Project 05EB000004: Booster Feed Forward Processor Version 1

N. Towne

Abstract

Draft report

INTRODUCTION

The Booster Dipole Power Supply is a fast ramping supply with tight regulation requirements for both injection and extraction. To accurately control the field a feed-forward (FF) system was developed [1, 2, 3]. This system functions by recording the tracking error from a given cycle, calculating a correction signal, and adding this correction to the setpoint ramp of the next cycle. The corrections are the current error, which is the difference between the dipole-magnet current sensed by a DC current transformer and the programmed ramp, filtered by a ten-pole, ten-zero digital filter defined by the response functions of the magnet, power supply, and analog electronics through the procedure outlined in ref. [4]. While the feed-forward filtering is in the digital domain, regulator feedback processing remains in the analog domain.

The existing feed-forward system is implemented in modules housed in the VME crate `brampm`. An I/O module in the crate handles analog input and output [**?**] and the digital signal processing is handled by a Pentek module containing a Texas Instruments TMS320C30 digital signal processor (DSP) [5]. The problem is that software and hardware obsolescence have each rendered the system nearly impossible to maintain, with parts out of production and software that can only operate on obsolete computers.

When this project started I had intended to not only build a FF-system replacement, but to also absorb dipole-regulator functions into the design and house the project on an FPGA. As it happened, however, the very limited processing required of the FF function and the inherent delay of one booster cycle made it easy to port the FF filter to a personal computer (PC) platform housing an analog I/O board, in this case a PC running NT 2000 and the National Instruments (NI) PCI-6052E board [6], which we had in hand. So the FF filter was first programmed in Visual Basic and talked to the board through the National Instruments Traditional NI-DAQ interface [7]. This arrangement worked well during a several-month trial run, during which very few glitches were detected. There seemed to be plenty of time available during the booster cycle for the program to do background processing.

This report describes the first FF processor's software.

OVERVIEW OF THE SOFTWARE

The Visual Basic software is organized into files that handle various functions:

- `FeedFwd` – the main program;
- `FFFilter` – core feed-forward filter processing;
- `AnalogIn(SCB-68C)` – analog input;
- `AnalogOut(DB)` – analog output; and
- `DigitalIO` – digital input and output (only input is currently implemented and used).

The program has a simple layered structure so that upper layers need not be concerned with the details of the function and configuration of the lower layers (Fig. 1).

In Fig. 1, the rectangular blocks above the NI-DAQ interface correspond to Visual Basic source files; the ramp block is only contemplated at this point. The transport block includes all transport between the NI-DAQ interface and PCI-6052E internals abstracted by the NI-DAQ interface, including the NI-DAQ layer, device drivers, the PCI bus and interfaces on both ends, and PCI-6052E controls and signal routing. The ramp-enable logic signal comes from `brampm` VME crate through a PCI-6052E digital-I/O pin.

The program and these files are structured very much as they would be in C, and translation to C could be done easily mostly on a line-by-line basis. Due to the driver-level buffering, it is probable that the program could also be coded in Labview with processing time to spare, although code translation would not be so simple. It would be preferable to use the multi-threaded and up-to-date NI-DAQmx interface. I am told that there are also performance gains to be realized through this route.

OVERALL SIGNAL FLOW

The PCI-6052E board [6] has eight differential ADC channels, two DAC channels, eight digital I/O lines, timers and flexible signal routing. On-board FIFOs are present for both input and output. In addition, driver-level output buffering is available and is used to provide the one-booster-cycle delay fundamental to the FF algorithm.

Once the program is running, triggering is a four-stage process.

- The program polls until a high level appears on the ramp-enable line from the VME chassis in reponse to

