Hardware/Software Co-design of Schedulers for Real Time Systems

Jorge Ortiz

Committee
David Andrews, Chair
Douglas Niehaus
Perry Alexander

Presentation Outline

Background
- Prior work in hybrid co-design
- FPGA hardware/software co-design
- Scheduling

RTFPGA Project Overview
- Design approach & functionality
- Design implementation plan

Scheduler Implementation
- Hybrid co-design for KURT-Linux event scheduler
- System properties

Testing and Analysis
- Functionality validation of design
- Data Stream testing
- Conclusions and future work

Contributions

- Design
  Mitchell Trope, Sweatha Rao
- Implementation
  Sweatha Rao, Jorge Ortiz
- Testing
  Mitchell Trope, Sweatha Rao, Jorge Ortiz

Background

Prior Work in Hybrid Co-design

- Average-case enhancements
- Worst-case scenarios
- Platform architectures for real time systems
- Systems that support real-time constraints

System Co-Design

Parallel systems

- Multi-processor, FPGA, System-on-Chip, ASIC, Reconfigurable systems

Hybrid systems

- Exploit recursive patterns
- Increase quality of service
- Meet real-time constraints

Real time operating systems

- Hardware support for the operating system
- OS function migration

FPGA Hardware/Software Co-design

Desired system properties

- Timeliness, concurrency, liveness, interfaces, heterogeneity, reliability, reactivity, predictability and safety
- Average-case, worst case scenarios

System flexibility and performance

- Reconfigurable
- No loss in performance

Scheduling

- *How* to allocate *which* resources *to whom* and for *how long*
- Handling real-time events
- Desire for finer granularity for servicing interrupts
- Overhead costs due to interrupt handling and event scheduling
- Design trade-offs

RTFPGA Project Overview

Project Motivation

- Provide higher resolution and finer granularity for event handling
- Minimize overhead processing
- Migrating key functions into hardware
- Time keeping hardware support
- Hardware/software co-design of event scheduler for real-time hybrid operating system

Design Approach

- Migrate shadow timer into FPGA
- Implement scheduler bookkeeping operations
- Provide processing for enqueuing and dequeuing events
- Allow entering, sorting and deletion of events
- Enable access to all queued events
- Move scheduling algorithm into the hardware

Design Functionality

- FPGA stores event information in local queue
- Run scheduling algorithm
- Identify the next event to run
- Sends event time to a match register
- Free-counting hardware clock provides interrupt generation

Design Implementation Plan

- Interface ported KURT-Linux to FPGA components
- Create FPGA-mapped registers
- Create basic event queue storage, interface and functionality
- Implement event scheduler
- Deliver FIQ interrupts to CPU
- Solve concurrency issues of hardware-based implementation

