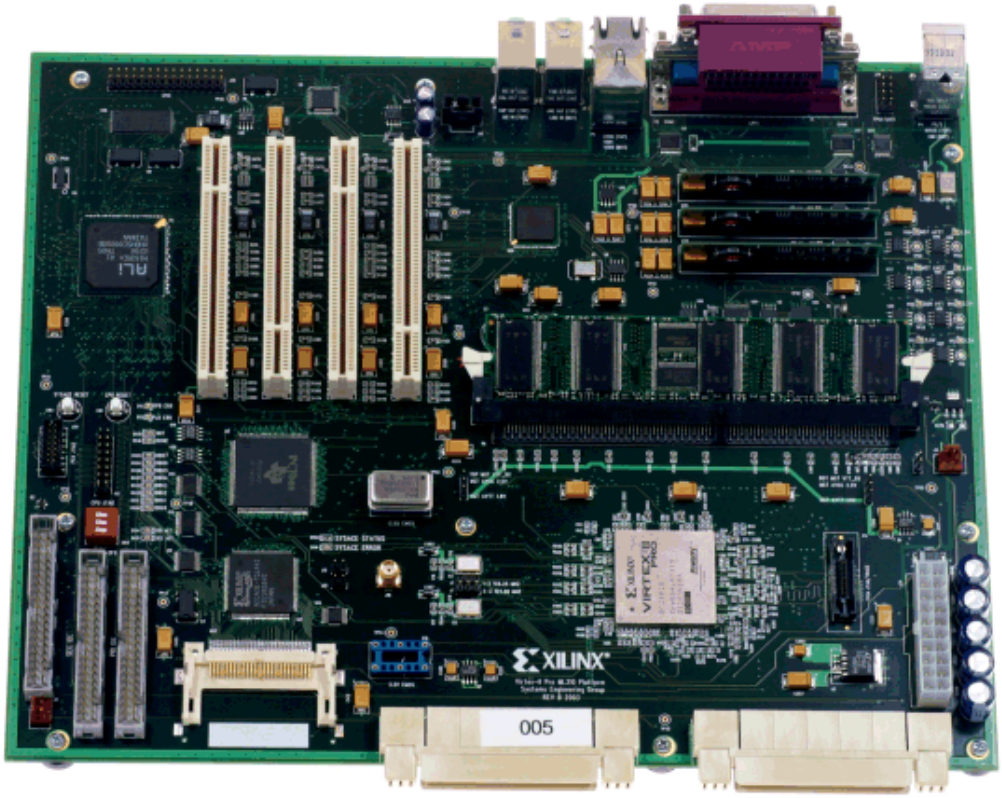
Run-Time Scheduling Support for Hybrid CPU/FPGA SoCs

Jason Agron

jagron@ittc.ku.edu



Acknowledgements

- I would like to thank…
  - Dr. Andrews, Dr. Alexander, and Dr. Sass for assistance and advice in both research and class work.
  - Wes, Garrin, and Erik for help with tools and coding problems.

Publications

- Jason Agron, Wesley Peck, Erik Anderson, Ed Komp, Ron Sass, David Andrews, **Run-Time Support for Hybrid CPU/FPGA Systems on Chip**, Submitted to *Transactions on Embedded Computing Systems*, December 2005.
- Erik Anderson, Jason Agron, Wesley Peck, Jim Stevens, Fabrice Baijot, Ed Komp, Ron Sass, David Andrews, **Enabling a Uniform Programming Model Across the Software/Hardware Boundary**, In *Proceedings of the Fourteenth Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2006),* April 2006.
- David Andrews, Wesley Peck, Jason Agron, Erik Anderson, Jim Stevens, Fabrice Baijot, Presentation on the **KU Hybrid Threads Project,** *MOCHA Design Conference*, January 2006
- D. Andrews, W. Peck, J. Agron, K. Preston, E. Komp, M. Finley, and R. Sass. **hThreads: A Hardware/Software Co-Designed Multithreaded RTOS Kernel**. In *Proceedings of the 10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2005)*, September 2005.
- J. Agron, D. Andrews, M. Finley, E. Komp, and W. Peck. **FPGA Implementation of a Priority Scheduler Module**. In *Proceedings of the 25th IEEE International Real-Time Systems Symposium, Works in Progress Session (RTSS WIP 2004)*, December 2004.
- W. Peck,J. Agron, D. Andrews, M. Finley, and E. Komp. **Hardware/Software Co-Design of Operating System Services for Thread Management and Scheduling**. In *Proceedings of the 25th IEEE International Real-Time Systems Symposium, Works in Progress Session (RTSS WIP 2004)*, December 2004.

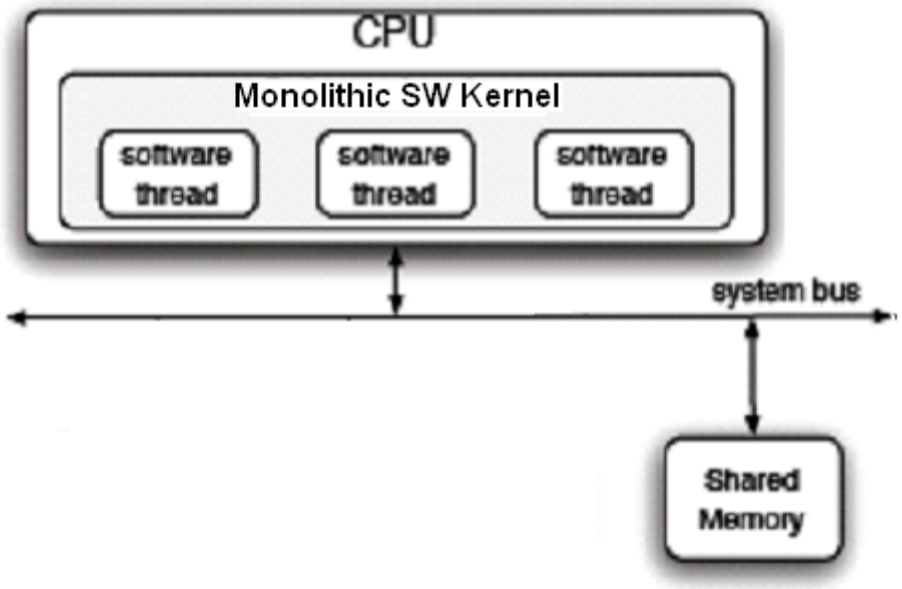
Overview

- HybridThreads Project.
  - Primary Goals
  - Basic Architecture
- Purpose of my work.
- Comparison to related works.
- Designs, Implementations, Results.
- Conclusion.
- Questions.

