THE LHC CENTRAL TIMING HARDWARE IMPLEMENTATION

P. Alvarez, J. Lewis, J. Serrano CERN, Geneva, Switzerland

Abstract

The LHC central timing requirements are very different from those of the injector chain; not only is machine safety and reliability critical, there are other important differences that have forced a new approach. Unlike the injector chain, the LHC processes can not be usefully broken up into basic time periods and cycles, rather they are independent, asynchronous, and of arbitrary duration. This paper presents the MTT, the new multitasking event generation hardware we developed to control the LHC machine processes.

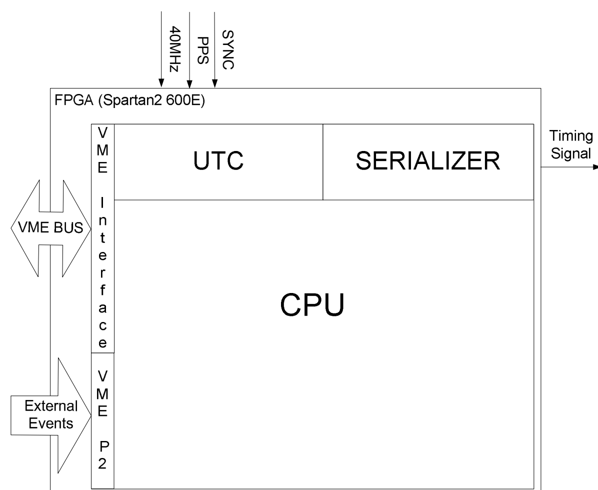

Figure 1: MTT Block diagram

MTT ARCHITECTURE

The MTT has been developed having in mind the new LHC sequencing requirements. In LHC, processes are independent from each other and do not adapt to a periodic structure as in the injector chain [1]. The MTT, implemented in the same hardware as the previous Timing Master (CTG), contains a CPU that runs up to 16 tasks in a round robin scheme. Tasks can be executed, loaded and stopped independently from each other in a deterministic way. Timing events are sent by writing to a special register of the Tasks Register map.

UTC AND SYNC REGENERATION

The MTT accepts as input a Sync Pulse, 1 Pulse Per Second (PPS) and a 40MHz clock, which is fed as reference to an onboard quartz Phase Locked Loop (PLL). The PPS is used to set the phase of a 40MHz counter which regenerates the millisecond and second. This counter is initialized via the VME bus with the correct UNIX time. The Sync pulse signals the beginning of the machine basic period (900 ms or 1200ms for the PS complex and 1000ms for the LHC).

SERIALIZER AND TIMING FRAMES

The serializer builds and broadcasts a 1MHz Manchester encoded timing signal containing UTC frames, millisecond frames and CPU send requests. Frames always contain 4 bytes of data sent in 125us slots. Millisecond frames are always sent periodically, whereas UTC frames are low priority and can be discarded. CPU send requests are stored in a 256 frames FIFO. Alarms are raised if the CPU send request is not treated in the corresponding millisecond, which can be allowed in normal operation, or if the FIFO is full.

Each 32-bit timing frame is broken up into two 16-bit parts; a header and a payload. The header identifies the kind of frame it is, for example a UTC time frame, a telegram frame, or a machine event frame. For some header types, the header contains a code; in particular machine and telegram events contain an 8-bit code. For telegram frames the code denotes which telegram parameter the frame represents, while event codes denote which event it is, for example *start-ramp* or *post-mortem*.

The payload carries specific information that depends on the header. The two UTC frames carry the 32-bit UTC time, and in the LHC, the machine event payloads vary according to the event code. For example the LHC beam intensity event carries the intensity per bunch encoded in units of 10^{10} protons; while the machine mode event carries the mode enumeration in its payload.

MTT CPU

The MTT CPU was inspired by the TMS9900, which maps all of its registers to external memory. A single workspace register (WP) points to a set of 16 registers. This provides a simple mechanism for context switching, where the store of 16 registers becomes equivalent to storing and updating the WP. In the MTT all the local registers and memory are mapped to internal RAM. A scheduler updates the WP every instruction cycle in a round robin scheme, resulting in a continuous and deterministic parallel execution of every task.

Tasks Register Map

The MTT task register Map consist of 256 locations:

- *32 Task registers.* Task register #0 is used as an index in relative move instructions.
- *218 Global registers* visible by every task.
- *6 Special registers:*
    - *MSFR: Millisecond free running counter, Read only.* Used to wait for a relative number of milliseconds.