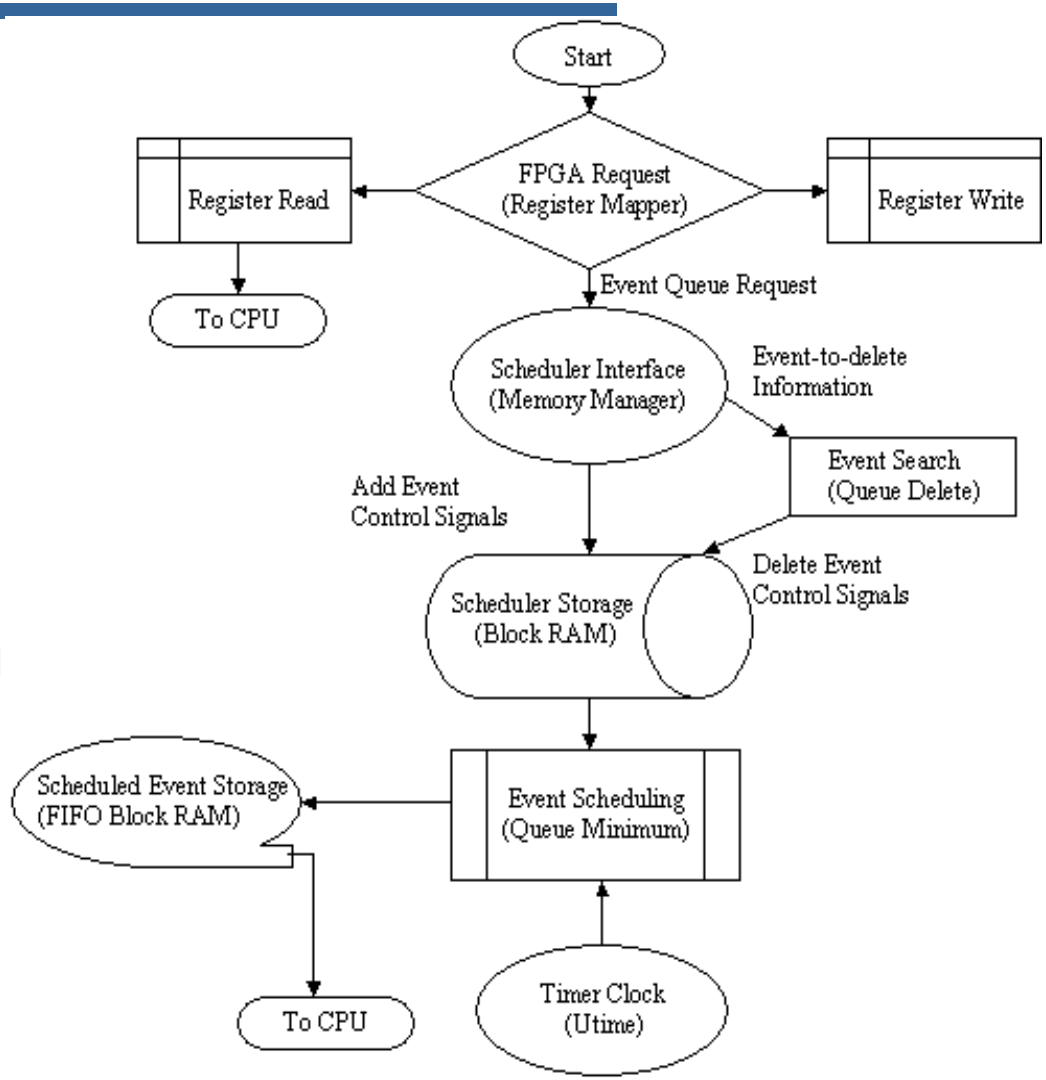
Scheduler Implementation

Hybrid Co-design for Kurt-Linux Event Scheduler

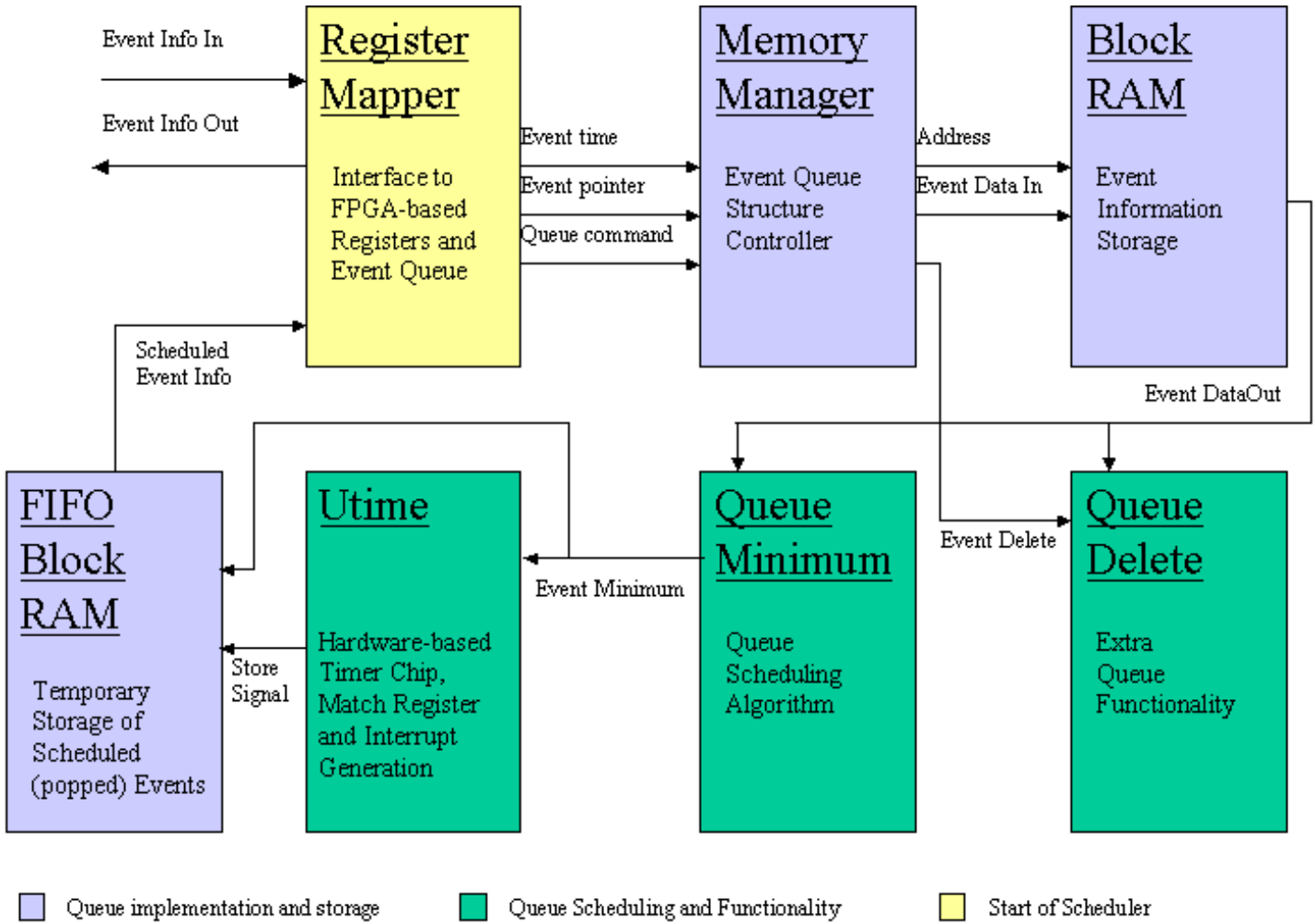
- Forward event requests from OS to hardware through command register
- Identify request type and event information:execution time and reference pointer
- Service request on event scheduler
- Raise interrupt flag upon execution time
- KURT-Linux runs appropriate ISR with retrieved event information