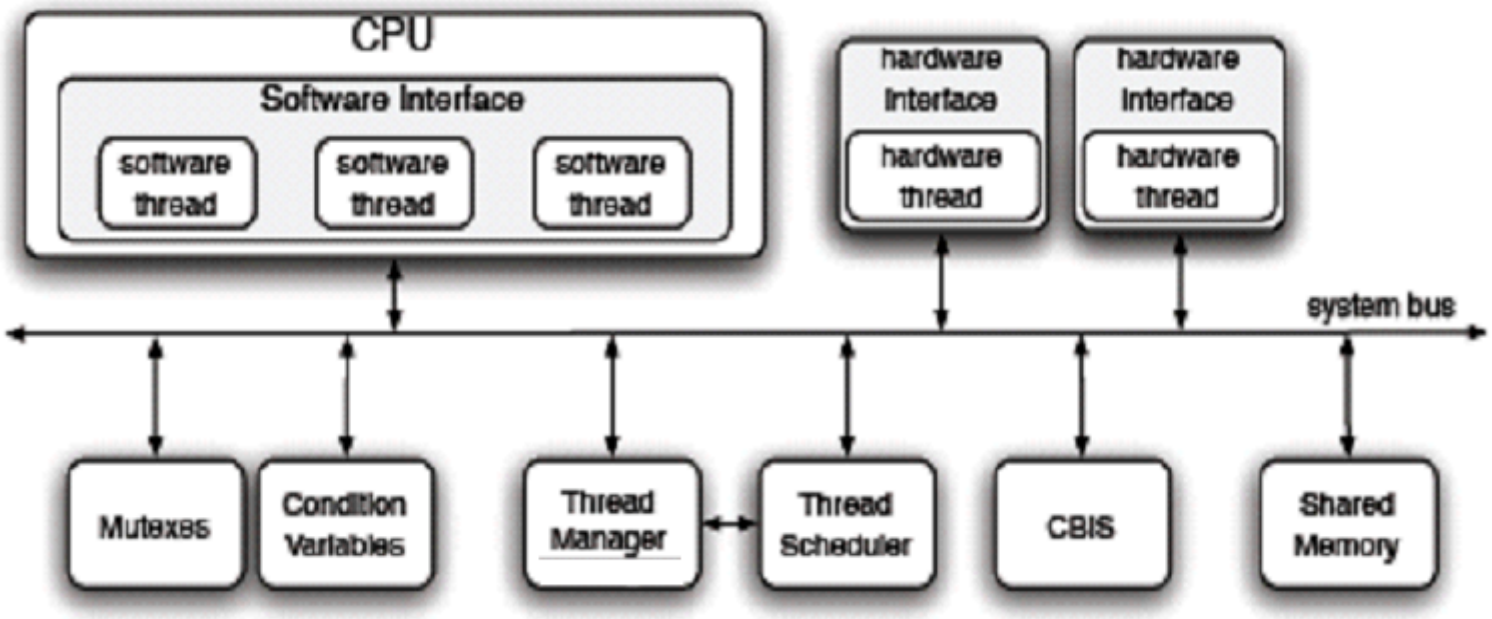
HybridThreads Project

- Hybrid architecture = CPU + FPGA.
  - Portions of programs can now be in SW or HW.
- Create a unified programming model.
  - Threaded programming model based on POSIX standard.
  - All computations are viewed as threads.
    - HW, SW, or both.
- HW/SW co-design of OS services.
  - OS services can be implemented in HW to give a uniform interface to hybrid computations.
    - Anyone that can “talk” on the bus can use the services.
    - No need to interrupt the CPU to access services.
  - HW implementations allow for more parallelism to be exploited.
    - OS services themselves run in parallel with application execution (coarse-grained).
    - Internals of each OS service can be parallelized (fine-grained)
  - Improves accessibility to resources of the FPGA.

Traditional Architecture (All SW)



