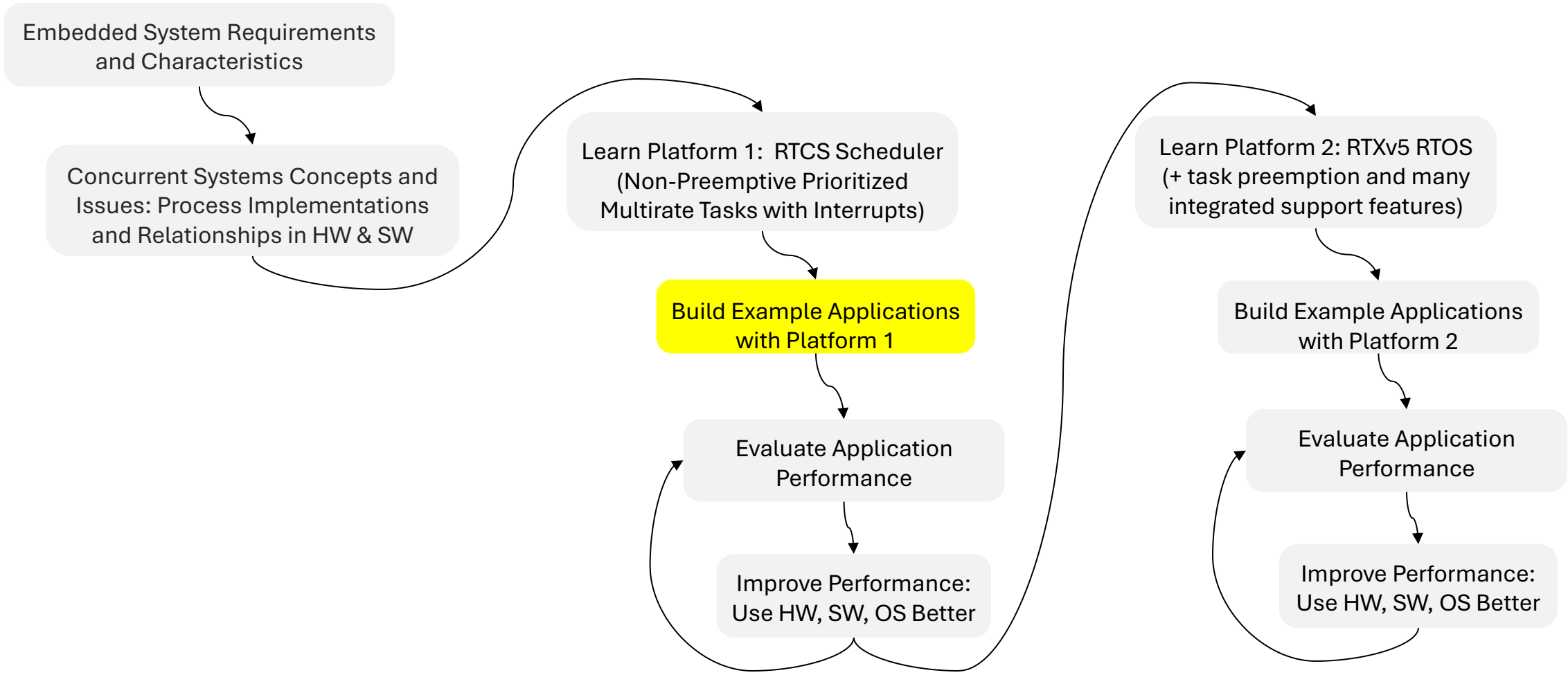
APPLICATION DESIGN PROCESS USING PLATFORM 1 (RTC SCHEDULER WITH INTERRUPTS)

