Readout Process & Noise Elimination Firmware for the Fermilab Beam Loss Monitor System

Jinyuan Wu, Alan Baumbaugh, Craig Drennan, Randy Thurman-Keup, Jonathan Lewis & Zonghan Shi

Abstract— In the Fermilab Beam Loss Monitor System, inputs from ion chambers are integrated for a short period of time, digitized and processed to create the accelerator abort request signals. The accelerator power supplies employing 3-phase 60Hz AC cause noise at various harmonics on our inputs which must be eliminated for monitoring purposes. During accelerator ramping, both the sampling frequency and the amplitudes of the noise components change. As such, traditional digital filtering can partially reduce certain noise components but not all. A non-traditional algorithm was developed in our work to eliminate remaining ripples. The sequencing in the FPGA firmware is conducted by a micro-sequencer core we developed: the Enclosed Loop Micro-Sequencer (ELMS). The unique feature of the ELMS is that it supports the "FOR" loops with pre-defined iterations at the machine code level, which provides programming convenience and avoids many micro-complexities from the beginning.

***Index Terms*—Digital Data Processing, Embedded System, Micro-processor, Micro-sequencer, FPGA, Reconfigurable Computing.**

I. Introduction

THE new Fermilab Beam Loss Monitor (BLM) readout system [1] 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 signals from the ion chambers are integrated for a short period of time, typically 21 μs, and digitized to 16 bits. The digital data are used to construct fast, slow and very-slow 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 the immediate measurement with thresholds. The system abort signal is made by checking the number of channels and types of abort request signals.

For the Main Injector BLM system, an integration sum for each channel is accumulated.

In addition to producing abort request signals, the sliding sums are also readout to monitor the beam loss. However, the accelerator power supplies employing 3-phase 60Hz AC cause noise at various harmonics on our inputs which can be larger than the beam loss data in some channels. Both analog methods, e.g., appropriate grounding scheme for the input cables and digital methods are employed for noise reduction. A special challenge for the digital processing in accelerator systems is ramping, i.e., accelerating particles from lower energy to higher energy. During accelerator ramping, both the sampling frequency and the amplitudes of the noise components change. A traditional digital filtering process was implemented and it partially reduced certain noise components but not all. A non-traditional algorithm was developed in our work to eliminate the remaining ripples.

An FPGA firmware core for our sequencing control, called the Enclosed Loop Micro-Sequencer (ELMS) is also described in this document. The primary difference between the ELMS and the regular micro-processor/micro-sequencer is that "FOR" loops with pre-defined iterations at the machine code level are supported in the ELMS making it self-sufficient to run multi-layer nested-loop programs.

II. The Digitizer Card

A Digitizer Card (DC) integrates, digitizes and processes 4 channels of ion channel inputs. The partial block diagram for the FPGA calculating the sliding sums is shown in Fig. 1.

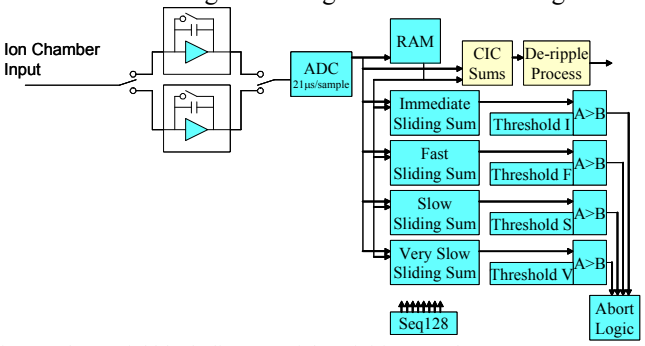

Fig. 1. The partial block diagram of the Digitizer Card

Each input from the ion chamber is integrated by two integrators alternately in ping-pong fashion. The output voltages of the integrators reflect the charges due to beam loss and are then digitized by an ADC device (AD7654AST) at about 21 μs per sample and input into an FPGA (with project

Manuscript received April 30, 2007. This work was supported in part by Universities Research Association Inc. under Contract No. DE-AC02-76CH03000 with the United States Department of Energy.

The authors are with Fermi National Accelerator Laboratory, Batavia, IL 60510 USA (phone: 630-840-8911; fax: 630-840-2950; e-mail: jywu168@fnal.gov).

name Sums03) for digital processing.

A total of 16 sliding sums are to be kept in the FPGA. (In addition to the fast, slow and very slow sliding sums, the immediate measurement is implemented as a sliding sum with sum length=1). They are compared with corresponding thresholds pre-loaded into the FPGA to produce 16 abort request signals indicating the channel and type of the abort request. The FPGA also produces several other "de-rippled" values for monitoring purposes which will be described in detail in later sections. 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 a 21 μs period, there are more than 1000 clock cycles at 50 MHz inside the FPGA. Clearly it is more economical to calculate 16 sliding sums and to perform other tasks sequentially using one set of data processing resources. The sequence is conducted by the "Seq128" block with an ELMS block inside.

