NC STATE UNIVERSITY

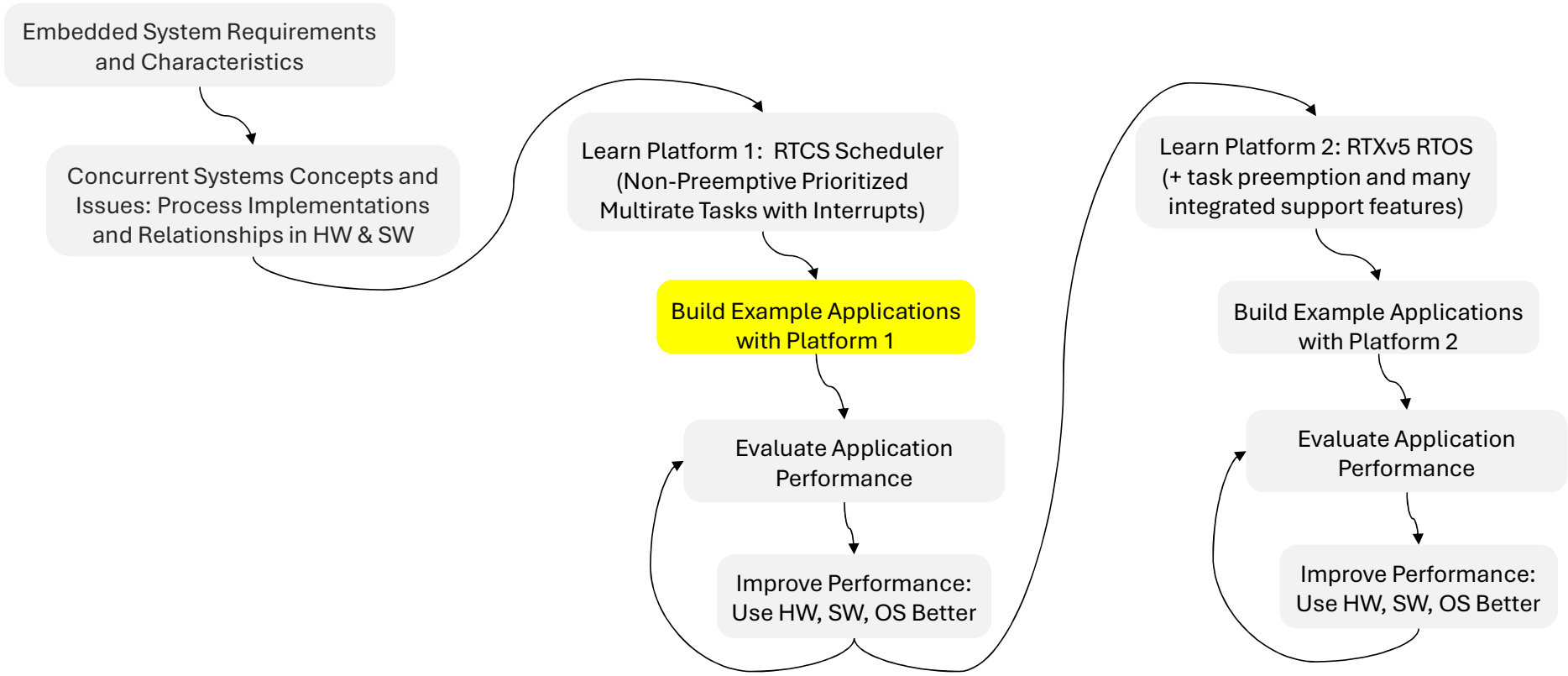
APPLICATION DESIGN PROCESS USING PLATFORM 1 (RTC SCHEDULER WITH INTERRUPTS)

