

ELMS--Enclosed Loop Micro-Sequencer for the Fermilab Beam Loss Monitor System (Abstract & Summary)

Jinyuan Wu, Craig Drennan, Alan Baumbaugh and Jonathan Lewis

Abstract—Most of program loops in micro-processors are implemented with conditional branches that are the origin of many micro-complexities like branch prediction. Intrinsicly, loops with constant iterations need not use conditional branches. The Enclosed Loop Micro-Sequencer (ELMS) supports the “FOR” loops with constant iterations at the machine code level, which provides programming convenience and avoids micro-complexities from the beginning. Another design goal of ELMS is to be compact so that it can be easily embedded into FPGA devices. Low resource consumption is achieved by separating program flow control functions from the data processing functions (i.e., the arithmetic logic unit (ALU) in most micro-processors). The ELMS is able to run multi-layer nested-loop programs without help from external arithmetic/logic resources used for data processing. Since the data processing resources are external and purely user defined, the ELMS is not a traditional micro-processor, which is why it is called a “micro-sequencer”. The ELMS is used in the digitizer FPGA for the Fermilab Beam Loss Monitor system with expected performances.

Index Terms—Embedded System, Micro-processor, Micro-sequencer, FPGA, IP core.

I. INTRODUCTION

FPGA computing has been broadly used in high-energy/nuclear physics experiments. Inside an FPGA, data processing resources are flexibly defined by the users and usually are application specific. The sequence control of the data processing resources is an important issue in FPGA design.

Sequence control is normally implemented using either finite state machines (FSM) or embedded micro-processor cores. When an input data item is to be fed through a fast and very simple process, typically using a few clock cycles, FSM is a suitable means of sequence control. FSM also responds to external conditions promptly and accurately. However, the sequence or program in the FSM is not easy to change and debug, especially when irregularities exist in the sequence. Also, the state machines occupy logic elements no matter how

rarely they are used. So it is not economical to use FSM to implement the occasionally-used sequences such as initialization, communication channel establishment, etc.

Embedded or external micro-processor is another option of sequence control. Today’s main stream micro-processors are ALU (Arithmetic Logic Unit) oriented. The ALU, being the center piece of the micro-processor, performs not only data processing, but also program control functions. The ALU oriented architectures have two drawbacks in FPGA computation. (1) When a micro-processor core is embedded in an FPGA, the ALU occupies large amount of silicon resources. In instances where the application specific data processing is implemented in dedicated logic for the sake of speed, the ALU is barely utilized. (2) The program loops are implemented using conditional branches, which are the primary source of the micro-complexity of pipeline bubble, branch penalty etc. that needs to be solved with further micro-complexities such as branch prediction. The micro-processor is a better choice only if a data item is to be processed with a very complicate program, typically using thousands of clock cycles.

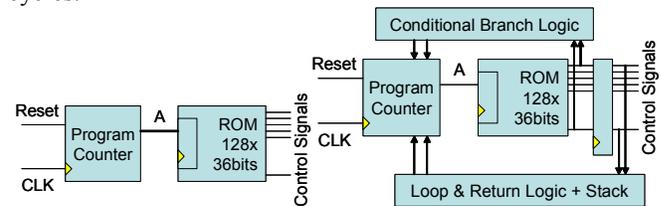


Fig. 1. Micro-Sequencers: When the program counter increases, the control signals changes states according to the sequence stored in the ROM. Left: PC+ROM structure. Right: the Enclosed Loop Micro-Sequencer (ELMS).

When a data item is to be processed with a medium length program, e.g., using a few hundreds clock cycles, the sequence control needed is not too much more than a PC+ROM structure (Fig. 1, left), which is the starting point of the Enclosed Loop Micro-Sequencer (ELMS) (Fig. 1, right). The primary difference between the ELMS and regular micro-processor is that in the ELMS there are no data processing resources like an ALU. The control signals for external data processing resources turn on and off according to the sequence stored in the ROM as the program counter (PC) increases. Obviously, supporting logic must be added to control the PC. In addition to the conditional branch logic that also exists in micro-processors, loop and return logic with an internal stack are added in the ELMS, so that it supports “FOR” loops with

Manuscript received May 15, 2006. This work was supported in part Operated by Universities Research Association Inc. under Contract No. DE-AC02-76CH03000 with the United States Department of Energy.

Jinyuan Wu, Craig Drennan, Alan Baumbaugh and Jonathan Lewis are with Fermi National Accelerator Laboratory, Batavia, IL 60510 USA (phone: 630-840-8911; fax: 630-840-2950; e-mail: jywu168@fnal.gov).

constant iterations at the machine code level and is self-sufficient to run multi-layer nested-loop programs.

II. THE FERMILAB BEAM LOSS MONITOR SYSTEM

A. Overview

The new Fermilab Beam Loss Monitor (BLM) readout system is designed to perform several tasks: to provide a flexible and reliable abort system to protect Tevatron magnets; to provide loss monitor data during normal operations of the Tevatron, Main Injector and Booster; and to provide detailed diagnostic loss histories when an abort happens. Beam losses are detected using ion chambers.

