Design Automation for Embedded Systems, 7, 341–364, 2002.

Reprogrammable Platforms for High-Speed Data Acquisition

J. LUDVIG jludvig@lbl.gov
Lawrence Berkeley National Laboratory, Berkeley, CA 94720

J. MCCARTHY jim@agiledesigntech.com
Agile Design, Inc., Walnut Creek, CA 94596

S. NEUENDORFFER neuendor@eecs.berkeley.edu
EECS Department, U.C. Berkeley, Berkeley, CA 94720

S. R. SACHS ssachs@eecs.berkeley.edu
EECS Department, U.C. Berkeley, Berkeley, CA 94720

Abstract. Complex embedded systems that do not target mass markets often have design and engineering costs that exceed production costs. One example is the triggering and data acquisition system (DAQ) integrated into high-energy physics experiments. Parameterizable and reprogrammable architectures are natural candidates as platforms for specialized embedded systems like high-speed data acquisition systems. In order to facilitate the design of specialized embedded systems, design strategies and tools are needed that greatly increase the efficiency of the design process. End-user programmability of reprogrammable platforms is essential, because system designers, without training in low-level programming languages, are required to change the base design, compare designs, and generate configuration data for the reprogrammable platforms. This paper presents a methodology for designing and evaluating high-speed data acquisition systems using reprogrammable platforms.

1. Introduction

High energy physics experiments study properties of elementary particles. Accelerator based experiments can currently produce particle energies up to a few TeV while cosmic rays are found up to 10^{21} eV.

The design of these experiments and specifically the detectors requires highly specialized systems engineering because they have to satisfy performance requirements which are often close to the physical limits of the available technology while the funding is often limited.

The data acquisition system (DAQ) of a physics experiment captures the data generated by a detector. A DAQ system simulation model includes a number of distinct components:

— An *event generator* simulates properties of the physical events under observation.

— A *detector model* simulates the detector technology and must include its sensitivity and the errors it produces.

- An *analog DAQ front end model* simulates the analog signal processing and digitization process, including its limitations.
- A *digital DAQ back end model* captures the performance of the low-level data acquisition system, e.g., buffer memory size, dead-time, latency etc.
- A *CPU farm*, an application specific multiprocessor system which controls the experiment, performs event triggering and building. An event trigger is a filter that analyzes critical parts of every data set, passing to the event builder only those events that satisfy certain *trigger conditions*, such as total energy, hit multiplicity, signal to noise ratio, etc. The event builder sorts, formats, and compresses the raw data in such a way that all relevant information can be stored on disk or tape media for later retrieval and analysis.

This framework is valid for the majority of DAQ systems in high energy physics and cleanly defines all interfaces between the physical and technical sub-systems.

An important assumption made to simplify the engineering process is that the performance required of each component is completely specified by the stage in front of it. While detector specific analog front-end circuits are required, generic digital backends based on a reconfigurable architecture and a library of physics specific digital function blocks like digital filters, feature extractors, trigger circuits, etc., can serve a variety of experiments. The historically "manual" system integration strategy of these designs can be automated if the DAQ is treated like an SoC (System on Chip) design. The latter can create systems of very high complexity by properly interfacing existing function blocks.

The key to creating complex designs from library elements is a design tool which uses interfaces which are correct by design and transparent to the user. In effect even less experienced (physics) users can model, design, modify and upgrade large parts of the DAQ after the system becomes operational.

2. The Ptolemy II Modeling Framework

Systems such as the one considered in this paper are *heterogeneous* in the sense that they are composed of subsystems with different characteristics which interact in a variety of ways. Many tools have been developed for modeling individual aspects of such systems, e.g., data and control flow, analog or digital subsystems. In order to evaluate the complete system, these models need to be composed, and the interactions between subsystems can lead to hard-to-analyze, undesirable and unexpected behavior. An in-depth discussion on the modeling of heterogeneous systems is presented in [4].

In the Ptolemy II framework [3], [10], subsystems are hierarchically composed so that the properties of the complete system can be simulated without having to resort to adhoc integration of multiple models. These subsystems are able to communicate through asynchronous, synchronous, buffered, and unbuffered mechanisms.

The basic Ptolemy II building blocks are called *actors*. Actors atomically execute a task and communicate with other actors through ports. Communication channels are established by connecting ports between actors. A *composite actor* is a set of connected actors, whose execution is the controlled execution of the actors it contains.

The model of computation associated with a composite actor is implemented in Ptolemy II as a *domain*. Domains define communication semantics and execution order among actors. The communication mechanism is implemented using *receivers*. Receivers are contained in input ports, and there is one receiver for each communication channel. Receivers could represent FIFO queues, mailboxes, proxies for a global queue, or rendezvous points.

The communicating sequential processes (CSP) domain represents actor processes that communicate by instantaneous rendezvous [5]. The continuous time (CT) domain [12] models ordinary differential equations (ODEs), extended to allow the handling of discrete events. Special actors which represent *integrators* are connected in feedback loops in order to represent the ODEs. In the discrete event (DE) domain, actors communicate through events placed on a (continuous) time line. Events have a value and a time stamp, and are processed in chronological order. The synchronous dataflow (SDF) domain [11] is a special case of a Process Network (PN) [7]. An actor executed in an SDF model consumes a fixed number of tokens from each input port and produces a fixed number of tokens to each output port. Finite-state machine (FSM) domain [10] entities are states, and inputs to a FSM actor result in state transitions. Outputs of a FSM actor are associated with state transitions. For a more detailed discussion of domains in Ptolemy II, see [10] , [3].

3. High Energy Physics Events

The DAQ system under investigation was primarily designed for neutrino astrophysics experiments. Since the physics of these experiments is very simple, they are good examples for applying the proposed design strategy. In order to better understand the requirements of the DAQ system, we briefly summarize the physics.

3.1. Neutrino Astronomy Experiments

Neutrino astrophysics experiments are intended to detect high energy neutrinos in the cosmic ray flux. The neutrinos are generated by violent cosmic processes like *Gamma Ray Bursters* (GRBs), believed to occur in the final seconds of the colliding components of binary neutron stars and *Active Galactic Nuclei* (AGNs), the massive black holes in the center of most galaxies.

Neutrinos are very weakly interacting particles and can penetrate galactic dust clouds and even the entire earth without being stopped. They are therefore ideal carriers of information about astrophysical objects which are shielded by large amounts of interstellar matter.

Despite their weak interactions, some neutrinos hit nucleons and generate high energy secondary muons. The electrically charged muons can be detected directly. At very high energies ($>$100 GeV) the average muon energy is approximately half the neutrino energy and the muon track points in the same direction as the neutrino's momentum vector (with an error of one degree or less). Hence a particle detector which can measure energy and momentum of the muons is a telescope which can map the angular distribution of neutrino sources in the sky with a resolution of approximately twice the angular diameter of the moon (which is 30 seconds). However, at these energies, the neutrino induced muon flux is so small that detectors have to cover an area of one km^2 and a volume on the order of one km^3 or more to detect astrophysical neutrinos with statistical significance.

Very few detector technologies can instrument such a large volume. The predominant method is based on optical detection of Cherenkov radiation. Cherenkov radiation is electromagnetic radiation generated by all charged particles (in this case the muons) moving faster through a polarizable medium than the local speed of light. Cherenkov radiation can be detected from the radio spectrum all throughout the far ultraviolet. The power spectrum of Cherenkov radiation is proportional to the frequency of the emitted photons and the visible fraction of the Cherenkov spectrum therefore looks blueish.

Liquid water and ice are the only Cherenkov media which are available in the required volume. Pure water and ice have very low optical scattering and absorption coefficients for blue and near ultraviolet light and are almost ideal Cherenkov media. Neutrino experiments must be shielded against light and background muons generated in the upper atmosphere by cosmic rays. They are either located in underground laboratories at the bottom of deep mines (like the Kamiokande and Super-Kamiokande experiments which are designed to detect low energy solar neutrinos) or in the deep ocean (Dumand, Nestor [13], Antares [1]) or the Antarctic ice shield (IceCube [6]). Cherenkov radiation is very faint; only a few tens of photons are generated per cm track length. These photons have to be detected by photomultiplier tubes (PMTs) with 8″–12″ diameter which are enclosed by transparent, pressure resistant optical module (OM) spheres. Approximately 5000 OMs will be used in the IceCube experiment which has 60 OMs on each of the 81 vertical strings. These strings are placed on a regular grid at a distance of 100 m from each other. The distance between adjacent OMs on the same string is 10–15 m.

