CPRE 558 Project Proposal

Jake Hafele, Thomas Gaul
jmhafele@iastate.edu, tvgaul@iastate.edu
CPRE 558 Section 1
Department of Electrical and Computer Engineering
Iowa State University, Ames, IA 50014

I. Project Type

Our proposed project will be an **implementation** focused project.

For our design, we are proposing to model a real-time RMS task scheduler on an FPGA development board. In our design, we would include a control module which would determine which periodic task is currently active, and if any ready periodic tasks should be preempted. This control module will drive a series of register buffers, which holds critical information on the state of each periodic task, such as its computation time and period. A counter will be designed to load the computed time of a task and increment the task based on a preset clock period. Finally, we will be required to design a register buffer module, which will hold the state of each task, such as if it has been completed in the current period, its current priority, etc.

FPGA's are useful since they can be used to prototype many different digital designs for cheap, since they are reconfigurable, unlike the ASIC manufacturing process, which is very expensive and time consuming. Compared to using a software based RTOS implementation, FPGAs can also be run at much faster clock speeds, which could be beneficial for modeling RMS systems with smaller task sets that require faster preemption.

II. Project Goal

Our project will be oriented around application driven goals, which will be dictated by the RTL design and FPGA implementation of an RMS scheduler which has been covered in class.

Overall, the goal of this project will be to apply what we have learned about RMS real-time system schedulers and apply them to a field of interest such as computer architecture and FPGA design. By using experience we have gained in digital logic and computer architecture design in classes such as CPRE281 and CPRE381, we will be able to leverage past experience in a new context such as real-time systems, for an idea that has not been fully explored in hardware implementation.

The following list of goals can be summarized as:

- Model periodic RMS task sets as learned in class
- Model RMS schedule on FPGA hardware
- Multiple queues of tasks based on priority
- Compile/Simulate RTL related to real-time system topics
- Synthesize RTL for FPGA configuration for real-time operation
- Preempt task sets based on periodic priority per RMS
- Save register states in between tasks when preempted
- Implement counter based on clock to monitor RMS ready times

III. Solution Approach

This project will expand on tools we have used in previous courses, with the intent to spend more time focusing on the real-time applications of an RMS scheduler within the context of reconfigurable hardware on an FPGA. We will be coding our project using the Verilog HDL language, following IEEE formatting. We will use tools provided to us with the Coover lab computers, including HDL simulator QuestaSim and FPGA synthesis tool Quartus Prime. Since we are using Quartus Prime to synthesize our design, we will need to use an Intel FPGA, which Coover has in stock with their DE-2 FPGA Development boards. In the Spring 2023 semester, we used the DE-2 boards to synthesize our 5 stage MIPS processor design successfully, and plan to recreate that toolflow.

The following list of tools will be utilized for our project:

- Coover RedHat Linux machines
- Verilog HDL language
- QuestaSim for RTL simulation
- Quartus Prime for FPGA synthesis
- DE2 FPGA Dev board

We will apply the same development process as previous classes for developing the RTL and FPGA synthesis, but the main focus of this project will be recontextualizing an RMS schedule with a hardware application. Work will be spent at the beginning of the project to define our requirements based on an example RMS task set, so that we can determine the needs of our FPGA implementation based on counter width, control signals, and how many task registers to create. We will then follow into designing a block diagram of our design and determining what I/O ports are needed between each module, which will be instantiated in our final FPGA wrapper interface to be reconfigured on the DE-2 development board. The Verilog HDL designs will be designed in a text editor of choice, and simulated using QuestaSim, to debug functional waveform errors. Using these wave forms, we can determine if our RMS task sets are being preempted properly, becoming active at the correct time, and are following the correct computation times based on a set synchronous design.

The following process will be used throughout our project for development:

1) Define RMS requirements
2) Create RMS task set for FPGA implementation
3) Define function of each module
4) Draw Block Diagrams for RTL modules
5) Define I/O ports for each module
6) Delegate designer/tester for each module
7) Design RTL using VSCode
8) Test/debug RTL using QuestaSim
9) Synthesize for DE2 board / debug synthesis
10) Evaluate on DE2 board

IV. Expected Outcomes

Our expected outcomes will be a mix of performance evaluation using simulation and demo evaluation using implementation. The simulation results will come from our RTL waveform results using Verilog testbenches and waveforms generated using QuestaSim. The demo/evaluation portion will be used from our final project on a DE-2 FPGA development board, which will be able to confirm the state of multiple period tasks in our implemented RMS scheduler, which should match our expected waveform results from simulation. All of the testbenches and the FPGA configuration for the development board should be reproducible as a final deliverable for the project.

Our expected outcomes can be summarized as below:

- Create reusable Verilog test-benches to create waveforms with RMS scheduler and task preemption
- Demonstrate simulated waveforms showing RMS preemption and task control
- Synthesize RMS scheduler on FPGA and run in real-time
- Demonstrate matching RMS scheduler results between constructed task, waveforms, and synthesized results
- Create FPGA configuration deliverable that can be reused on DE-2 FPGA development board

V. Related Work

Our idea of hardware-based scheduling is novel within the scope of course content in CPRE 55, but it has been explored in many different applications by researchers. The idea was initially suggested in 1995 in the paper "Hardware implementation of a real-time operating system" [1]. From there, it was researched in several ways for the last 28 years. We explore a handful of these previous works as inspirations for our implementation of hardware-based scheduling.

The first paper in 1995 proposed implementing major parts of a real-time operating system in hardware and comparing it to its hardware counterparts. They implemented their primary hardware kernel on an FPGA and found that it performed overwhelmingly well. It performed many times faster than its software counterpart, and the kernel took up less than half the space. The only cost of this concept is the hardware development costs. The next paper we looked at explored the pros and cons of three setups: software, software with a coprocessor, and hardware-software [2]. They found that the hardware outperforms either of the software-exclusive versions and takes up less area and power than a coprocessor solution. Once again, the only issue with this solution is implementation complexity.

Some of the more modern papers look into some other aspects of application outside of overall performance. In the paper "A Hardware Scheduler Based on Task Queues for FPGA-Based Embedded Real-Time Systems," they suggest the possibility of creating a scheduling accelerator that could run on an FPGA in addition to a processor-implemented on an FPGA, which could outperform a hardcore processor due to the advantages of a hardware scheduler [3]. In addition, the paper explores the option of more complicated algorithms that don't take bandwidth due to the hardware accelerator. The next paper we looked at, "A High-Performance Real-Time Hardware Scheduler" looked into algorithms for scheduling a multicore system, which is typically processor-heavy for optimization [4]. Like the early papers, the authors found that these hardware options outperformed software options in every category except implementation complexity.

References

[1] T. Nakano, A. Utama, M. Itabashi, A. Shiomi, and M. Imai, "Hardware implementation of a real-time operating system," in *Proceedings of the 12th TRON Project International Symposium*, pp. 34–42, 1995.

[2] M. Vetromille, L. Ost, C. Marcon, C. Reif, and F. Hessel, "Rtos scheduler implementation in hardware and software for real time applications," in *Seventeenth IEEE International Workshop on Rapid System Prototyping (RSP'06)*, pp. 163–168, 2006.

[3] Y. Tang and N. W. Bergmann, "A hardware scheduler based on task queues for fpga-based embedded real-time systems," *IEEE Transactions on Computers*, vol. 64, no. 5, pp. 1254–1267, 2015.

[4] D. Derafshi, A. Norollah, M. Khosroanjam, and H. Beitollahi, "Hrhs: A high-performance real-time hardware scheduler," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 4, pp. 897–908, 2020.