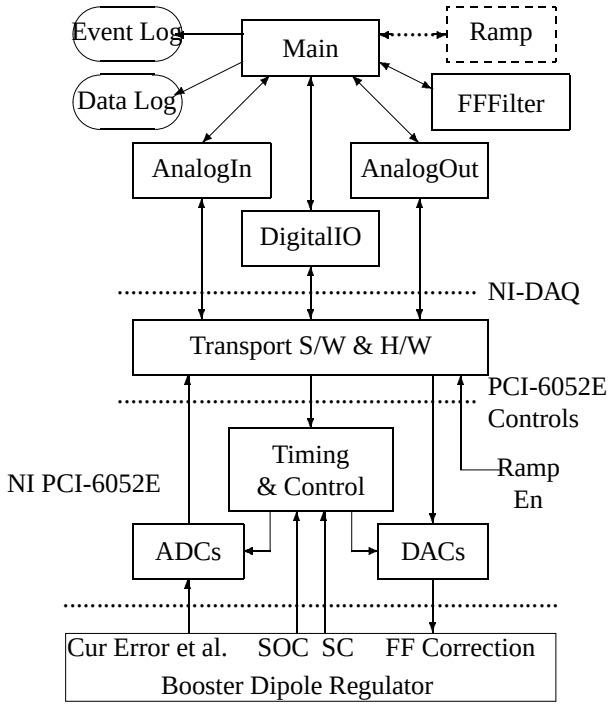

Figure 1: Feed-forward system architecture. SOC is the start-of-cycle signal and SC is the sample clock.

control-room command to ramp. This can occur at any time.

- Then, the hardware waits for a start-of-cycle low-to-high transition. There can be up to a booster cycle delay before this happens since there is no synchronization between the ramp-enable and the the start-of-cycle signals.
- From there, the hardware *scan* triggering is controlled by the sample clock, the first of which happens 600 μs after the start-of-cycle transition.
- ADC *sample* triggering is provided by an on-board timer. The hardware scans through the activated ADC channels in the configured order. A gain can be independently assigned to each channel.

Input and output of samples are handled in blocks $n/2$ long, where n is the number of scans per booster cycle. (The number n must be even.) Two blocks are processed per booster cycle. Output buffering is handled on the driver level and below; the program need only send the correct number of scans to the driver interface in each half-cycle time slot. Input buffering is also handled on the driver level and below but `AnalogInSCB-68C` polls the driver for the number of available scans until $n/2$ scans are available, at which time the scans are retrieved and processed. To set up the booster cycle delay, the output is initially loaded with n scans; input and output are clocked synchronously and are then started simultaneously (the start of output is slaved to the start of input).

The existing FF cycle algorithm also has a waveform advance of 30 scans that is ramped in at the start of cycling. This is necessary because the FF-filter response function has characteristic delay of about 30 scans. This waveform advance is implemented in this program by sending, in the first handful of booster cycles, buffers of data to the driver interface that are a few scans shy of $n/2$. Over these cycles the waveform advance accumulates to the 30 scans.

FILTER BLOCK

The FF filter block is defined to mimic the filter defined in the C source code for the TI DSP. This code is or will be on the `//nslsnt1/ElectSys/Power Systems Group/Linac-Booster` directory for inspection. This directory with file sizes and date stamps are listed below. The correct source file is `FF_NEWAU.C` and `FF_NEWAU.OUT` is the binary currently in operation.

```
12/17/1992  05:17p  164,923 FF_NEW
12/23/1992  03:43p  21,792  FF_NEW.C
12/17/1992  05:17p  97,813  FF_NEW.LST
12/24/1992  03:22p  64,821  FF_NEW.MAP
12/24/1992  03:22p  8,850   FF_NEW.OBJ
12/24/1992  03:23p  171,607 FF_NEW.OUT
01/25/1993  06:08a  23,215  FF_NEWAU.C
01/25/1993  06:13a  164,131 FF_NEWAU.OUT
05/16/2005  02:32p  1,452   README
```

Figure 2 shows the structure of the filter. The K1 and K2 filters are structured as three and five cascaded biquad filters (Fig. 3), respectively. In code, these are modeled as a set of taps, each tap of which is modeled as simply a coefficient (corresponding to a_i or b_i) and a memory element (corresponding to z^{-1}). Each is of double precision real data type. The procedure `biquad` steps a sample, applied as the first argument, through the biquad filter specified by the second argument.

Biscardi prepared a presentation on how the coefficients are calculated based on the characteristics of the power supply, magnets, and regulator [4].

Initialization of the filter requires a handful of steps. The biquad filters require defining an array of eight biquad data types, at compile time; at run time, each of its coefficients is set to its value, the memory elements are set to zero, the circular buffer is cleared, the advance, which is later ramped in, is set to zero, and the memory element 'ylast' in Figure 2 is cleared.

A speed test of the Visual Basic program showed that a 1.5-GHz Athlon computer can cycle the filter at 2.5 MHz, corresponding to 190 MFlops. So computing power is not a problem.