For the Main Injector BLM system, integrations must be computed. In order to compute the integrations properly, the pedestal for each channel is first calculated. At the beginning of each beam cycle when there is no beam, about 752 readings for each channel are accumulated to measure the pedestal.

For the Main Injector the very-slow sliding sum of each channel has a sum length of about 64 and now represents a smoothed version of the input. For each measurement, the pedestal is subtracted from the very-slow sliding sum with an appropriate scaling and the difference can be optionally compared with a user-defined value called the "squelch level". If the difference is bigger than the squelch level, the input signal is considered bigger than noise and it is added into the integration sum. Otherwise, the input signal is considered below the noise level and the integration sum is kept unchanged.

III. The De-ripple Process

In addition to calculating the 16 sliding sums and generating corresponding abort request signals, the Digitizer Card also outputs various values for monitoring of the beam loss.

The signal cables in the MI tunnel pick up noise generated by equipment powered by 3-phase 60Hz AC. It contains primarily harmonics of 60Hz, 180Hz and multiples of 360Hz. The noise level is higher than the desired signal and the noise peaks exist at both higher and lower frequencies compared to the signal spectrum. A typical set of raw measurement data and their spectrum are shown in Fig. 2 and Fig. 3.

A natural approach of eliminating noise is filtering. In fact, calculating sliding sums can be viewed as a digital filtering process. The fast sliding sum over a length of 128 shown in Fig. 2 has reduced the noise level from about 10 ADC counts in the raw data down to about 2 ADC counts. Some beam loss can be seen in the fast sliding sum plot.

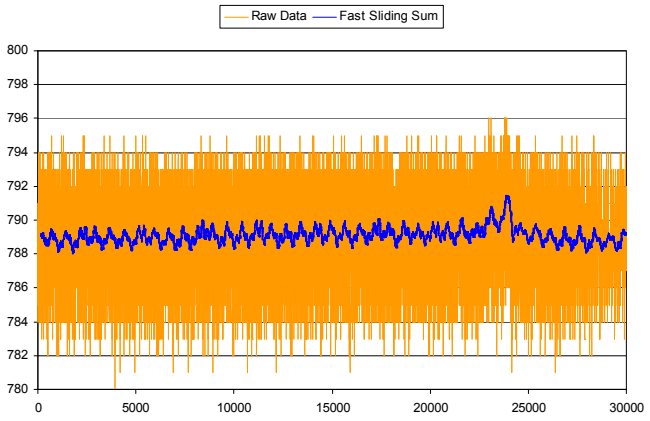

Fig. 2. The raw measurement data and the fast sliding sum

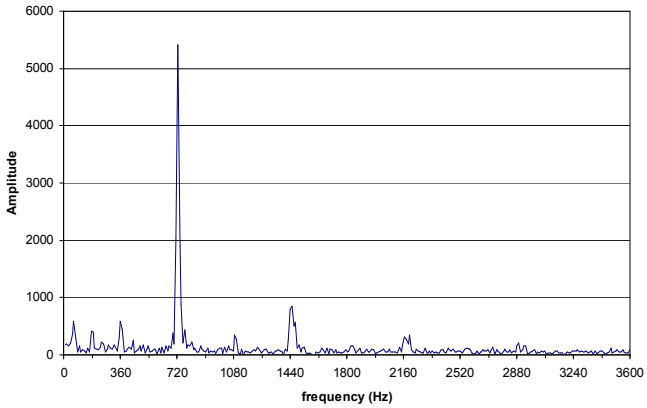

Fig. 3. The frequency spectrum of raw measurement data

However, the sliding sum is a low pass filter with a sinc(x) function shaped frequency response. The zeros of the sinc(x) are very sharp and it is hard to align them with the noise peaks, especially when the sampling frequency is not an integer multiple of 60Hz and when the accelerator ramps which varies the sampling frequency. Also the side lobes of the sinc(x) are not low enough. Therefore, the plot of the fast sliding sum still contains glitches along with unfiltered 60Hz and 180Hz components.

The de-ripple process further eliminates remaining noise components so that smaller beam losses become visible. The process takes the following steps:

1. Calculating the Cascaded Integrator-Comb (CIC) sums.
2. Waveform extraction, storage and validation.
3. Waveform subtraction.

A. Calculating the Cascaded Integrator-Comb (CIC) Sums

The cascaded integrator-comb (CIC) digital filter [2] of order N contains N cascaded stages. Each stage is a moving average filter which is a CIC filter of order 1. The sliding sum can be viewed as a CIC filter of order 1 with un-normalized gain. The CIC sums implemented in the Sums03 FPGA firmware are CIC filters of order 2, which can be viewed as the sliding sum of the sliding sum of the raw data.

