IEEE Transactions on Nuclear Science, Vol. NS-32, No. 5, October 1985

TEVATRON EXTRACTION MICROCOMPUTER

L. Chapman, D. A. Finley, M. Harrison, W. Merz
Fermi National Accelerator Laboratory*
Box 500
Batavia, IL 60510

Extraction in the Fermilab Tevatron is controlled by a multi-processor Multibus microcomputer system called QXR (Quad eXtraction Regulator). QXR monitors several analog beam signals and controls three sets of power supplies: the "bucker" and "pulse" magnets at a rate of 5760 Hz, and the "QXR" magnets at 720 Hz. QXR supports multiple slow spills (up to a total of 35 seconds) with multiple fast pulses intermixed. It linearizes the slow spill and bucks out the high frequency components. Fast extraction is done by outputting a variable pulse waveform. Closed loop learning techniques are used to improve performance from cycle to cycle for both slow and fast extraction. The system is connected to the Tevatron clock system so that it can track the machine cycle. QXR is also connected to the rest of the Fermilab control system, ACNET. Through ACNET, human operators and central computers can monitor and control extraction through communications with QXR. The controls hardware and software both employ some standard and some specialized components. This paper gives an overview of QXR as a control system; another paper [1] summarizes performance.

Design Goals

There were many challenges in designing QXR. First among them were the high speed real-time requirements. The basic slow spill bucker cycle runs at 5760 Hz. Combined with long spill lengths this demands significant memory for data taken during slow spill. These data are used between machine cycles to do closed-loop learning and smoothing.

Another requirement was incremental implementation of features as they became necessary. For example, slow spill was implemented long before fast. The sequencing needed by QXR as it tracks the machine cycle is not trivial. Communications with ACNET and with the power supplies used to control extraction were also needed.

The goals listed above are somewhat contradictory. How they were acheived is described below.

Software

Figure 1 shows a block diagram of the software modules linked to form QXR. About half of them are generic and are used in other systems. A few are borrowed from other systems (PROTO and NIL). The remaining are specific to QXR. The connections indicate closely associated modules. Typically a generic module is connected to a module defining QXR's use of that generic software. For example, the generic module PAN provides a software interface to a front panel, including routines such as LED_ON which turns a light on. The modules QXRPAN, PROPAN, and NILPAN all define uses for a front panel, including routines such as SHOW_STATE, which knows what LEDs to use to display the current state.

Modules below the horizontal line are specific to the Multibus hardware used by QXR (equivalent modules exist for other hardware environments). Interrupts, for example, use Multibus-independent generic modules

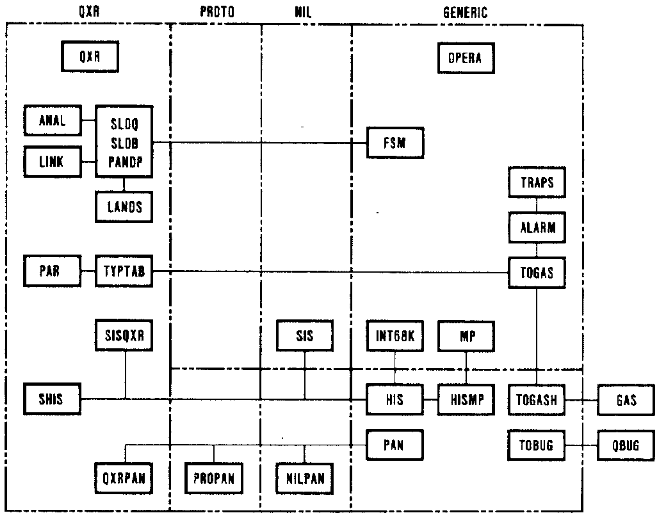

Figure 1

INT68K and MP; Multibus-dependent generic modules HIS and HISMP; and applications modules SIS, SHIS, and SISQXR.

Generic Software

Some generic software systems were already available to solve some of the design problems. These include QBUG, a software debugging tool; GAS, a software package which implements the micro end of the standard communications protocol; and PROTO, which uses GAS to download code into microcomputer RAM. All three of these programs are prom-based and reside in the micro.