The function interface is:

- `FFFilter` – the filter itself. Its argument is the input sample as a 'double' data type and it returns the

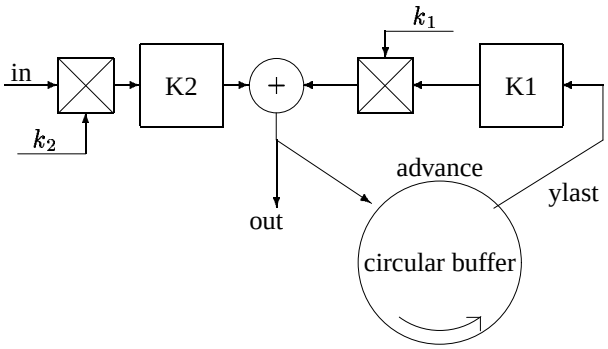

Figure 2: Feed-forward filter shown with one booster cycle delay and waveform advance. K2 is five cascaded biquad filters and K1 is three such filters cascaded; k_1 and k_2 are multiplicative factors; and the circular buffer contains samples corresponding to one booster cycle.

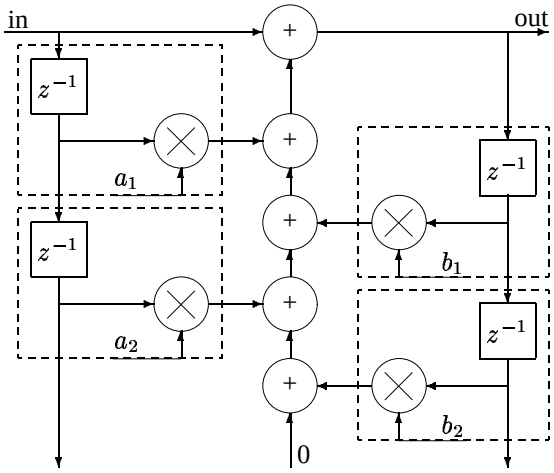

Figure 3: Biquad filter having two FIR taps and two IIR taps. All processing implemented in the program is in double-precision floating-point data type. The z^{-1} blocks are one-sample memory elements. The dashed boxes are taps, which are discussed in the text.

filtered output as a double. There is no cycle delay of the output since the I/O board provides that delay. Lines can be commented and uncommented to insert the delay.

- `FFFilter16` – the filter for 16-bit integers. Its argument is the sample as a 'short' data type and the output is a short windowed to the 16-bit range.

- `FFFilter32` – the filter for 32-bit integers. Its argument is the sample as a 'long' data type and the output is a long windowed to the 32-bit range.

- `FFInit` – for filter initialization. Defines the filter according to code in the procedure and initializes the internal states of the filter elements to zero. Its inputs are the booster cycle delay (as long) and the final advance in samples (as long). It returns zero.

- `FFFilterSaveState` – saves the state of the filter. Used each time there is a good ramp so that it can be restored and used over again to skip over a bad ramp. See `FFFilterRestoreState`.

- `FFFilterRestoreState` – restores the state of the filter.

All arguments are passed by reference.

Feed-forward-specific board and driver interface

The application-dependent software layer that manages the details of the board through the NI-DAQ interface consists of three files, one for analog input and its configuration (`AnalogInSCB-68C`), one for analog output and its configuration (`AnalogOutMC`), and the last for digital input (and potentially output) (`DigitalIO`). These blocks are independent of each other. The analog-out start trigger is, however, slaved to the analog-input start trigger in hardware by `AnalogOutMC`.

Analog Input

Analog input is encapsulated in the file `AnalogInSCB-68C.bas`. It handles channel, timing, and trigger configuration as described earlier, as well as starting and stopping acquisition and acquiring data.

- `AnalogInInit` – initializes the board for channel, timing, and trigger configuration as described in Sec. . It requires for arguments:
  - the device number as defined by the Measurement and Automation Explorer (M&AE) (as short);
  - the number of input channels (as short);
  - a list of ADC channels to be used as an array of integers (as short);
  - a list of gains corresponding to the channels (as short); and
  - the cycle length (as long).

- `AnalogIn` – reads sets of scans from the board. It requires for arguments the number of scans to read (as long) and a scan buffer (as short) into which data are placed.

- `AnalogInStart` – starts acquisition. There are no arguments.

- `AnalogInReset` – resets the analog input. Be sure to end acquisition started by `AnalogInStart` or your system may become unstable. There are no arguments.