The frequency response shape of the CIC sums is $\text{sinc}^2(x)$ in

which the zeros become the second order ones that provide deeper attenuation to the noise peaks even though the peaks are not precisely aligned with the zeros. The side lobes also become lower. In Fig. 4, the sliding sums (FS) and the CIC sums of a set of typical measurement data are plotted (with appropriate scales and an artificial offset). It can be seen that the CIC sums are significantly smoother than the sliding sum.

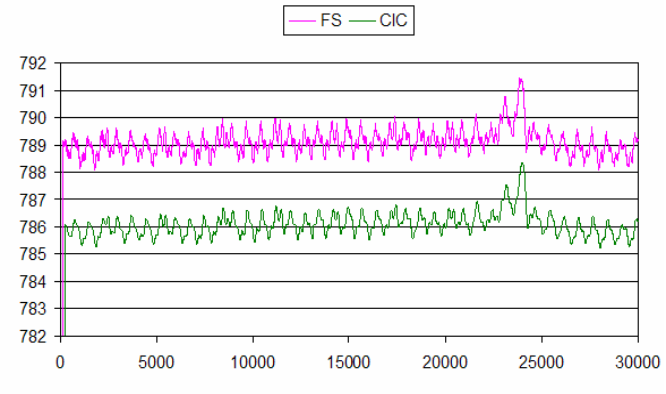

Fig. 4. The sliding sum and the CIC sum

The CIC sum y[n] and sliding sum s[m] of input sequence x[j] with sum length K in our work are defined as:

$$y[n] = \sum_{m=n}^{n-(K-1)} s[m] \quad s[m] = \sum_{j=m}^{m-(K-1)} x[j]$$

The sum length K for the CIC sums in our firmware is chosen to be 124-128, which brings the first zero to 360Hz. It can be reasonably assumed that the CIC sums are band limited to 360Hz.

In the practical firmware, the accumulations above are implemented recursively as shown in Fig. 5.

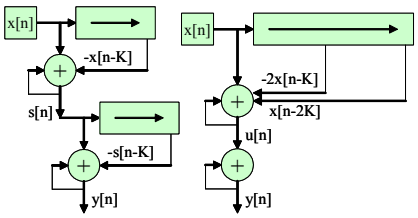

Fig. 5. The CIC sum calculation

Both left and right diagrams in Fig. 5 are valid CIC sum implementations and the resource usages are comparable. However in our application, we would like avoid adding a separate storage for s[n], given that a record of up to 64K raw measurement points x[j] are available. The formula for the CIC sum is altered as shown in the right diagram, which contains two recursive accumulations:

$$u[n] = u[n-1] + x[n] - 2x[n-K] + x[n-2K]$$

$$y[n] = y[n-1] + u[n]$$

This way, the only additional storage is the intermediate value u[n], which takes only one memory space. This way, no additional long record of intermediate values needs to be stored.

B. Waveform Extraction, Storage and Validation

The "de-ripple" process involves simply subtracting the noise waveform from the current CIC sum. The waveform is a record of previous CIC sums of one period (1/60Hz = 16.7ms) long which are believed not to be contaminated by abrupt beam loss (although a DC or very slow beam loss is allowed). In the BLM digitizer FPGA firmware, the calculated CIC sums are directly stored as the waveform data.

Since the CIC sums are band limited to about 360Hz, it is possible to decimate the y[n] sequence to save storage space without losing information. The decimation counter is a 24-bit accumulator that increases by 22336 for every input point or about every 21μs. The top 7 bits are used as address to the waveform WF storage memories. This way, 128 CIC sum points are stored for the time period of 1/60Hz. The separation of two decimated points is 5 or 6 raw data points. The effects of non-uniform decimation are negligible for our application, although interpolation algorithms are available to reduce the effects. The block diagram of the de-ripple processor is shown in Fig. 6.

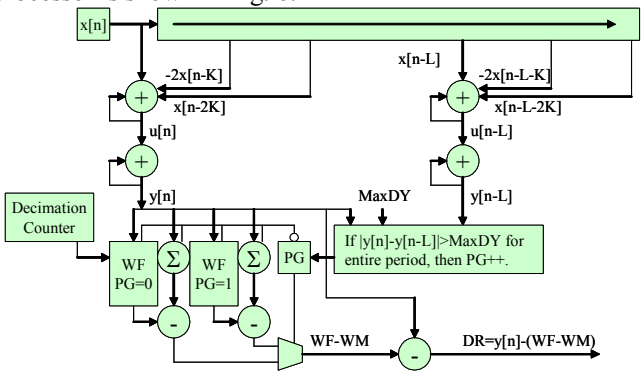

Fig. 6. The de-ripple process

The waveform storage memories are split into two pages for each channel. The CIC sums are written into the page as the tentative waveform which must be validated through the period. Meanwhile, for each channel, a sum of the waveform is accumulated to calculate the waveform mean value, WM. After accumulating 128 points for the entire period, the sum is simply the waveform mean scaled by a factor of 128, or 7 bits. This is the reason for choosing the decimation scheme mentioned above.