The inputs from ion chambers are integrated for a short period of time, typically 21 μ s, and digitized to 16 bits. The digital data are used to construct several sliding sums, which are a measure of the integrated loss over a variety of time scales up to 64k cycles. The abort request signals for each channel are made in firmware by comparing these sums as well as immediate measurement with thresholds. The system abort signal is made by checking number of channels and types of abort request signals.

B. The Digitizer Card

A Digitizer Card (DC) integrates, digitizes and processes 4 channels of ion chamber inputs. The block diagram for the FPGA calculating the sliding sums is shown in Fig. 2.

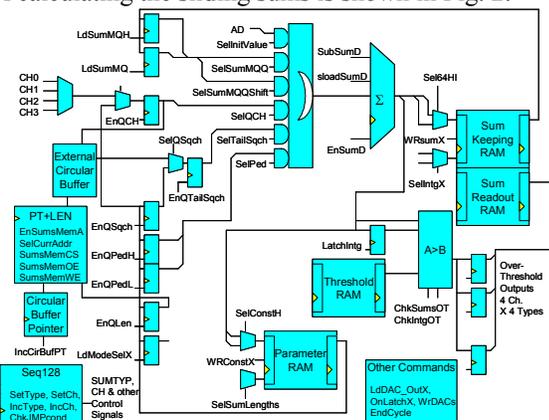


Fig. 2. The partial block diagram of the SUM03 FPGA

A total of 16 sliding sums are to be kept in the FPGA. If all sums were kept using accumulators, the FPGA would easily consume several thousand logic elements, out of 5980 logic elements in the Altera Cyclone EP1C6 device we use.

On the other hand, during 21 μ s period, there are more than 1000 clock cycles at 50 MHz inside the FPGA. Clearly it is more economical to calculate 16 sliding sums sequentially using one set of data processing resources. The control signals shown in Fig. 2 are turned on and off to perform various functions by the “Seq128” block with an ELMS block inside.

III. THE ENCLOSED LOOP MICRO-SEQUENCER

A. Description:

Detailed block diagram of the ELMS is shown in Fig 3.

The program is stored in a 36-bit x 128-word ROM in our example.

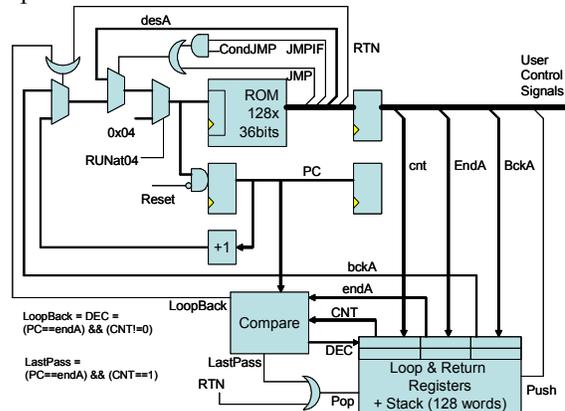


Fig. 3. Detailed block diagram of the Enclosed Loop Micro-Sequencer (ELMS): The Loop & Return Registers + Stack block provides support of the “FOR” loop with constant iterations.

Both unconditional and conditional branches are supported as in regular micro-processors. We have used a non-pipelined design in our example. The Loop & Return Registers (LRR) along with a 128-word stack are the primary elements designed to support the constant iteration “FOR” loops.

B. Program Control Instructions

Some ELMS instructions are shown in Table I.

TABLE I
PROGRAM CONTROL INSTRUCTIONS

	35	34	33	32	31:24	23:16	15:8	7:0	Notes
JMP	1	0	0	0					desA Unconditional go to desA
JMPIF	0	0	0	1					desA Conditional go to desA
FOR	0	0	1	0		BckA	EndA	cnt	Repeat cnt+1 times form BckA to EndA
CALL	1	0	1	0		BckA	EndA	desA	Go to desA, upon PC=EndA, go BckA
RTN	0	1	0	0					Return, pop stack
	0	0	0	0	X	X	X	X	User instructions

The ELMS instructions are 36-bit words. When any of the bits 32-35 is set, the word represents a program control instruction. Especially, when bit 33 is set, the instruction starts a FOR loop in which the bit fields BckA, EndA and cnt are pushed into corresponding LRR/stack. The PC is incremented until reaching EndA, and then it is set back to BckA. This continues for (cnt+1) passes. Then the stack is popped on the last pass of the loop.

The CALL instruction is implemented as a combination of the FOR and JMP instructions with cnt automatically set =1. At the CALL instruction, the PC jumps to desA while BckA and EndA are pushed into the LRR/stack. When PC reaches EndA or when a RTN instruction is seen, the PC jumps back to BckA and the stack is popped. Note that in addition to a regular return instruction, the return point from the subroutine is also pre-defined to be EndA, which allows an alternative means of subroutine return that provides extra convenience.

Multi-layer FOR or CALL loops can be nested. When an inner layer starts, the parameters of the unfinished outer loop are pushed into the stack, which allows the outer loop to continue after the inner loop finishes. Up to 128 layers of loops can be nested.