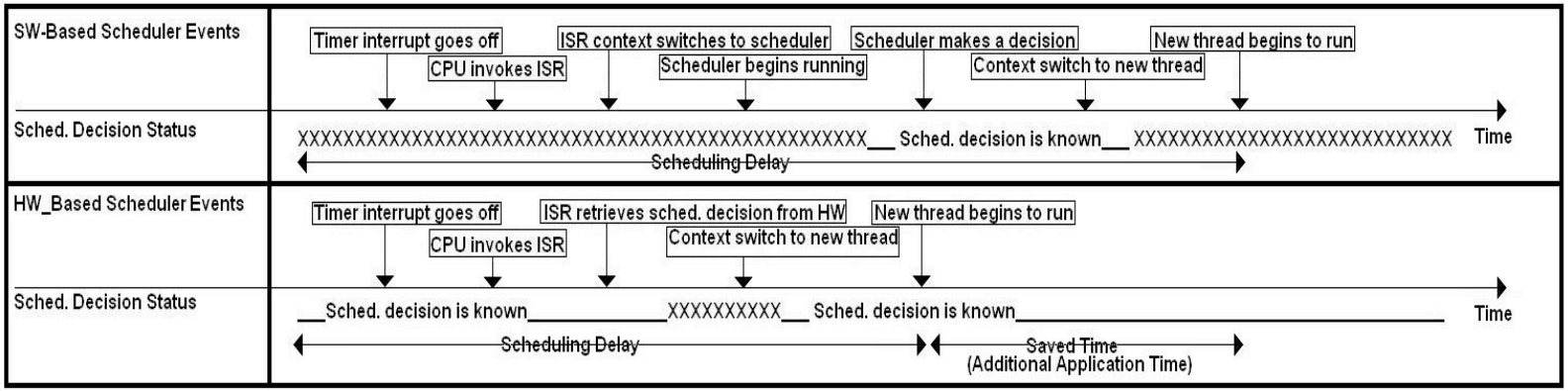
HybridThread Architecture



Purpose of my work

- Provide scheduling support for “hybrid” threads.
  - Uniform APIs, regardless of thread “type”.
- Allow for high-level scheduling policies.
  - Add priority scheduling.
  - Separate OS policy concerns
    - More modular (scheduling != management).
    - Easier to extend and scale within the framework.
- Minimize overhead and jitter!!!
  - Reduce overhead/jitter of system by streamlining the scheduler.

Benefits of a HW-based Scheduler



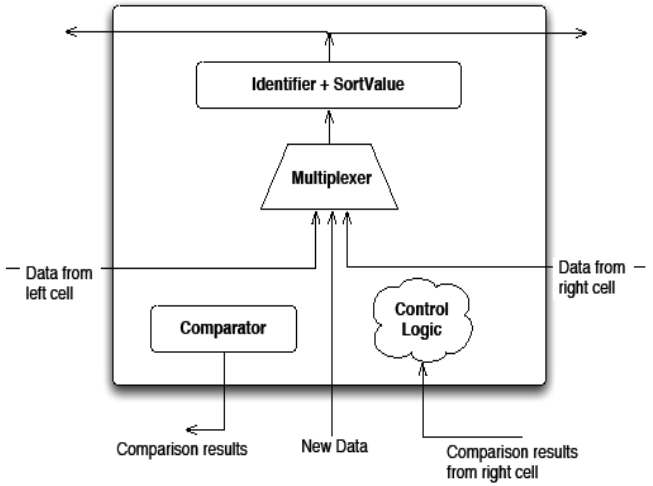
- SW-based scheduler is “invoke on demand”.
  - Starts to get “ready” when needed.
- HW-based scheduler is “ready on demand”.
  - Always “ready” ASAP.

Benefits of a HW-based Scheduler

- Scheduler can be invoked without interrupting the CPU.
  - Scheduler is the sole "bookkeeper" of the R2RQ.
    - Scheduler doesn't require CPU time to execute.
- Traditional ISRs are translated into ISTs (Interrupt Service Threads).
  - Traditional ISRs are akin to threads with priority level of ∞.
  - CPU can be shielded from external interrupts by transforming them into scheduling requests.
    - Interrupts are then fielded by the scheduler, not the CPU.
    - Scheduler decides when to interrupt CPU based on status of R2RQ.
    - Jitter from interrupts can be controlled:
      - Critical interrupts = high priority → interrupt all user threads.
      - Non-Critical interrupts = low priority → interrupt some user threads.
- Scheduling algorithm can be parallelized.
  - Reduction of overhead and jitter.

Related Works

- RealFast/Malardalen – RTU: Real-Time Unit, A Real-Time Kernel in Hardware.
  - Systolic array implementation of R2RQ.
  - EDF, PRI, RM capable.
  - Can handle 16 tasks with 8 priority levels.
- Georgia Tech – Configurable Hardware Scheduler for Real-Time Systems.
  - Systolic array implementation of R2RQ.
  - EDF, PRI, RM capable.
  - 421 logic elements (slices), and 564 registers for queue of size 16.
- Valencia/Valle – A Hardware Scheduler for Complex Real-Time Systems.
  - Systolic array implementation of R2RQ.
  - EDF capable.



Problems with Systolic Arrays

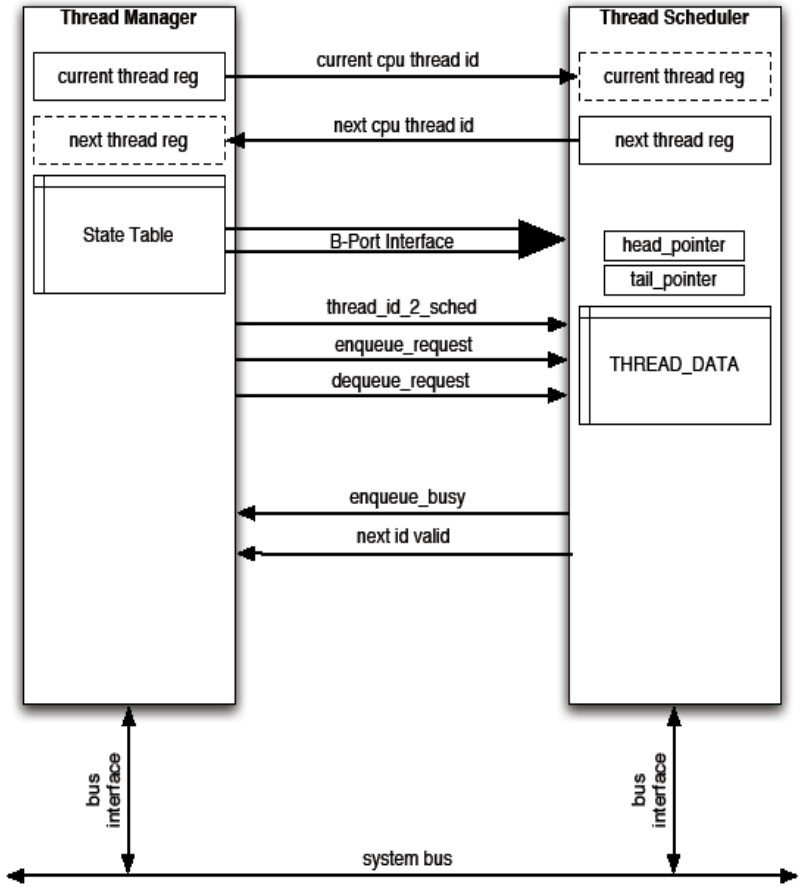
- Systolic arrays are fast and easily allow for dynamic changes in priority.
  - Registers within cells allow for parallel accesses
- Systolic arrays are easily scaled through cell concatenation.
  - Each cell requires registers, multiplexers, comparators, and control logic.
    - Scaling systolic arrays requires lots of logic resources!
- But HW threads require logic resources of FPGA!
  - HybridThread OS modules need to be as small as possible to save space for HW threads.
  - BRAMs are to be used instead of registers to hold ready-to-run queue structure.
  - Pros:
    - Scalable: more space → use more BRAM and slightly more logic.
      - BRAMs don't take up CLBs
      - BUT address decode logic and pointers will grow slightly.
    - Fast: 2 clock cycle reads, 1 clock cycle writes.
      - Almost as fast as registers.
  - Cons:
    - Serial: Only 1 or 2 accesses at a time.
      - Dynamic priority changes can't happen in parallel.