Data Flow Chart in the RTFPGA Scheduler

- Request to FPGA
- Send request to scheduler
- Identify request
- Add or delete in Block RAM storage
- Schedule events in parallel
- Queue expired events for CPU to read



RTFPGA Scheduler Modules



RTFPGA Scheduler Modules

Register Mapper

- Interface to all registers
- Interface to internal functionality in the FPGA board
- Modular and portable
- Two operational blocks - read and write
- Timing, event information, scheduler commands, debugging information.

RTFPGA Scheduler Modules

Memory Manager

- Control storage and handling of Block RAM-implemented event queue
- Block RAM Address generation for event addition
- Service Scheduler requests
- Dirty bits indicate usage of Block RAM addresses

RTFPGA Scheduler Modules

Block RAM

- Dual-read/write port synchronous Block RAM FPGA storage memory
- First port for Memory Manager requests
- Second port for continuously polling information in BRAM addresses for scheduling and searching purposes

RTFPGA Scheduler Modules

Utime

- Shadow of KURT Utime clock implementation
- Set for microsecond resolution
- Provide match register for next scheduled event time
- Create FIQ interrupts to CPU

RTFPGA Scheduler Modules

Queue Minimum

- Earliest Deadline First algorithm
- Reads event values output by Block RAM
- Output information for next event to be scheduled
- Send appropriate control signals

EDF pseudo-code

```
START
SET next_event = max value
LOOP
IF last deleted value = next_event
   GOTO START
IF last expired event = next_event
   GOTO START
IF no elements in queue THEN GOTO START
ELSE
   READ new data from BRam event queue
   IF data = event (dirty bit is set)
      READ scheduled time
      IF scheduled time < next_event
         next_event = scheduled time
      END IF
   END IF
END IF
GOTO LOOP
```

RTFPGA Scheduler Modules

Queue Delete

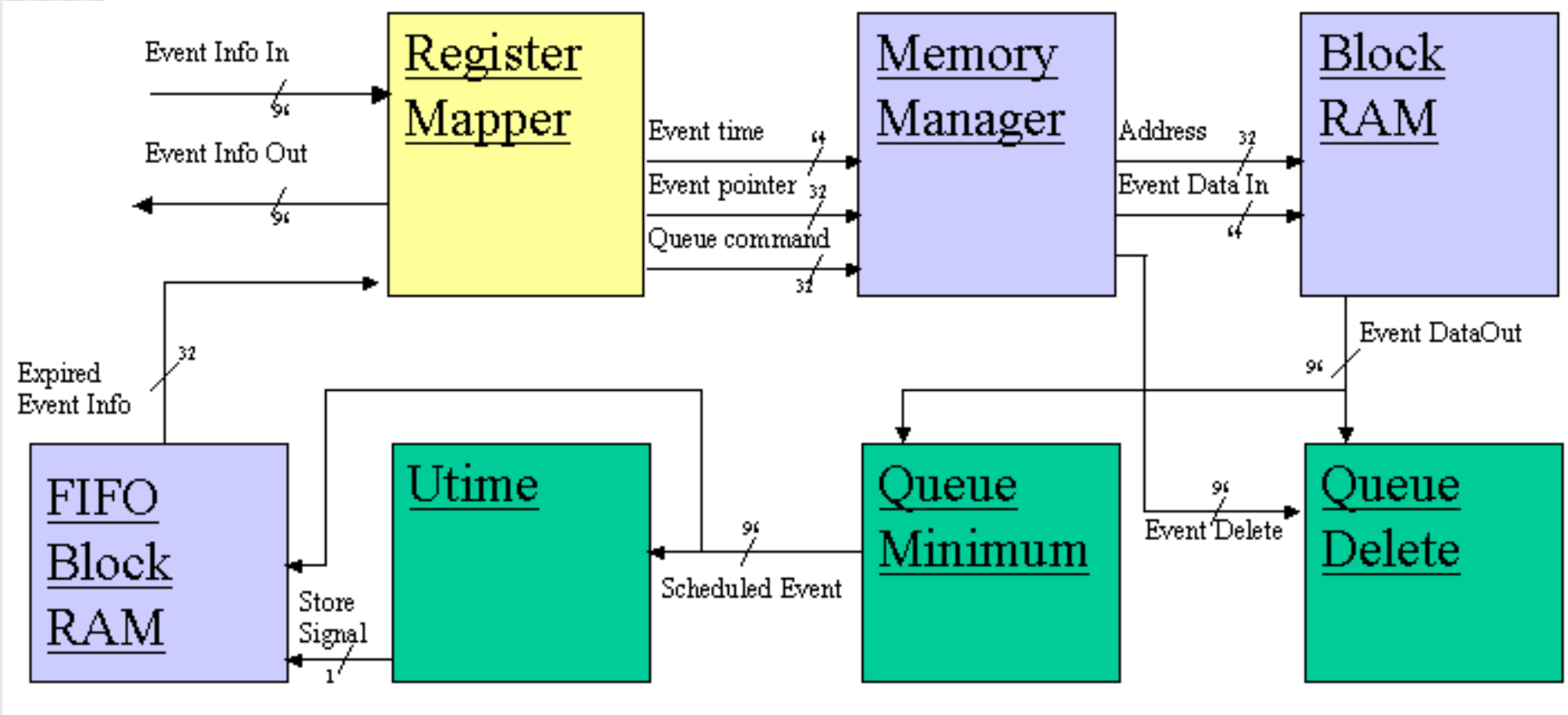
- Continuously receive polled information from Block RAM
- Perform a deletion using linear search & comparison
- Found or lost signal sent back to Memory Manager
- Check for concurrency with interrupt generation and event queue popping

