ESF Textbook Overview

Dr. Alexander G. Dean
Dept. of Electrical and Computer Engineering
My Background

- Associate Professor of ECE at North Carolina State University (Raleigh, NC)

- Course development and teaching
  - Embedded Systems Fundamentals
  - Embedded System Architectures
  - Embedded System Optimization
  - Advanced Embedded Systems

- Industry Engagement
  - Consulting: Over 60 on-site embedded system software design reviews and training engagements since 2001
  - Five courses developed for university programs of three MCU/CPU companies

- Research in embedded systems
  - Compilers (back-end): software thread integration, stack frame splitting, code transformation
  - Real-time systems: static timing analysis, predictable memory systems
  - Low-cost digital control of switching power converters
Basics of the Book

- Targets ECE or CS undergrads with modest C programming background and basic circuit knowledge
- Exercises at end of each chapter, solutions available to instructors.

Table of Contents
1. Introduction
2. General Purpose Input/Output
3. Basics of Software Concurrency
4. ARM Cortex-M0+ Processor Core and Interrupts
5. C in Assembly Language
6. Analog Interfacing
7. Timers
8. Serial Communications
9. Direct Memory Access

Appendix. Measuring Current, Power and Energy on the FRDM-KL25Z

Platform – FRDM-KL25Z (<$15)
- 32-bit ARM Cortex-M0+ CPU in NXP Kinetis MCU
- 48 MHz clock, 128 KB Flash ROM, 16 KB RAM
- Onboard Debug MCU via USB
- Board peripherals: 3-axis accelerometer, RGB LED, capacitive touch slider, virtual serial port (USB)
- Arduino footprint expansion ports
- mbed.org enabled - online development toolchain
Why Pick This Book?

- Companion to ARM Education Kit on Efficient Embedded System Design and Programming
- ARM Cortex-M is dominant architecture for MCUs.
  - Highly scalable
  - Synergy with curriculum and research
- Easy to read
- Covers crucial embedded system topics guided by industry experience
  - Strong endorsements from industry and academia
- Hands-on approach
Reason #1: Easy to Read

- Clear, concise writing style focuses on most critical points
- Goes just deep enough into critical concepts (but no deeper)
Reason #2: Covers the Right Material

- Chapter 1: Introduction
  - Embedded System Concepts: Why and How
  - Target Platform
- Chapter 2: General Purpose Input/Output
  - Outside the MCU: Ones and Zeros, Voltages, and Currents
    - Input Signals, Output Signals
    - Interfacing with a Switch and LED Lights
  - Inside the MCU
    - Preliminaries: Control Registers and C Code
    - Configuring the I/O Path
    - GPIO Peripheral
    - Putting the C Code Together
- Chapter 3: Basics of Software Concurrency
  - Creating and Using Tasks
  - Improving Responsiveness
    - Interrupts and Event Triggering
    - Reducing Task Completion Times with FSMs
  - Using Hardware to Save CPU Time
  - Advanced Scheduling Topics
    - Waiting
    - Task Prioritization
    - Task Preemption
    - Real-Time Systems
Reason #2: Covers the Right Material

- Chapter 4: ARM Cortex-M0+ Core and Interrupts
  - CPU Core and Architecture
  - Registers
  - Accessing Memory
  - Instructions
  - Exceptions and Interrupts
    - CPU Exception Handling Behavior
    - Hardware for Interrupts and Exceptions
    - Software for Interrupts
      - Program Design
      - Interrupt Configuration
      - Writing ISRs in C
      - Sharing Data Safely Given Preemption

- Chapter 5: C in Assembly Language
  - Software Development Tools
  - C Language Fundamentals
  - Making Functions
  - Controlling the Program’s Flow
  - Accessing Data in Memory

- Chapter 6: Analog Interfacing
  - Introduction
    - Quantization
    - Sampling and Aliasing
    - Digital-to-Analog Conversion
    - Analog Comparator
    - Analog-to-Digital Conversion
Reason #2: Covers the Right Material

- **Chapter 7: Timers**
  - Concepts
  - Timer Circuit Hardware
  - Example Timer Uses
  - Timer Peripherals
    - SysTick Timer
    - Kinetis KL25Z COP Watchdog Timer
    - Kinetis KL25Z Timer/ PWM Module

- **Chapter 8: Serial Communications**
  - Concepts
    - Why and How
      - Serialization, Symbol Timing, Message
  - Framing, Error Detection, Acknowledgments, Media Access Control, Addressing
  - Protocol Analyzers
  - Software Structures for Communication
    - Supporting Asynchronous Communication
    - Queue Implementation
    - Queue Use
  - Serial Communication Protocols and Peripherals
    - Synchronous Serial Communication (SPI)
    - Asynchronous Serial Communication (UART)
    - Inter-Integrated Circuit Bus (I2C)
Reason #2: Covers the Right Material

- Chapter 9: Direct Memory Access
  - Concepts
  - KL25Z DMA Controller and Multiplexer Peripherals
    - DMA Multiplexer and Trigger Sources
    - DMA Controller
    - Basic DMA Configuration and Use
    - Examples
      - Bulk Data Transfer
      - Analog Waveform Generation