Two CIC sums are calculated, the current one y[n] and the one a period before y[n-L], where L is the length corresponding to a period which is about 752. During the period, the absolute value of the difference of the two CIC sums is constantly compared with a parameter MaxDY. If the difference between the two CIC sums is bigger than the predefined limit MaxDY, there may be an abrupt beam loss in the period. The waveform then is considered invalid. If the differences in the entire period are within the predefined limit, the waveform becomes valid.

At the end of each period, if the waveform is valid, a 1-bit counter PG flips, which swaps the tentative waveform page to the usable waveform page. The new tentative waveform is

stored in another page until it becomes a valid waveform at the end of another period.

At the initial time after reset, the firmware logic forces the value (WF-WM)=0. After the waveform of a period becomes valid, it outputs the latest valid one.

C. *Waveform Subtraction*

Once the waveform becomes valid, waveform subtraction can be performed to get the de-rippled sum DR. The stored waveform WF contains a DC component, which represents the slow beam loss during the period when the waveform was recorded. To assure the slow beam loss at current time is correctly preserved, the DC balanced waveform (WF-WM) is used in the subtraction. As mentioned above, the WM is the waveform mean accumulated during the waveform recording period and therefore the mean of (WF-WM) over the same period is zero or DC balanced. Results of the full de-ripple process is shown in Fig. 7.

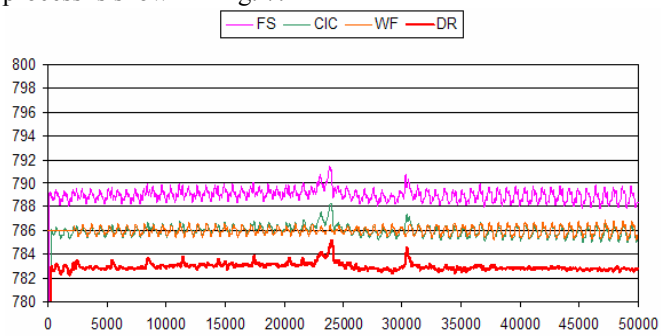

Fig. 7. The de-ripple results

In order to see the curves clearly, each curve is added with an offset. The second curve is the CIC sum with a sum length of 128. For reference, the sliding sum FS with the same sum length is also shown as the first trace. The waveform WF in general takes 3 periods to become valid. In the first 3 periods, WF and WM are forced to be 0. The de-rippled output DR, the bottom curve, first follows the CIC sum for 3 periods since WF is invalid. Then it becomes a smooth curve with 60Hz and 180Hz ripples canceled. It can be seen that in the DR sum, both abrupt and slow beam losses become visible.

IV. The Enclosed Loop Micro-Sequencer

A. *Sequence Control Options*

As mentioned earlier the functions in the Sum03 FPGA are performed sequentially. 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 microprocessor is another option of sequence control. The drawback of a microprocessor is the large resource usage. The micro-processor is a better choice only if a data item is to be processed with a very complicated program, typically using thousands of clock cycles.

When a data item is to be processed with a medium length program, e.g., using a few hundred clock cycles, a micro-sequencer becomes a better option. We have developed a micro-sequencer in our FPGA called the Enclosed Loop Micro-Sequencer (ELMS). The primary difference between the ELMS and regular micro-processor/micro-sequencer is that the ELMS supports "FOR" loops with predefined iterations at the machine code level and is self-sufficient to run multi-layer nested-loop programs.

B. *Description*

A detailed block diagram of the ELMS is shown in Fig 8. The program is stored in a 36-bit x 128-word ROM in our example. Clearly the instruction width and memory depth can be flexibly chosen for different applications as necessary. Also, ROM's in FPGA are typically implemented with dual-port random access memories (RAM's), which allows the users to overwrite its contents so that new programs can be loaded. However, if the program is not to be changed during operation, a block memory organized as a ROM with the program pre-stored is more convenient.

