overriding factor dictating system performance. In this paper, a configurable data path processor is presented which can be modified to optimize performance for a wide class of problems. The new processor is specifically designed for arbitrary data path operations and can be dynamically reconfigured.**

1 Introduction

High performance computers are increasingly in demand in areas of weather forecasting, structural analysis, etc.. These often require architectures which are different from the standard von-Neumann's machine also called the Standard Stored Program Computer. The stored program computers are designed to be general purpose and is not optimized for any specific problem. Fully customized architectures can be optimized to achieve maximum performance for a specific problem, but such processors cannot usually be adapted to produce solutions to different problems.

Modern technology opens new dimensions to the designer of high performance systems, by providing low cost VLSI modules which have high computational throughput. For a given functionality, there are two major dimensions of performance:- Delay and Throughput. High throughput is the most critical factor in real time processing of massive amounts of data, for example in Digital Signal Processing, Data Base operations, etc.. Since general purpose parallel computers cannot offer real time processing speeds, special purpose computers become the only appealing alternatives.

Special purpose processors can be of two types: 1) Dedicated Processors and 2) Reconfigurable/Programmable Processors. While the former are characterized by high processing speeds, inflexibility, long design time and high design cost, the latter have advantages of greater flexibility in coping with changes in the object problem, system specification and greater design economy with some reduction in throughput.

This paper presents a general purpose accelerator which is an enhancement over [1], that allows a variety of data path configurations, each characterized by its own topology of activated interconnections and hence applicable to a wide range of applications.

This configurable architecture combines the general purpose advantages of the stored program machine with the optimization of a fully customized architecture to achieve maximum performance for a broad class of problems. Every functional unit, data path and

[1]This research was supported (or partially supported) by NASA under Space Engineering Research Center Grant NAGW-1406.

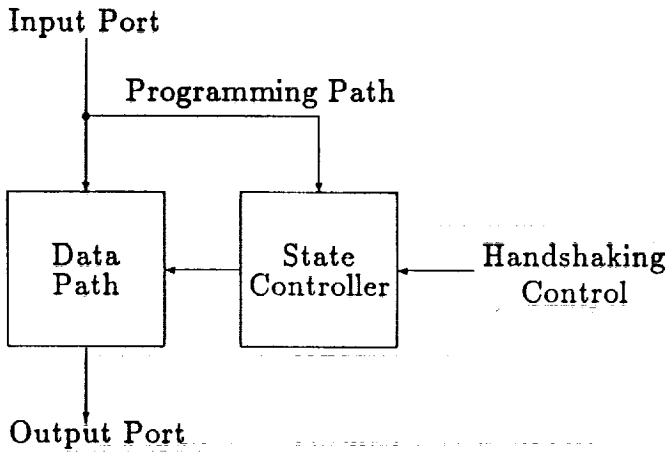

Figure 1: Block Diagram

control structure can be individually optimized for a given algorithm. The architecture presented is capable of operating in parallel, pipelined or sequential modes. The user configures the data path through programming. The architecture can be altered during operation by reprogramming or can be initialized and fixed for dedicated processing or can be attached to a host processor.

The reconfigurable processor differs from the stored program computer in the sense that there is no instruction fetch-decode-execute cycle. Moreover, an operation can be executed every clock pulse in every data path element.

2 Processor Design

The data path and the control structure have been designed to allow sequential, pipelined or parallel operation. The processor is configured as a set of identical data path elements with an overall controller. The top level block diagram of this processor is shown in Figure 1. There are two major components: the data path, which is an ALU-register stack to manipulate the data, and the state controller, which controls the register stack. The actual hardware configuration of the data path is specified during the programming of the State controller.

2.1 Data Path

Each data path element is as shown in Figure 2. Let there be m data path elements, each n bits wide. Direct communication between each data path element is an essential feature to achieve pipeline or parallel operation. Therefore, to allow all possible register to register communications, the data path bus must be $m \times n$ bits wide. This complete connectivity results in the flexible reconfigurability, but also limits the number of data path elements.