- Appendix: Measuring Current, Power, and Energy on the FRDM-KL25Z
  - Power System Architecture
  - Current Measurement and Power Calculation
  - Power Reduction
  - Energy Measurement
  - Circuit Modification Summary
Key Differentiator: Understanding Concurrency

Hardest concept for students to master

- Typical embedded system manages multiple activities
  - Use software on CPU
  - Use peripheral hardware
  - Activities have varied timing requirements

- Provide mental framework for students
  - What work can the peripherals do?
  - How do we share the CPU's time among the software activities and get things done on time? (task scheduling)

- Textbook tackles concurrency early with a tangible, hands-on example
  - Chapter 2: Starting point of flashing an LED and reading a switch using GPIO
  - Chapter 3: Applying concurrency and interrupts to improve responsiveness
  - Chapter 4: In-depth coverage of interrupts
Motivating Example for Concurrency: LED Flasher

Overview of program design

- Requirements
  - Two modes
    - LEDs lit in Red/Green/Blue Sequence
    - LEDs lit in White/Black sequence
  - Pressing SW1 speeds up flashing or sequencing rate
  - Pressing SW2 forces flash mode
- Program uses busy-wait loops for time delays
- Responsiveness: How long from switch change to LED change?
Using HW and SW Concurrency

1. Starter Program
2. Tasks + Cooperative Scheduling
3. Shorter tasks with FSMs
4. Use switch interrupts
5. Use timer peripheral
6. Task prioritization
7. Task preemption

Response Time
- Faster
- Slower

CPU Overhead
- Less
- More
### Comparing Response Times

<table>
<thead>
<tr>
<th>Starter Code</th>
<th>LED Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R G B R G B R G B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cooperative Tasks</th>
<th>LED Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task_Read_Switches</td>
<td>R G B</td>
</tr>
<tr>
<td>Task_Flash</td>
<td>R G B</td>
</tr>
<tr>
<td>Task_RGB</td>
<td>R G B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cooperative Tasks and Switch ISR</th>
<th>LED Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>PORTD_IRQHandler</td>
<td>R G B</td>
</tr>
<tr>
<td>Task_Flash</td>
<td>R G B</td>
</tr>
<tr>
<td>Task_RGB</td>
<td>R G B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cooperative FSM Tasks</th>
<th>LED Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task_Read_Switches</td>
<td>R G B</td>
</tr>
<tr>
<td>Task_Flash</td>
<td>R G B</td>
</tr>
<tr>
<td>Task_RGB</td>
<td>R G B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cooperative FSM Tasks and Hardware Timer</th>
<th>LED Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task_Read_Switches</td>
<td>R G B</td>
</tr>
<tr>
<td>Task_Flash</td>
<td>R G B</td>
</tr>
<tr>
<td>Task_RGB</td>
<td>R G B</td>
</tr>
<tr>
<td>PORTD_IRQHandler</td>
<td>G B R G</td>
</tr>
<tr>
<td>Task_Flash</td>
<td>R G B</td>
</tr>
</tbody>
</table>
Delay with ISR

- **Switch Inputs**
- **PORTD_IRQHandler**
- **g_Flash_LED**
- **Task_Flash**
- **Task_RGB**
- **LED Output**

Switch 1 pressed

- $T_1$
- $T_2$

Time
Splitting the Work between ISR and Task

**Interrupt service routine**

**Option 1: Short ISR**
- Did switch 1 change?
  - Determine new value of switch 1
- Did switch 2 change?
  - Determine new value of switch 2
- Update `g_RGB_delay`, `g_w_delay`

**Option 2: Medium ISR**
- Update `g_flash_LED`
- Optional: Update LEDs immediately

**Option 3: Long ISR**
- Task_RGB: Light LEDs in RGB sequence
- Task_Flash: Flash LEDs white and black

**Task or main-line code**
Reason #3: Hands-on Examples

Course material contains extensive demonstration code

- LEDs with switches (GPIO, interrupts and concurrency)
- Driving a speaker (GPIO)
- Voltage transition monitor (comparator)
- Thermistor-based thermometer (ADC)
- Infrared proximity sensor (ADC, GPIO)
- 1 Hz tick (timer)
- LED dimmer (PWM)
- Time-out tilt detector (I²C, watchdog timer)

- Serial console interface with PC (UART)
- Loopback test (SPI)
- Analog waveform generator
  - Ch. 6: V1 (DAC, software delay loop). 100% CPU utilization
  - Ch. 7: V2 (DAC, timer and ISR). 14% CPU utilization
  - Ch. 9: V3 (DAC, timer and DMA). 0.48% CPU utilization
- Bulk data transfer example (DMA)
Thank You!

- **Why this book?**
  - Covers critical topics, including concurrency
  - Easy to read
  - Hands-on approach
  - Targets inexpensive hardware
  - Companion to ARM Education Kit course

- **Links**
  - Available on Amazon (search for 9781911531036)
  - My book support website (has these slides): https://embeddeddean.wordpress.com/
  - agdean@ncsu.edu
  - https://people.engr.ncsu.edu/agdean/