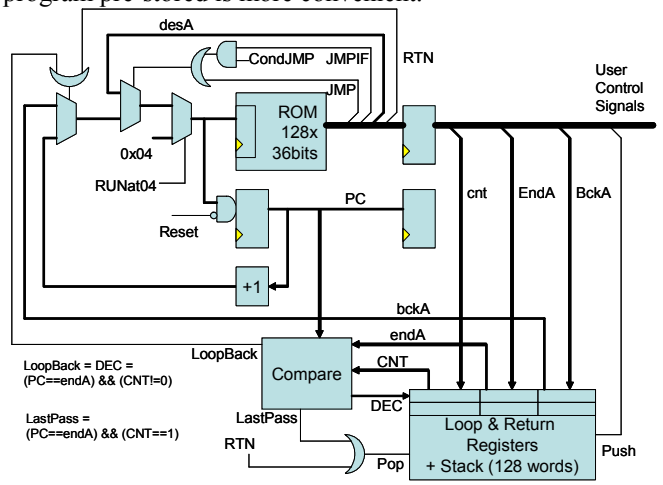

Fig. 8. Detailed block diagram of the Enclosed Loop Micro-Sequencer (ELMS): The Loop & Return Registers + Stack block provides support of the "FOR" loop with constant iterations.

Both unconditional and conditional branches are supported as in regular micro-processors. We have used non-pipelined branch logic in our example for simplicity.

The Loop & Return Registers (LRR) along with a 128-word stack are the primary elements designed to support the constant iteration "FOR" loops.

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. Otherwise, it is treated as a user instruction. In the ELMS, the only built-in instructions are the program control instructions. All other instructions can be freely defined by the users.

C. The Branch Instructions

The unconditional branch instruction JMP is implemented as in typical micro-processors. When bit 35 is set, bit field desA (only the lower 7 bits are used in our example) is selected as the PC for next clock cycle.

The conditional branch instruction JMPIF is signified when bit 32 is set. An input line CondJMP is supplied from external user logic as the branch condition, i.e., the PC jumps to desA only when CondJMP is high. The branch condition in the ELMS is treated as a result from the external data processing resources. It is the users' responsibility to generate this signal and assure that it is valid when reaching the conditional branch instruction. This design arrangement allows us to avoid using an ALU in the sequencer.

In the non-pipelined design, the branch logic is the most latency critical part. When a JMP or JMPIF instruction is present at the output of the ROM, the signals must flow through several layers of multiplexers, arriving at the address registers of the ROM with sufficient setup time. We have been able to compile the non-pipelined design in an Altera Cyclone FPGA device EP1C6Q240C6 [3] with a 153 MHz maximum operating frequency.

To increase the operating frequency further, a pipelined design can be used, i.e., assigning registers on both input and output ports of the ROM. We have compiled a pipelined version in the same device with a 250 MHz maximum operating frequency. However, a pipeline bubble (no-op instruction) or out-of-order time slot must be added after the JMP or JMPIF instructions.

In our application, the clock inside the FPGA is 50 MHz. That's why we chose a non-pipelined design in our example.

The branch instructions are to be used only when it is necessary.

D. The FOR Instruction

Supporting FOR loops with predefined iterations at machine code level is a special feature of the ELMS.

When bit 33 is set, the instruction starts a FOR loop in which the bit fields BckA, EndA and cnt are pushed into the 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.

A program segment with a FOR loop may look like the following:

```
            FOR  BckA1  EndA1  5
               Initialization Processes
BckA1
               Repeating Processes
EndA1
```

After the FOR instruction, the instructions before PC = BckA1 are executed once, essentially serving as initialization. Then the instructions between PC = BckA1 and EndA1 (inclusive) are executed (cnt+1) or 6 times in this example. Note that there is no conditional branch instruction at EndA1. The ELMS conducts the loop sequence by itself.

Another interesting point is that the LRR + stack structure appears like a Branch Target Buffer (BTB) in advanced micro-processors [4]. Indeed, the LRR + stack stores information of the targets to be branched to. However, the PC jumps in ELMS are pre-defined by the FOR instruction and are not based on predictions. Thus the sequencing performance of the ELMS is deterministic rather than statistical.

E. The CALL and RTN Instructions

The CALL instruction is implemented as a combination of the FOR and JMP instructions with cnt automatically set equal to 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.

A program segment with CALL/RTN instructions may look like the following:

```
            CALL  BckA1  EndA1  DesA1
BckA1
               Processes after Subroutine Return
DesA1
               Subroutine
EndA1  RTN (optional)
```

After the CALL instruction, the PC jumps to DesA1 to execute the subroutine. Once PC reaches EndA1, it returns to BckA1. The instruction at EndA1 does not need to be RTN. Therefore any program segment can be called as a subroutine.

The RTN instruction is provided primarily for possible early returns in the subroutines. The RTN instruction may also be used when early breaks are needed in the FOR loops.

F. Nesting Loops

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.

Note that in the FOR loops, inner loops can be nested not only in the repeating processes, but also in the initialization processes. This design arrangement provides convenience for the programmers when subroutine calls or FOR loops are

needed in the initialization, such as presetting an array.

