A PROGRAMMABLE FINITE STATE MODULE FOR USE WITH THE FERMILAB TEVATRON CLOCK

D. Beechy
Fermi National Accelerator Laboratory *
Batavia, Illinois 60510 USA

Introduction

The timing requirements of the Fermilab system of accelerators have become increasingly complex with the addition of the Pbar source and subsequent collider operation of the Tevatron. To date, timing events have typically been treated as isolated markers rather than as part of a sequence of events on the Tevatron Clock.[1] However, it is often important to know not only that a clock event has occured, but also that it has occured as part of a sequence of clock events. A finite state machine is ideally suited to this task. A VME module has been designed which implements several programmable finite state machines that use the Tevatron Clock signal as inputs. This module is known as the VME Finite State Machine or VFSM. The machines of the VFSM are completely dynamic. They may be created, altered, or destroyed while in service. In addition to normal finite state machine type outputs, the module records a history of changes of state so that the exact path through the state diagram can be determined. There is also provision for triggering and recording from an external digitizer so that samples can be taken and recorded under very precisely defined circumstances.

Finite State Machines

Finite state machines have been used in the control system at Fermilab for everything from controlling the cool-down procedure of superconducting magnets to gating on counting equipment at very specific times during the Tevatron cycle. It is the ability to activate an output after an exact sequence of input combinations that make state machines applicable to a wide range of control problems.

A finite state machine is a machine that can only be in one of a finite number of states at any given time. The inputs to the machine provide the means for moving the machine from one state to another. These changes in state of the machine are called transitions. The VFSM has 8 inputs which are used to input the 8 bits of the Tevatron Clock signal (TClk) to the machine. The VFSM changes states and produces outputs in response to TClk events.

The outputs of the machine can either be triggered by certain transitions between states, or they can be active while the machine is in a particular state or states. In the first case the outputs are pulsed active for a short time while in the second they are active for as long as the machine remains in the selected state.

TClk

The TClk signal is the source of synchronization for many components throughout the Fermilab accelerator complex. It is distributed by a network of repeaters and is decoded by a large variety of modules. TClk uses modified Manchester coding to combine an 8 bit 'clock event' with a 10 MHz 'carrier' signal. The transmission of the 8 bit code is the means of synchronizing various machine processes and functions.

All major machine functions have been assigned clock events and these codes are transmitted as the various accelerators ramp through their cycles. Any equipment that has been programmed to listen for a specific set of codes can take appropriate action when any of those codes are detected.

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

State Diagrams

Finite state machines are described with the help of a state diagram. The state diagram shows all possible states that the machine can occupy along with the input combinations that cause the various transitions between the states. The outputs of the machine are also indicated on the state diagram and are either associated with the transitions between states or with the states themselves.

Fig. 1 shows a state diagram for a simple finite state machine designed to be driven by TClk. The machine has 4 possible states, as indicated by the four ovals, and only one output. The transitions between the states are labeled with both the input combination (TClk event) that causes the transition and the state of the output during the transition. The inputs are given in hexadecimal format and are just the 8 bit clock event as received from the Tevatron Clock. The state of the output is given by the 0 or 1 following the slash in each of the transitions in the diagram.

The state machine of Fig. 1 is designed to be reset to state 0 by clock event $00, and to then output pulses on the first two $E0 clock events which occur after event $D0 if event $D0 is preceded by event $C0. Note that other events may occur between $C0 and $D0. The $E0/1 convention means that the transition from one state to another is in response to clock event $E0 and that the output of the machine is to be pulsed active. The convention $ELSE/0 is intended to show that for all other clock events not explicitly shown on the state diagram, the output of the machine is to be zero. $ELSE/1 is an equally valid statement but specifies that the output of the machine is to be pulsed high.