3.2. Cherenkov Cone Geometry

In an optical medium with refraction index n ultra-relativistic, charged particles emit Cherenkov radiation at a fixed angle α with $n\cos(\alpha) = 1$ relative to the momentum vector of the radiating particle. For water and ice and blue to near UV wavelengths this angle is approximately 42 degrees. The particle travels at the tip of the cone shaped Cherenkov wavefront. The effect is the optical analog of a sonic boom.

Vertical strings of detector modules (like the ones used in IceCube) intercept Cherenkov cones along cone sections, i.e., a wedge or a hyperbola. This intersection creates a characteristic hit time profile along the string: the relative timing of photons registered by OMs is a function of the position of the OM and the direction of the particle track. If the distance between the particle track and the string is called r, the projection of the nearest point onto the string coordinate (here chosen to be z) is named z_0, the angle of the track relative to the

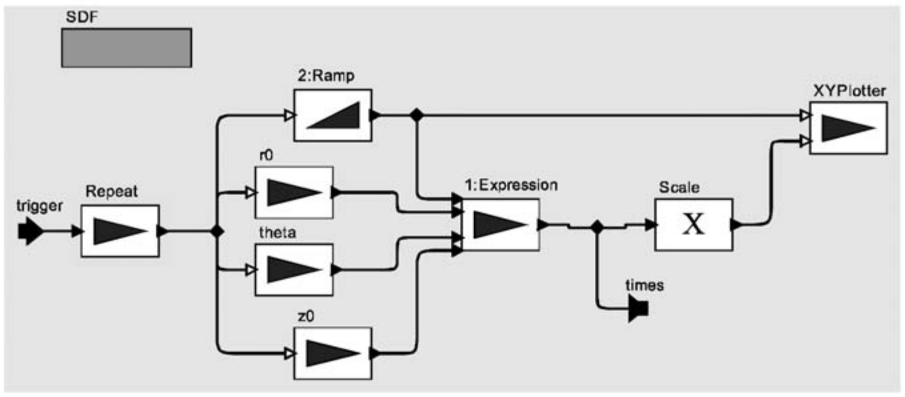

Figure 1. Model to calculate the timing of the Cherenkov cone.

string (the zenith angle) is called θ, and c denotes the speed of light, the equation for the hit time $t(z)$ becomes:

$$ct(z) = (z - z_0)\cos(\theta) + \frac{\sqrt{r_0^2 + (z - z_0)^2 \sin^2(\theta)}}{\tan(\alpha)} \tag{1}$$

While this geometric model is an oversimplification of the physics, it is sufficient for an initial exploration for systems engineering purposes.

A Ptolemy II model calculating the timing is shown in Figure 1. This model creates the coordinates of equidistant modules and the hit times for given particle parameters using formula 1. These times are used by the top level DE *discrete event simulation* model of the system. The Ptolemy II *synchronous data flow* (SDF) Cherenkov model itself executes in zero time. The SDF domain orders operations logically but does not assign an execution time to them. It can be used whenever calculations are needed as part of a timed model which have to be available instantaneously. The calculated times are converted into timed discrete events by the *timedDelay* actors of the system model shown in Figure 9.

Figure 2 shows the simulated hit times of modules in the case of a muon track passing perpendicular to the string in a distance of 100 m, i.e., close to an adjacent string. The timing of a track of a particle passing through a string would have a sharp tip. The further away the particle track is from the string, the more the timing curve takes on a smooth hyperbolic shape. The total event length in this case is approximately 800 ns between the first hits near the center of the string and the last hits at the ends. In an experiment of the size of IceCube, the muon, traveling at the speed of light, traverses the 1.7 km diagonal of the detector in approximately 5 μs. A physical event can therefore last up to 6–7 μs.

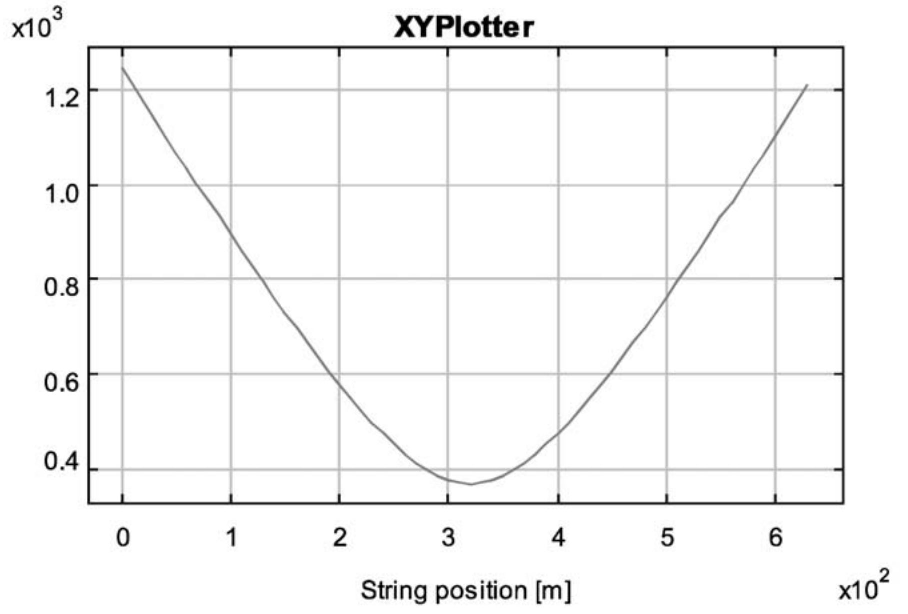

Figure 2. The Cherenkov cone timing [ns] vs. string position for a horizontal track 100 meters from the string.

4. Cherenkov Detector Model

In a Cherenkov detector, very faint light has to be detected with ns time resolution. The photomultiplier tubes used in these detectors are close to being ideal photon counting devices. Despite this fact, the technical performance of the PMTs limits the detector efficiency, which has to be understood in detail to perform the physics analysis of detector data. The timing and stochastic properties of PMTs can be simulated with a combination of DE and CT models.

4.1. Stochastic Properties

Photomultiplier tubes are stochastic amplifiers: a photon creates a single photoelectron at the surface of the photocathode with a probability of approximately 10–25%. This *quantum efficiency* (QE) depends on the photocathode material and the wavelength of the photon. Most PMTs have their peak sensitivity in the blue and near ultraviolet. The photoelectrons are accelerated towards an electron multiplier chain by a strong electrostatic field where they create (a few) multiple secondary electrons on the activated surface of the multiplier electrodes. PMTs with 10–14 secondary electron multiplier stages have an average gain of 10^5 to 10^6. Because only 3–5 secondary electrons are generated by the primary photoelectron, the amplitude of the output anode current pulse fluctuates strongly, even though the pulse shape is relatively constant. The resulting *pulse height distribution*

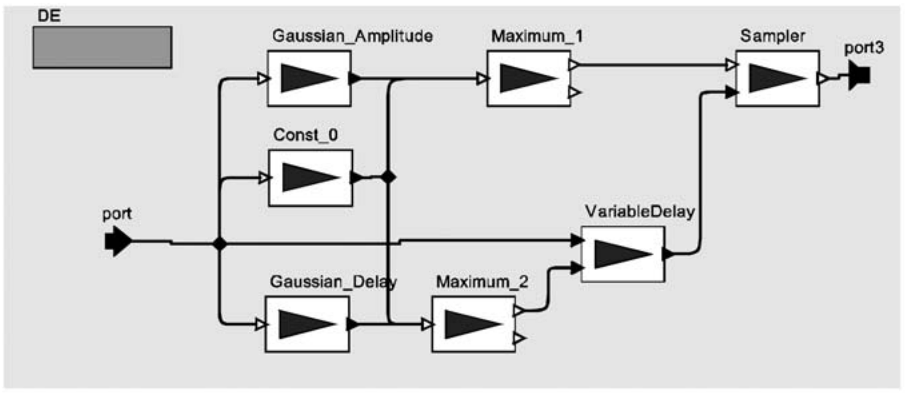

Figure 3. Discrete event model of the stochastic amplification properties of a PMT.