RTFPGA Scheduler Modules

FIFO Block RAM

- FIFO queue for expired events that have not been read yet by CPU
- Keep track of number of expired events
- De-queue until empty

Inter-Module Interfaces



System Properties

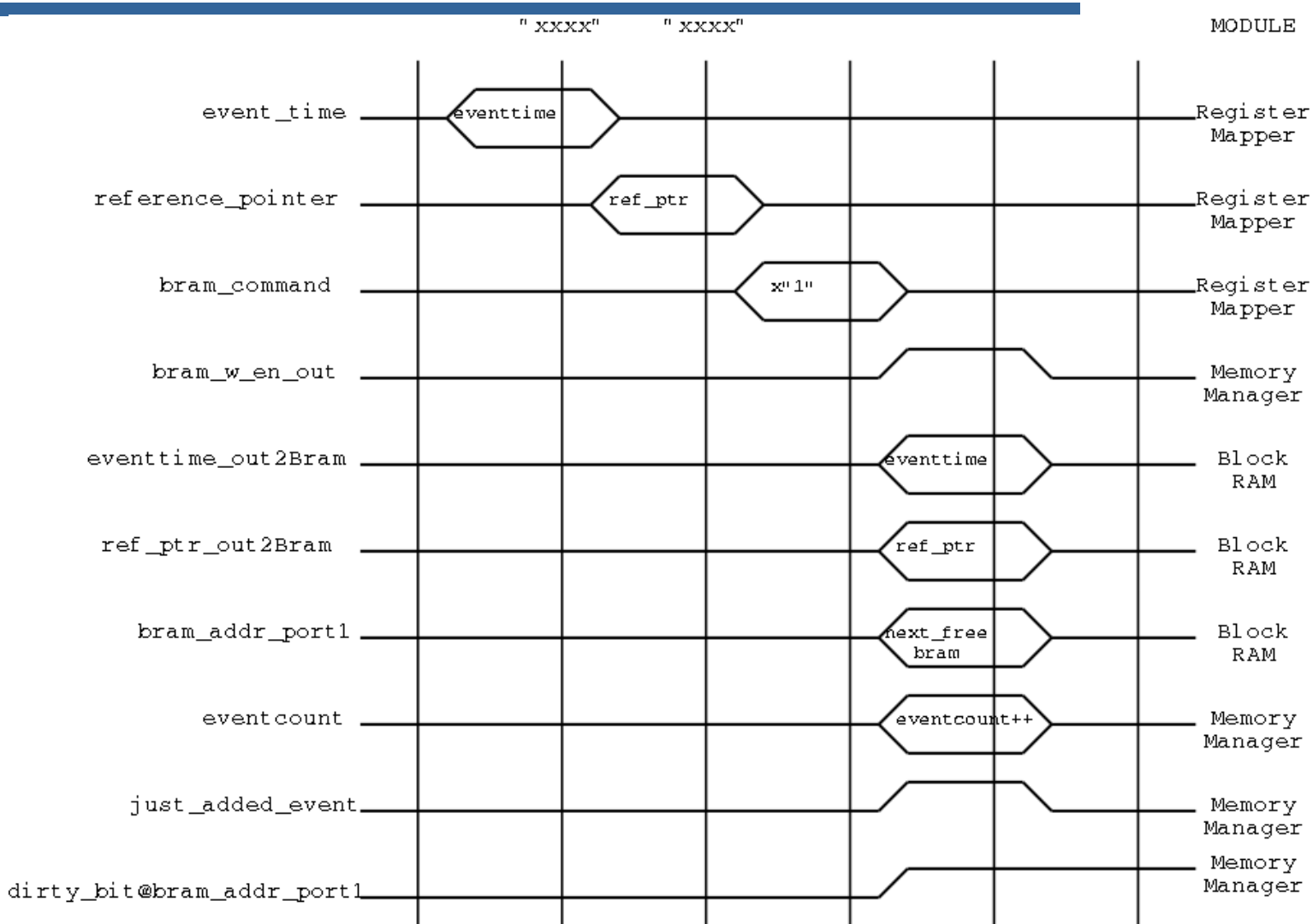
- High interdependency of modules
- Concurrency of deletions and interrupts solved by linear approach
- Highly cohesive scheduling algorithm
- Tight coupling due to system requirements