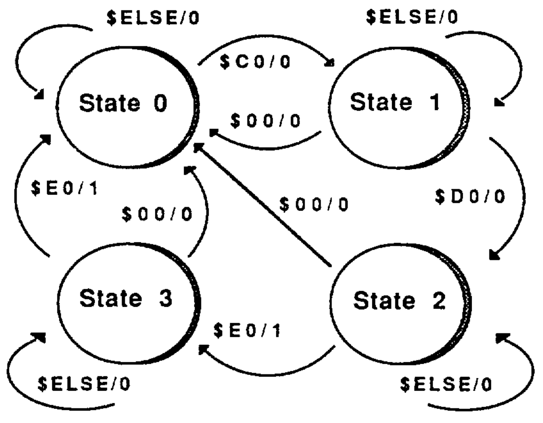

Fig. 1. State diagram of a simple 4 state finite state machine and the transitions that can occur between any two states.

Ram Based Finite State Machines

Finite state machines can be implemented with both software algorithms and in hardware using a latching circuit preceded by some amount of combinatorial logic. Software machines have the advantage of providing large numbers of states and transitions along with the ability to modify the machine without changes in wiring or the burning of new pals, etc. The major disadvantage of software based machines is that they are relatively slow compared to hardware devices and will not respond correctly to rapidly changing input combinations. Hardware machines on the other hand are fast. They can run at MHz speeds, but as implied above, they are not programmable. They cannot be modified without taking the machine out of service.

It is possible to have the advantages of both types of machines if the combinatorial logic of the hardware based state machine is replaced with random access memory. Because the contents of Ram is easily changed, the machine can be modified dynamically and since the access time of Ram is now quite short, the Ram machine can be very fast. The VFSM uses Ram to implement a finite state machine that can operate at TClk speeds. TClk events can occur as rapidly as 1 event every 1.2 microseconds and even slow Rams have access times less than this allowing less expensive versions to be used.

Fig. 2 shows the basic components of a Ram based state machine. In typical finite state machine fashion, the outputs of the state register are fed back and combined with the clock event inputs to form the total input to the Ram address lines.

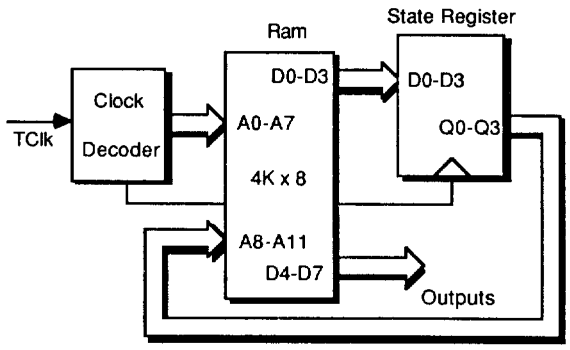

Fig. 2. Basic Components of a Ram based State Machine.

The output of the Ram is latched by the state register just prior to the end of each clock event. Programming the Ram based state machine in Fig. 2 is just a matter of filling the 4096 locations of the the Ram with the appropriate data values. The state machine of Fig. 1 provides a good example for showing how these values are derived.

The state diagram of Fig. 1 shows that for state 0 there is a transition labeled $C0/0 which points toward state 1. This transition has a corresponding Ram entry at address $0C0. Table 1 show that the data stored at this location is $01. This corresponds to the state diagram which shows that the transition is to produce no output but is to take the machine to state 1. In a similar manor, there is a transition from state 2 to state 3 labeled $E0/1. Table 1 shows the corresponding entry at location $2E0 to be $13 which will produce an active output and change the state to state 3. Notice that most table entries produce no outputs and also do not cause the machine to change state. These entries correspond to the transitions labeled $ELSE/0 on the state diagram.

There is some overhead associated with Ram based state machines that is not shown in Fig. 2. This overhead is the multiplexing that must be done to the address and data lines of the Ram to allow the normal state machine functions to take place and also to provide access to the Ram by a processor for reading and writing the data values.