V1

9/17/2025

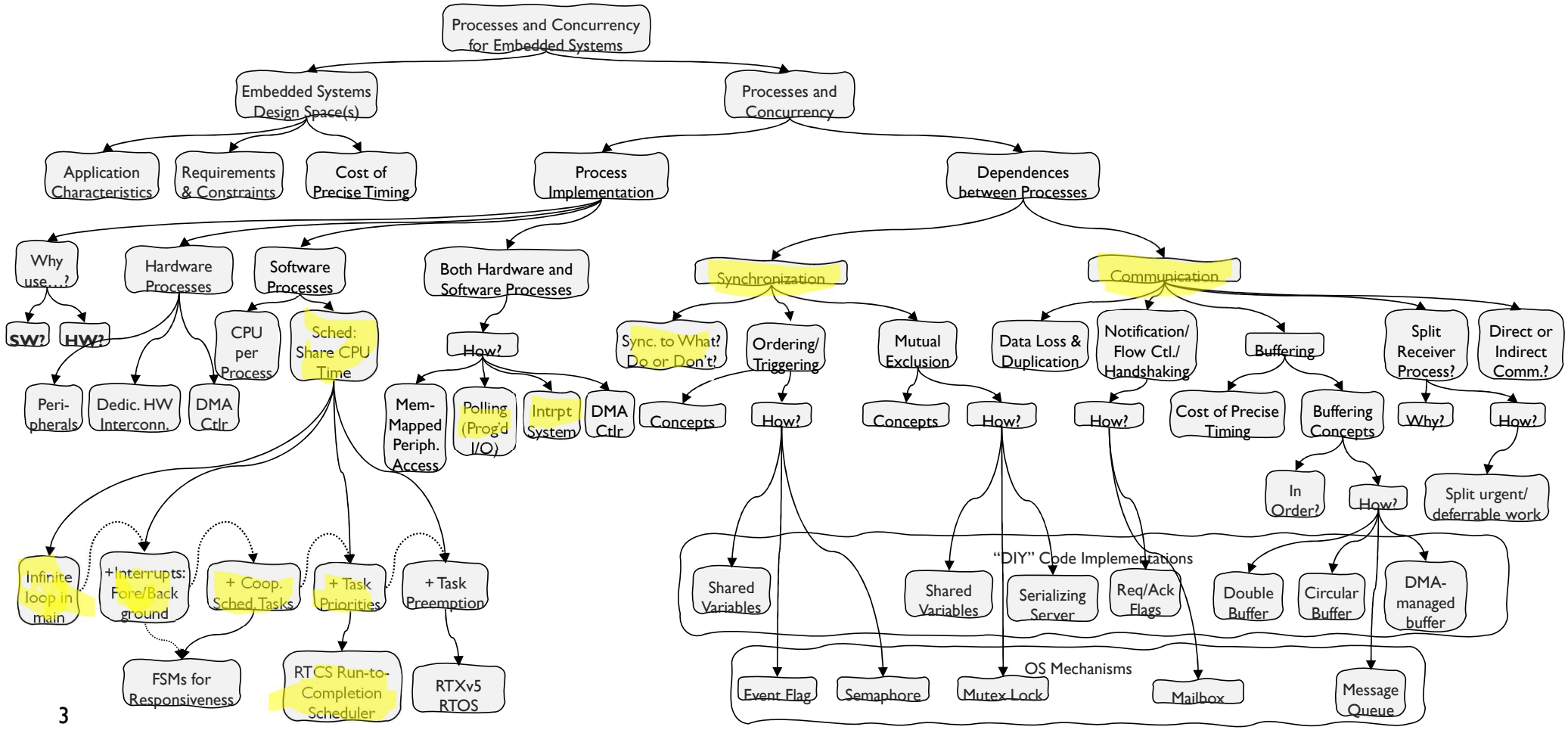
NC STATE UNIVERSITY

Where are we in the class?