Testing and Analysis

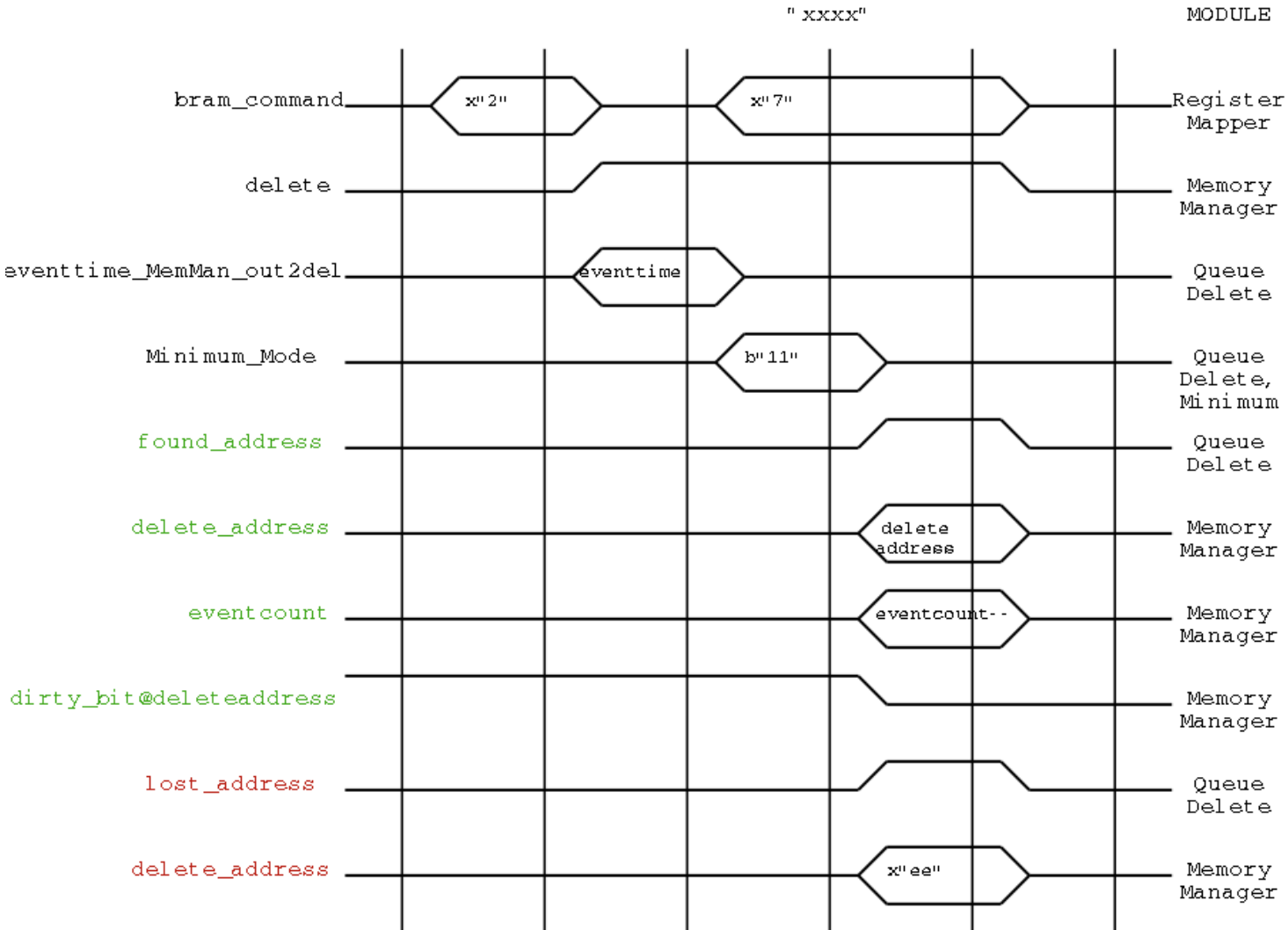
Functionality validation of design

- Initial setup bootstraps FPGA timer shadow
- Check for:
  - Event addition
  - Event deletion
  - Event scheduling