Up to 128 layers of loops can be nested. It is the users' responsibility not to nest more than 128 layers of loops. This should be sufficient for most applications. For example, if 64 layers of FOR loops, each iterating 2 times, are nested together, it will take the sequencer, operating at 250 MHz, more than 2000 years to complete.

G. The User Instructions

When the bits 32-35 of the instruction word are all 0, the word represents a user instruction. The users have maximum flexibility to define their own instruction sets based on the application. We present the instruction set we used for the Fermilab BLM system as an example shown here in Table II.

TABLE II
THE USER INSTRUCTION SET USED IN FERMILAB BLM SYSTEM

Bit 35:32	31:28	27:24	23:20	19:16	15:8	7:0
0000	SEQA	SEQB	SEQC	SEQDQQ	ADH	ADL

	Branch Instruction	SEQA[]	SEQB[]	SEQC[]	SEQDQQ[]
0					
1	JMPIF	IncCirBufPT	SetType	SetCh	EnQLen
2	FOR	ChkJMPcond	IncType	IncCh	EnQCH
3		SelSumLengths	SubQLen	SelQWF	
4	RTN	EnSumsMemA	SelCurrAddr	ShiftM1	LdModeSelX
5		SumsMemCS	SumsMemOE	SumsMemWE	LdDAC_OutX
6		WRsumXa	SelQSqch	EnQTailSqch	
7		WRsumXb		Sel64HI	LdSumMQH
8	JMP		SubSumD	SelInitValue	LdSumMQ
9		EnSumD	sloadSumD	SelSumMQQ	EnQSqch
10	CALL	LatchIntg		SelSumMQQShift	EnQPedL
11		WRsumX	SelIntgX	SelQCH	EnQPedH
12	BRK	ChkSumsOT	ChkIntgOT	SelTailSqch	
13		WRwaveform		SelPed	
14		WRconstX	SelConstH	OnLatchX	
15			WrDACs	EndCycle	

A user instruction contains four instruction fields, 4 bits each: SEQA, SEQB, SEQC and SEQDQQ; and two address/data fields: ADH and ADL. Each instruction field is decoded into up to 15 control signals with names shown in Table II. The SEQDQQ are delayed by a pipeline step before being decoded. The control signals generated from SEQDQQ are essentially register enable signals for reading out contents from the Parameter RAM and the Sum Keeping RAM that are registered on input port. The ADH and ADL fields are used to provide addresses for memory access or to specify initial values for some registers.

Sometimes, several control signals must be turned on simultaneously. While defining the instruction set, signals that might be turned on simultaneously are carefully assigned to different columns in Table II.

H. Sample Codes

The ELMS codes for calculating 16 sliding sums in our application are shown in Table III.

After reset, the PC starts from 00. The sequencer runs into a dead loop at PC = 03. The unconditional instruction, JMP to 03 is "executed" every clock cycle. However, there is no bit flipping at all. The sequencer and the logic it controls are effectively in a sleep mode that consumes no dynamic power.

When an external "do sums" signal arrives, the "RUNat04" signal in Fig. 8 is turned on for a clock cycle that forces the PC to become 04. The ELMS then goes through the sequence of calculating the sliding sums. The FOR instruction at PC = 07 sets the outer loop for the 4 types of sliding sums (immediate, fast, slow and very-slow). Then the FOR instruction at PC = 0A sets the inner loop for 4 input channels. The type and channel of the sliding sums are indexed by two counters that are initialized and incremented by the SetType, IncType, SetCh and IncCh instructions, respectively.

The "compiler" we used is a Microsoft Excel spread sheet. The search and index functions are used to find labels and instructions. Each row is composed as a 36-bit integer in the column "code". The columns "PC" and "code" are copied to another worksheet, which is then saved as a text file. The text

TABLE III
SAMPLE CODES OF THE ELMS

PC	Label	BR Instr.	BckA	EndA	cnt/desA	SEQA	SEQB	SEQC	SEQDQQ	ADH	ADL	code	Notes
00												000000000	
01												000000000	
02												000000000	
03	DeadBk3	8 JMP			DeadBk3 03							800000003	dead loop after reset
04												000000000	do sums begins at 0x04
05						1 IncCirBufPT						010000000	
06							1 SetType			0		001000000	*** sliding sums begin ***
07		2 FOR	TypeBgn1	08 TypeEnd1	17 3	3						200081703	
08	TypeBgn1					3 SelSumLengths			1 EnQLen		40	030010040	load sum length of the type
09								1 SetCh			0	000100000	
0A		2 FOR	ChBgn1	0B ChEnd1	16 3	3						2000B1603	
0B	ChBgn1								2 EnQCH		48	000020048	current hit
0C									8 LdSumMQ	80		000088000	stored sum
0D						4 EnSumsMemA			4 LdModeSelX		68	040040068	
0E						5 SumsMemCS	5 SumsMemOE					055000000	
0F						5 SumsMemCS	5 SumsMemOE	6 EnQTailSqch				055600000	load tail
10						9 EnSumD	9 sloadSumD	9 SelSumMQQ				099900000	old sum
11						9 EnSumD		11 SelQCH				090B00000	+current value
12						9 EnSumD	8 SubSumD	12 SelTailSqch				098C00000	-tail = new sum
13												000000000	
14						11 WRsumX				80		0B0008000	
15						12 ChkSumsOT						0C0000000	
16	ChEnd1							2 IncCh				000200000	
17	TypeEnd1						2 IncType					002000000	*** sliding sums fin ***