NC STATE UNIVERSITY

Extended Topic Map: Class 09



EXAMPLE APPLICATIONS: FIRST VERSIONS WITH RTC SCHEDULER AND INTERRUPTS

NC STATE UNIVERSITY

Applications: Functionality First, then Performance

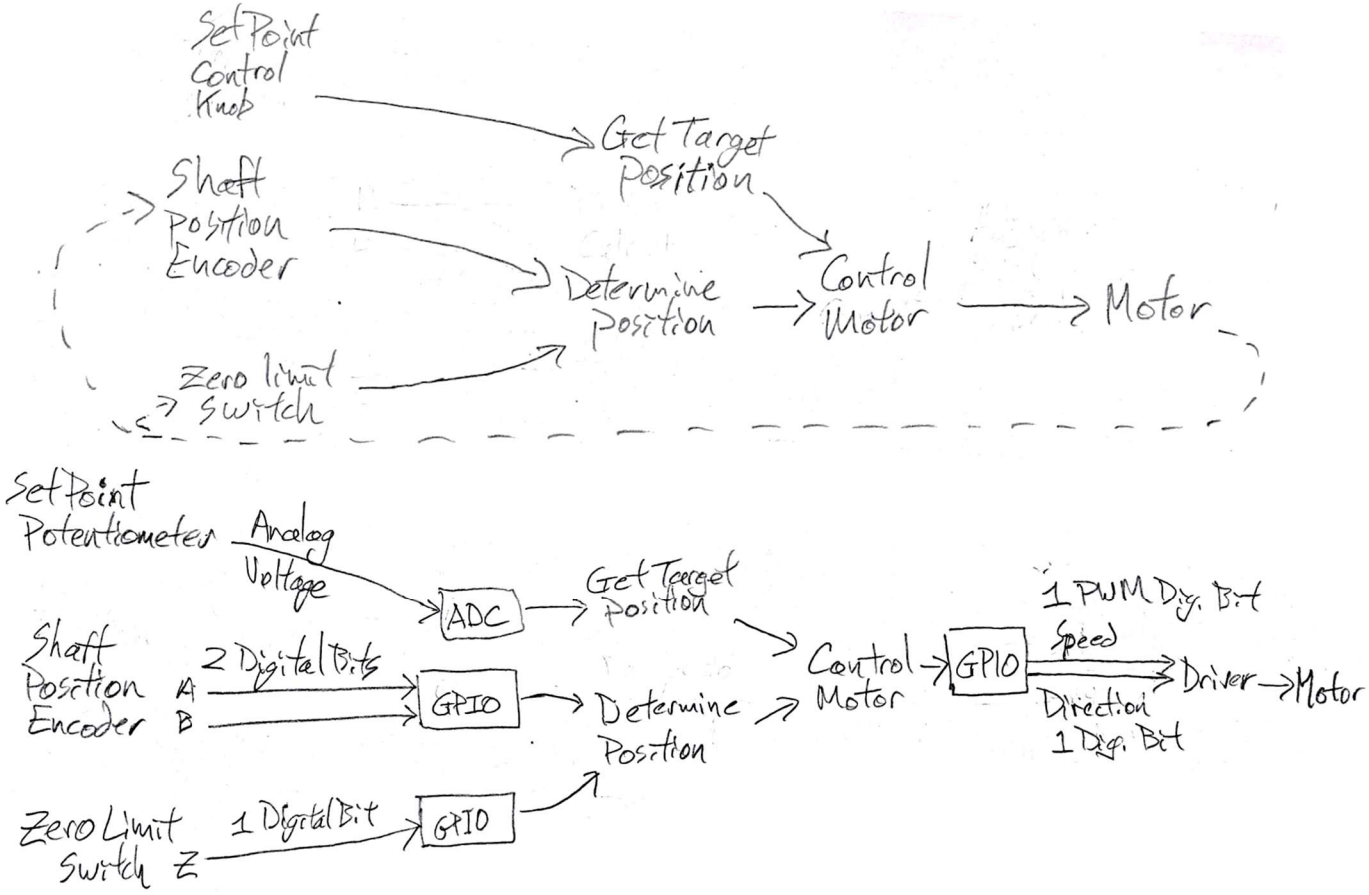
			Blinky Control Panel	Quad. Dec. w/Z Limit Switch	Waveform Generator	LCD Controller	Touch screen	Scope	Serial Comms.	I^2C Comms.	µSD via SPI Comms.	SMPS Controller
Providing Functionality	Interfacing	Simple Digital	In, Out	In			Out					PWM Out
		Complex Digital	PWM			Bus Out			In, Out	In, Out	In, Out	
		Analog	ADC In, CMP In, DAC Out		Out		ADC In	ADC In, Cmp In				ADC In with Sync. Sampling
	Async	# Processes for async. exec.	4	1,2	1,2	0	0, 1	1,2	2	1	2	1
	Sync and …	Sync and Do: Coarse Triggering		Digital Edge Detection	Periodic Output Updates			Ana. Edge Det., Periodic In. Smplg., Buffer mgt.	Tx Rdy, Rx Done events. Producer & Consumer	Data Producers & Consumers. I2C Device read response.	Tx Rdy, Rx Done events. Prod. & Cons.	ADC In with Sync. Sampling
		Internal, Fine Grain Block/Sched/Trig	ADC conv. time				ADC conv. time			I2C message internal events & timing reqts. for conditions, data		
		Sync and Don't: Sharing & Races						LCD Ctlr Sharing, Data buffer mgt.	Tx, Rx byte queues	Tx, Rx Msgs	Tx, Rx byte queues	
	IPC	Inter-Process Comm.		Shared Position Variable				Data buffer				
Meeting Perf. Reqts.		Timing Stability	1		1			2				1
		Responsiveness	1					1			1	1
		Reducing SW Overhead			2	1- Perf. Optimiz.		2		1 – series of timed I/O events per message. FSM vs. RTOS		2
		Tolerating Timing Mismatches			2			2	2		2	