(PHD) is the most important performance characteristic of a PMT. A good PHD is nearly Gaussian with width $\sigma^2_{PHD} \approx 1$.

Individual photoelectrons are also delayed by a random drift time due to the inhomogeneity of the accelerating electrostatic field. The average drift time ($\approx$20 ns) and the time spread ($\approx$2 ns) of the drift time distribution can be modeled with a normal distribution $N(t_{drift}, \sigma^2_{drift})$.

The Ptolemy model shown in Figure 3 uses two Gaussian random sources to create discrete events which approximate the pulse height as well as the drift time statistics. These discrete events trigger the following pulse shaper models which simulate the time dependence of the PMT current pulses.

The model can be refined in multiple ways. A well known problem is so called "flashers," leaking PMTs which generate light when they are triggered. Flashers can be seen by other optical modules and can be modeled by feeding the output of the PMT model back into the trigger event chain.

Figure 4 shows a histogram of a pulse height distribution generated by the model in Figure 3. The model agrees reasonably well with the measured distribution in the data sheet of the photomultiplier R5912 from Hamamatsu [8].

4.2. *Electronic Properties*

The normalized, time dependent output current waveform of a PMT can be approximated with an electrical model of a bandwidth limited Dirac pulse. The electrical waveforms are best described by a *continuous time* (CT) model which can solve the systems of time dependent differential equations describing analog electronic circuits. The PMT waveform generation and the simulation of the analog front-end circuit are very similar and have been integrated into a single CT model, which will be discussed below.

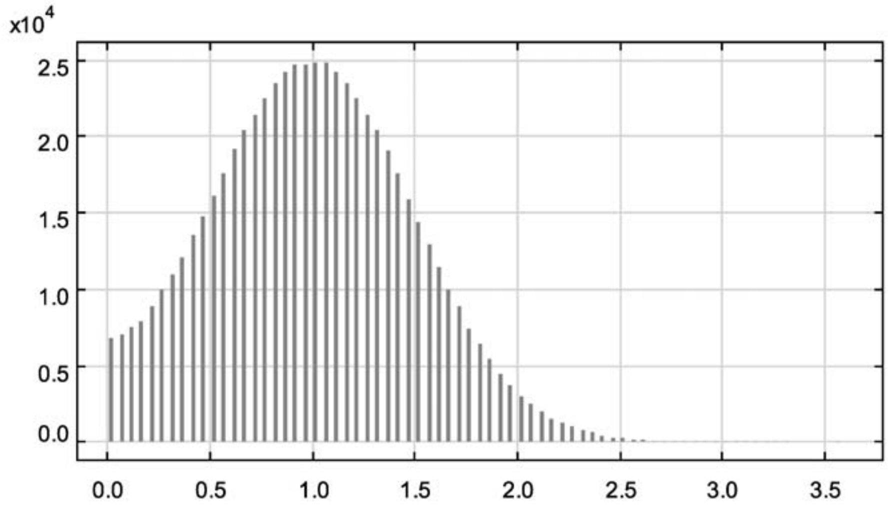

Figure 4. Pulse height distribution calculated by the PMT model in Figure 3 (number of events vs. pulse height).

5. DAQ Front-End Simulation

A typical DAQ front-end consists of a pre-amplifier, a signal shaper to limit the bandwidth of the circuit and improve the signal-to-noise ratio of detector signals, a sample/hold (S/H) circuit, and an analog-to-digital converter (ADC). The specific implementation of these circuits is chosen in such a way that the relevant information carried by the detector signals can be captured with the given precision without violating the size, power and cost constraints of the experiment. In the case of Cherenkov detectors, most information is in the timing of the signal.

5.1. Generating the Sampled Pulse Shape

The PMT waveform and pre-amplifier model in Figure 5 converts a discrete event at its *Trigger* input into a step function using a *Zero-Order-Hold* actor. The step function is then shaped into an exponential decay with an integrator with negative feedback and further filtered with a fourth-order low-pass filter with 5 ns time constants. The resulting waveform corresponds to the pulse response of a well damped pre-amplifier with $\approx$60 MHz bandwidth excited by PMT pulses.

The pre-amplified and shaped waveforms shown in Figure 6 are sampled at 10 ns intervals and the samples are quantized with an ADC resolution of 12 bits, i.e., the output of this model is an integer with a range between 0 and 4095. The model does not account for ADC errors like noise and non-linearity but these could be integrated if necessary.

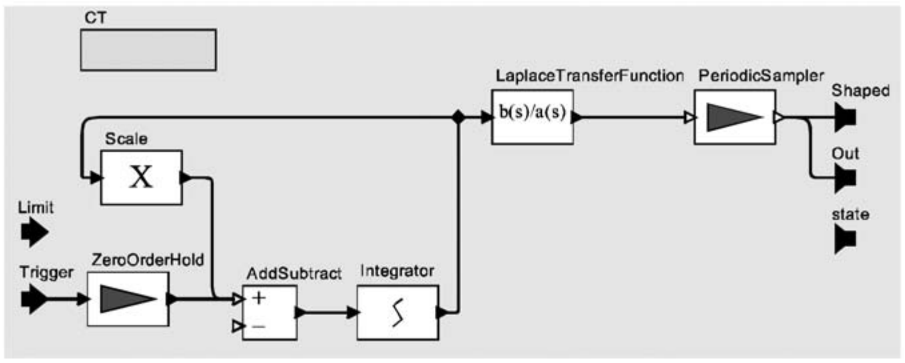

Figure 5. Continuous time model of the PMT pulse shape.

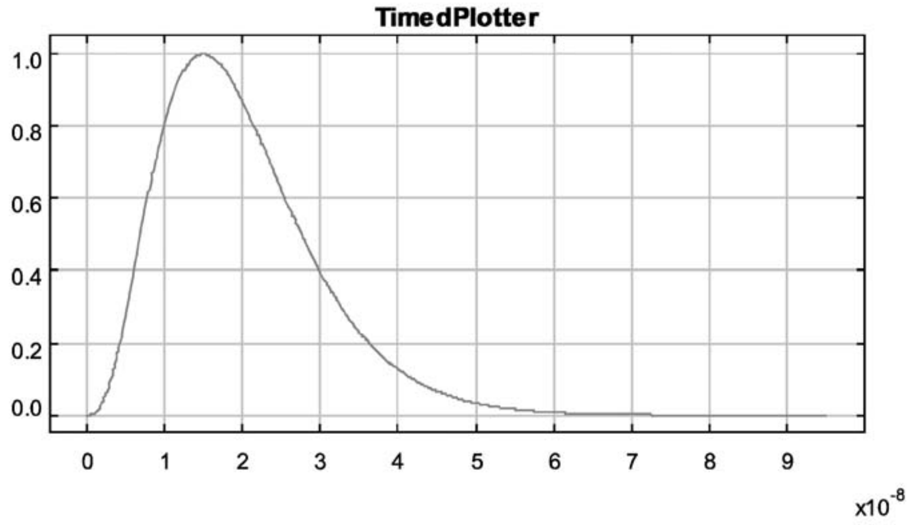

Figure 6. PMT pulse shape generated by the model in Figure 6 (relative current vs. time [ns]).

5.2. *Simulation Efficiency*

An important issue for mixed DE/CT simulations is efficiency. While a DE simulation is only evaluated at times at which at least one state-variable changes due to the model dynamics, CT models are evaluated continuously at time intervals which are set by the differential equation solver of the simulator. A S/H stage model is achieved with a periodic sampling actor, which also requires the simulator to evaluate the model every 10 ns. This is very inefficient since the average hit rate of a PMT is 1 kHz and the PMT pulse is only 100 ns long. A free running CT model would create unnecessary computational overhead, evaluating the model in time increments of a few ns.