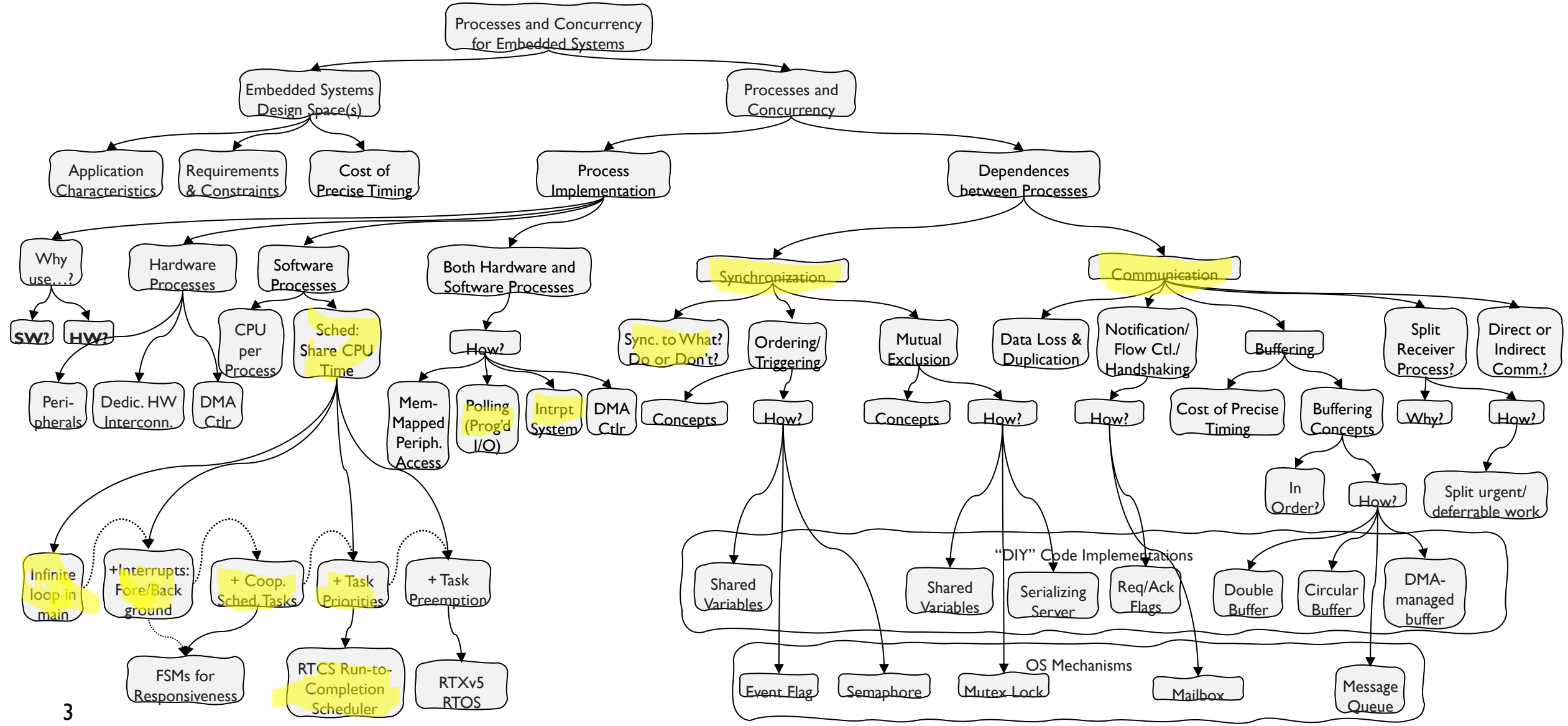
V2

9/18/2025

Where are we in the class?