Comparison to Related Works

- HybridThreads is compatible with the POSIX (Pthread) thread standard.
  - RealFast/GaTech/Valencia use their own custom APIs.
    - Doesn't allow for easy portability between systems.
- HybridThreads R2RQ is of size 256.
  - RealFast/GaTech only have R2RQs of size 16.
    - Systolic queue of size 16 requires ~421 slices
    - BRAM queue of size 256 requires ~484 slices.
- HybridThreads must support scheduling of both SW and HW resident tasks.
  - Other systems only handle SW threads.
- HybridThreads system is real → simulatable, synthesizable, and usable.
  - Valencia's scheduler is only theoretical.
  - RealFast/GaTech have real systems
    - BUT you must learn their custom APIs to use their systems.
    - Pthreads applications can be ported to HybridThreads for "free"
      - Using our pthread to hthread wrapper.

Initial Design

- Break scheduling services out of TM.
  - Define a standard interface.
  - Create a R2RQ that is separate from management data structures.
- Add priority scheduling services (while still remaining backwards compatible).

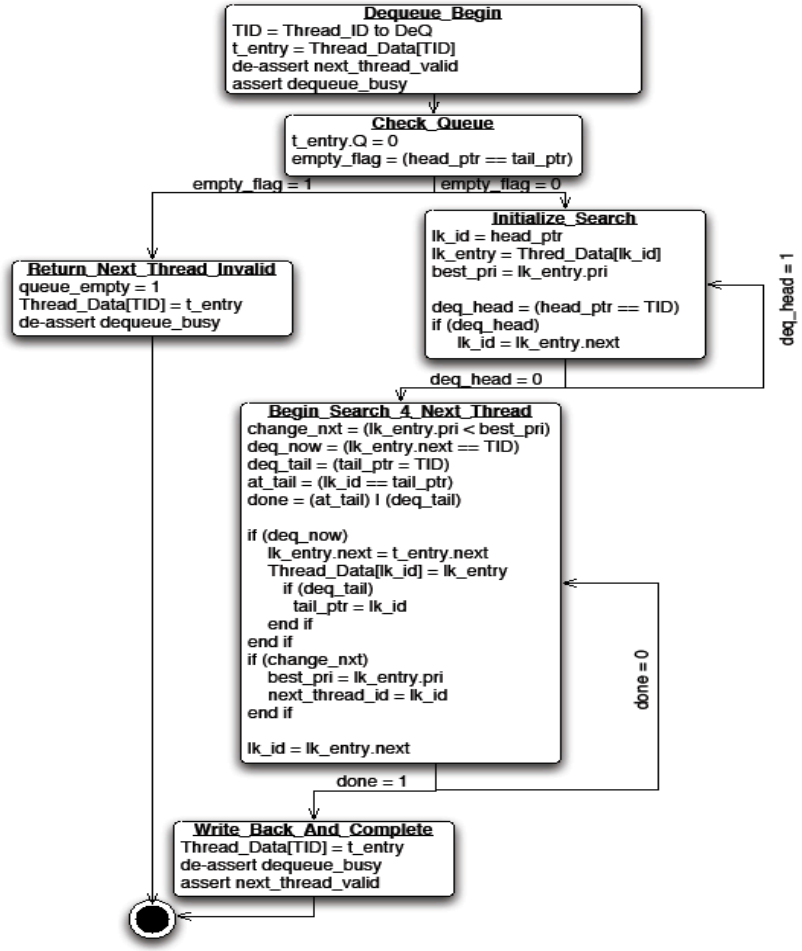


Internals of First Redesign

Thread_Data BRAM

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Q?	N0	N1	N2	N3	N4	N5	N6	N7	L0	L1	L2	L3	L4	L5	L6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Field	Width	Purpose
Q?	(1-bit)	1 = Queued, 0 = Not Queued.
N0:N7	(8-bit)	Ready-to-run queue next pointer
L0:L6	(7-bit)	Scheduling priority-level



Initial Results

- R2RQ of size 256 with 128 priority levels.
  - Linear traversals → O(n).
- Synthesized on Virtex-II Pro 30:
  - 484 out of 13,696 slices, 573 out of 27,392 flip-flops, 873 out of 27,392 4-input LUTs, 1 out of 136 BRAMs.
  - Max. operating frequency of 166.7 MHz.
- Scheduling decision requires ~40 ns per thread in R2RQ.
  - Variable execution-time based on R2RQ length.