Each data path element consists of a Multiplier Accumulator (MAC) which multiplies two eight bit numbers and also adds two sixteen bit numbers to the product. (a.b+c+d). This output is stored in a globally accessible register of the data path element. Also

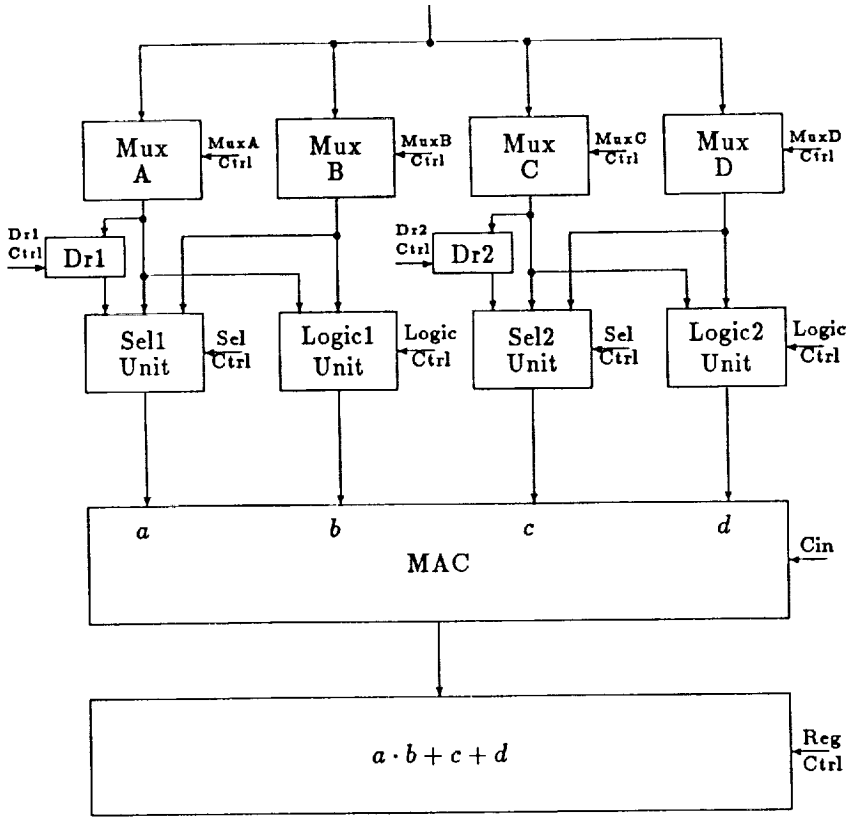

Figure 2: Data Path Element

contained within each data path element are two dedicated registers, which are used for operations local to that data path element. The addition of these dedicated registers is one of the improvements over [1]. This avoids the use of an entire data path element for the purpose of storage only. Since the area of a data path element is constrained by the $m \times n$ interconnect bus the addition of these registers should have little impact on the overall chip area.

The data path also contains a set of ALU and selector units. The ALU can implement an arbitrary arithmetic/logic operation. The operations of the first logic unit is as shown in Table 1. The selector unit selects the output of its respective multiplexors or the output of the respective dedicated register as shown in Table 2. The m to 1 multiplexors can select the output of any of the m globally accessible registers. The MAC operates on the output of the selector unit and the logic unit to allow a mixture of arithmetic and logic functions. Table 3 shows example of ALU operations that can be performed. CI is the carry in data bit.

Each globally accessible register is controlled as defined in Table 4. The dedicated registers are controlled as shown in Table 5.

The control word for each data path element structure is shown in Figure 3. For 16 data path elements, the control word is 33 bits wide.

Logic Control	Logic Operation
0 0 0 0	0
0 0 0 1	A AND B
0 0 1 0	A AND B'
0 0 1 1	A
0 1 0 0	A' AND B
0 1 0 1	B
0 1 1 0	A XOR B
0 1 1 1	A OR B
1 0 0 0	A NOR B
1 0 0 1	A XNOR B
1 0 1 0	B'
1 0 1 1	A' NAND B
1 1 0 0	A'
1 1 0 1	A NAND B'
1 1 1 0	A NAND B
1 1 1 1	1

Table 1: Logic Unit 1 Control

MC	Selector Output
00	A Mux
01	B Mux
10	Dr1
11	0

Table 2: Selector Unit 1 Control

Logic Unit	Sel Unit	CI	Output
0 0 0 0	A	1	A + 1
1 1 1 1	A	0	A + 1
0 0 1 1	-	1	A + 1
0 1 0 1	A	0	A plus B
1 0 1 0	A	0	1's complement A - B
1 1 0 0	-	1	2's complement A
1 0 1 0	A	1	2's complement A - B