◄── Address ──►		◄── Data ──►	
A11 A8	A7 A0	D7 D4	D3 D0
Present State	Clock Event	Outputs	Next State
$0	$00	$0	$0
· ·	· ·	· ·	· ·
· ·	· ·	· ·	· ·
$0	$C0	$0	$1
$0	$C1	$0	$0
· ·	· ·	· ·	· ·
· ·	· ·	· ·	· ·
$0	$FF	$0	$0
$2	$00	$0	$0
$2	$01	$0	$2
· ·	· ·	· ·	· ·
· ·	· ·	· ·	· ·
$2	$E0	$1	$3
$2	$E1	$0	$2
· ·	· ·	· ·	· ·
· ·	· ·	· ·	· ·
$2	$FF	$0	$2

Table 1. Partial contents of a Ram based finite state machine described by the state diagram of Fig. 1.

Output Records

In addition to providing the standard type of finite state machine outputs which can then be used to trigger other equipment, the VFSM has the capability of providing to other bus masters, a record or history of the states that the finite state machine entered before it produced an actual output signal. An output record is produced each time an output of the machine goes active. The VFSM also has an on board receiver for inputting a serial data value from an external digitizer. Additionally, the clock event that triggered the actual output from the finite state machine is combined with the external data value and the state machine history to form a complete output record. These three pieces of information: the digital input value, the clock event which triggered the output from the finite state machine, and the history of the finite state machine provide a means of storing a large array of data samples along with the state of the accelerator at the time each sample was taken. For example, a beam intensity sample could be taken both at injection time and at flattop and stored for later analysis. The history attached to each sample makes it possible to keep the various samples separated and effectively off loads one of the tasks of the crate processor to the processor on the VFSM.

Histories

Essential to the process of producing an output record which correlates a data sample with the state of the accelerator (as defined by a sequence of clock events) is the idea of keeping track of history. A history is a list of the states that a finite state machine enters as it responds to clock events on the Tevatron Clock. It is important to point out that the history is not merely a list of clock events. Rather, it is a list of states. Clock events cause changes in the state of a finite state machine, and as the machine changes states, these changes are recorded to form the history. When talking about the VFSM, the term 'history' means the keeping track of the changes in state that the machine undergoes as clock events are presented as inputs to the machine.

It is also necessary that the start of history is defined as one or more of the states on the state diagram that describes the machine. This simply means that history starts over each time the machine enters one of the designated states. For some state machines, this means that history can grow to be quite long. As long as the machine does not re-enter one of the selected start of history states, the history associated with that machine will continue to accumulate. The special states are programmable and can include one or all of the states on the state diagram. The state machine described by the state diagram in Fig. 1 will be used to help clarify these ideas.

For the purpose of this example, suppose that the special state that restarts history is selected to be state 0 and that the following sequence of clock events is received: $00, $2D, $07, $0F, and $C0. Clock event $00 will return the machine to state 0 from whatever state it is in initially and will also cause history to be restarted. Events $2D, $07, and $0F do not cause the state machine to change state. These clock events are included in the looping arrow labeled $ELSE/0 that is associated with state 0. The machine responds to these events by merely staying in state 0 and keeping the output at zero. The last clock event in the sequence, $C0, however, causes the state machine to make a transition to state 1. The output of the machine stays inactive, but since a change of state has occured, a history buffer is used to record the change. The history buffer is just a segment of the microprocessor's memory that is set aside for recording history. The processor updates the history buffer each time it reads the the fifo. The contents of the history buffer are shown in Fig. 3.

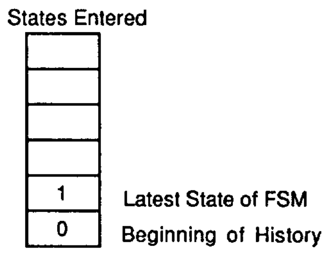

Fig. 3. History buffer showing the history of the finite state machine in Fig. 1 after the sequence of clock events $00, $2D, $07, $0F, and $C0.

Now suppose that the sequence of clock events continues with events $07, $0F, $C0, and $D0. In a similar manner as before, events $07 and $0F do not cause the machine to change state or to produce an active output pulse. There is a difference, however, in how the machine reacts to event $C0. Previously this event caused the state machine to change from state 0 to state 1, but the same clock event now causes no change in either the state of the machine or its output. The first three events of this sequence, $07, $0F, and $C0, are included in the looping arrow associated with state 1 labeled $ELSE/0. The last clock event in this second sequence, $D0, causes a similar reaction as event $C0 did in the first sequence: The machine makes a change in state from state 1 to state 2 but the output remains inactive. The change of state is recorded in the history buffer. Fig. 4 shows this addition to the buffer.