Extended Topic Map: Class 09



EXAMPLE APPLICATIONS: FIRST VERSIONS WITH RTC SCHEDULER AND INTERRUPTS

Applications: Functionality First, then Performance

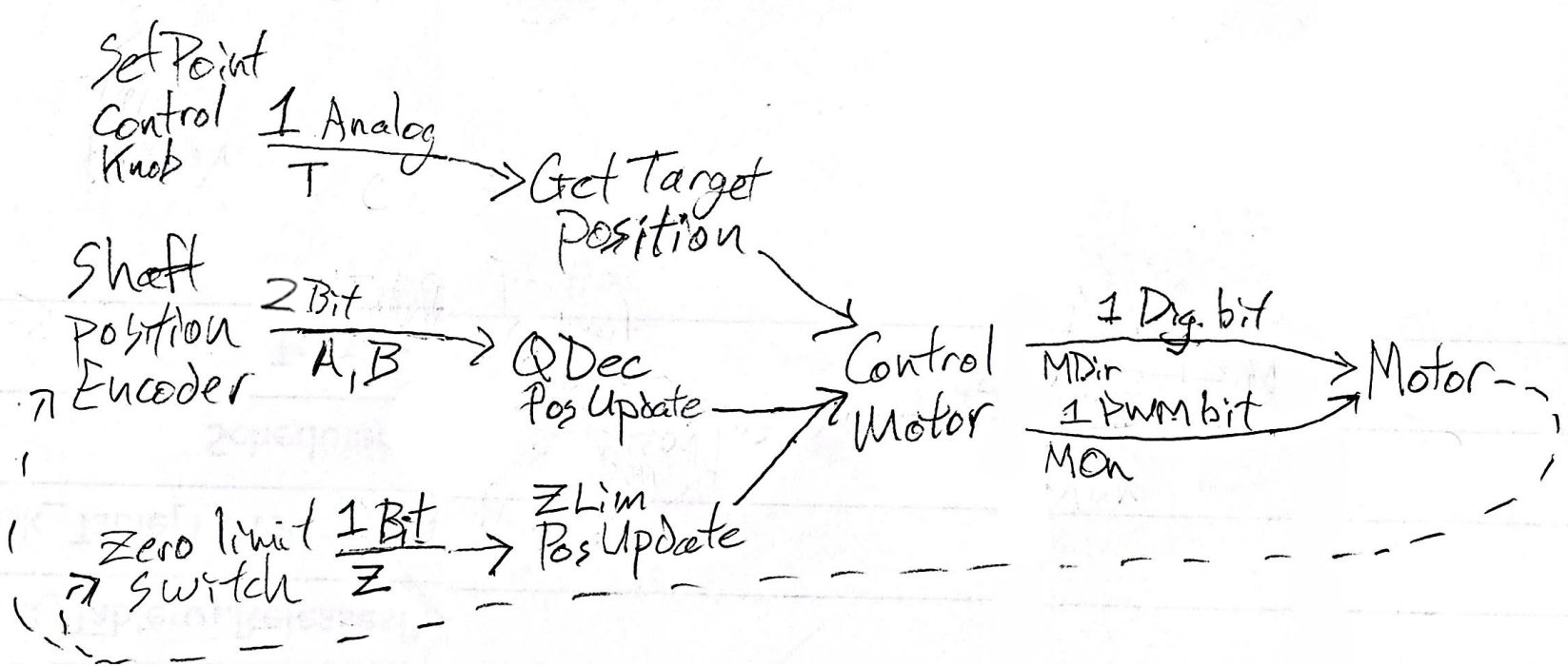
			Blinky Control Panel	Quad. Dec. w/Z Limit Switch	Waveform Generator	LCD Controller	Touch screen	Scope	Serial Comms.	I^2C Comms.	µSD via SPI Comms.	SMPS Controller
Providing Functionality	Interfacing	Simple Digital	In, Out	In			Out					PWM Out
		Complex Digital	PWM			Bus Out			In, Out	In, Out	In, Out	
		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

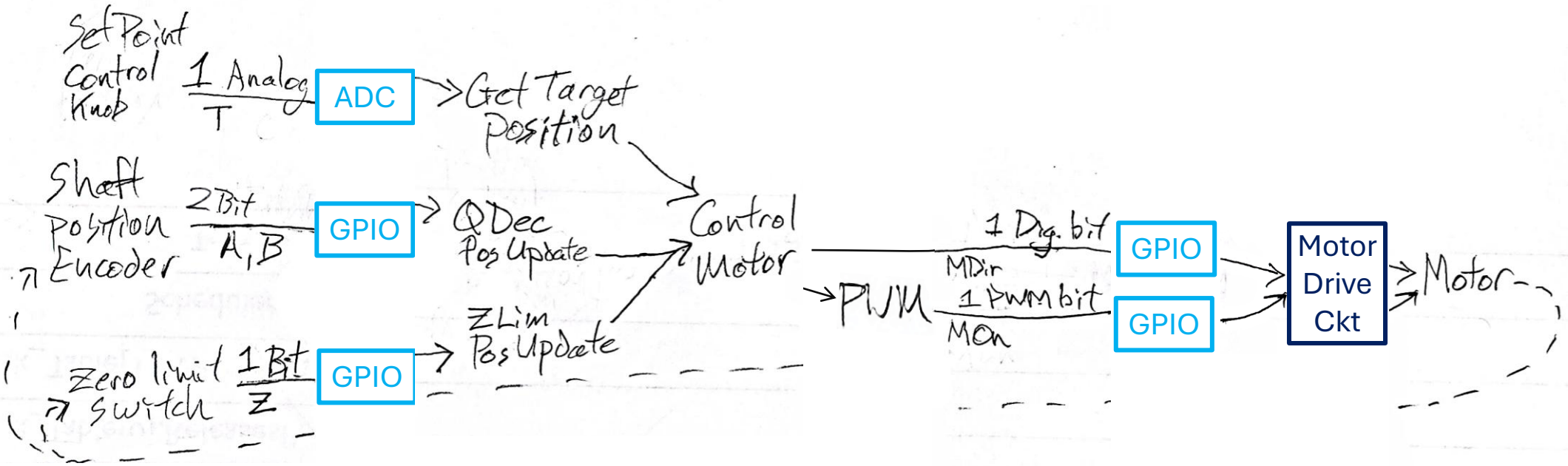
- *Note: Provide **just enough detail**, but not too much. Later design stages and iteration will address details as needed.*
- Identify system's **inputs, outputs** and **processes**, and their key connections
- Identify key **hardware, software stages** in each process
  - Initially consider only peripheral's 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 I/O Signals & Devices, Processes, Major Connections