In addition to the generic software systems just described, many generic software modules were used. (These modules are linked to form systems.) Some of them were created during QXR development but were written in a generic way and have since been used in other systems.

The most important generic software module is OPERA, a simple round-robin non-preemptive multitasking operating system.

A finite state machine model was chosen to implement the complex sequencing needs. Generic code, FSM, was written for this job. FSM is used by specifying a state table (including legal transitions), and, for each state, an entry routine, cyclic routine, and exit routine. The state table and these routines are written for a specific application and specify what should happen on entering a state, while in the state (at some frequency), and on leaving the state.

State changes are usually caused either by timing out or by interrupts from the clock system. They may also be caused by operator intervention but this is only done during debugging.

Other generic modules provide services for multiprocessor systems, interrupts, alarms, device drivers for the various pieces of hardware used,

*Operated by Universities Research Association, Inc. under contract with the U.S. Department of Energy.

and interfaces to the other resident software systems QBUG and GAS.

QXR-Specific Software

In addition to the generic modules several modules were written specifically for QXR. These modules define how QXR uses its I/O hardware, how it uses interrupts, what it is willing to discuss with ACNET, and most importantly, how it does learning and smoothing, and the structure of the finite state machines used. There is also a module containing default values for all parameters. These values may be changed by ACNET, by operators for example.

The heart of the extraction algorithm is in the cyclic routine for state 7 (slow spill). This algorithm is executed at 720 Hz and controls the QXR power supplies. A similar algorithm, run at 5760 Hz, controls the bucker power supplies. A simplified version is given here:

```
E (cell)  <- (I-Id)*G
Id        <- Id-dI
Q         <- M(cell) + E(cell)
```

where		
I	=	present intensity
Id	=	desired intensity
dI	=	change in desired intensity
G	=	gain
E	=	error
M	=	integrated error memory
Q	=	output to power supplies

Simple learning and smoothing techniques are used for linearizing the slow spill, removing high frequency components from the slow spill, and for adjusting fast spill levels. The learning and smoothing algorithms are an OPERA task triggerred to run between machine cycles. For each "cell" (720 Hz or 5760 Hz cycle) the following is done:

```
Learning:
            M(cell) <- L*E(cell+P) +
                       R*M(cell)

Smoothing:
            M(cell) <  -  N*(M(cell-1)+M(cell+1))
                +    (1-N)*M(cell)
```

where		
M	=	integrated error memory
L	=	learning rate
E	=	error memory
P	=	phase shift
R	=	retention rate
N	=	neighbor gain

For fast extraction, learning is done for each pulse as follows:

```
Ir <- Ib - Ia
E  <- Id - Ir
V  <- V + L*E*G
```

where		
Ir	=	intensity removed by pulse
Ib	=	actual intensity before pulse
Ia	=	actual intensity after pulse
Id	=	desired intensity for pulse
E	=	error
V	=	level output to power supplies
L	=	learning rate for pulses
G	=	gain

Software Development

Assembly language (Motorola 68000) was used for all programming. This was necessary to achieve the high speed required for the real-time code. It was also necessary for the learning and smoothing between cycles. Even though they do not run in real-time, they do have to complete before the next cycle begins, and the huge amount of data to be processed precludes the use of a high-level language. The disadvantages of using only assembly language were slower development time and awkwardness of coding (in particular, the learning and smoothing had to use a sort of fixed point arithmetic; floating point would have been too slow).

The software was developed using the BSO (Boston Systems Office) cross assembler and linker on the Fermilab Development VAX. (Serious bugs in the BSO software caused some delays.) A BSO simulator running on the VAX was used to test some code initially but was too unwieldy for testing the entire system. For that purpose, an accelerator simulator was built. A test QXR was connected to this system instead of to the real accelerator, allowing QXR to attempt to "extract" beam even before the Tevatron was operational. This made it possible for the initial version of QXR to be debugged and ready when needed.

QXR code is downloaded into RAM rather than burned into PROM. This was done mainly to speed up software development.

Hardware

Both QXR and the bucker systems use Motorola MC68000 microprocessors. The processors reside on a single board computer designed at Fermilab [2] and used in multiple applications. Each board contains 16K bytes of RAM and 32K bytes of ROM. In addition the computer board has four programmable parallel I/O ports which are used by the processor to drive a status display and as control inputs.