All arguments are passed by reference.

Analog Output

Analog output is encapsulated into the file `AnalogOutDB.bas`. It handles channel and trigger configuration, starting and stopping output, and outputting data.

- `AnalogOutInit` – initializes the board and driver for channel and trigger configuration. For arguments, it requires:
  - the device number as defined by the Measurement and Automation Explorer (M&AE) (as short);
  - the number of output channels (as short);
  - a list of ADC channels to be used as an array of integers (as short);
  - a list of polarities for the channels (bipolar=0, unipolar=1, as short);
  - the cycle length (as long);
  - a full-length output buffer (as short).
- `AnalogOut` – outputs data to the device. Its arguments are the number of output samples (as long) and a buffer containing the output samples (as short).
- `AnalogOutStart` – starts analog output in the sense that output is enabled to respond to its start trigger, i.e., it is armed. The start-of-analog-output trigger is slaved to the start-of-analog-input trigger.
- `AnalogOutReset` – stops analog output and sets the output channels to zero. Be sure to call this function to end analog output after starting output with AnalogOutStart, or your computer may become unstable.

All parameters are called by reference.

Digital Input

At the moment, the program only uses the ramp-enable signal to start FF corrections. This signal is tied into the first digital I/O terminal and is polled by the program when waiting (after the `Go` button is clicked). The interface is as follows:

- `DigInit` – initializes the digital port. It arguments are the board device number (as short), which is determined by M&AE, the bit number (0 to 7) (as short), and whether input or output (as boolean, True = input, False = output).
- `DigInput` – reads a digital I/O bit. It requires a bit number (as short) as an argument. The function returns the state of the bit as a boolean. The bit specified should have been previously configured.
- `DigClose` – sets the direction of the digital I/O bits back to input. Use this call to avoid contention when the bits are later used for other purposes.

All parameters are called by reference.

FF-TEST – THE APPLICATION LEVEL

The main task of the application is feed forward, which requires the program to acquire current-error samples, filter them, and output them in blocks to/from the interface layer of Sec. . Other tasks are handled:

- Allow the specification of M&AE device number and a configuration file.
- Read the configuration file that specifies input and output channels and gains and the final waveform advance in cycles, and configure analog input and output accordingly.
- Manage the waveform advance at the start of ramping. Part of this function is also embedded in `FFFilter` (which breaks the layered-software model).
- Skip over cycles in which the ramp fails to start.
- Display a simple graph of one of a choice of signals input and output by the program. The scale is fixed at $\pm 2^{15}$. A button permits enabling and disabling the plot. The `MSCHART` OCX control is used.
- Stream acquired and processed data to a file. All input channels specified by the configuration file plus the filtered feed-forward correction are output to the file (as binary short data type).
- Display a cycle count and a diagnostic input block size each for block.

The control-panel buttons function as follows:

- `Go` tells the program to enter a run state where it either runs corrections or waits for a ramp-enable trigger telling it to do so. This button is grayed out while the program is in this state.
- `Standby` means to go to standby mode where the program waits for user input, i.e., it stops responding to triggers and waits for either a `Go` command or an `Exit` command. It is grayed out while in this state.
- `Exit` quits the program. It is grayed out unless the program is in standby state.

When the program first comes up, it immediately goes to the FF correction cycle where it waits for the ramp enable. The computer is set up to log in to my account and run the FF program when it boots so that it and the program recover automatically from power failure.

The configuration file is specified in the ‘configuration file’ field of the program’s graphical user interface (GUI) and must reside in the same directory as the program. Its format is:

- The first line has the number of analog input channels in use. For the PCI-6052E, this number may be 1 through 8.
- For each input channel, there is a line specifying the channel number and gain. The order of these lines specifies the order in which the channels are scanned. Valid input gains for the PCI-6052E are -1 (for 0.5), 1, 2, 5, 10, 20, 50, and 100.
- The next line has the number of output channels to use. For the PCI-6052E, this number may be 1 or 2.
- For each output channel, there is a line specifying the channel number and polarity (0 for bipolar, 1 for unipolar). During each scan of the channels, the PCI-6052E reads the channels in the order specified by these lines.
- This line specifies the waveform advance in scans. It should be a multiple of three.
- This line specifies an overall gain factor to apply to the filter output (double precision). This number may be overridden on 'Output gain factor' entry of the program's GUI.