Now let the string of clock events continue with events $07, $0F, $C0, $D0, and $E0. As before, $07 and $0F along with $C0 and $D0 cause no change of state or output and are included in the looping $ELSE/0 arrow of state 2. Event $E0, however, causes a transition to state 3 and at the same time triggers the machine output to pulse high for the duration of the clock event. The history buffer is again updated because of the change of state of the machine.

Finally let the string of clock events continue with events $07, $0F, $C0, $D0, and $E0. Note that this sequence is identical to the series just previous to this. The response is also identical except that the transition is to state 0. The history buffer following this final sequence is shown in Fig. 4. After this last state is recorded, it is copied into the beginning of the buffer to show that history has again started over.

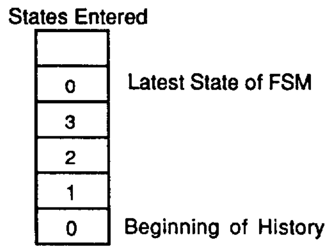

Fig. 4. History buffer showing the history of the finite state machine in Fig. 1 after the final sequence of clock events.

Block Diagram

A block diagram of the VFSM is shown in Fig. 5. In the diagram the thicker black or gray lines indicate bus lines. Fig. 5 shows that communication between the VFSM and the VME bus can either be through a dual port ram located on the VFSM or the VFSM can act as a bus master in which case communication is through shared memory located elsewhere in the crate. In either case, the VFSM uses DMA to speed up the transfer of commands and data records between the VFSM and other bus masters.

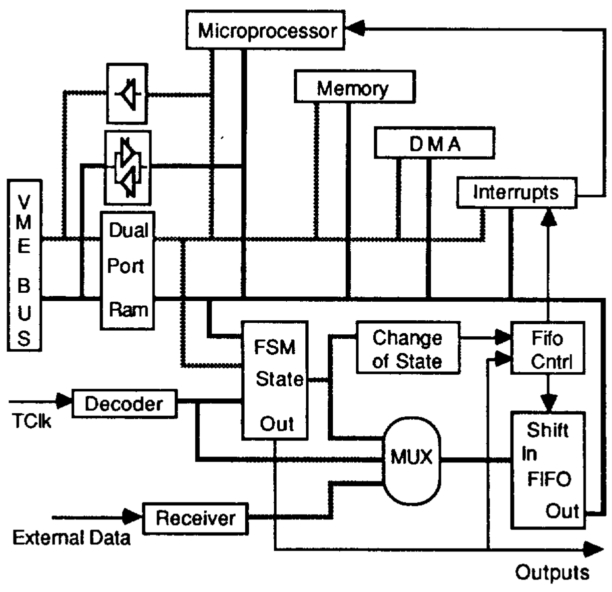

Fig. 5. Simplified Block Diagram of the VME Finite State Machine.

The block diagram shows the finite state machine as a single block with access provided to both the microprocessor and the input clock event. Access to the state machine by the processor is for the purpose of programming the state machine. This allows the characteristics of the machine to be changed without reburning proms, pals, etc. The clock decoder naturally has access to the state machine since the machine responds to Tevatron clock events.

An important component in Fig. 5 that has not yet been discussed is the fifo and the associated control and multiplexing circuitry. It is this circuitry that is the mechanism by which the VFSM provides an output record. The record does not consist of just a dump of the contents of the fifo, but rather, the contents of the fifo are used in the process of keeping track of history and the building of an output record. The fifo performs two important functions: First, it provides for high speed capture of three sources of data, and secondly, because the data are shifted into the fifo sequentially in time, the data are correlated. The multiplexor just before the fifo steers the three data sources to the fifo. The three sources of data for the fifo are: 1) the state of the finite state machine, 2) the Tevatron clock decoder, and 3) the externally supplied digital value.