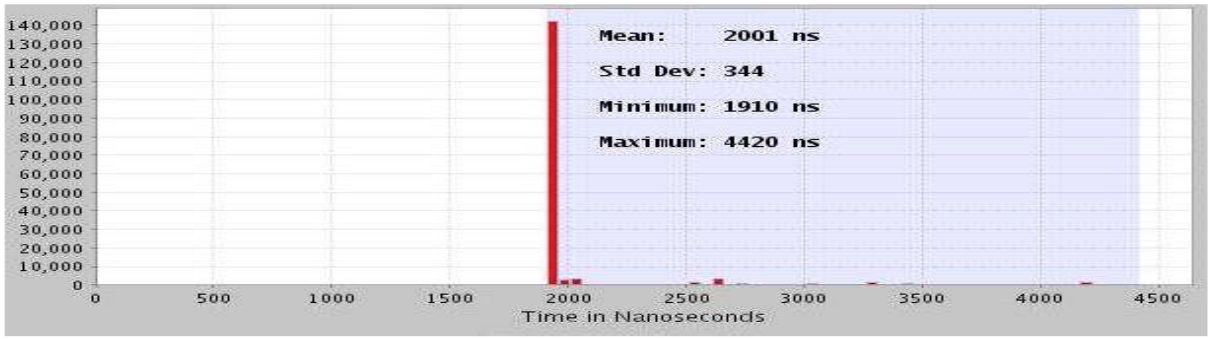
O(n) Timing Results

ModelSim Timing Results of Dequeue Operations, Build 1

No. of Threads in R2RQ	Time (ns)	Est. Time/Thread (ns)
250	10060	40.24
128	5140	40.16
64	2610	40.78
32	1330	41.56
16	690	43.13
2	130	65

- O(n) → Variable scheduling decision delay based on R2RQ length.
- Context switch requires ~ 2µs
  - (R2RQ ≤ 32 threads) → decision completes before C.S. completes
  - (R2RQ > 32 threads) → decision completes after C.S. completes
- As R2RQ length increases so does the possibility of a scheduling event occurring while the next scheduling decision is still being calculated, thus introducing jitter into the system.

O(n) Timing Results



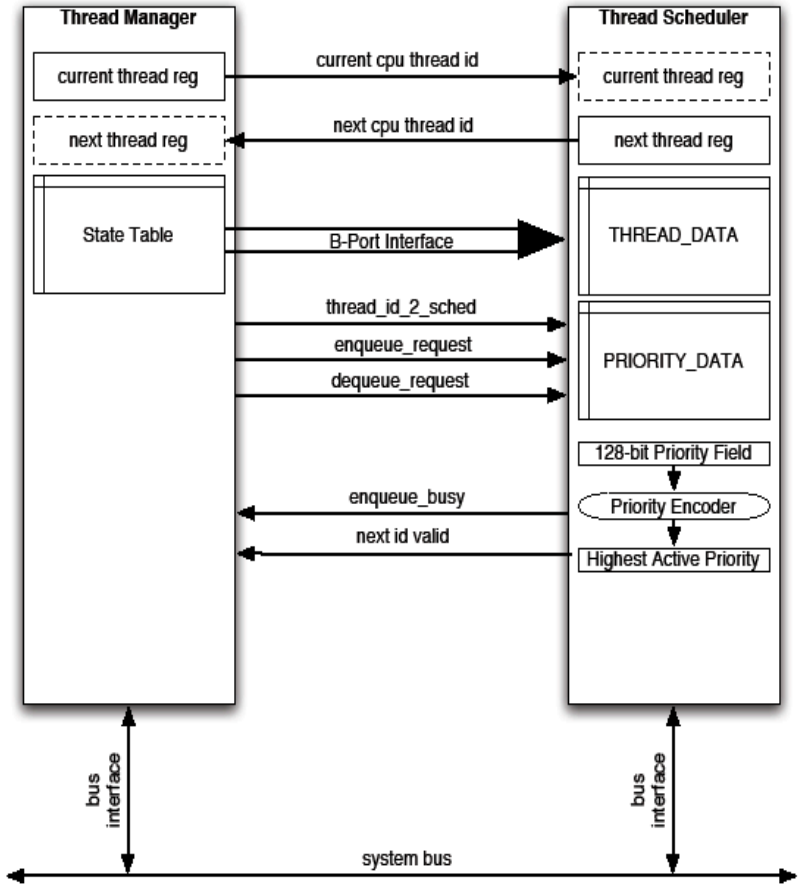
- Raw interrupt delay = time from when an interrupt fires to when the CPU enters the ISR.
  - System dependent, cannot be changed – due to variable length execution times of atomic instructions that delay interrupt acknowledgement.
    - Not affected by number of threads in R2RQ
  - Mean = 0.79 µs. and Jitter = (Max – Mean) = 0.73 µs.
- End-to-end scheduling delay = time from when an interrupt fires to when the C.S. is about to complete (old context saved, new context about to be loaded).
  - Mean = 2.0 µs. and Jitter = (Max – Mean) = 2.4 µs with 250 threads in R2RQ.
    - Jitter is caused by scheduler module and cache.
  - Raw interrupt delay makes up a significant portion (~ 1/3) of end-to-end scheduling delay.