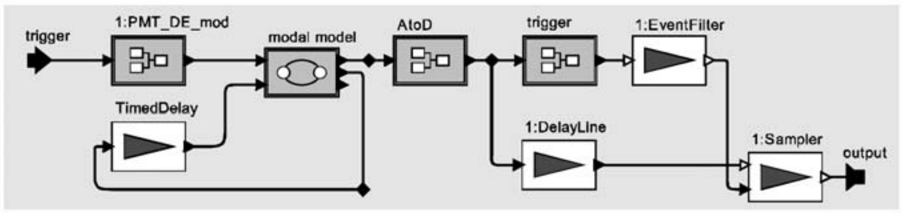

Figure 7. Model of the PMT and DAQ front end.

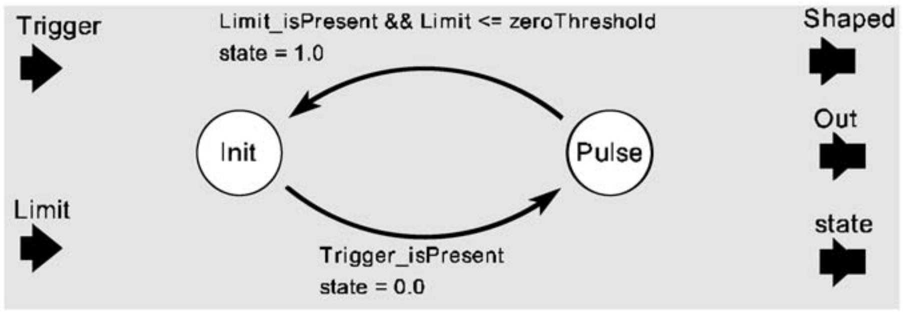

Figure 8. Controller of the PMT and DAQ front end.

By embedding the pulse shaper into a *modal model*, a *FSM* determines when the CT model has to be executed. The controller of the embedded modal model is a state machine with two states, an inactive *Init* state and a *Pulse* state which activates the CT PMT model. The transition from the *Init* to the *Pulse* state is triggered by the presence of a trigger signal, the transition back into the *Init* state occurs as soon as the output of the CT model falls below a given threshold. This allows low overhead simulation without losing any relevant information. The technique is shown in Figures 7 and 8.

The remainder of the front-end model is the ADC and control flow logic which ensures that at least eight samples are generated for the following DAQ back-end models. In this work a down-scaled front-end with eight channels was simulated. The complete model is shown in Figure 9.

5.3. *Timing Extraction*

Timing can be extracted from a PMT pulse by sampling it with high resolution (e.g., 12 bit) at a high rate (e.g., 100 MSPS) and fitting a parameterized analytical expression for the expected waveform to the sampled data. For PMT pulses three free parameters—baseline, pulse height and pulse time—are used and at least three samples are necessary to estimate the parameters. In practice the signal is sampled continuously and the average of many samples is used to predict the baseline. Two or more non-zero samples are thus sufficient to estimate the pulse height and the pulse time. PMT pulses are the result of a stochastic process and show slight variations in shape which limit the precision with which the timing

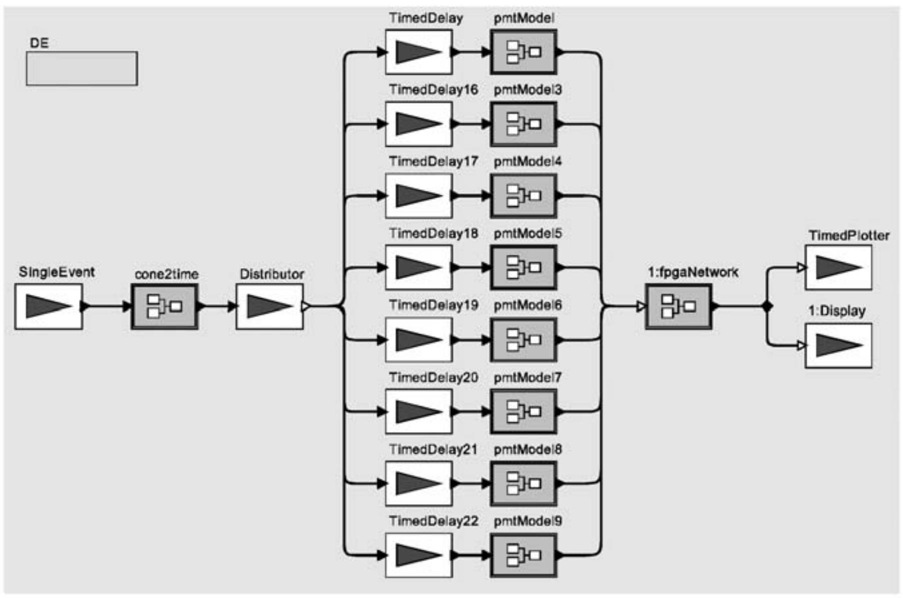

Figure 9. Complete system model.

can be estimated. Theoretical considerations and experience with this technique show that with 3 to 5 non-zero samples the timing can be resolved with an error of approximately 20% of the pulse width. A PMT pulse of 15–20 ns width gives 3 to 5 non-zero samples at 100 MSPS, which is a good choice for the sampling rate. The statistical error for the time parameter fit is then 20% of the sampling period or about 2 ns.

6. DAQ Back-End Simulation

High-speed data acquisition systems are platforms for acquiring, processing, and moving data around. These platforms have limited bandwidth, latency, memory size, logic density, etc. They may use different digital signal processing algorithms, data rates, data formats, and network topologies.

The problems to be solved by DAQ system designers are: *(i)* how to get the data; *(ii)* how to store it; *(iii)* how to move it efficiently to a destination for processing; *(iv)* how to process the data; *(v)* how to test the correctness of the result; and *(vi)* how to ensure that the quality of the process is within given constraints.

A reconfigurable platform is an economic way to solve these problems because it can be adapted to a new application without the need to redesign physical hardware. One of the problems of interest is to create a reusable hardware design strategy for a variety of applications that operates efficiently within the platform specific bandwidth, latency, and resource limitations.

In this section we describe the architecture of a reconfigurable hardware platform for DAQ applications and an efficient programming model in the Ptolemy II framework. Using the high level of abstraction provided in Ptolemy II, user-level programming can include all components of a heterogeneous architecture that entails fabrics of FPGAs, CPUs, memory, network links, algorithms, firmware and software while hiding low level details. Without knowing how to program reconfigurable platforms users can easily modify parts of the architecture.

In Section 7, we show how an application model is mapped to the platform model. After the application is mapped to the hardware platform, users receive feedback on how they are utilizing the hardware. Based on this feedback, the system designer can make changes to the platform, which allows him to create different variants of the design and to compare and optimize performance, functionality, cost etc.

6.1. The Back-End Network Architecture

The communication of data and control messages between the DAQ front-end hardware—analog electronics and ADC—and the back-end hardware—digital signal processing and storage (on a CPU farm)—is accomplished with a static network of transmitting and receiving nodes.

For the application in question, the platform has up to 64 input channels with 12 bit ADC running at 100 MHz and with latency of 100 clock cycles.

The digitized signals from eight analog channels are fed into one FPGA[1] which writes them into 27 Mbit DDR SRAM waveform memories. Digital signal processing algorithms inside the FPGA are used to extract trigger information such as event energy and timing. Processing of the $8 * 12$ bit $* 100$ MHz $= 1.2$ GByte/s data stream is continuous and dead-time-free. Waveform data in the memory is associated with a 64 bit time stamp, a 32 bit integer counting seconds of universal time and a 32 bit integer resolving fractions of a second with 0.24 ns resolution. A network of eight FPGAs acquires the input from the 64 channels. In this architecture, we connect the eight FPGAs in a ring topology which is favorable due to the simplicity of the board design and the nearly optimal electrical line length, which helps minimizing noise problems in the analog section. Figure 10 illustrates the network of FPGAs.