- Identify system's **input** and **output devices** and **signals** and **processes**, and their **key connections**
  - Note feedback from output (motor) to some inputs
  - Information for key signals
    - Name
    - Format: analog? digital? comm. protocol? bit width? …
    - Does it have challenging timing or performance requirements?



Identify Hardware and Software Stages

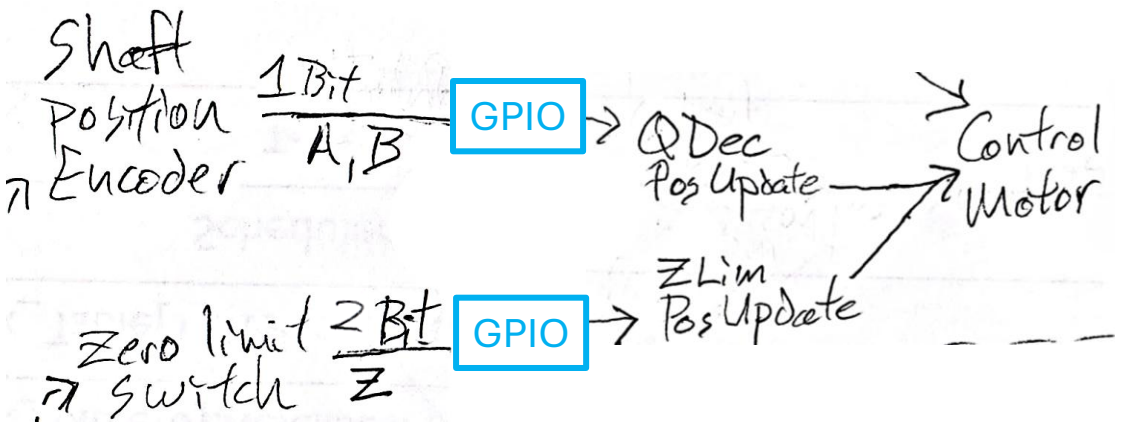


- Identify key **hardware** and **software stages** in each process
  - Add hardware peripherals known to be needed
    - Conversion from/to analog
    - Generating or processing complex signals
  - Initially consider only peripheral's fundamental features (later will examine enhancements as needed)

Examine Sync. Requirements from Inputs and Outputs

Tough to do with general-purpose computers – need hardware processes and good sync/comm support

- Analysis may identify challenges early, steer design
  - Use interrupts, HW peripheral features, etc? Covered soon in **Define Architecture** stage.
- Synchronization events (triggers) may be based on **time** or **events**
  - Can convert event trigger to time trigger if sampled often enough.
  - Frequency depends on minimum event duration (pulse width) and maximum detection latency allowed
- Synchronization: What **allows** scheduler to run process (e.g. code to do work): ***Event*** or ***time?***
- Start with shaft position sensing
  - Quadrature Decoder
  - Handle Zero Limit Switch
- Consequences of timing errors?



Process	Urgency	Synchronization	
		Trigger	Processing
QDecPosUpdate	High	Event: A ↑	Read Z, B. Inc/Dec/Zero pos variable
ZLimPosUpdate	High	Event: Z ↑	Zero pos variable

Examine Sync. Requirements from Inputs and Outputs

Process	Urgency	Primary Synchronization		Additional Synchronization	
		Trigger	Processing	Trigger	Processing
GetTarget Position	Low	Time, periodic (100 ms)	Start ADC Conversion	Event: Conversion Complete	Read result, scale, update target_position