Table 3: Example ALU operations

MuxA Ctrl	MuxB Ctrl	MuxC Ctrl	MuxD Ctrl	Logic1 Ctrl	Logic2 Ctrl	Sel1 Ctrl	Sel2 Ctrl	Reg Ctrl	C I	Dr1 Ctrl	Dr2 Ctrl
32 29	28 25	24 21	20 17	16 13	12 9	8 7	6 5	4 3	2	1	0

Figure 3: Data Path Element Control Word

RC1	RC2	Register Function
0	0	Hold Present Data
0	1	Load MAC Output
1	0	Shift MAC Right and Load
1	1	Shift MAC left and Load

Table 4: Global Register Load Control

Dr1	Register Function
0	Hold Present Data
1	Load Mux Output

Table 5: Dedicated Register Load Control

2.2 Control

The state controller specifies the control words for each data path element. The hardware compiled control words are contained in a control store memory as depicted in Figure 4. The output of each word from the control store drives each data path element. A total of 536 bits are needed in each control store word to control the data path elements in a 16 element, 16-bit data path structure. Program control within the control store is implemented with a program location counter. The control store can be of an arbitrary depth; here, it is depicted as 256 words deep. To perform a jump within the control store, an 8-bit jump address is provided in each control store word as depicted in Table 6.

The control store must be specified prior to operation. This specification (hardware compilation) can be achieved through the input port, 16 bits at a time. After the control store is specified, the processor is ready to operate in real time.

3 Operation

The control store word defines the operation and the source of data (registers) for each data path element. The output of any pair of registers R_i and R_j, i,j = 0,1,2,...,15 can be input to a data path element. In general, the operation can be specified as

$$R_i[ALUoperation]R_j \rightarrow R_k \tag{1}$$

33 bits	33 bits	...	33 bits	8 bits
Data Path Control Word Cell 0	Data Path Control Word Cell 1	Data Path Control Word Cell i	Data Path Control Word Cell 15	Program Counter Address