The DAQ hardware is controlled exclusively by messages which are transmitted using a custom datagram protocol. This protocol is specifically optimized for the FPGA architecture so that network nodes and routing circuits can be implemented in a small fraction of available FPGA logic.

A network node is either a transmitter or a receiver circuit. A transmitter node contains a control state machine, a source and a destination address generator, a data shift register and an input register. A receiver node contains a slightly different state machine, an address discriminator, a shift register, and an output register. Packets are routed between transmitter and receiver nodes by router and combiner circuits. Figure 11 illustrates the network structures of an FPGA.

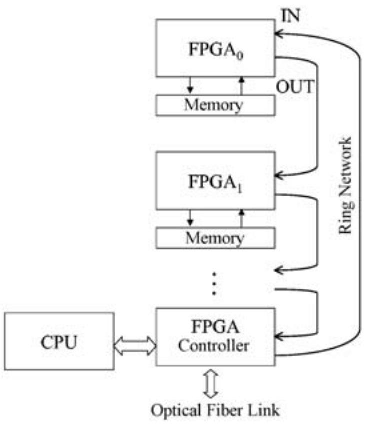

Figure 10. Network of FPGAs.

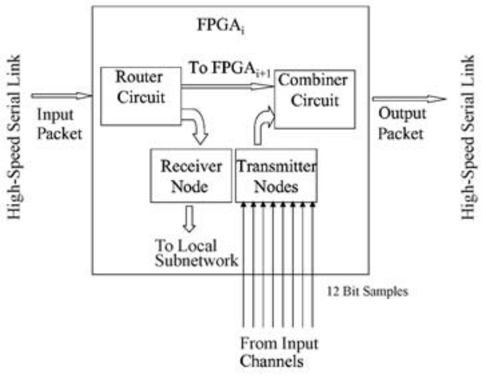

Figure 11. Network structures within a FPGA.

NETWORK TRANSMITTER NODE

The transmitter circuit is a synchronous design driven by a clock signal *CLK* and can be triggered by a transmit enable signal *TxEnable*. After being triggered, the transmitter node starts transmitting blocks of data available on its inputs to the selected destination, formatting the data into packets. Flow control codes are used to indicate packet start, stop and idle line states. Transmitters have a fixed number of clock delays between the time transmit is activated and the time data appears at the output.

The physical interface of the transmitter node has as inputs:

$CLK =$ clock signal
$DestAddr[(AddressWidth - 1)..0]$
$SrcAddr[(AddressWidth - 1)..0]$
$IN[(n - 1)..0]$
$=$ a bit string of fixed length n
$TxEn$ signal
$=$ transmit enable signal

and as outputs:

$FCF = 1$ bit flow control flag
$DATA[6..0] = 7$ bit data
$TxRdy =$ transmit ready signal.

The transmitter state machine needs one clock cycle between assertion of $TxEn$ and the first output byte. If the flow control flag is inactive $FCF = 1$), $DATA[6..0]$ is interpreted as 7 bit wide sections of the input bit string $IN[(n - 1)..0]$.

A sequence of 8 bit wide output words forms a packet which is submitted on one of the network sub-nets. The packet structure is the following:

$IDLE = 0$ or
$PACKET\_STOP = 64$
$PACKET\_START = 1$
$DestAddr[(AddressWidth - 1)..0]$
$SrcAddr[(AddressWidth - 1)..0]$
$IN[6..0]..IN[7 * nwords - 1..7 * (nwords - 1)]$
$nwords = ceil(n/7)$
$PACKET\_STOP$
$IDLE$ or $PACKET\_START$

The IN portion of a typical packet will carry either time stamped signal energy information or sampled data. Packet error is not modeled in this paper. Future refinements of our models will include overflow detection. When a packet is partially lost due to FIFO overflow, a $PACKET\_ERROR$ flag can be appended to the partially lost packet.

Network Receiver Node

Receiver nodes listen to packets on their sub-net and store message data in a parallel output register when they identify their own address in the $DestAddr$ field of a packet header. In a DAQ application most messages are short (8–16 bytes) and most nodes are specialized to transmit or receive a single, fixed format message type.

The receiver node has as inputs:

$$\boxed{\begin{array}{l} CLK = \text{clock signal} \\ RecAddr[(AddressWidth - 1)..0] \\ FCF = 1 \text{ bit flow control} \\ DATA[6..0] = 7 \text{ bit data} \end{array}}$$

and as outputs:

$$\boxed{\begin{array}{l} SrcAddr[(AddressWidth - 1)..0] \\ OUT[(n - 1)..0] \\ RxRdy = Rx \text{ ready signal} \end{array}}$$

One clock cycle after $PACKET\_STOP$ is received, the receiver asserts $RecRdy$.

Network Router and Combiner Structures

The datagram network uses independent data paths for reception and transmission, is collision free and loss-less. A simple high performance packet routing strategy with static routing tables is used to transport messages between nodes of the hardware network. *Router circuits* split nets into sub-nets. A router has one input port, two output ports, and two constants for network address, $DestAddr$ and sub-net mask, $NetMask$. It reads every packet destination address and reroutes the packet to the internal network if the packet's destination address matches $NetAddr$—the address of the local sub-net, i.e., if $DestAddr\&NetMask == NetAddr$. Otherwise, the packet is routed to the next node in the ring.

In the current implementation, a router requires 7 clock cycles to route data from its input to one of its outputs, and never looses or drops data. If a packet is wrongly addressed, it is dropped. Applications which require control over correct packet reception may modify the token passing implementation to include a "return to sender" mechanism.

Data from multiple sub-nets are merged by *combiner circuits* into one output stream. Since the transmitter architecture is non-blocking, a combiner uses FIFOs on its inputs to buffer incoming data until the output line becomes available. The arbitration controller state machine of the combiner circuit always selects the input FIFO with the most data as the next source to optimize FIFO efficiency. If both input buffers have the same number of words in them, the controller state machine toggles in a ping-pong scheme. The combiner circuit is lossy. Due to the stochastic nature of the data sources, the network begins to drop packets when the input rate exceeds $\delta * R$, where R is the maximum data rate and δ is a constant <1. One of the goals of modeling and simulating this network is to find safe limits for the network load, depending on the chosen sub-net topology and the stochastic properties of the data. Figure 12 illustrates the structure of the combiner circuit.

The minimum delay of a combiner is one clock cycle, and the maximum delay depends on the number of words in the FIFO buffers. The worst case delay is unbounded if a short packet is waiting to be transmitted in one input FIFO while the other input FIFO is filled

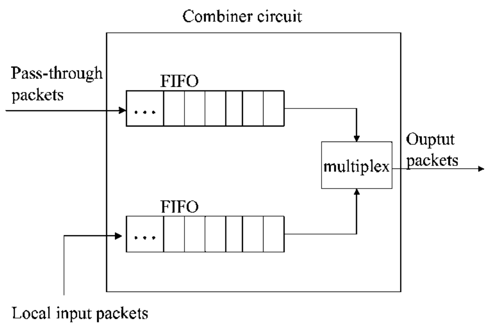

Figure 12. Combiner circuit.

constantly with new incoming packets at the same or a higher rate. As part of future work, we plan to use other combiner schemes to trade optimal FIFO utilization against guaranteed latency or guaranteed throughput.

6.2. The Ptolemy II Back-End Model

In this section we model the data flow across the network to collect statistics on the queue lengths used in combiner nodes. Modeling the receiver and router nodes is only necessary when modeling the control flow across the network. In this paper we do not study the control flow of the system.

Transmitter Model

The transmitter model is a hierarchical model that combines the Synchronous Data Flow (SDF) [11] and the Finite State Machine (FSM) [9], [10] models of computation. The high-level model is shown in Figure 13.

The top-level model, executing in an SDF domain, takes as inputs event data and a global clock, which in turn are inputs to a FSM.

The transmit controller has a single state with transitions depending on whether event data, a clock signal (trigger), or both inputs exist at the time(s) when execution of the transmit node is scheduled. This controller makes sure that data is only taken into the transmitter when a valid clock signal is present. The controller generates as outputs a data variable and a Boolean variable to indicate whether or not the data is valid. The transmit controller model is shown in Figure 14.