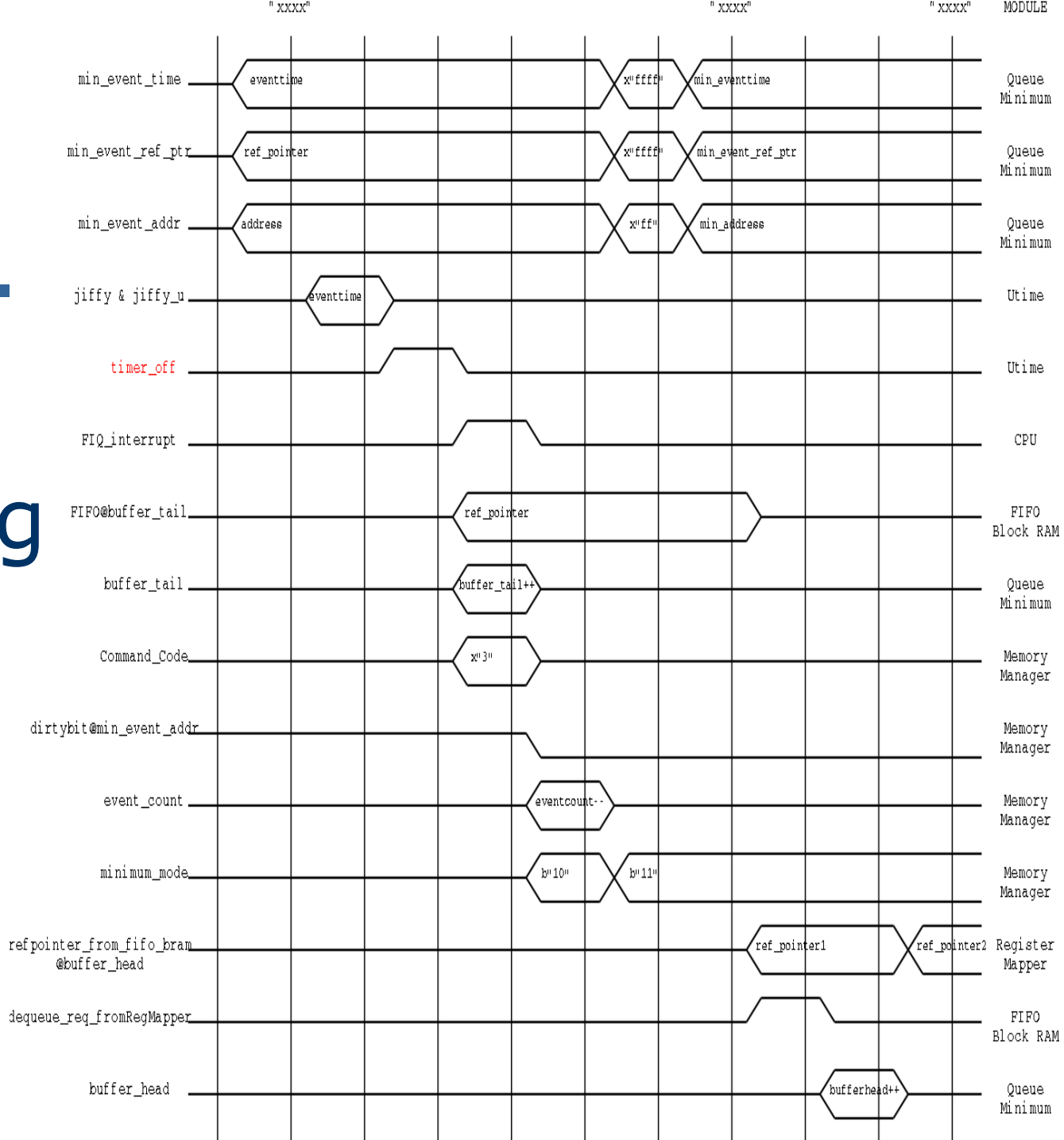
Event Addition Timing Diagram



Event Deletion Timing Diagram



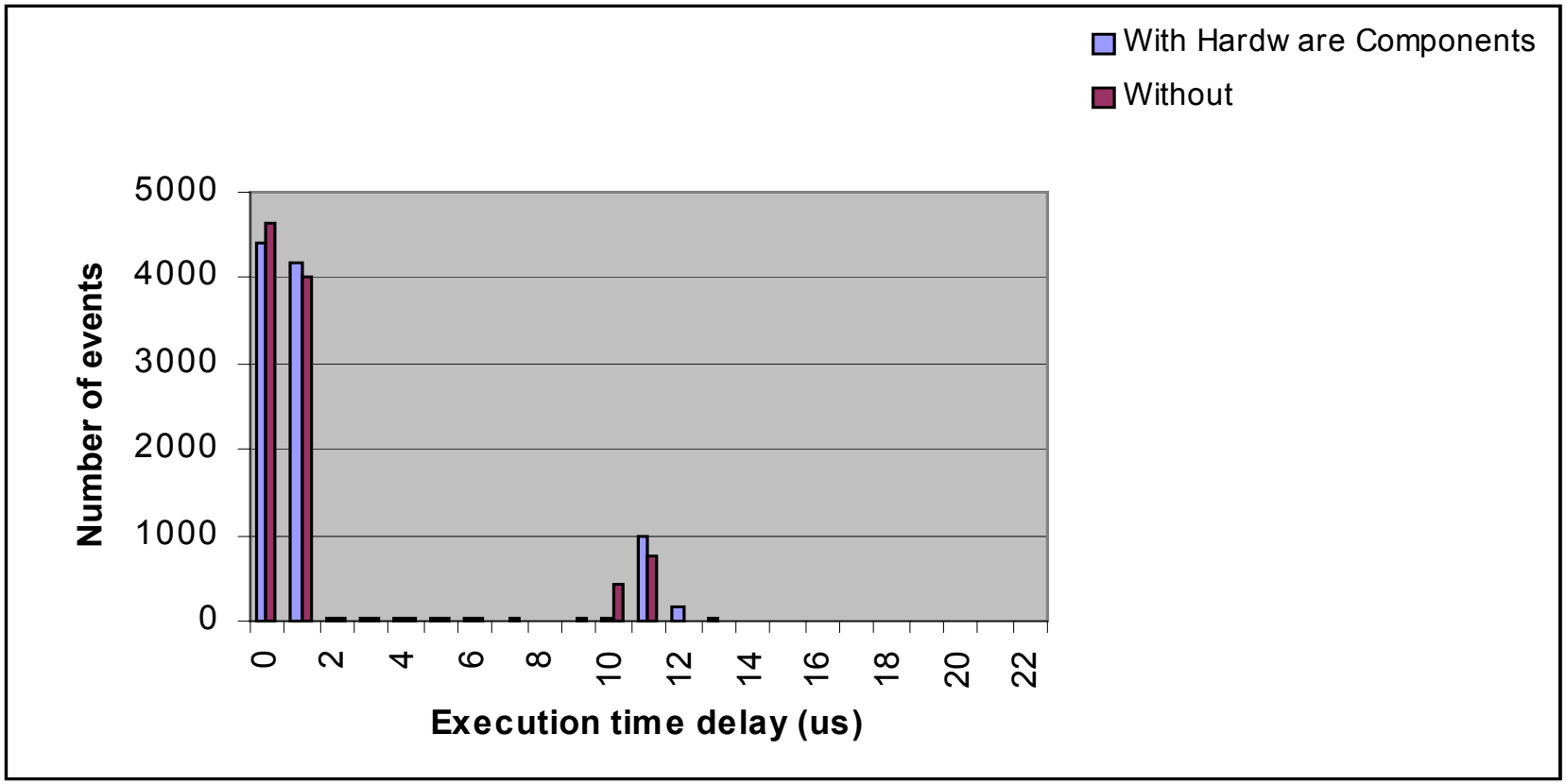
Event Scheduling Timing Diagram



Integrated Scheduler Test

1. Initial Set up
2. Add Event1, Add Event2, Add Event3, Add Event4, Add Event5
3. Find Minimum
4. Delete Event2, Delete Event3
5. Find Minimum
6. Read current time from FPGA timer registers (jiffy, jiffy_u)
7. Write event_time = jiffy + offset
8. Write ref_ptr = loop variable
9. Repeat the above three steps for loop variable = 0 to 2
10. Wait on jiffy + large offset
11. WHILE fifo_ref_ptr != x"00000000" (queue not empty)
12. Read fifo_ref_ptr (confirm all the three timers expired)