A useful configuration file is

```
4 [number of input channels]
4 5 [gain of five for the current error]
5 -1 [gain of 0.5 for others]
1 -1
3 -1
1 [number of output channels]
0 0 [bipolar output for the correction]
30 [waveform advance in scans]
0.97 [overall gain factor]
```

The current error is typically small so a gain of 5 (10 times the other channels) is specified for it on the second line. Bracketed comments are not part of the file.

The program's window must not be moved during operation. When this happens, the program appears to be preempted, which means that the output buffer goes empty and output stops. Another output operation generates an error, which then causes a programmed FF-cycle restart.

TESTS

The program was operated in a mirror mode leading up to the operation test (below). In this mode, the normal VME-based feed-forward system was running and controlling the power supply as normal, but the new program was operating in parallel. It was also recording the current error, ramp, dipole current, and feed-forward corrections from both programs. Its feed-forward correction was recorded in analog loop-back mode through another ADC channel to provide an accurate comparison (Fig. 4).

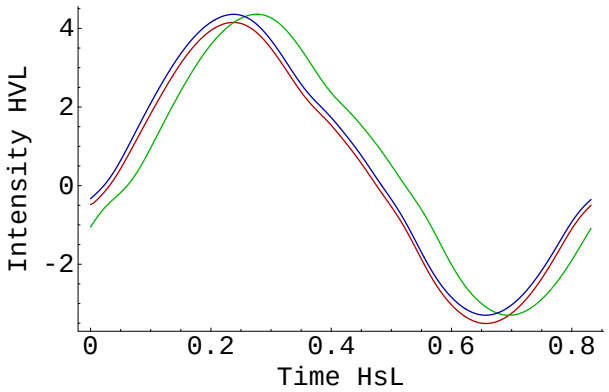

Figure 4: Steady-state FF corrections from the existing DSP (red), this processor (green) and this processor but before waveform advance (blue) recorded during a booster ramp cycle. The existing DSP is controlling the power supply and the overall gain factor used to match the peak-to-peak swings is 0.97.

The program was successfully tested on May 2, first controlling the power supply without injection, and then with injection into the VUV ring, which appeared normal. It was not left in place since changes were needed to make the program suitable for operations. Since then bells and whistles were added (Sec.) necessitating more tests. The system was put into trial operation in early June and has been in operation since. Operations people say that it has worked well and is preferred over the VME system.

Table 1: Connections to the National Instruments SCB-68 breakout box.

Src/dst signal	SCB-68 Pins	Board signal	Signal direction
Current error	28/61	ACH4/12	to PC
Beam current	60/26	ACH5/13	to PC
VME Ramp	33/66	ACH1/9	to PC
VME correction	30/63	ACH3/11	to PC
Magnet current	31/65	ACH10/2	to PC
Start of cycle	68/34	ACH0/8	to PC
	11/44	PFI0/DGnd	to PC
Convert	43/9	PFI2/DGnd	
Scan clock	41/7	PFI4/DGnd	to PC
Ramp enable	52/18	DIO0/DGnd	to PC
PC correction	22/55	DAC0/AOGnd	from PC
PC Ramp	21/54	DAC1/AOGnd	from PC

REFERENCES

[1] R. Olsen, J. Dabrowski, and J. Murray Prepared for 1993 IEEE Particle Accelerator Conference (PAC 93), Washington, DC, 17-20 May 1993.

[2] R. Olsen, J. Dabrowski, and J. Murray Prepared for 1993 IEEE Particle Accelerator Conference (PAC 93), Washington, DC, 17-20 May 1993.

[3] J. Murray, R. Olsen, and J. Dabrowski Prepared for 1993 IEEE Particle Accelerator Conference (PAC 93), Washington, DC, 17-20 May 1993.

[4] R. Biscardi, J. Dabrowski, and M. Fulkerson, *Booster power Supply Upgrade Analysis Using Matlab*, Presentation (1996).

[5] Pentek, Inc., One Park Way, Upper Saddle River, NJ 07458, *Model 4283 Operating manual*, g.1 ed. (1998).

[6] National Instruments, 11500 North Mopac Expressway Austin, Texas 78759-3504, *PCI-6052E User Manual*, may 1999 ed., http://www.ni.com/.

[7] National Instruments, ni.com, *Traditional NI-DAQ User Manual*, april 2003 edition ed., part Number 321644L-01.