Each controller output drives a *SamplerWithDefault* actor that generates the most recent input token when its trigger port receives a token. The output of the samplers and the (constant) source/destination network addresses are fed into the SDF transmitter model shown in Figure 15. If no token has been received on the input port when a token is received on the trigger port, the initial value of the sampler is produced.

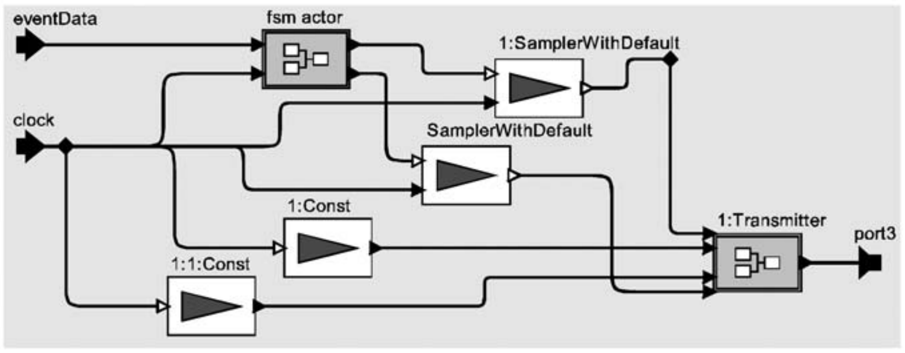

Figure 13. Model of the transmitter node.

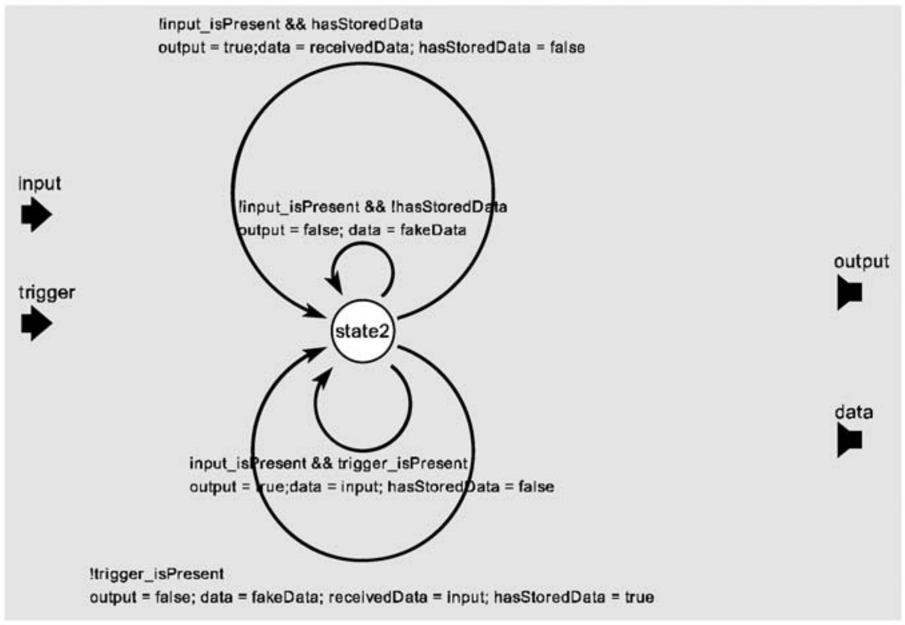
!input_isPresent && hasStoredData
output = true;data = receivedData; hasStoredData = false
input
!input_isPresent && !hasStoredData
output = false; data = fakeData
output
trigger
state2
data
input_isPresent && trigger_isPresent
output = true;data = input; hasStoredData = false
!trigger_isPresent
output = false; data = fakeData; receivedData = input; hasStoredData = true

Figure 14. Model of the transmit controller.

The packet data is first combined with the source address, destination address, *PACKET_START*, and *PACKET_END* flags to form a complete packet. This task is performed by the SDF model named *aggregate* in Figure 15. The transmission policy, represented in Figure 15 by the *transmit policy* actor, is a SDF model that takes as input the queue length of the last cycle of the transmitter. If the queue length is zero and if *transmit-Enable* input is true, the policy generates a true output. The policy output is used by the *BooleanMultiplexor* actor to select either the assembled packet or a sequence of *IDLE* flags for transmission. The data packet or the sequence of *IDLE* flags is stored in a synchronous queue.

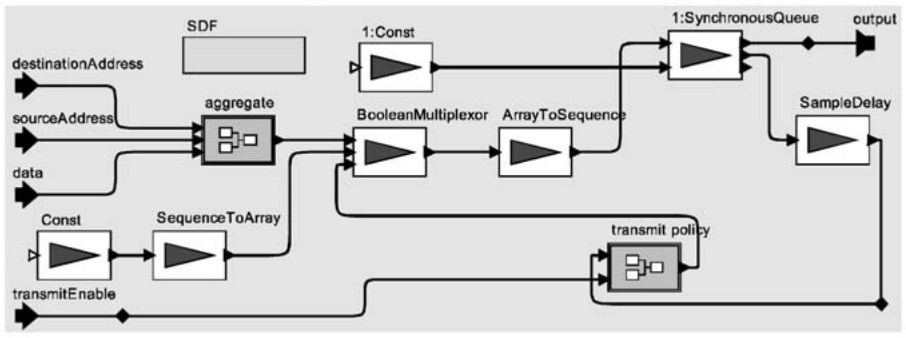

Figure 15. Model of the transmitter actor of Figure 13.

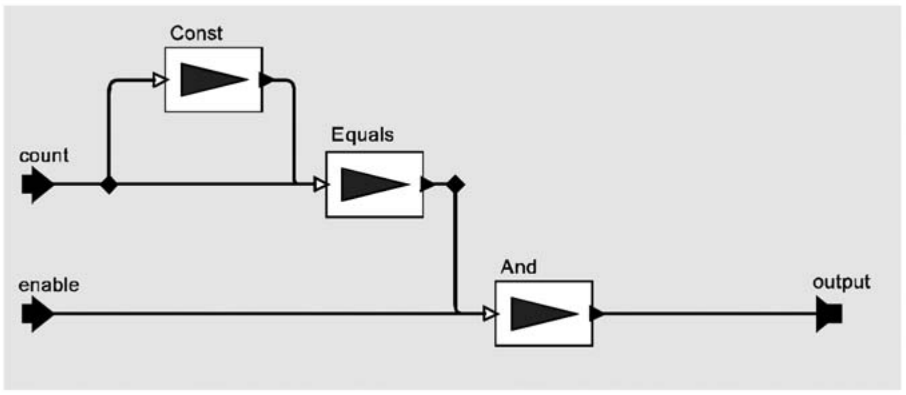

Figure 16. Transmit policy used by the transmitter node.

The Figure 16 depicts the Ptolemy II model of the transmit policy used by the transmitter node.

COMBINER MODEL

The combiner model is also a hierarchical model using the SDF and FSM models of computation. Figure 17 depicts the top level model of a combiner node. The top level model is a SDF model that takes as inputs data stored in two queues, modeled by *Synchronous-Queue* actors. For each cycle of the model, the output of one queue is selected by a FSM controller. *SampleDelay* actors are used to break dependency cycles in directed loops of SDF models. This actor declares an initial production parameter in its output port that is used by the SDF scheduler to properly schedule the model. The initial outputs permit the computation to get started. The *SynchronousQueue* actor generates a queue length output and allows the controller to look ahead to the next queued value. The combiner controller, shown in Figure 18, looks at the queue length of both queues and decides to empty the one that has the largest number of packets. It starts in the *packet start* state, staying there while both queues are empty. In this state, the queue selection is not important, thus the

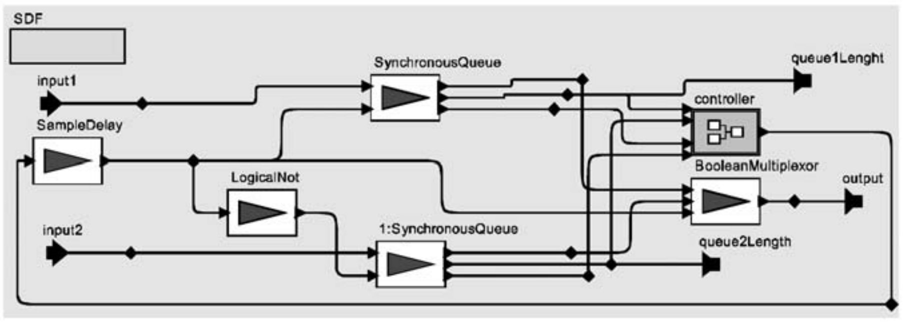

Figure 17. Model of the combiner node.

