ECE 460/560 Midterm Exam Study Guide

- **Process Concepts (Lectures and Theory HW1)**
  - Concepts for systems of concurrent processes
    - Process relationships: Concurrency, synchronization, communication
    - Process types: Software. Threads (inc. tasks), handlers (ISRs). Hardware peripherals
    - Inter-process synchronization and communication: SW->SW, SW->HW, HW->SW, HW->HW
    - Aspects of inter-process communication: How many senders? How many receivers? Notify receiver of new data? Notify sender of reception? Allow data loss or buffer it? How much buffering?
    - Software process scheduling: implicit program instruction sequence, interrupt system, task/process scheduler
  - Use of concurrent processes for embedded systems
    - Web of processing chains connecting inputs and outputs
    - Processing chain stages
    - Stage implementations in software and/or hardware
  - Analyzing timing for example system: quadrature decoder for shaft encoder (THW1)
    - Determine signal timing behavior and sampling requirements (deadline window)
    - Determine CPU timing behavior (interrupt response, input sampling, processing)
    - Determine maximum shaft rotation speed based on signal timing requirements, CPU timing behavior, and required free CPU utilization
- **Process Synchronization (Lectures and Theory HW 2)**
  - Why synchronize?
    - Synchronizing because of I/O – Key feature for embedded and real-time systems
      - Synchronize to event or time
      - Synchronizing software process reduces its timing variability (possible timing error) of when a given instruction might execute relative to the sync operation
      - Timing variability grows after synchronization due to software and hardware complexities (different control flow path durations, preemption, pipeline stalls, memory system latency, etc.)
    - Synchronization with other processes to allow/prevent progress, enable correct communication
  - Synchronize what?
    - Which parts in a software process can be synchronized (Start, certain parts, anywhere) to an event?
    - How is it done (in-line code, interrupt system, task scheduler, finite state machine, OS)?
  - **Triggering a Process : "Sync and Do"**
    - Processing Stages for "Sync and Do"
      - Sync
        - *Trigger Conversion if needed (external signal, converter not free-running)*
        - *Convert Data if needed (external signal)*
          - *Sample input*
          - *Quantize to digital value*
        - Detect Event
          - Analyze sampled, quantized data
          - Decide if event was detected
        - Schedule "do" work
          - Select processing based on event detection, possibly other factors (priority, etc.)

        - Dispatch it – start it running
    - Do: Execute "do" code in process or interrupt/exception handler, or trigger "do" hardware processing
  - Implementations of processing stages
    - Software. Process/task code, scheduler/OS
    - Hardware: peripherals, interrupt system, peripheral event interconnect, DMA
  - Synchronization and communication between stages in processing chain
- **Safely Sharing Resources among Processes - "Sync and Don't"**
  - Volatile data type modifier: What it does, why to use it, and how to use it
  - Critical sections of code for a shared resource
    - Shareable resources: data variables, hardware peripherals
    - Why mutual exclusion is needed
    - Identify critical sections (and their instructions) in program with multiple processes
    - Understand how certain factors affect which code sequences are critical sections
      - CPU instruction set architecture (load/store?)
      - Software process preemption and prioritization across thread-level code (thread, process, task) and handler-level code (interrupt and exception handlers)
      - Hardware peripherals

❖ **Application Analysis Process**

- Identify inputs, outputs and processes, and their key connections
- Identify key hardware and software stages based on fundamental peripheral features
- Analyze processes for synchronization and communication driven by I/O requirements
- Analyze processes for sync and comm with each other
- Define initial architecture considering most critical/difficult processing stages and interactions.
- Refine to detailed design and implementation. (Functionality first, then performance). Iterate as needed.

❖ **Software Scheduling (Lectures and Theory HW 3)**

- Interrupt System: Peripherals, Interrupt Controller, Interrupt Handlers
- Task Scheduler Evolution from while(1) loop to RTCS (Run To Completion Scheduler):
  - Code generalization, modularization and abstraction
  - Scheduler interface for data protection
  - Task prioritization
  - Timer tick and periodic tasks
- Evaluate impact of different process scheduling approaches on responsiveness in event-triggered system built on interrupt system and software non-preemptive task scheduling
  - Static task ordering
  - Event detection with polling or interrupts
  - Dynamic task ordering by priority
  - Processing work in task or interrupt handler

- **ESF** Textbook
  - Chapter 1 – Introduction
    - General concepts, MCU vs. MPU, peripherals
  - Chapter 2 – General-Purpose Input/Output
    - How to read digital inputs, how to write digital outputs
      - PORT peripheral, pin multiplexing concepts
      - GPIO peripheral configuration and use, especially control registers **PDDR, PDIR, PDOR, PSOR, PCOR, PTOR**
    - Chapter 3 – Basics of Software Concurrency
      - Software scheduling approaches
      - Turning code into finite state machines
      - Why peripherals generate interrupts (e.g. THW2)
      - Polling hardware peripherals for status (e.g. timer)
      - Port interrupt concepts (e.g. for input switches)
    - Chapter 4 – ARM Cortex-M0+ Processor Core and Interrupts
      - High-level understanding of CPU and instruction set architecture, especially:
        - Memory access is only load/store
        - Processor normally operates in thread mode, switches temporarily to handler mode when executing interrupt/exception handlers
      - Interrupt system, especially:
        - Concepts of operation (pause thread, save some context, execute handler, restore context, resume thread)
      - Process synchronization and scheduling
        - Role of peripherals, interrupt system and CPU
    - *Chapter 5 – C in Assembly Language – not covered in midterm*
    - Chapter 6 – Analog Interfacing
      - Understanding sampling and quantization,
        - Use a transfer function to convert a voltage to/from its quantized (digital) value
      - Understanding how analog comparator, DAC and ADC work, and their relative conversion speeds
      - Understand concepts in peripheral code examples
    - Chapter 7 – Timers
      - Only to extent covered in lectures and homework
        - Measure elapsed time
        - Generate periodic interrupts
    - Chapter 8 – Serial Communications
      - Only to extent covered in lectures and homework
        - Asynchronous serial communication
          - Data format and timing
          - UART peripheral
    - Chapter 9 – Direct Memory Access
      - Only to extent covered in lectures and homework
        - With each trigger event, copy data item(s) in memory from source to destination, potentially incrementing addresses
        - Triggerable by peripherals