Application Design Overview

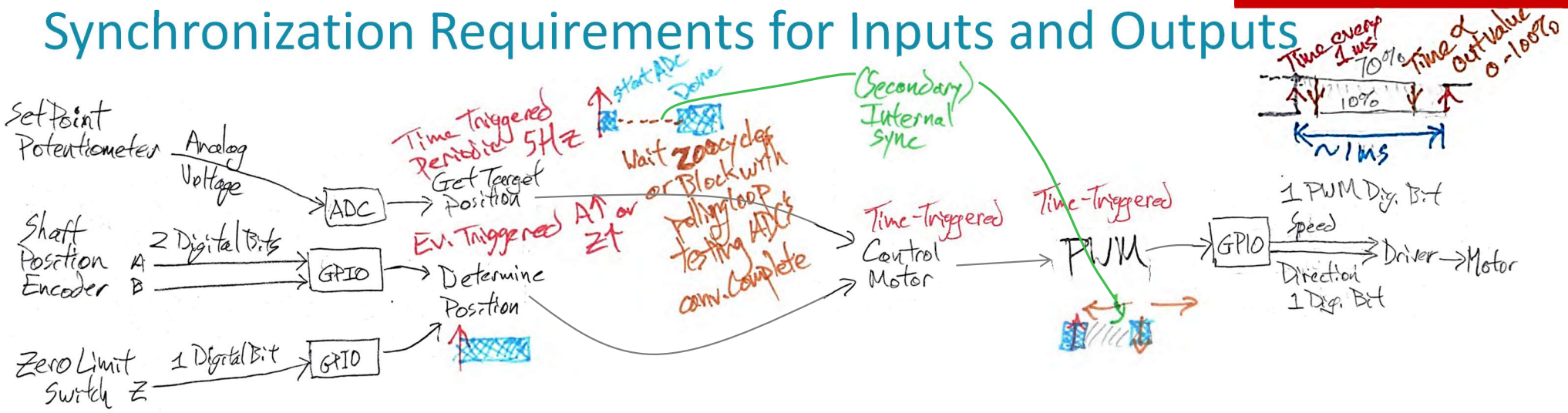
- *Note: Provide **just enough detail**, but not too much. Later design stages and iteration will address details as needed.*
- Identify system's **inputs, outputs** and **processes**, and their key connections
- Identify key **hardware, software stages** in each process
  - Initially consider only peripheral's fundamental features (examine enhancements later as needed)
- Analyze processes for **synchronization** and **communication** driven by **I/O requirements**
  - May be ***event-triggered*** or ***time-triggered***
  - What triggers/releases/allows process (e.g. thread's work code) to start execution: ***event*** or ***time?***
    - Do A once for every time E1 happens
    - Do B every 100 ms
  - Is there any ***internal*** synchronization (*wait until*)? Again, may be triggered by event or time
    - Do X, wait at least 3.2 ms but no more than 3.8 ms, do Y, Wait for event E, do Z.
- Analyze **process interactions** for **sync.** and **comm.**
  - Identify **explicit** sync. and comm. **between** processes
    - Triggering, shared data and resources (one-way data flows, other data flows)
  - Identify **implicit sync**. required by any **communication**
    - Notification of new data? Buffer old data? How much? how to manage buffer? How to handle overrun condition?
- Define **architecture** (high-level design)
  - Select how to implement most critical/difficult processing chain steps and interactions
  - Draw from toolbox of methods
    - Hardware: peripherals, DMA, programmable logic
    - Software: Program structure, algorithms
    - Both: Interrupts, task scheduler, support features from OS
- Continue with detailed design and implementation
  - Functionality first, then performance
  - Iterate design to improve performance

Identify System Fundamentals

- Identify system's **input** and **output devices** and **signals** and **processes**, and their **key connections**
  - Information for key signals
    - Format: analog? digital? comm. protocol? bit width? …
    - Challenging timing and performance requirements
  - Note: Feedback from output (motor) to some inputs
- Identify key **hardware** and **software stages** in each process
  - Initially consider only peripheral's fundamental features (later will examine enhancements as needed)