Accomplishments of 1st Design

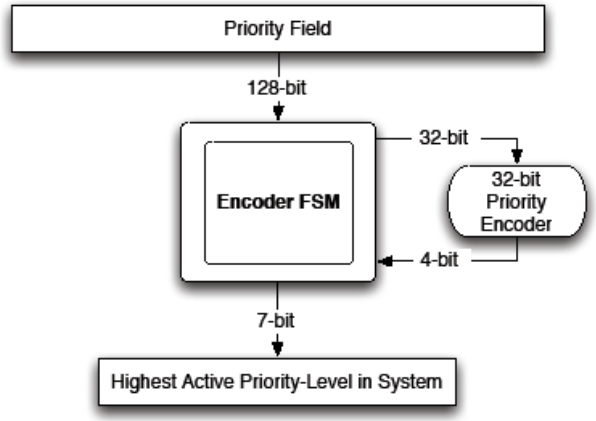
- Developed a standard scheduling interface.
  - Enforces policy, while leaving the mechanism abstract.
- Provides HPF, FIFO, Round-Robin scheduling services.
- O(n) – scheduling decisions.
  - FIFO R2RQ – requires traversal.
  - Decision is slower than context switch sometimes.
    - Long R2RQ = Long traversal.
- Conclusion:
  - Performance could be better.
    - 2.0 µs. end-to-end scheduling delay with 250 threads with 2.4 µs. of jitter is pretty good.
      - Linux delay is in the millisecond range!
      - RealFast/Malardalen's RTU is in the 40 µs range!
  - Still need control of both SW and HW threads.

Second Redesign

- Change R2RQ structure to reduce overhead and jitter.
- Solution = Partitioned R2RQ + Priority Encoder!



Priority Encoder



- Priority Encoder calculates the highest priority level active in the system.
  - Input register: 1-bit per priority level. (1 = active, 0 = non-active).
  - Output register: highest active priority level in the system.
- Requires 4 clock cycles to execute.

Internals of Second Redesign

Thread_Data BRAM

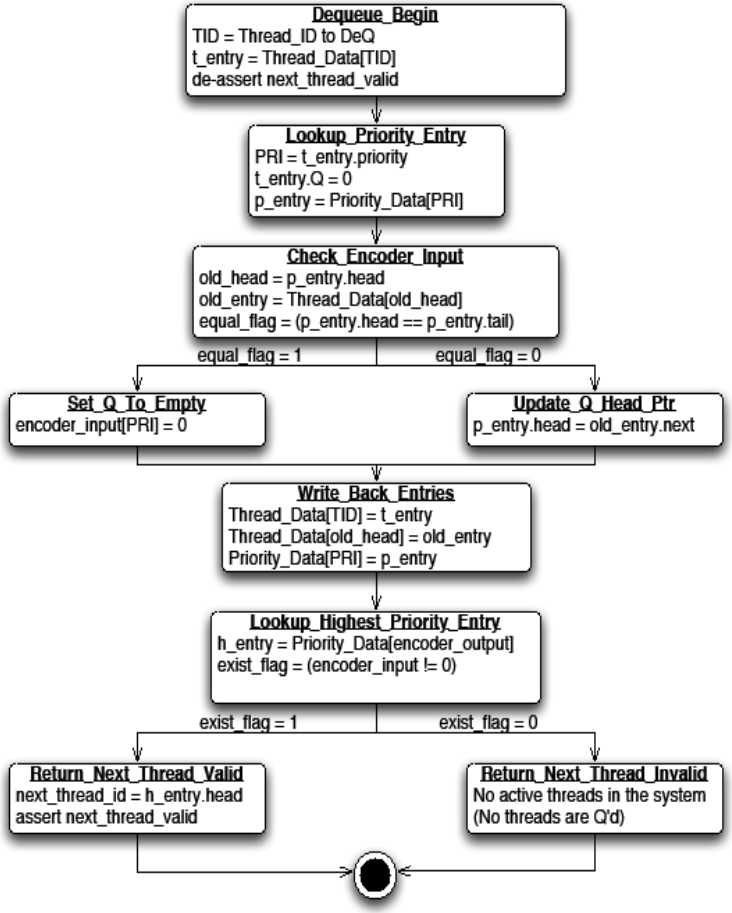
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Q?	N0	N1	N2	N3	N4	N5	N6	N7	L0	L1	L2	L3	L4	L5	L6	P0	P1	P2	P3	P4	P5	P6	P7	-	-	-	-	-	-	-	-

Field	Width	Purpose
Q?	(1-bit)	1 = Queued, 0 = Not Queued.
N0:N7	(8-bit)	Ready-to-run queue next pointer
L0:L6	(7-bit)	Scheduling priority-level
P0:P7	(8-bit)	Ready-to-run queue previous pointer

Priority_Data BRAM

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
H0	H1	H2	H3	H4	H5	H6	H7	T0	T1	T2	T3	T4	T5	T6	T7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Field	Width	Purpose
H0:H7	(8-bit)	Priority-queue head pointer
T0:T7	(8-bit)	Priority-queue tail pointer



2nd Redesign Results

- R2RQ of size 256 with 128 priority levels.
  - No traversals needed → fixed execution times.
  - O(1).
- Synthesized on Virtex-II Pro 30:
  - 1,034 out of 13,696 slices, 522 out of 27,392 flip-flops, 1,900 out of 27,392 4-input LUTs, 2 out of 136 BRAMs.
  - Max. operating frequency of 143.8 MHz.
- Scheduling decision executes in fixed amount of time (~ 24 clock cycles).