- *MSMR: Millisecond modulo Read/Write.* Sent as payload in the millisecond frame.
- *SYNC: Sync free running counter Read only.* Used to wait for a relative number of sync periods.
- *TSYNC Time from sync, Read/Write.* Milliseconds since last sync arrived.
- *EVOUT: Timing frame out Read/Write.* It writes into the frame serializer FIFO and reads the frame being sent on the cable.
- *VMEP2: External events read only, clear on read.*

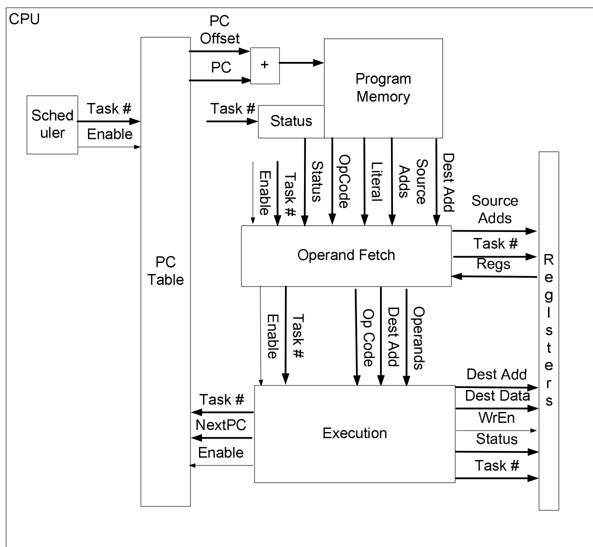

Figure 2: MTT CPU Block diagram

Instruction Format

The instruction format is triadic with a fixed length of 64 bits. There are three possible formats: Register, Register to Register and branch instructions.

	63..40	39..32	31..24	23..16	15..8	7..0
a)		Source1	Dest	Source2	OpCode	Reserved
b)	Literal	Literal	Dest	Source2	OpCode	Reserved
c)		Program Memory Add			OpCode	Reserved
		43..32	31..24	23..16	15..8	7..0

Figure 3: Instruction Format: a) Register, Register to Register; b) Literal, Register to Register and c) Branch instructions

Program memory

Instruction Set

The MTT instruction set consists of:

- No operation.
- Logical operations: Or, And, Xor, Left shift, Right shift.
- Arithmetic operations: Add, Substract.
- Move operations: Value to a register, register to a register, indexed register to register, register to indexed register.
- Program flow: Halt, Jump, Branch if equal, Branch if not equal, Branch if less than, Branch if greater than equal, Branch if less than equal, Branch if Carry.
- VME Interrupt.
- Wait instructions: Wait relative, Wait Or and Wait Equal.

A significant effort has been put in designing a generic instruction set. All logical, arithmetical and wait instructions can use *Source1* as a literal or a register address. Sending an event on the timing cable is equivalent to writing to the EVOUT register. Thus adding a new IO device, such as an additional serializer, just requires a connection to the Memory Map.

In *Wait Equal* the task waits until the two sources are equal; in *Wait Or* it waits until *(Source1 and Source2)* $\neq$ *0*; finally *Wait Relative* is used to wait on a free running counter. It adds *Source1* to *Source2* and stores it into a reference register (*milestone*). The wait state ends when *Source2 = milestone*. This scheme allows the 16 tasks to have independent programmable delays using a single counter and internal RAM.

CPU Block Diagram

- *Scheduler.* Every time a task is enabled or stopped the scheduler builds an execution table. Tasks are executed from lowest task number to highest. The scheduler provides the executed task number and the task enable bit, which are pipelined to the rest of CPU blocks.
- *Program Memory.* The MTT can store 4K instructions in internal memory. The program memory address read by a task is calculated by adding to the task's PC a second register called PC offset. Thus a task can be reallocated dynamically in memory without need to stop it.
- *Status Block.* It keeps flags such as Illegal opcode, illegal register, illegal value, running, waiting or halted. It also stores the milestone for the *Wait Relative.* The status block is read at the same moment as the Program Memory and updated by the Execution Unit.
- *Operand Fetch Unit.* The Operand Fetch unit decodes the operation code and fetches the corresponding registers.
- *Execution Unit.* The Execution Unit calculates the result data, the new PC and status register.
- *Register Block.* The register block stores local, global and special registers. The access is transparent to the Operand Fetch Unit.