It was not clear at the outset whether a single 68000 processor could handle the speed requirements. Partially for this reason, Multibus was chosen to allow the possibility of splitting the job among multiple processors. The software was also designed with this in mind. It did become necessary eventually and was done, but not without considerable hardware and software problems.

The computer boards are located in a 12-slot Multibus chassis along with a number of other support boards. Figure 2 shows a block diagram of all system hardware. Two 512K byte memory boards provide data storage necessary for the long spill lengths used. RAM was chosen over disk for faster speed. An eight channel D/A card is used to generate diagnostic signals

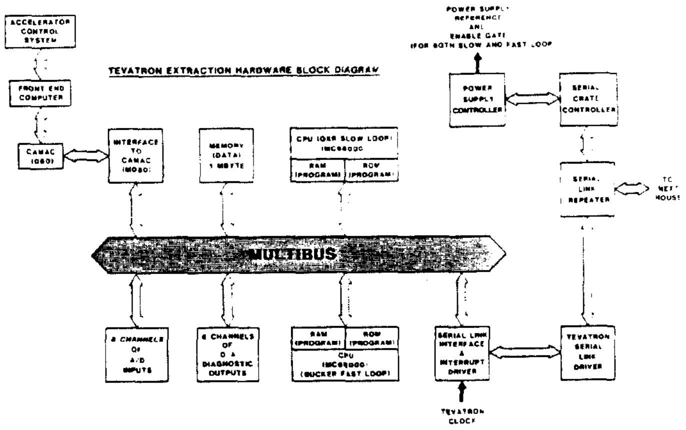
ACCELERATOR CONTROL SYSTEM
TEVATRON EXTRACTION HARDWARE BLOCK DIAGRAM
POWER SUPPLY REFERENCE AND ENABLE GATE (FOR BOTH SLOW AND FAST LOOP)
FRONT END COMPUTER
POWER SUPPLY CONTROLLER
SERIAL CRATE CONTROLLER
CAMAC (080)
INTERFACE TO CAMAC (M080)
MEMORY (DATA) 1 MBYTE
CPU (QXR SLOW LOOP) (MC68000)
RAM (PROGRAM)
ROM (PROGRAM)
SERIAL LINK REPEATER
TO NEXT HOUSE
MULTIBUS
8 CHANNELS OF A/D INPUTS
8 CHANNELS OF D A DIAGNOSTIC OUTPUTS
RAM (PROGRAM)
ROM (PROGRAM)
CPU (MC68000) (BUCKER FAST LOOP)
SERIAL LINK INTERFACE & INTERRUPT DRIVER
TEVATRON SERIAL LINK DRIVER
TEVATRON CLOCK

Figure 2

such as ideal spill, error, and power supply reference waveforms used in the main control room. An interface board containing a DMA establishes the link between the Multibus environment and ACNET's CAMAC dataway. The interface board was desinged in-house and is used in nearly all other Multibus systems at Fermilab. A second interface card connects the processor to a serial data link used for transmitting power supply waveforms around the ring. All the pieces of hardware in this dedicated link including the power supply controllers are standard items used in other systems. The second interface card also contains the electronics for decoding clock events and generating processor interrupts. Enabling and disabling of the interrupts is under processor control.

There are two four-channel A/D boards that are used for inputs to the servo loops. The A/D's run asynchronously and buffer their data into registers that the processor can access as memory locations. This eliminates time wasted waiting for conversions to finish. Each A/D channel has a gain stage on its input which is under microprocessor control. The gain is adjustable from 0-10 in .01 volts/volt steps. Some gains are adjusted automatically by the processor on each extraction cycle either as part of a signal normalization or to maximize resolution of the input. Other gains are adjustable parameters set by operators to optimize system performance.

Conclusions

The Tevatron extraction control system was implemented using as many generic hardware and software components as possible. During implementation, new generic components were designed when possible. These components are now in use in other Fermilab systems.

References

[1] W. Merz et. al., "Operation of the Tevatron Extraction System," these Proceedings.

[2] R. W. Goodwin et. al., "An MC68000 Multibus Compatible Computer Board" FN-330 Internal Fermilab Note 1980