The microprocessor has access to the fifo in order to empty it in response to an interrupt from the control circuit. The data in the fifo are compiled into various records and made available to other VME masters.

The block diagram also shows that the fifo is clocked or triggered to record these data by two different sources. The first source is a change of state detector. Its function is to clock the new state of the finite state machine into the fifo each time the machine changes state. This provides a record or history of how the state machine traverses its state diagram. The second trigger source for the fifo is the output of the finite state machine. This output is in turn a function of how the machine is programmed. This second trigger source clocks both a clock event and the externally supplied digital value into the fifo. As the finite state machine reacts to the Tevatron clock, changes in the state of the machine are detected and clocked into the fifo. Each time the state machine changes state, the number of the new state is loaded into the fifo. This 'history' makes up one of the fields of the output record. Eventually, the state machine produces an output which triggers the recording of the remaining two fields. The first of these is the clock event that triggered the finite state machine to produce an output and the second field is the external digital input value. The control circuitry appends a type code bit field to each entry as it is pushed into the fifo. The fifo makes building the record straight forward because the data are recorded in the fifo in chronological order.

Processor Tasks

A multitasking real-time executive is used to schedule the various tasks that the VFSM microprocessor performs. Some of the more important tasks are listed below:

1. Read Fifo Contents.
2. Update History Buffer.
3. Assemble Output Records.
4. Process Commands from Bus Master(s).
5. Service a Local Terminal.

Most of the above tasks are interrupt driven. For example, the microprocessor will only read the fifo when it receives an interrupt from the fifo control circuit.

Item #4 in the list above is a major task. A large number of finite state machine related commands are supported. A representative sample is given below:

1. Create/Destroy machine M.
2. Create/Destroy state S, of machine M.
3. Create/Destroy transition T, of state S, of machine M.
4. Force machine M to state S.
5. Enable/Disable machine M.
6. Return Output Record for machine M.

OOC

New system software[2], written at Fermilab for a multiprocessor VME environment, is used in the VFSM. This software is called Object Oriented Communications or OOC for short. The OOC code is just another task that runs under the multitasking executive. OOC was written to provide a standard communications protocol between VME bus masters. It treats both the source and the destination of inter-processor messages as objects. OOC provides the standard functions associated with object oriented languages. It can create and destroy objects, it allocates space for instance variables, and it implements an inheritance type class structure by maintaining tables of pointers to the various access methods associated with each of the classes. Messages to objects are processed by the OOC task by searching the class structure to find a pointer to the access method required by the message type. Fig. 6 below shows the format for an OOC message.

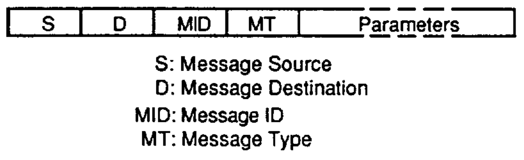

Fig. 6. OOC message format.

In the VFSM, the finite state machines become objects as do the states of the machines. Even the transitions between states are objects. The instance variables of a transition include the state from which the transition originates and the state towards which the transition points. A standard set of access methods is provided by OOC as well as a provision for user supplied routines.

Conclusion

The VFSM extends the usefulness of TClk by providing a finite state machine that can be programmed to recognize very specific sequences of clock events corresponding to well defined accelerator conditions. The output records produced by the VFSM allows data samples to be precisely tagged with the state of the accelerator at the time the samples were taken. Future enhancements will include a companion module that will allow any set of digital input signals to control the finite state machines and not just TClk. This system will then provide a programmable high speed interface between the VME environment and external devices.

Acknowledgments

The author wishes to acknowledge the design contributions of C. McClure, J. Ticku, and J. Utterback whose hard work and enthusiasm toward this project have added significantly to the capabilities of the VFSM.

References

[1] D.G. Beechy and R.J. Ducar, "Time and Data Distribution Systems at the Fermilab Accelerator", presented at the Second International Workshop on Accelerator Control Systems, Los Alamos, NM, October 7-10, 1985.

[2] L.J. Chapman, "Object-Oriented Communications", Fermilab internal memo (1987), unpublished.