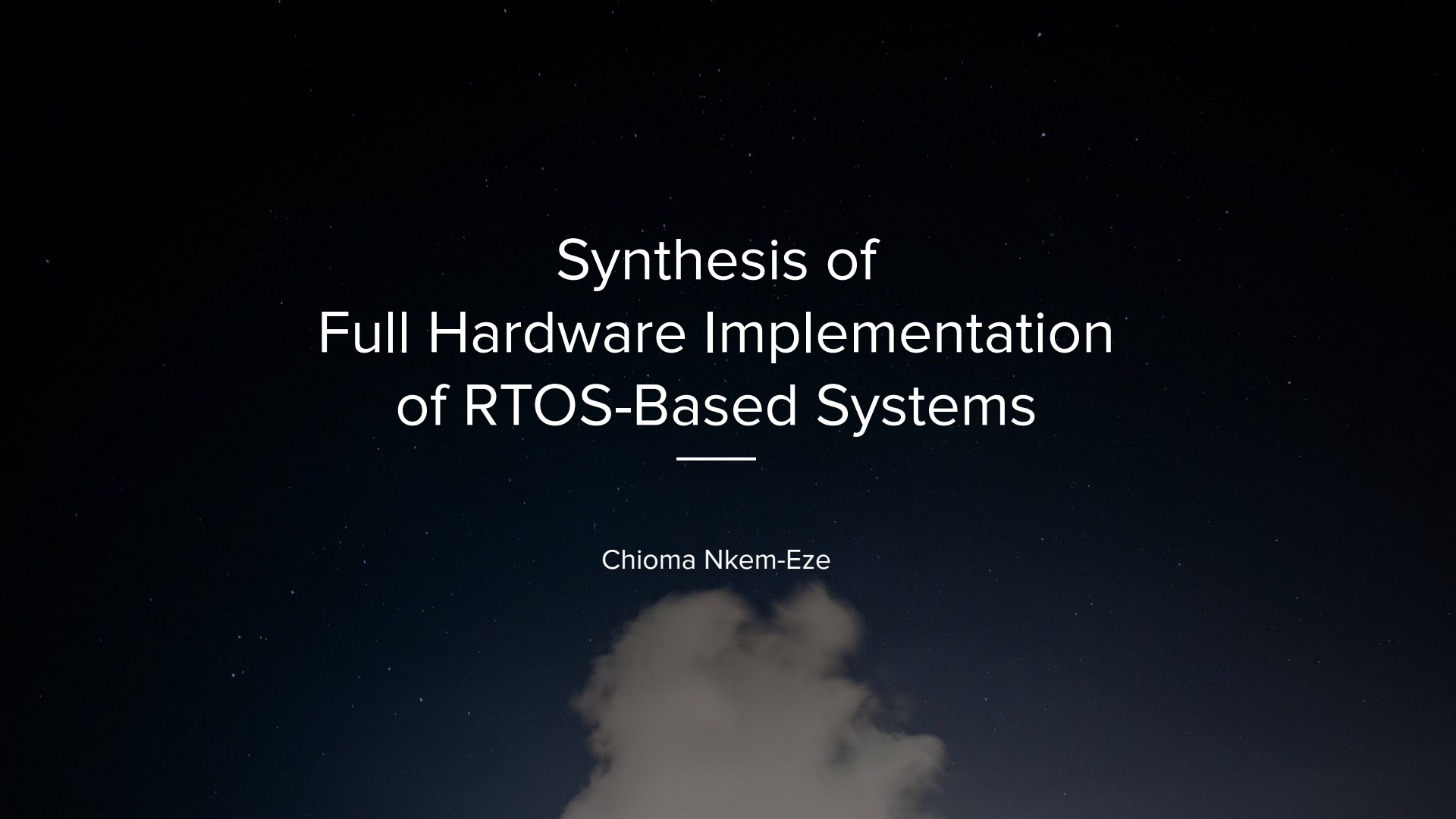
Synthesis of Full Hardware Implementation of RTOS-Based Systems

Chioma Nkem-Eze



Paper’s Objective

Present a method of automatically synthesizing a hardware design from a set of source codes for a real-time system utilizing a Real-Time Operating System (RTOS)

🤔; the WHAT?

An overview of concepts

What is a real-time system?

- Has hardware and software components
- Can interact with the real world
- 🚨 Subject to timing constraints