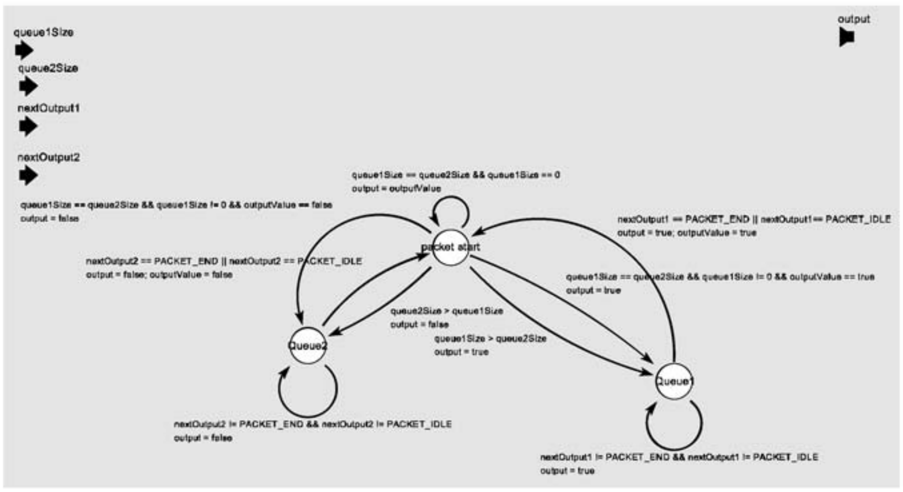
queue1Size
queue2Size
nextOutput1
nextOutput2
output
queue1Size == queue2Size && queue1Size == 0
output = outputValue
queue1Size == queue2Size && queue1Size != 0 && outputValue == false
output = false
nextOutput1 == PACKET_END || nextOutput1== PACKET_IDLE
output = true; outputValue = true
packet start
nextOutput2 == PACKET_END || nextOutput2 == PACKET_IDLE
output = false; outputValue = false
queue1Size == queue2Size && queue1Size != 0 && outputValue == true
output = true
queue2Size > queue1Size
output = false
queue1Size > queue2Size
output = true
Queue2
Queue1
nextOutput2 != PACKET_END && nextOutput2 != PACKET_IDLE
output = false
nextOutput1 != PACKET_END && nextOutput1 != PACKET_IDLE
output = true

Figure 18. Model of the controller for the combiner node.

controller simply outputs a true value or the same Boolean value used before transitioning back into this state. If the length of the second queue is greater than the length of the first queue or if they have the same length, are not empty, and the second queue was last selected, the controller moves to the *Queue2* state, selecting the second queue for output. The controller moves to the *Queue1* state if the same conditions are true for the first queue, selecting the first queue for output. While in state *Queue1* or *Queue2*, it selects the corresponding queue until a *PACKET_END* or a *PACKET_IDLE* tag is seen, at which time the controller goes back to the *packet start* state, indicating what the last output selection was.

Figure 18 depicts the Ptolemy II model of the controller for the combiner node. The model of the network of FPGAs is formed by connecting the output ports of one FPGA model to the input port of the next FPGA in the ring. Figure 19 depicts the Ptolemy II model of this network of FPGAs.

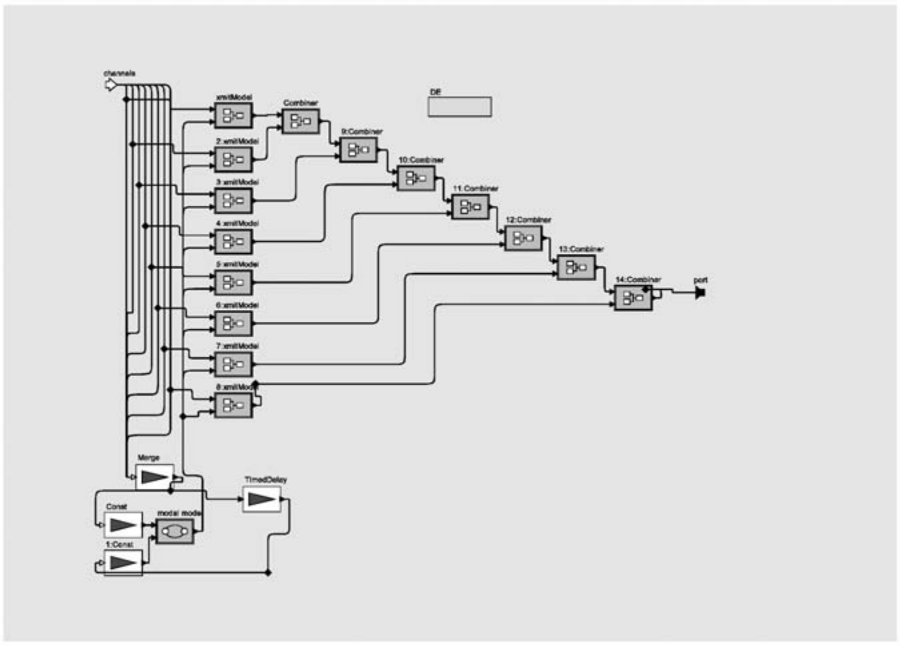

Figure 19. Model of a network of FPGAs

6.3. *Simulation Results*

The performance of the system is closely related to the performance of the configurable hardware platform. The two implementations of the system under design at Lawrence Berkeley National Laboratory are an eight channel and a 64 channel DAQ board.

The eight channel board uses Altera Mercury FPGAs running at a clock frequency of 100 MHz. This board can achieve 100 MByte/s on any part of the network internal to one FPGA, and implements four sub-networks in each of the two chips used for data acquisition. This results in an aggregate data rate of 400 MByte/s per chip.

The external ring bus topology uses the SerDes circuits and LVDS drivers of the Mercury family to achieve 1 GByte/s. With two FPGAs collecting data from 8 channels this is a very well balanced network design. The 64 channel board will use Altera Apex II chips and improve upon the performance of the 8 channel system, although the ratio between source data rate and network bandwidth has to be smaller.

On the eight channel board a packet with 24 words has a duration of 240 ns and passes a switch in 70 ns. The minimum round-trip time of a packet in the proposed network structure is $\approx 12 \times 70$ ns $= 840$ ns. The combiner FIFOs have a depth of 256 words and a packet can be held up by an average of 200 clock cycles per combiner when the network is running near its limits without dropping packets. In this case the round-trip time on the ring-bus will still be ≤ 48 μs. The main limitation of the non-blocking protocol is the finite FIFO depth resulting in occasional packet loss.