file can be directly used as a "memory initialization file" that specifies the ROM contents in the FPGA.

V. FPGA Implementations

The firmware with project name Sums03 that calculates the sliding sums, generates abort request signals and performs de-ripple functions has been implemented in a low cost FPGA device, EP1C6Q240C6. The ELMS has been used in the block Seq128 in the FPGA for sequence control. For evaluating the idea of the ELMS, a bare ELMS circuit plus three 8-bit accumulators has also been compiled and simulated in a test project ELMS1 using the same FPGA device. Compiled results are shown in Table IV.

TABLE IV
Silicon Usage of The ELMS

Device Price: (April 2007)	EP1C6Q240C6 $28	
	Logic Elements (5980 total)	M4K memory blocks (20 total)
Whole Sums03 FPGA	2486 (41%)	20 (100%)
Seq128 (ELMS + etc.)	212 (3.5%)	2 (10%)
ELMS1 (ELMS+ 3 8bit-accumulators)	193 (3%)	2 (10%)

It can be seen that the resource usage of the ELMS is very small, leaving most portions of the FPGA for data processing functions defined by users. As a result of using ELMS, a significant portion of the resources for calculating the sliding sums and the integration sums are reused multiple times for each measurement. Without resource reusing, the whole function would not fit our FPGA.

It is possible to find off-the-shelf micro-processor/micro-sequencer IP with comparable resource usage. However, the supporting of predefined FOR loops at machine code level is a special and convenient feature of the ELMS. In fact, because of its simplicity, the ELMS itself can become an off-the-shelf solution for future projects. On the other hand, it is not difficult to add the FOR loop support to a future version of an existing IP.

Again because of the simplicity, it is very easy to compile the ELMS with a high operating frequency. The non-pipelined and pipelined versions of the ELMS1 project are compiled with 153 MHz and 250 MHz maximum operating frequencies, respectively, where 250MHz is the upper operating limit of the M4K memory block in the device. The project Sums03 does not need a high operating frequency since its internal clock is only 50 MHz. The Sums03 project is compiled with a maximum operating frequency of 61 MHz.

VI. Discussion

Several design considerations of the firmware are to be discussed in this section.

A. Several Remarks about the De-ripple Process

Traditional digital filtering performs well on eliminating high frequency noise. In our firmware, we used a CIC filter of order 2, i.e., the CIC sums to eliminate noise above 360Hz. To eliminate low frequency components, however, traditional digital filter needs a sufficiently long record, usually many (1/60Hz) periods, of measurement data. In our system, there are not very many periods after a reset and the amplitudes and phases of the 60, 120 and 180 Hz noise components can be different over a few (1/60Hz) periods before and after the accelerator ramping.

Therefore, a noise waveform subtraction approach is chosen for our de-ripple process in which a period of noise waveform is stored for subtraction from the later measurements. Since the noise components may change over a few periods, the waveform is updated constantly. To assure the waveform is free of abrupt beam loss, it is being validated using a periodic condition (assuming the abrupt beam loss is not periodic) while it is recorded. Finally, the effect of DC beam loss in the waveform is canceled by subtraction of the waveform mean from the waveform.

B. The Sequencer without Data Processing Resources

In history, there are computers employing the "Harvard" architecture [5] in which storages of program and data are physically separated. Most of today's general purpose micro-processors use the "Princeton" architecture in which the program and data are stored in the same external memory. However, inside the micro-processor, the program and data are usually stored in separate caches and at this level it is the Harvard structure again.

In the ELMS, the data and program are further separated beyond the Harvard architecture. A micro-sequencer is not a CPU since the sequencer itself does not have capabilities for general purpose data processing. The micro-sequencer controls external data processing resources by toggling control signals.

In FPGA computing, this arrangement allows maximum flexibility in the data domain. The widths of data words, addressing modes and number of processing channels etc. can be chosen by the designer without any restrictions as in general purpose micro-processors.