13. END WHILE
14. Read jiffy & jiffy_u

Data Stream Kernel Interface Tests



Timer Execution Delay for Software RTOS and Hardware/Software RTOS

Data Stream Kernel Interface Tests

The Software-based RTOS showed expected better performance

- Lag between CPU and FPGA timers of 2-4 microseconds
- FIQ handler schedules an IRQ, downgrading the interrupt priority

Data Stream Kernel Interface Tests

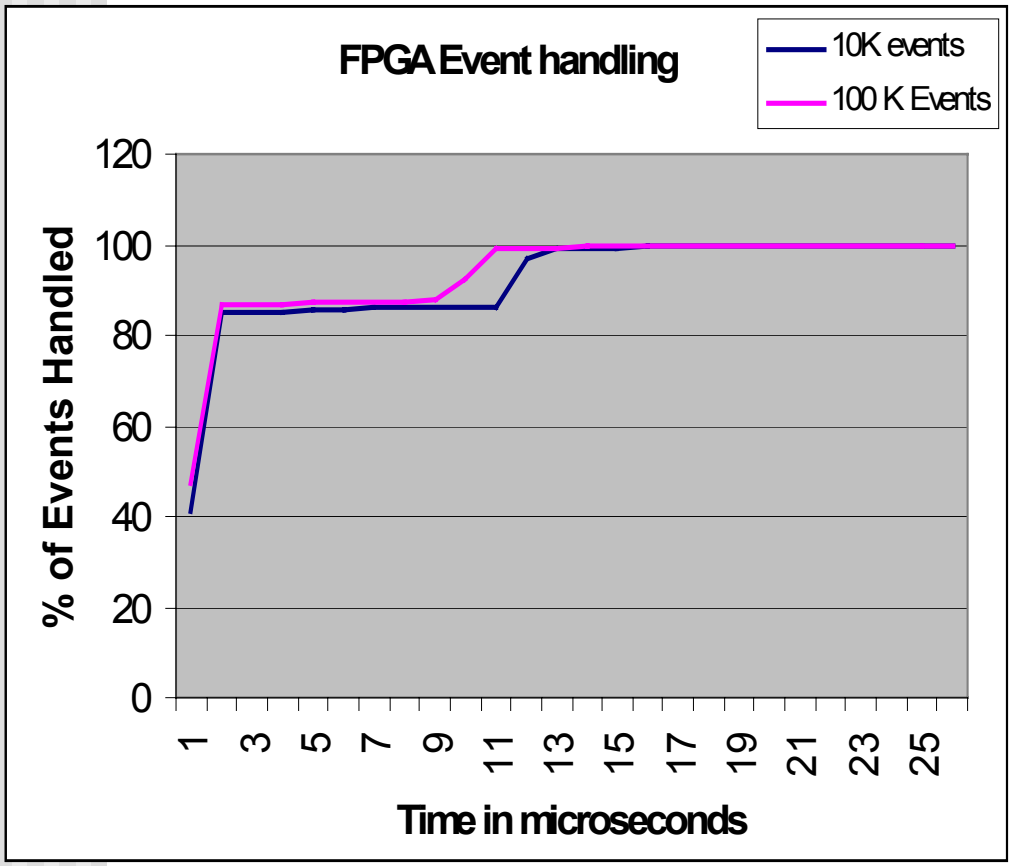
```
[root@george dski]$./dskihists -i 900 -f 4 -h 1 -n 100
Waiting for 900 seconds...
Name of the histogram: HIST_TIMER_EXEC_DELAY
Number of events logged: 10000
Minimum Value: 0
Maximum Value: 7137329
Lower Bound: 0
Upper Bound: 100

   Num of Buckets: 102
```