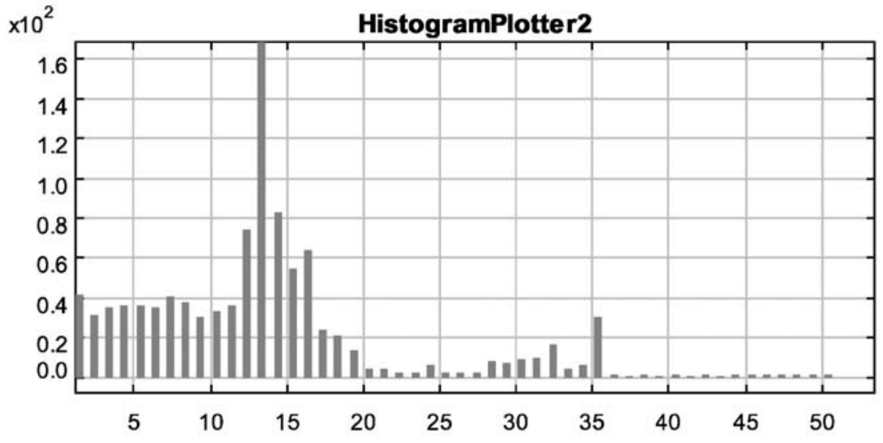

Figure 20. Histogram of queue2 length

In a high energy physics DAQ environment, most of the events are simply noise and contain no interesting physics information. Thus, packet losses are tolerable if they are limited to a small fraction (e.g., 10^{-3}) of the total data flow. However, the system design has to make sure that packet losses do not occur frequently under "normal" circumstances, i.e., if the quality of data generated by the detector is as good as or better than specified by the design requirements. The simulation model can help to address these questions by creating histograms of the FIFO congestion. Figure 20 shows the distribution of the number of bytes in the FIFO of the last and therefore most congested combiner circuit. The highest observed congestion is well below the physical FIFO depth of 256 bytes, therefore no packets were lost in this simulation.

7. Mapping the Simulation Model to Hardware

In the previous sections the modeling and general structure of the system design were discussed. The important result was that a complete and sufficiently realistic system model could be expressed within the Ptolemy II framework. It remains to be shown how this simulation approach lends itself in a natural way to design automation.

7.1. Analog Front End Design Automation

Generally analog circuits, especially high-speed, low-noise, mixed signal board-level designs are hard to generate automatically. However, the design space of waveform-recording DAQ applications is relatively limited and design space exploration is possible. A high energy DAQ front-end can be divided into two independent parts: the input amplifier and signal shaper and the digitizer. The main parameters of a digitizer are sampling frequency, effective ADC resolution and precision. In the case of high speed digitizers, currently available analog chips limit the designer to combinations of these parameters which are below the current technological limits of approximately {18 bit, 1 MSPS}, {16 bit, 10 MSPS}, {12 bit, 100 MSPS} and {8 bit, 1 GSPS}. While this limit is moving constantly, doubling the maximum sampling rate at a given precision every couple of years, the general shape of the design space boundary is given by the physical noise, stability and speed limits of converter circuits based on silicon technology and remains relatively constant. Given the few possible choices, the decision for a certain speed/resolution tradeoff is usually straight forward. Input amplifier noise and the transfer function of the shaper amplifier have the biggest impact on physics performance. Both have to be optimized carefully. The required noise density and transfer functions depend on the specific detector type and have to be adjusted for any single application.

If a sufficiently well defined detector model exists, the choice of the ideal filter coefficients, expressed as Laplace actor parameters in Ptolemy II, is a well understood linear optimization problem which can be automated. Deriving a Spice or APLAC model [1] from these parameters and choosing one or several integrated amplifier chips from the small set of available high-speed operational amplifiers can also be automated. By using the optimization routines of a circuit simulator like APLAC, the required transfer functions can be approximated with off-the-shelf components by tuning component values. Research to this effect is currently done by one of the authors and is only slightly different from similar attempts to automate analog chip design.

7.2. Digital Logic Code Generation

In case of the digital back-end design, the previously discussed Ptolemy II simulation models are ideally suited for code generation and design automation of the digital back-end design. In general, models using the *SDF* and *FSM* domains can be synthesized if the semantics is suitably extended to represent that of a digital system with a single clock domain. SDF actors performing integer and logic operations can be trivially mapped onto digital circuits. A similar relationship exists between a Ptolemy II FSM model and a digital implementation of a state machine. Apart from naming conventions, a Ptolemy II state machine can be mapped directly onto a VHDL description.

Most *DE* models do not correspond in a trivial way to hardware elements (even if asynchronous digital circuits were to be used, the restrictions are very limiting). However, their synthesis is usually not required because they either describe physical models, i.e., parts of the system design which are not synthesized into hardware at all, or they are used to increase the performance of embedded CT and SDF models by triggering them only when required. Ptolemy II cleanly assigns a single domain to each (sub-)model. Therefore the code generation facility can trivially distinguish between sub-models which need to be synthesized and those which constitute the model of the physical environment of the actual DAQ system.

7.3. Network Synthesis

The most relevant difference between a "classical" DAQ system and the presented design is the exclusive use of datagram packets on a custom network to represent and transport non-local information. This networked approach maps abstract data structures, i.e., the objects that system designers and users care about, unambiguously to an implementation on the hardware level. The presented implementation is simple, correct by design, reasonably efficient in terms of bandwidth and resource usage and can be code generated easily. Network transmitters and receivers are essentially shift registers. Packet router and combiner circuits use minimal FPGA resources in terms of logic cells and memory blocks.

We developed a network code generator program that maps a list of packet data words to ports and registers of parameterized VHDL models of the transmitter and receiver blocks for synthesis of the FPGA designs. Since the packets have to be processed by an instance of a software algorithm running on a trigger and event builder CPU farm, the code generator also creates Java and C++ classes with methods to access elements of the packet data structure. While specialized actors correspond to the network nodes, Ptolemy II relations are equivalent to a physical net-list of interconnections and can be translated to VHDL mapping files which connect multiple network circuits and other code generated function blocks to a complete DAQ system design. The Ptolemy II simulator is dynamic and interactive. A direct coupling of code generated hardware and a Ptolemy II model by means of a hardware interface (serial port, parallel port, 10/100 BaseT etc.) is possible. Thus Ptolemy II can also be used to test and debug a code generated design on a programmable platform.

Notes

1. Recently available FPGA products have transmitter/receiver circuits that can carry up to 1.2 GByte/s per link.

References

1. ANTARES. A Deep Sea Telescope for High Energy Neutrinos, Technical Proposal 99–01, The ANTARES Collaboration, DAPNIA, 1999.
2. APLAC. APLAC 7.60 Reference Manual Vol. 1 Programming, Analysis, and Optimization. Reference Manual, APLAC Solutions Corporation, 2000.
3. Davis, J., C. Hylands, B. Kienhuis, E. A. Lee, J. Liu, X. Liu, L. Muliadi, S. Neuendorffer, J. Tsay, B. Vogel, and Y. Xiong. Heterogeneous Concurrent Modeling and Design in Java. Technical Memorandum UCB/ERL M01/12, Electronics Research Lab, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley California, USA, 2001.
4. Eker, J., J. W. Janneck, E. A. Lee, J. Liu, X. Liu, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming Heterogeneity—The Ptolemy Approach. *IEEE Proceedings, Special Issue on Modeling and Design of Embedded Software*, scheduled for publication June 2002.
5. Hoare, C. A Theory of CSP. *Communications of the ACM*, vol. 21, no. 8, 1978.

6. IceCube. IceCube: A Kilometer-Scale Neutrino Observatory. Proposal to the National Science Foundation, The IceCube Collaboration, LBNL, 1999.
7. Kahn, G., and D. B. MacQueen. Coroutines and Networks of Parallel Processes. In *Proceedings of the IFIP Congress 77*. Paris, France, North-Holland Publishing Company, 1977, pp. 993–998.
8. K. K., H. P. Photomultiplier Tube R5912. Data sheet, HAMAMATSU PHOTONICS K. K., 1998.
9. Lee, B. and E. A. Lee. Interaction of Finite State Machines with Concurrency Models. In *Proc. of Thirty Second Annual Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, Nov. 1998.
10. Lee, E., J. Liu, X. Liu, J. McCarthy, S. Neuendorffer, S. Sachs, and Y. Xiong. *Designing Systems with Ptolemy II*, unpublished manuscript, Feb. 2002.
11. Lee, E., and D. Messerschmitt. Synchronous Data Flow. *Proceedings of the IEEE*, pp. 55–64, 1987.
12. Liu, J. Continuous Time and Mixed-Signal Simulation in Ptolemy II. Memo M98/74, UCB/ERL, EECS UC Berkeley, CA 94720, 1998.
13. Monteleoni, B. NESTOR A Deep Sea Physics Laboratory for the Mediterranean. In *Proceeding of the 17th International Conference on Neutrino Physics and Astrophysics (NEUTRINO'96)*. Helsinki, Finland, 1996.