Instruction Flow

The execution stages of the MTT CPU are the following (Figure 4): PC Fetch (PCF), Operand Fetch 1 and 2 (OPF1, OPF2), Execute (EX) and Store (ST). If a branch instruction is executed, the PC should be read in PCF before it has been stored in ST; otherwise the task will jump to an incorrect location. If less than 3 tasks are executed, branch dependencies are avoided by inserting void instruction cycles.

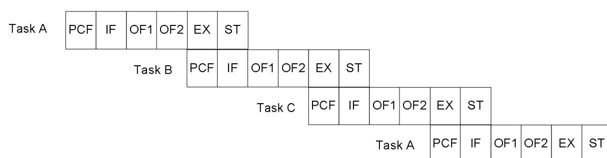

Figure 4: Instruction Flow

VME ACCESS

VME access is D32A24. Registers implemented in internal RAM, such as Program Memory, TCB table, local, global registers and task's status can be read or written at any time by the VME interface. This is achieved by time multiplexing the access of the MTT CPU and the VME interface to the corresponding Block RAM.

Table 1: MTT FPGA Device utilization summary

Device Utilization Summary (xc2s600e-6fg456)			
Logic Utilization	**Used**	**Available**	**%**
Number of Slice Flip Flops	2,268	13,824	16%
Number of 4 input LUTs	3,804	13,824	27%
Logic Distribution			
Number of occupied Slices	2,673	6,912	38%
Number of Slices containing only related logic	2,673	2,673	100%
Number of Slices containing unrelated logic	0	2,673	0%
Total Number of 4 input LUTs	4,294	13,824	31%
Number used as logic	3,804		
Number used as a route-thru	276		
Number used for Dual Port RAMs	212		
Number used as Shift registers	2		
Number of bonded IOBs	154	325	47%
Number of Block RAMs	72	72	100%
Number of GCLKs	2	4	50%
Number of GCLKIOBs	3	4	75%
Total equivalent gate count for design	1,239,614		

FPGA IMPLEMENTATION

The MTT has been fully developed in VHDL. It was fitted in the same PCB used by the previous timing master, the CTG. The target FPGA is a Spartan2E 600. The design makes extensive use of internal RAM, either distributed RAM (i.e. task's status, task's milestones) or block RAM (i.e. program memory, task's registers). The MTT maximum frequency is 50MHz and runs at 40MHz.

TASKS EXAMPLES: EVENT TABLES

An event table is a list of entries, each of which contains the name of the event to be broadcasted over the LHC timing cable, the relative time to wait before sending it, and its payload. These tables model LHC machine processes and are controlled from the LHC Software Architecture LSA timing service. The event table is sent from LSA to the LHC central timing gateway as a byte stream. On arrival the event table compiler builds an ASCII file suitable for input to the MTT assembler that will become the event task. The compiler first translates each entry in the table into wait and move instructions.

```
WRLV < time>   MSFR          ;Wait <time> ms
MOVV <frame> EVOUT           ;Send <frame>
```

The resulting block of instructions is then included into an event table task template to become a full event task suitable for assembly. The template contains code that manages the task synchronisation and its run count. The resulting task is assembled into position independent object code. The task loader then places it in memory, initialises its control block and runs it. LSA must now specify a synchronisation event and run count to start producing event frames from it.

The length of an event table can vary form only one event entry to thousands, and event tables are loaded and unloaded under LSA control. The task loader must manage MTT program memory so that each task has the memory it needs. The task PC offset register permits the task loader to pack fragmented memory containing running tasks into contiguous memory. This can be achieved by copying a task from one address to another unused portion of memory and changing its offset register so that it executes at the new location without being disturbed.

CONCLUSIONS

The MTT satisfies the LHC sequencing requirements. Tasks can be uploaded, started and stopped without disturbing other tasks. A case example has been presented where event tables are compiled by operators and dynamically reloaded on the MTT. The program memory, register map, instructions set and number of serializers could be easily scaled in future versions; opening the door to a compact integration of the CERN's General Machine Timing System.

REFERENCES

[1] Julian Lewis, Pablo Alvarez, Jean-Claude Bau, Stephane Deghaye, Ioan Kozsar, Javier Serrano "The CERN LHC Central timing, a vertical slice" ICALEPCS 2007, CERN, Geneva

[2] TMS 9900 Microprocessor data Manual http://www.bitsavers.org/pdf/ti/_dataBooks/TMS9900_DataManual.pdf