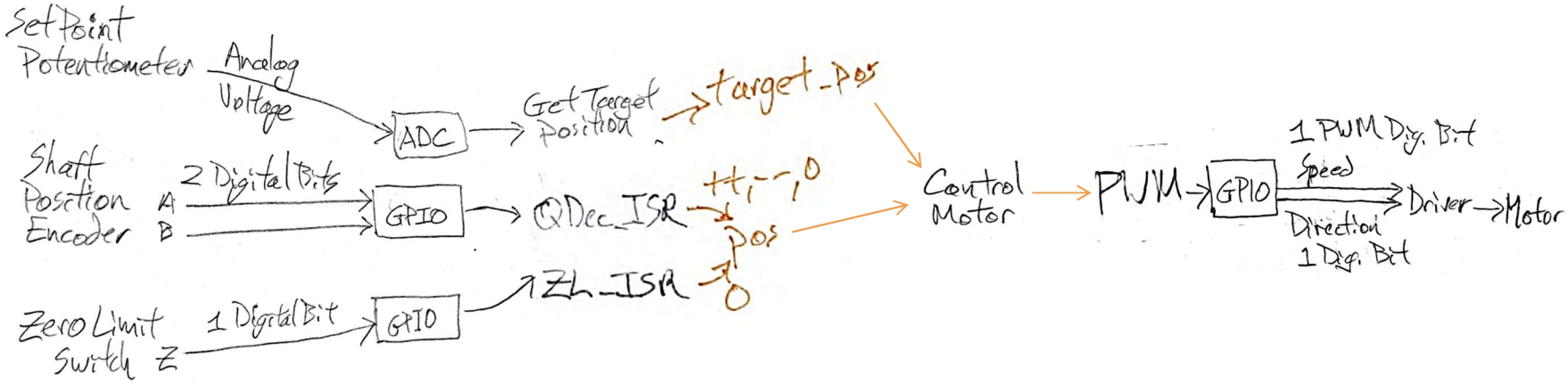


Synchronization Requirements for Inputs and Outputs

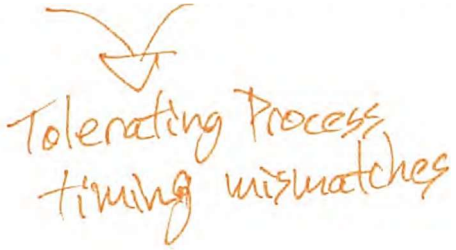


- Analysis may drive early decisions: using interrupts, HW peripheral features, etc. Covered in **Define Architecture** soon.
- First synchronization: What triggers/releases/allows process (e.g. thread's work code) to start execution? ***Event or time?***
  - Get Target Position: Time-triggered, periodic: run at least 5 times /second (at most every 200 ms)
  - Detect Position: Event-triggered: run on rising edge of A or rising edge of Z.
    - Could also be time-triggered if polled frequently enough
  - Control Motor: Time-triggered, periodic: run every 1 ms to set Speed to 1 (1 kHz PWM output signal) and update Direction
- Is there any more synchronization within the process after the first sync.? ***Event or time?***
  - Control Motor: Time-triggered, delay. Clear Speed to 0 after 0 to 1 ms, depending on drive effort requested

Sync. and Comm. from Key Interactions between Processes



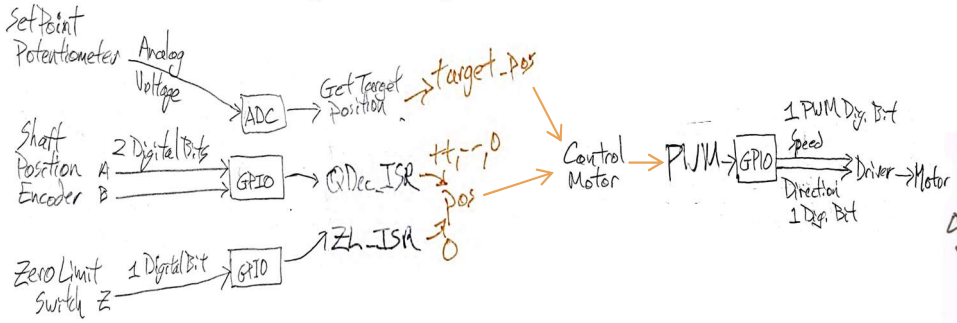
- Identify and describe ***synchronization*** and ***communication*** **between** processes
  - Additional triggering, shared data and resources (one-way data flows, other data flows, …)
- Identify implicit ***sync.*** required within the ***communication***.
  - Notification of new data? Handle over-run how? Buffer old data? How much? How to manage buffer? etc.