Table 6: Control Store Word

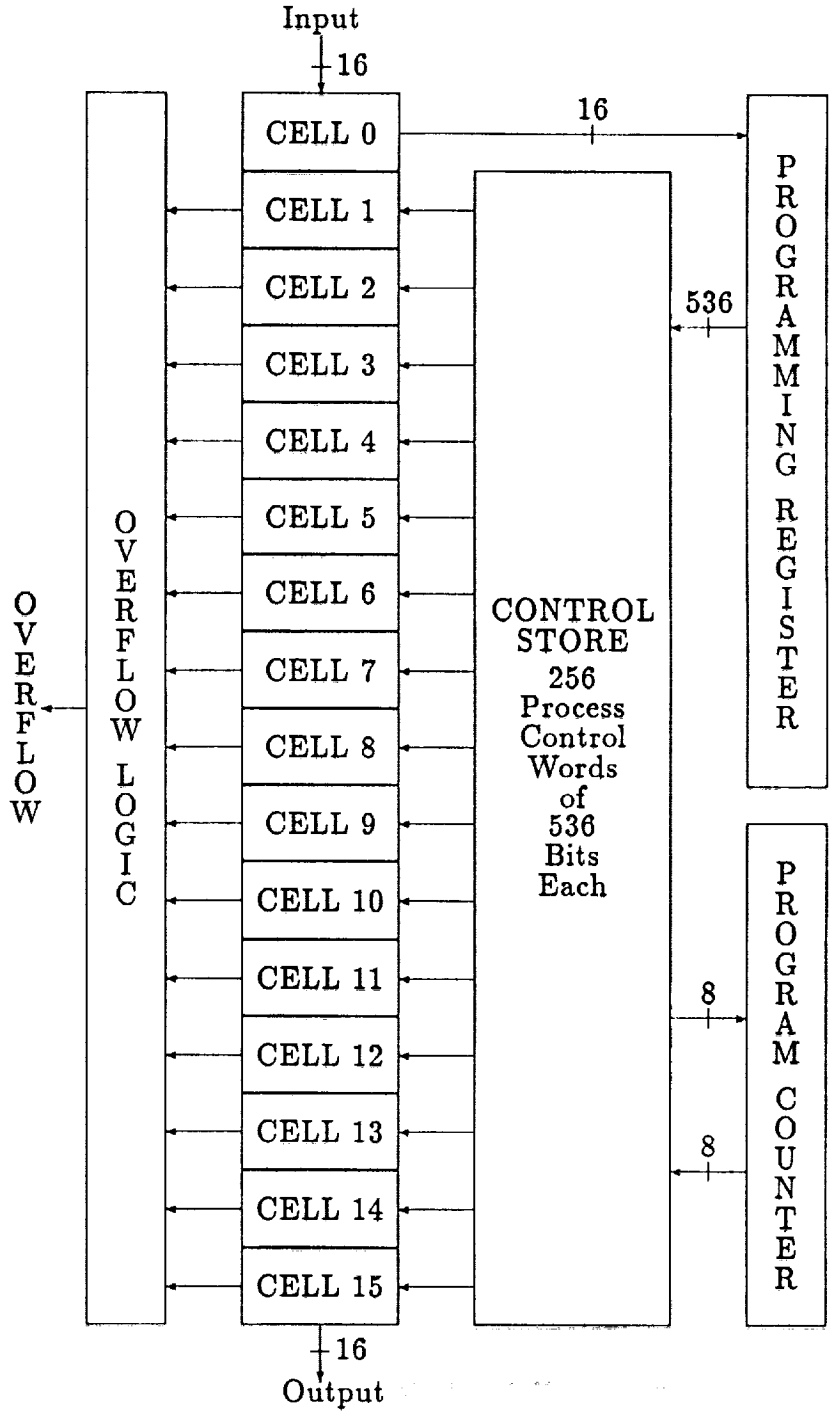

Figure 4: Control Store and Data Path

which means that the result of an ALU operation upon the contents of any register pair R_i and R_j can be placed into register R_k. This is true for any and all registers in the data path and all operations occur simultaneously. Since each data path element can function as an independent element, the entire data path can be configured to operate in the sequential, pipelined or parallel modes. The controller also specifies the next state of the controller and provides handshaking for external input and output control functions. The memory can be ROM for dedicated processing or RAM or EPROM where field programmability is desired. Depicted in Figure 4 is a feature where the control store can be programmed via the input data port. The entire control store can be initialized in a 16 bit word serial manner.

The control store is specified prior to operation. Once the control store is specified, the processor executes at the rate specified by the system clock. With static cells, the system clock can range from d.c. to the maximum allowable by the IC process.

3.1 Examples

Consider the following Digital Filter examples to illustrate the use of this processor. The general second order difference equation is

$$\begin{aligned} y(n) &= \alpha_0 x(n) + \alpha_1 x(n-1) + \alpha_2 x(n-2) \\ &\quad -\beta_1 y(n-1) - \beta_2 y(n-2). \end{aligned} \tag{2}$$

This implements an IIR filter. For an FIR filter the equation simplifies to

$$y(n) = \alpha_0 x(n) + \alpha_1 x(n-1) + \alpha_2 x(n-2). \tag{3}$$

To implement the FIR filter in the architecture presented in this paper, let Dr_{61} contain α_0, Dr_{51} contain α_1 and Dr_{41} contain α_2 as shown in the simplified block diagram of Figure 5. Also let R_4, R_5, R_6 and R_0 be initially reset. The operations can be described in a register transfer language where each P_i is a control state that defines the data transfers that take place when P_i is active.

P_0: $Data \rightarrow R_0$

P_1: $R_0 \cdot Dr_{61} + R_5 \rightarrow R_6$, $R_0 \cdot Dr_{51} + R_4 \rightarrow R_5$,
$R_0 \cdot Dr_{41} \rightarrow R_4$

Assuming that constants are preloaded into the registers and that 2's complement arithmetic is used, the control word for each data path element (R_i) is shown in Table 7. Each control state P_i represents one parallel control word; the portion of the control word for each R_i is shown on a series of lines for the sake of simplicity. Register control for all other registers not shown in Table 7 , the register control bits in their control words are 00, indicating no operation, for that control state, P_i.