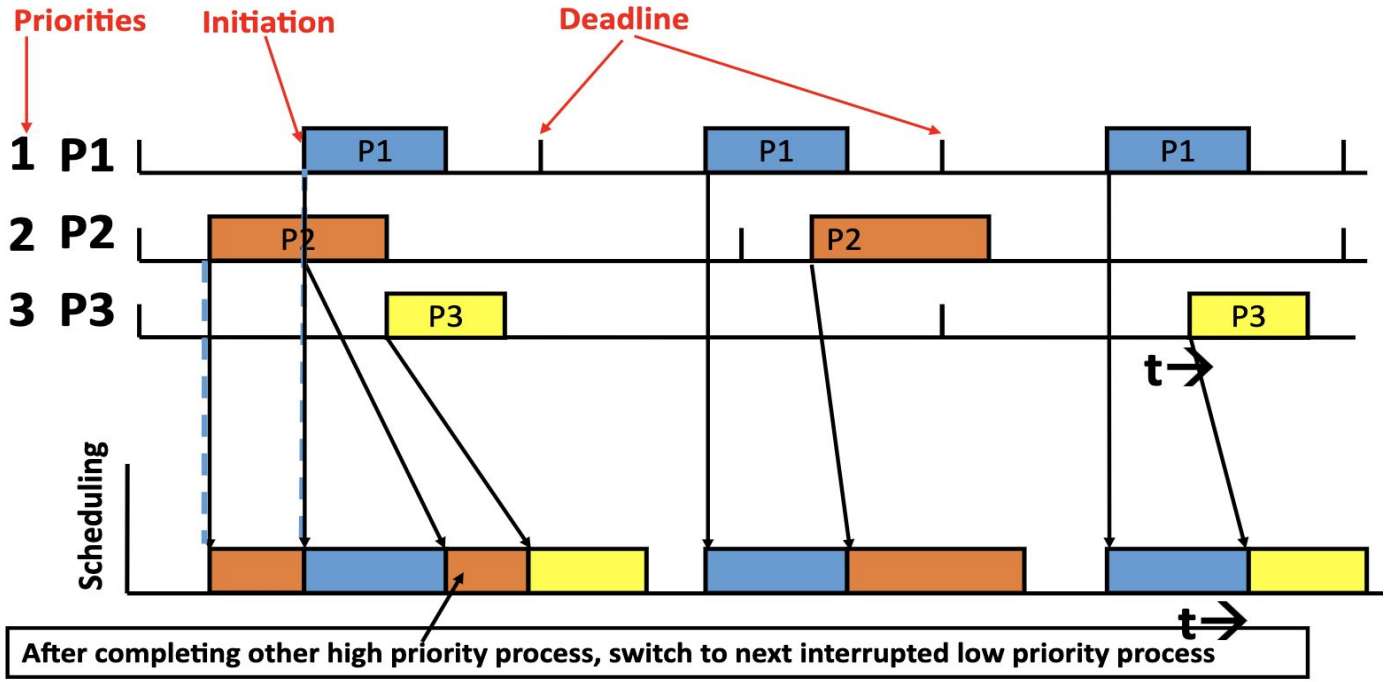
An Illustration

Imagine for a moment that the
airbag doesn't deploy until after 1 minute

RTOS: An Overview

- What is an RTOS?
  - An operating system for real-time systems
- What makes RTOS suited for real-time systems?
  - Scheduler is designed to provide a predictable execution pattern
- How does the RTOS ensure predictability?
  - Tasks and priorities
- Highly responsive system
  - Pre-emptive scheduling

Preemptive Scheduling: A Visual Aid



🤔; the WHY?

RTOS seemed so great

RTOS Overhead

- RTOS is software
- Scheduling Overhead
- Context switching overhead
- Waiting for CPU availability

🤔; the HOW?

Hardware Implementation - A solution

What’s different in this paper?

Implementing both the RTOS functionalities and tasks/handlers into hardware

- No scheduling queues
- **ALL** tasks and handlers are independent hardware modules

Paper's Objective

Present a method of automatically synthesizing a hardware design from a set of source codes for a real-time system utilizing a Real-Time Operating System (RTOS)

Given a set of C source codes that runs on top of an RTOS (TOPPERS/ASP3 kernel in this case), we can generate Verilog HDL codes

How does this work?

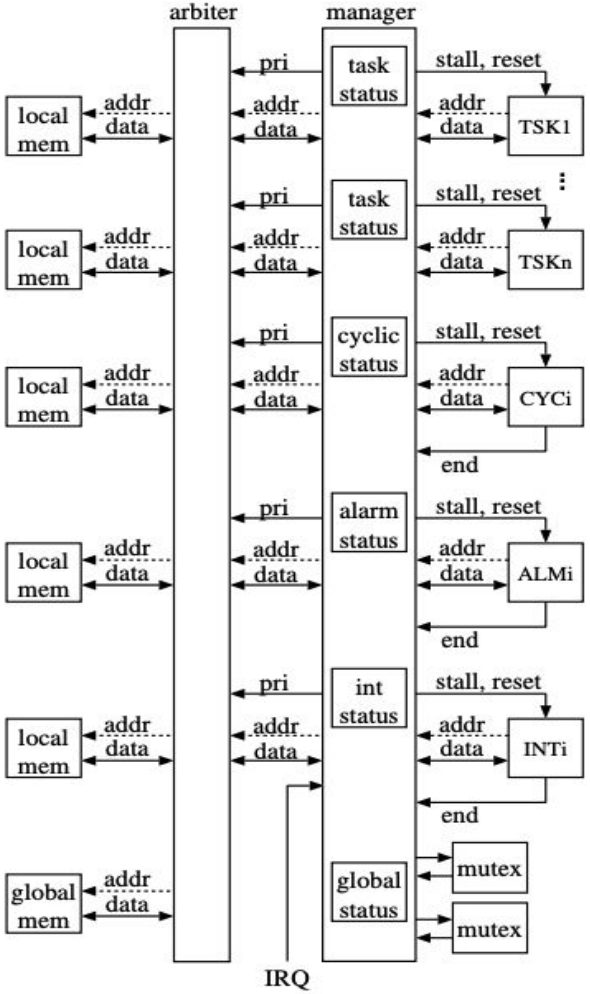
1. Original System: Tasks (TSK), handlers (CYC, ALM, INT), and an RTOS are stored in the instruction memory (IMEM)

2. The system converts IMEM and CPU into equivalent hardware

3. A manager module that controls the run/stall of the task/handler modules

4. An arbiter module to arbitrate data memory (DMEM) accesses

5. Hardware Mutexes



Overhead

- RTOS is software
- Scheduling Overhead
- Context switching overhead
- Waiting for CPU availability

Solved

- Application logic is implemented as hardware
- No wait queues
- Computational resources are not shared between tasks
- Parallel Execution

Conclusion

- Hardware model synthesized from sample code was able to achieve very quick responses
- The size of synthesized hardware is little too large
- Extend applicability of method to other RTOSs such as FreeRTOS

Any Questions?