Stability and Bounded Execution Time

- Maximum execution delay for a software-based timer was in the range of millions of microseconds
- In the hardware version, all events in all tests were executed within the time delay range of 23 microseconds
- Hardware delay attributed to startup delays, worst case scenario checking, interrupt servicing, communication costs across ports

Event handling under different loads



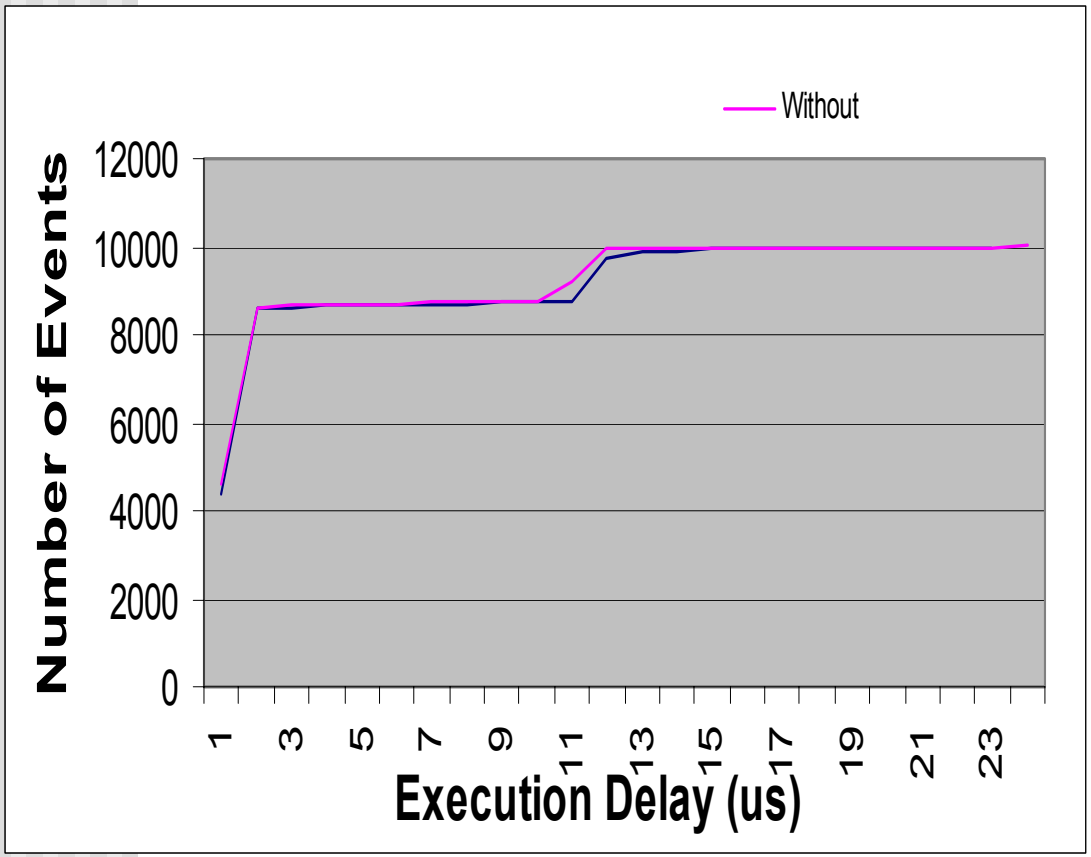
- Increase of 6.4% for events handled with a zero microsecond delay
- 1.7% increase for all events handled within one microsecond
- 12.6% increase after the serial port polling delay between 10-12 microseconds

Device Utilization (V2P)

- EDF scheduling function
  - Logic utilization: 11% SLICES, 17% of LUT, 22% Block RAM, Gate Count of 405,637
  - Device utilization: 24% SLICES
- Total FPGA SLICE Utilization
  - 83% or
  - 2510 out of 3008

Conclusions and Future Work

System behavior



- Aggregate performance characteristics of both systems are comparable
- Robust base for hardware based event scheduler correctness, behavior and performance

System characteristics

- Modular and Portable, IP encapsulated
- Flexible and reconfigurable
- COTS Linux environment
- Tested in different architectures, ADI Engineering's 80200EVB and the Xilinx Virtex-II Pro

Future Work

- Thread Scheduling
- Hardware-based Threads
- Hybrid Threads
- Smarter scheduler functionality
- Only necessary interruptions to CPU

Questions