There are a total of 5 operations that occur in 2 clock pulses. If this processor operated at 20 MHz, 50 million operations per second would be performed.

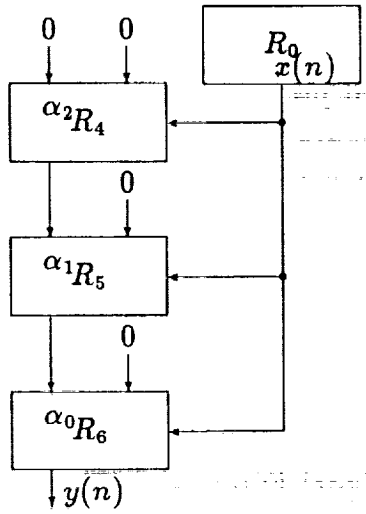

Figure 5: FIR Filter Block Diagram

State	Reg	MuxA	MuxB	MuxC	MuxD		
P_0	R_0	A	-	-	-		
P_1	R_4	R_0	-	-	-		
	R_5	R_0	-	R_4	-		
	R_6	R_0	-	R_5	-		
State	Reg	ALU1	ALU2	SC1	SC2	CI	RC
P_0	R_0	1111	0000	00	11	0	01
P_1	R_4	0011	0000	10	11	0	01
	R_5	0011	0000	10	00	0	01
	R_6	0011	0000	10	00	0	01

Table 7: FIR Filter Control Word Programming

State	Reg	MuxA	MuxB	MuxC	MuxD		
P_0	R_0	A	-	-	-		
P_1	R_2	R_0	-	R_6	-		
	R_6	R_0	-	R_7	-		
	R_7	R_0	-	-	-		
	R_{10}	R_9	-	-	-		
	R_9	R_9	-	R_2	R_8		
	R_8	R_9	-	-	-		
State	Reg	ALU1	ALU2	SC1	SC2	CI	RC
P_0	R_0	1111	0000	00	11	0	01
P_1	R_2	0011	0000	10	00	0	01
	R_6	0011	0000	10	00	0	01
	R_7	0011	0000	10	11	0	01
	R_{10}	1111	0000	00	11	0	01
	R_9	0011	0101	10	00	0	01
	R_8	0011	0000	10	11	0	01

Table 8: IIR Filter Control Word Programming

Dr_{21}	α_0
Dr_{61}	α_1
Dr_{71}	α_2
Dr_{91}	β_1
Dr_{81}	β_2
R_{10}	y(n)
R_9	y(n-1)
R_8	y(n-2)

For an IIR filter, consider the following register assignment. A register transfer language description of the operations to implement the IIR filter equation would be

P_0: $Data \rightarrow R_0$

P_1: $R_0 \cdot Dr_{21} + R_6 \rightarrow R_2$, $R_0 \cdot Dr_{61} + R_7 \rightarrow R_6$,
$R_0 \cdot Dr_{71} \rightarrow R_7$, $Dr_{81} \cdot R_9 \rightarrow R_8$,
$R_2 + R_8 + R_{91} \cdot R_9 \rightarrow R_9$, $R_9 \rightarrow R_{10}$.

Assuming again that constants are preloaded into the registers as 2's complement numbers and that 2's complement arithmetic is used. The control word for each data path element is shown in Table 8. There are a total of 9 operations that occur in 2 clock pulses; operating at 20 MHz, 90 million operations per second would be performed.

4 Summary

A new architecture has been presented which allows for sequential, pipelined, or parallel operation. A control-data path structure consists of m identical data path elements. The data path elements can be independently specified to allow parallel or pipelined operation. The control of the data path is specified by the control store memory. The processor can be

a dedicated stand alone machine or attached to a general purpose processor. As an attached processor, it can be dynamically modified to assume different data path configurations if the control store is RAM based. It is proposed that this architecture is a first step in producing a processor that allows the digital designer the same kind of flexibility in altering data path configurations as field programmable gate arrays offer alternatives to the logic designer.

Acknowledgement This research was supported in part by NASA under grant NAGW-1406 and grant NAG5-1043.

References

[1] G. Maki, S. Whitaker and G. Ganesh, "A Reconfigurable Data Path Processor", Proceedings of the IEEE ASIC Conference, Rochester NY, Sept., 1991.