NEW DEVELOPMENTS ON THE FAIR DATA MASTER

M. Kreider, R. Bär, D. Beck, W. Terpstra, GSI, Darmstadt, Germany
J. Davies, V. Grout, Glyndŵr University, Wrexham, United Kingdom

Abstract

During the last year, a small scale timing system has been built with a first version of the Data Master. In this paper, we will describe field test progress as well as new design concepts and implementation details of the new prototype to be tested with the CRYRING accelerator timing system. The message management layer has been introduced as a hardware acceleration module for the timely dispatch of control messages. It consists of a priority queue for outgoing messages, combined with a scheduler and network load balancing. This loosens the real-time constraints for the CPUs composing the control messages noticeably, making the control firmware very easy to construct and deterministic. It is further opening perspectives away from the current virtual machine-like implementation on to a specialized programming language for accelerator control. In addition, a streamlined and better fitting model for beam production chains and cycles has been devised for use in the data master firmware. The processing worst case execution time becomes completely calculable, enabling fixed time-slices for safe multiplexing of cycles in all of the CPUs.

OVERVIEW AND SYSTEM LAYOUT

As discussed in our previous papers [1, 2], the FAIR accelerator will be a highly complex system which needs a control system to match. This suggests a design that supports high performance, flexibility and deterministic command generation and distribution. While all machine commands for beam production will be calculated from physics data ahead of time, all final scheduling and deterministic delivery is the responsibility of the Data Master (DM). The DM itself is a hybrid of an industrial PC and a Field Programmable Gate Array (FPGA) based embedded real-time system with hardware acceleration modules. We will now discuss the system layout of the current implementation and the inner workings of the sub-modules in greater detail.

CONTROL DATA

Structure

The control data received by the DM broadly resembles a flowchart for beam production. It consists of $2..I$ alternative beam production scenarios, called plans. Each plan has $1..J$ event chains in it. Chains are the basic building blocks. They each contain $0..K$ command messages. They also come with the means for simple control structures. The input format is currently XML based and converted to a binary format for the embedded system.

Time

From the start of a plan, all times are relative offsets. Each chain has a given duration, and chain start times are calculated by adding up previous durations. The only exception to the rule is a conditional wait. Here, the start time of the next chain is set to time the condition was fulfilled, plus a fixed offset. An absolute execution time is calculated for each command message at the moment it is dispatched. This is done by adding the message offsets to its chain's start time.

EMBEDDED SYSTEM

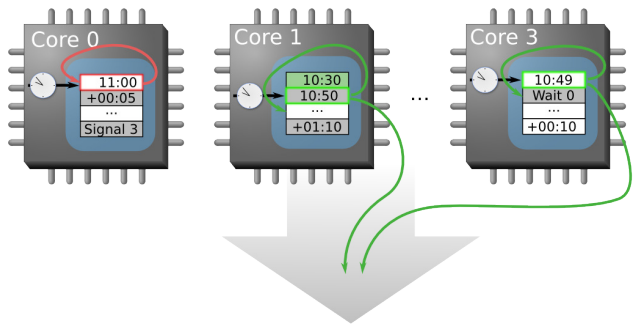

Figure 1: Scheduling commands in the Soft-CPU Cluster.

Layout and Firmware

The LM32 (Lattice Mico 32 Micro-controller) is a 32 Bit RISC processor for use in Field Programmable Gate Arrays (FPGAs) and Application Specified Integrated Circuits (ASICs) and a good choice for a control system [3]. Timer Interrupts were unsuitable for dispatch due messages, because saving and restoring all 32 registers takes considerable time and not all instructions have the same execution time. Furthermore, it would have meant the processors being idle most of the time. In order to build a deterministic system with a low reaction time, a polling approach with fixed time slices was used and multiple Soft CPUs were instantiated to deal with different parts of beam production in parallel, aided by hardware acceleration cores. Each of the LM32 runs very simple firmware with three responsibilities for each iteration. The first is synchronization, meaning checking conditions or signalling another process(or). Next comes processing the current chain. This means sending a due command message to the priority queue. The third is to check the command interface for external instructions from the control system. All worst case execution times are completely deterministic. The only exception would be dispatch, because the network interface is a shared resource for all cores. The current testbed only features one thread

Copyright © 2014 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-146-5

per LM32 as yet. Figure 1 shows how the control data is evaluated in each iteration by the CPU cluster.

DEDICATED HARDWARE

This leaves the problem of arbitration and gathering these messages. They must also be ordered by urgency before adding them to network packets because different commands will have different requirements for control lag. Kicker Magnet control at FAIR will require a lag below 500 µs while commands to the vacuum system can easily be implemented with a margin of seconds. Last but not least the payload/packet-size ratio should be high in order to make the most of the limited bandwidth. An additional core with a priority queue was introduced to deal with these requirements, making the whole system a heap on top of a calendar queue. The messages need to be wrapped in the EtherBone (EB) [4] network protocol when adding them to a network packet. In our earlier prototype, we used a full port of the x86 EB library. This is very impractical for various reasons and a wasteful solution in terms of RAM. Instead, an EB Master was implemented as a hardware core, providing a simple and fast bridge from the local Wishbone (WB) [5] bus over the network interface to the remote WB bus of the controlled endpoints.