Without data processing resources, conditional branches are harder to be implemented and therefore are discouraged in the ELMS, while loops using the built-in FOR loop support are encouraged. On the other hand, because of the FOR loop support in ELMS, a further separation of program and data than the Harvard architecture becomes practical and feasible.

C. Predefined Iteration FOR Loop Support

Using loops in the program is a primary means of code reuse. Supporting block-styled predefined iteration FOR loops without using a conditional branch instruction is a unique feature of the ELMS. Of course, the ELMS must still support conditional branch instruction JMPIF since the FOR loops can replace conditional branches in many but not all instances.

In advanced micro-processors, branch penalties [6][7] become more serious as the pipeline becomes deeper and

deeper. An attempt to solve the problem is to use branch prediction with additional resource and there are good algorithms in this area.

When a FOR loop with predefined iteration is programmed, the execution route including where and how many times to loop is determined in advance. There should be no branch penalty at all. However, when using conditional branch to conduct the loop, the originally known sequence becomes unknown and the branch condition must be evaluated each time the end of the loop is reached. With FOR loops available at the machine code level, it helps to ease the branch penalty problem.

In an FPGA, the clock speed difference between pipelined and non-pipelined ROM is not very significant. In cases such as in our example, a clock frequency as low as 50 MHz is sufficient which makes non-pipelined structures preferable. Hence the benefit of the FOR loops on reducing the branch penalty is not very obvious. Nevertheless, the FOR loop is still a convenient program instruction to achieve silicon resource and code reuse.

In practice, indexes must be kept in loops to distinguish different passes of the loops. In the ELMS, the pass counter for the FOR loop can be viewed as an index. However, we chose for the user to implement external user indexes rather than supporting them inside the micro-sequencer. The pass counter is in the program domain while user indexes belong to the data domain. It is more convenient for the users to specify the parameters of the index counters such as number of bits, incremental difference, reset or preload features etc.

In our example, the number of iterations "cnt" is an immediate value that comes with the FOR instruction. However, there is no fundamental reason why this value can not be stored in a user register. This way, FOR loops with a variable number of iterations can be supported, which is very useful in applications like matrix computation.

D. Software Issues

The operation of the ELMS is conducted by a pre-stored program. Just as in micro-processor computing, the software must be appropriately coded and compiled for given computing tasks. Based on experience with micro-processor computing, it is known that software engineering could become a major effort in certain tasks.

In many cases complexity of software is only partially necessary for the computing tasks and is partially artificial, essentially due to complexity of the hardware or firmware. Therefore, the best way to reduce software complexity is to simplify the hardware or firmware design.

The architecture of the ELMS is directly reflected in its instruction set. There are only a handful program flow control instructions that are native to the ELMS. All the remaining ones are user instructions that are application specific. Unlike in micro-processors where the users write programs using an existing instruction set, in an FPGA with the ELMS, the users design the instruction set as well as program them into the desired sequence.

For our practical design, we have used spread sheets as our tools for keeping track of the instruction set design, program coding, compiling as well as documenting. This way, the effort of software design is controlled within a reasonable fraction of the entire work.

VII. CONCLUSION

The firmware in the digitizer card for the Fermilab BLM system has been commissioned and operates with specified functions.

The de-ripple process described in this document can be a useful noise elimination tool for systems with periodic noise and non-periodic signals. The signals can be brief abrupt ones or slow moving ones.

The ELMS provides an option of sequence control in an FPGA with very low resource usage. It has been used for the Fermilab BLM system with specified performance and a flexible reprogramming ability.

The FOR loop support in machine code level in the ELMS also provides some hints on fighting branch penalty problems for advanced micro-processor development. Clearly, there is a whole array of associated issues that must be studied in the future.

REFERENCES

[1] C. Drennan, et. al., "Development of a new data acquisition system for the Fermilab beam loss monitors," in *Nuclear Science Symposium Conference Record*, Date: 16-22 Oct. 2004, Pages: 1816 - 1819 Vol. 3.

[2] R. Lyons, *Understanding Digital Signal Processing*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2004.

[3] Cyclone FPGA Family Data Sheet, Altera Corp., San Jose, CA, 2003 [Online]. Available: http://www.altera.com/

[4] G. Hinton, et. al., "The Micro-architecture of the Pentium 4 Processor," in *Intel Technology Journal*, Vol. 5 Issue 1 (February 2001).

[5] J. Hilburn & P. Julich, *Microcomputers/Microprocessors: Hardware, Software and Applications*, Englewood Cliffs, NJ: Prentice Hall, 1976.

[6] D. Comer, *Essentials of Computer Architecture*, Upper Saddle River, NJ: Prentice Hall, 2005.

[7] Arvind et. al., "6.823 Computer System Architecture" MIT Open Course Ware, [Online]. Available: http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-823Fall-2005/CourseHome/