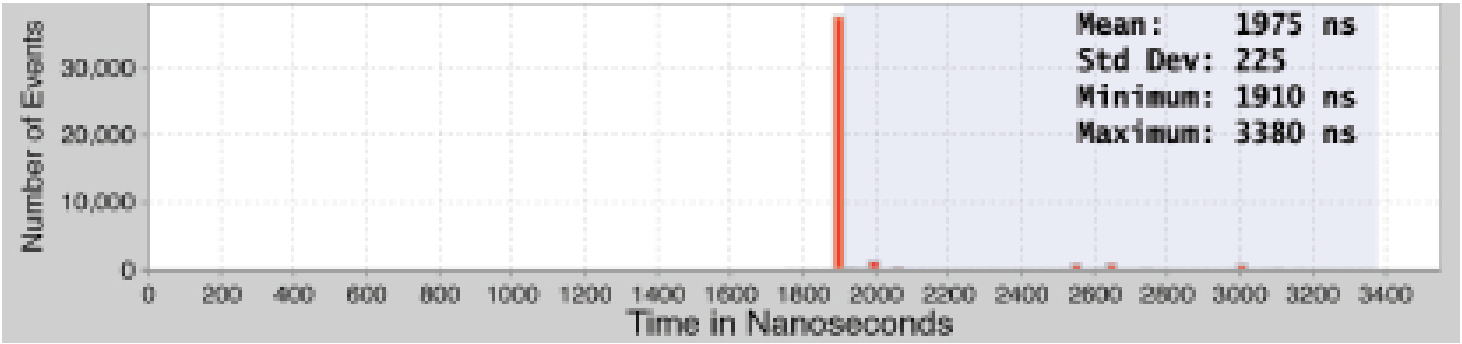
O(1) Timing Results

ModelSim Timing Results of Dequeue Operations, Build 2

No. of Threads in R2RQ	Time (ns)
250	240
128	240
64	240
32	240
16	240
2	240

- O(1) → Constant scheduling decision delay regardless of R2RQ length.
- All scheduling operations execute in fixed amount of time that is less than C.S. time.
  - Scheduling operations do not inject any jitter into the system!

O(1) Timing Results



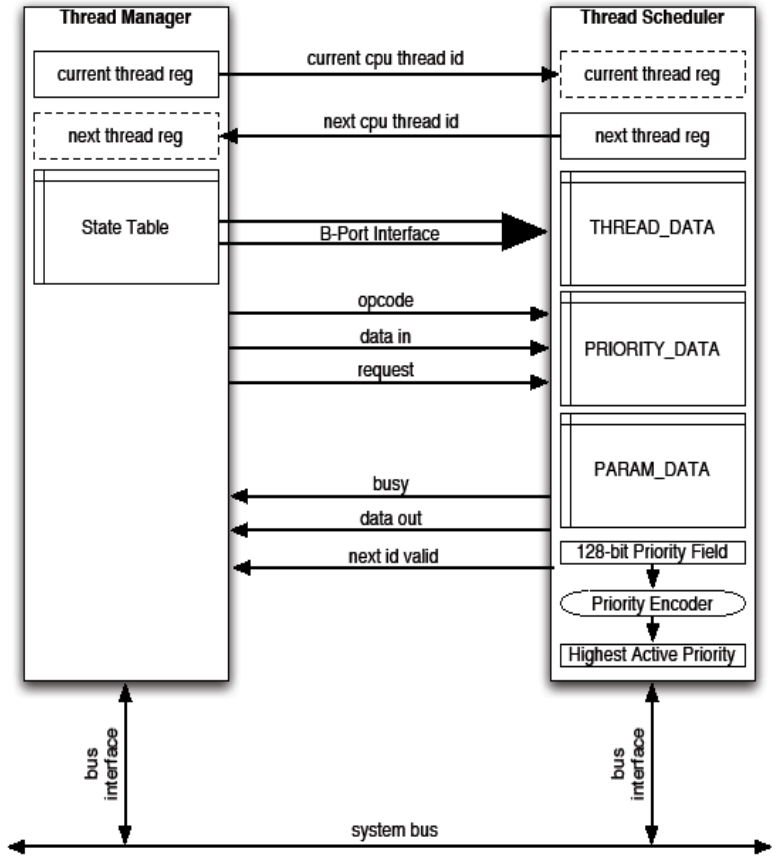
- Raw interrupt delay was re-measured and found to be the same as in the system with O(n) R2RQ.
  - Mean = 0.79 µs. and Jitter = (Max – Mean) = 0.73 µs.
- End-to-end scheduling delay changed based on redesign of scheduler.
  - Mean = 1.9 µs. and Jitter = (Max – Mean) = 1.4 µs with 250 threads in R2RQ.
    - Jitter is caused by the cache.
  - O(1) R2RQ helped to reduce the jitter by approximately 1 µs!

2nd Redesign Accomplishments

- Partitioned R2RQ + Priority Encoder:
  - Executes quickly and in constant time.
    - O(1).
    - ~ 24 clock cycles.
- Priority Encoder:
  - Responsible for scheduling decision (priority level selection).
    - Functionality can be changed (hierarchical).
- Conclusion:
  - Scheduling overhead and jitter have been reduced.
  - Still need support for both SW and HW threads.

Third Redesign

- Provide services for “hybrid” threads.
  - SW threads – covered.
  - HW threads - ???.
- What else changes?
  - All policies deal with threads; which ones need to know “where” they are?
    - Management – allocation, creation, status of TIDs.
    - Scheduling – which TID should run when and where.
- Only the scheduler needs to be changed in order to “hybridize” the system!
  - Could these changes be encoded in the scheduling parameter...