Priority Queue

All command messages consist of device parameters and their execution timestamp. The priority queue uses the timestamp part to sort the messages in order of urgency, device parameters are payload and will reside in an extra RAM. There are several ways to implement priority queues, depending on the focus. In our case, we aim for minimal execution time of the get-Min and extract-Min operations, which returns the minimum key element.

In order to get a better performance than sequential reordering in software solutions can provide, multiple parallel accesses to the RAM are necessary. The underlying constraint for efficient sorting is the ability to read both child nodes at the same time and write the falling node to the former parent. FPGAs natively only provide dual port memory (DPRAM). An ASIC design can have more read ports to the same content without any problem. If this is to be emulated in an FPGA, memory and routing costs increase considerably.

RAM-based Heap At least two independent read ports are required on the RAM holding the sorting keys, so both left and right child node can be read at the same time. With the moving element in a register, it is possible to compare parent and both children in a single cycle. For reordering, an independent write port is required, so the minimum is a 3 port RAM (2 read, 1 write). Many manufacturers provide macro cores which emulate a multi-port RAM with 2 read and 2 write ports. However, the synthesizer can only achieve this by replication. So instead, it is more efficient to exploit the fact that the read ports never access the same child node and keep left and right children in two different DPRAMs.

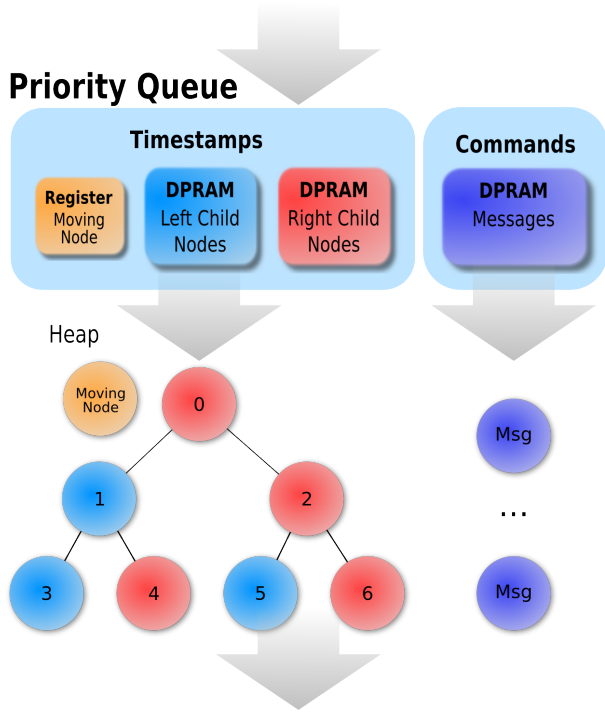

Figure 2: Heap-based priority queue in FPGA DPRAMs.

As Figure 2 shows, no redundant copy of the keys is necessary, and the arbitration logic is mostly Multiplexers and Write enable signals controlled by the least significant bit of the node index. The path taken through the heap is a sequence of indices. Placed in shift register, it can be used as addresses to order the message payload with constant delay to the key sorting. This simple design can implement *INSERT* and *REMOVE* Operations in $O(log(n))$ time, with one comparison/move per clock cycle per heap level plus two cycles delay and minimal memory cost. But there is still much room for improvement.

Optimizing the Solution Firstly, *REMOVE* operations can only be pipelined if all heap levels are accessible in parallel [6]. Secondly, because *REMOVE* and *INSERT* normally work in different directions, pipelining of mixed operations is prevented. At the cost of fan-out and additional arbitration logic, one can reformulate the *INSERT* operation to work from top to bottom as well. In addition, two DPRAMs, one for left children, one for right, are necessary per level so each level is a pipeline stage that can move one element and all operations must work from top to bottom.