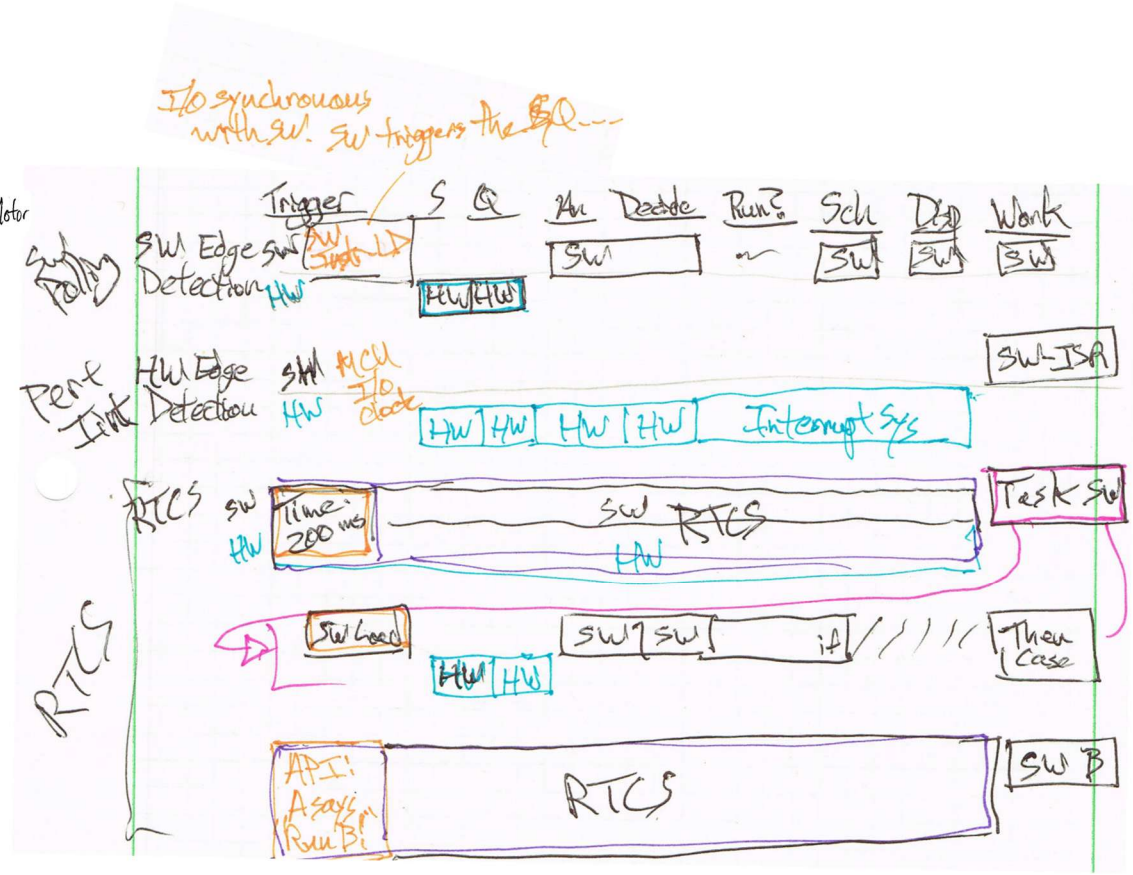
NC STATE UNIVERSITY

Define Architecture (High-Level Design)

- We have a high-level plan for **what to do**



- Next, refine that plan to decide **how to do it**
  - Decide **how** key stages in key processing chains will be performed
  - Pick from toolbox of methods
    - Hardware, software, scheduler, OS, and combinations
  - Note that a stage in the processing chain may contain **another** processing chain



NC STATE UNIVERSITY

Example Applications and Subsystems

- Blinky Control Panel
- Quadrature Decoder w/Limit Switch
- Waveform Generator
- LCD Controller
- Touch screen
- Scope
- Communications
  - Basic: Serial Comms., SPI
  - Higher protocol layers: I^2C, Secure Digital via SPI
- SMPS Controller