Internals of Third Redesign

Thread_Data BRAM

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Q?	N0	N1	N2	N3	N4	N5	N6	N7	L0	L1	L2	L3	L4	L5	L6	P0	P1	P2	P3	P4	P5	P6	P7	-	-	-	-	-	-	-	-

Field	Width	Purpose
Q?	(1-bit)	1 = Queued, 0 = Not Queued.
N0:N7	(8-bit)	Ready-to-run queue next pointer
L0:L6	(7-bit)	Scheduling priority-level
P0:P7	(8-bit)	Ready-to-run queue previous pointer

Priority_Data BRAM

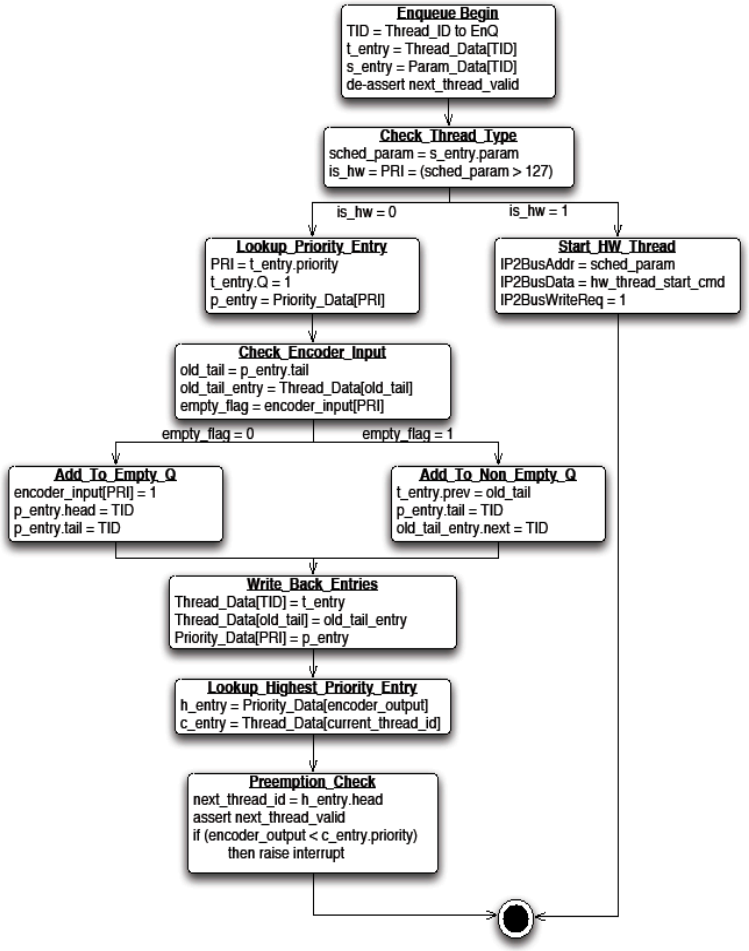
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
H0	H1	H2	H3	H4	H5	H6	H7	T0	T1	T2	T3	T4	T5	T6	T7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Field	Width	Purpose
H0:H7	(8-bit)	Priority-queue head pointer
T0:T7	(8-bit)	Priority-queue tail pointer

Param_Data BRAM

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
s0	s1	s2	s3	s4	s5	s6	s7	s8	s9	s10	s11	s12	s13	s14	s15	s16	s17	s18	s19	s20	s21	s22	s23	s24	s25	s26	s27	s28	s29	s30	s31

Field	Width	Purpose
s0:s31	(32-bit)	Scheduling parameter



3rd Redesign Results

- R2RQ of size 256 with 128 priority levels.
  - O(1).
- Synthesized on Virtex-II Pro 30:
  - 1,455 out of 13,696 slices, 973 out of 27,392 flip-flops, 2,425 out of 27,392 4-input LUTs, 3 out of 136 BRAMs.
  - Max. operating frequency of 119.6 MHz.
- Scheduling decision executes in fixed amount of time (~ 24 clock cycles).
- Uniform support for both SW and HW threads.
  - Thread type encoded in scheduling parameter
    - Low-overhead and jitter scheduling support for SW and HW threads!

3rd Redesign Accomplishments

- Encode HW/SW distinction in sched. parameter.
  - SP < 128 → (SW thread, priority)
  - SP >= 128 → (HW thread, address of cmd. reg.)
- Only the scheduler had to change!
  - Add Master-IPIF.
  - Add storage for larger SP.
  - ENQ – check SP.
    - SW threads: Add SW thread to R2RQ.
    - HW threads: Send "START" command to HW thread.
- Entire system is now truly "hybridized".

Hybrid O(1) Timing Results

ModelSim Timing Results of Scheduling Operations, Build 3

Operation	Time (clock cycles)
Enqueue(SWthread)	28
Enqueue(HWthread)	20 + (1 Bus Transaction)
Dequeue	24
Get_Entry	10
Is_Queued	10
Is_Empty	10
Set_Idle_Thread	10
Get_Sched_Param	10
Check_Sched_Param	10
Set_Sched_Param(NotQueued)	10
Set_Sched_Param(Queued)	50

- Timing results for tests involving only SW threads remain the same as for non-hybrid O(1) scheduler.
- System is now “hybrid” compliant.
  - Operations can be used by both SW and HW threads.

Overall Accomplishments

- HW/SW co-design of scheduling services allows for:
  - Complexity of OS to be pushed into hardware.
    - Relieves CPU of duties
      - OS coprocessors do the work.
      - Less kernel code.
  - More parallelism within the system can be exploited.
    - Internals of scheduler are parallelized, and application and OS run in parallel.
- Breaking up management and scheduling through a standard interface allows for:
  - Greater maintainability
    - Mechanisms of each can be modified independently.
  - Separation of concerns
    - All OS components became “hybridized” by “hybridizing” the scheduler.

Results & Conclusion

- Three design iterations:
  - 1st - Enable priority scheduling → O(n).
    - Still needed to improve performance.
  - 2nd - Improve performance → O(1).
    - Needed to provide hybrid support.
  - 3rd - Combine O(1) with "hybrid" features.
    - Provides FULL system support for ALL threads in a system!!!
- The Result:
  - OS services with generalized support for SW and HW threads.
    - Super low overhead and jitter.
    - Highly scalable due to on-chip BRAMs.
  - Standard interface defined for scheduling operations and data storage.

Questions???

- More information at…
  - http://wiki.ittc.ku.edu/hybridthread.
- I can be contacted at…
  - jagron@ittc.ku.edu.