Both solutions combined are likely to be the most high-performance implementation of this design, managing one *INSERT* or *REMOVE* per two or three clock cycles, depending on desired design frequency. This is, however, slightly overpowered for our scenario for two reasons. The first lies in the layout of memory blocks in modern FPGAs, which are too big for the top heap levels and would result in wasted

ISBN 978-3-95450-146-5

Copyright © 2014 CC-BY-3.0 and by the respective authors

memory [7]. The second reason is possible maximum speed. As we want a low system reaction time and elements in the heap are committed to be sent, it is not sensible to provide more than two EB packets(1500 Bytes - header) [4] of command messages. This makes the max. heap elements $n = \frac{2(1500B-8B)}{(8+2)4B} = 74.6$ and max. heap depth would be $\log_2(n) = 7$. For our design, this equals a min. dequeueing time of $T_{d\ min} = n + 2 = 9$ clock cycles. Taking a look at the timing constraints for queuing and dequeuing, we have relaxed conditions on the input side and leave a larger margin for packet processing. On the output side, we want the possibility to dequeue with line speed and catch up after delays. In our scenario, the 64 bit wide keys to be sorted come with a payload field of 192 bits, which equals 8 words at 32 bit. Absolute minimum output would be line speed of the Gigabit Ethernet interface, which is $8\ bit/125\ MHz \rightarrow 256\ ns$ dequeue, bus speed is $32\ bit/62.5\ MHz \rightarrow 128\ ns$ dequeue $\rightarrow$ 8 Clock cycles. So a dequeuing time of $8 <= T_d < 16$ clock cycles already works well with a heap depth of 6 to 13, making the simpler implementation a valid approach for our timing requirements.

EtherBone Master

The EtherBone Master(EBM) core translates WB bus operations into the EB network protocol [4]. The main obstacle was a basic property of WB, which is it being a cycle based bus. This means that all operations have to be completed/acknowledged before a cycle can be finished. Over the network, this would lead to very high lag, stalling a local bus device, with the added risk of packet loss and therefore unacknowledged operations stalling the bus indefinitely. Our solution was to design a core which presents a simple interface to the user and fully conforms to the wishbone standard [5] and avoids the risk of freezing the local bus.

EBM Design It consists of two distinct WB slaves. The first is used for configuration, such as source and destination addresses for the network layer and additional information needed by EB [4]. The second slave acts as a write-only FIFO Buffer for WB operations to be sent. This way, the EBM can acknowledge all incoming data immediately, since it is not waiting for remote data. All replies go to the local EB Slave core. This now leads to several problems, first and foremost that directly using address and value of local WB operations is not possible. The reason is that the EB Master is itself a WB slave, occupying a certain address. The high address bits depend on its own position in the crossbar hierarchy and might be different from those of the remote target device. Another two bits are needed because control and data interface as well as read and write operations have to be distinguished. Our solution is therefore to replace all high address bits by the content of a control register which must be set according to the remote target address.

In the case of a WB write operation, address and value have the normal meaning except for the high address bits. A read operation is different. It is turned into an EB write on the destination platform, writing back the requested values to the source. For this purpose, the address is the location to be read on the destination, but the value is the location on the source's bus where the return value is to be written. A flush command to the control register delivers the content to the network interface.

Fitness for DM Because command messages will always follow the same 8-word-write scheme, it is true that meta data is known in advance and a faster and much simpler core would have sufficed for the DM. The fixed format still makes encoding command messages deterministic. The EBM therefore turned out to be a good fit with added flexibility, obviating the need for a DM specific solution. It is also a powerful component, suitable for generic remote WB access.

The introduction of a generic representation for accelerator control in form of an XML string has already proven a great help for rapid development and generating test data. It is also invaluable for debugging. With the addition of hardware modules for sorting and sending command messages, most sources for non-determinism could be removed from the DM design. Both modules have been tested with the new data format on a quad-core embedded system in a lab environment and tests to generate timed pulses on a timing endpoint were satisfactory.

The DM has yet to be fully integrated with the FAIR control system and further to prove its ability to control the real CRYRING accelerator by early 2015.

REFERENCES

[1] M. Kreider, R. Bär, D. Beck, J. Davies, and V. Grout. The fair timing master: A discussion of performance requirements and architectures for a high-precision timing system. In *Proc. of the International Conference on Internet Technology and Application ITA*, September 2013.

[2] R. Bär, T. Fleck, M. Kreider, and S. Mauro. The timing master for the fair accelerator facility. In *Proc. of the International Conference on Accelerator and Large Experimental Physics Control Systems ICALEPCS*, pages 642–645, October 2011.

[3] W. Terpstra. The case for soft-cpus in accelerator control systems. In *Proc. of the International Conference on Accelerator and Large Experimental Physics Control Systems ICALEPCS*, pages 642–645, October 2011.

[4] M. Kreider, R. Bär, D. Beck, W. Terpstra, J. Davies, V. Grout, J. Lewis, J. Serrano, and T. Wlostowski. Open borders for system-on-a-chip buses: A wire format for connecting large physics controls. *Phys. Rev. ST Accel. Beams*, 15:082801, Aug 2012.

[5] Wishbone b4 wishbone system-on-chip (soc)interconnection architecture for portable ip cores. Technical report, OpenCores, 2010.

[6] Wojciech M. Zabołotny. Dual port memory based heapsort implementation for fpga. volume 8008, pages 80080E–80080E–9, 2011.

[7] Embedded memory blocks in arria v devices. Technical report, Altera Corperation, 2014.

Copyright © 2014 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-146-5