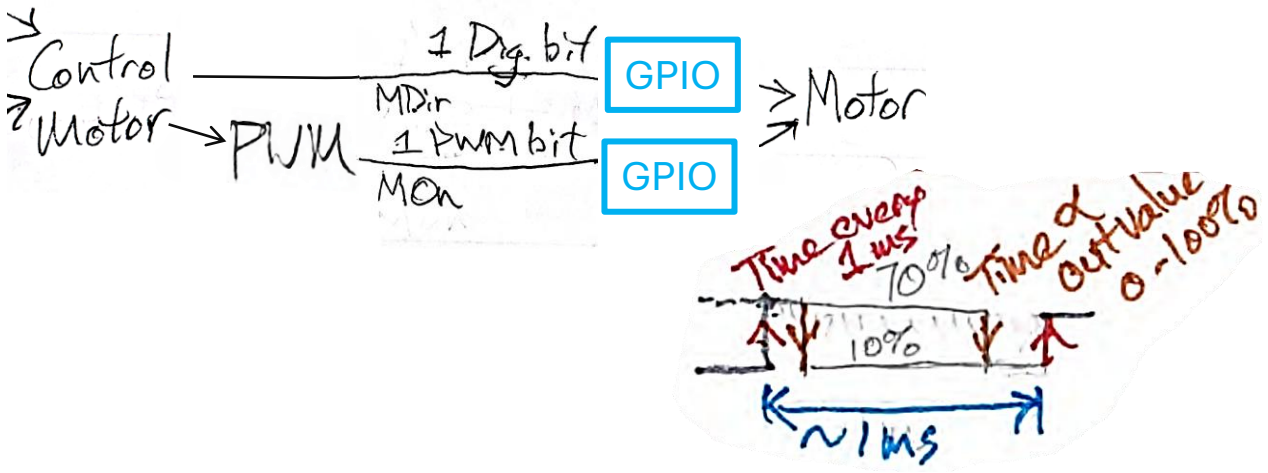
- What triggers **GetTargetPosition**?
  - Time-triggered, periodic: run at least at 10 Hz to make it responsive for operator (every 100 ms)
- **GetTargetPosition** must trigger A/D conversion
  - ADC is shared with other processes
  - Two kinds of ADC: **very slow** (vs. CPU), or **expensive** (as fast as CPU)
  - Our ADC's 3 us conversion takes multiple CPU clock cycles (144 CPU clock cycles in our 48 MHz MCU)
- More synchronization: code needs to **wait** for end of A/D conversion. How?
  - Blindly wait in a loop for a fixed amount of time?
  - ADC provides "conversion complete" flag (COCO). Loop code can poll COCO, repeat (blocking rest of this code) until done.
    - 144 << 48,000,000 per second. Many applications can tolerate this waste.
  - COCO flag can request interrupt, trigger ISR
    - Put rest of work from that process in the ISR? Now need to support **even more** synchronization and communication.


```
GetTargetPosition() {
  Configure A/D converter
  Start A/D conversion
  Wait for end of conversion
  Read conversion result
  Convert result to target position
}
```

Examine Sync. Requirements from Inputs and Outputs

- Use PWM Motor On (MOn) signal: pulse-width modulated
  - Motor runs more smoothly.
    - Switches between accelerating, coasting each period.
    - Make period short compared with inertial time constant for motor
  - Power electronics now are simple, low-cost, efficient
  - Doesn't generate big bursts of electromagnetic noise
- What triggers GeneratePWM?
  - Time-triggered. Needs to run periodically (e.g. every 1 ms)
  - Set MOn to 1
- Is there any more sync. within the process?
  - Yes. Time-triggered, one-shot delay.
  - Clear MOn to 0 after on_time delay (e.g. 0 to 1 ms)
- Consequences of timing errors?
- Feasibility?
  - *Possible* to implement in software but not scalable: limited timing precision and processor sharing
  - Most MCUs include timer peripherals which can generate PWM output signals, since useful across so many applications

Process	Urgency	Primary Synchronization		Additional Synchronization(s)	
		Trigger	Processing	Trigger	Processing
Generate PWM	High	Periodic: 1 ms	Raise MOn signal to 1	Time Delay: on_time	Lower MOn to 0



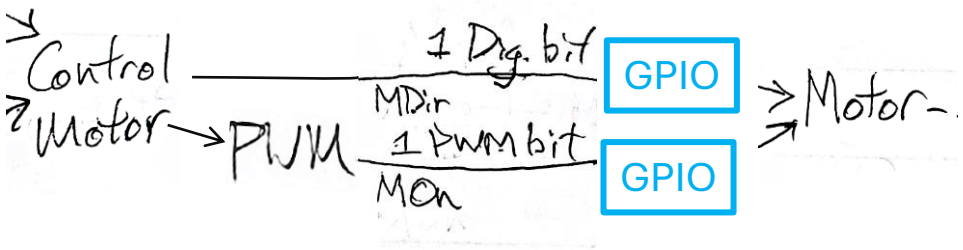

```
GeneratePWM() {
  MOn = 1
  Wait for on_time
  MOn = 0
}
```

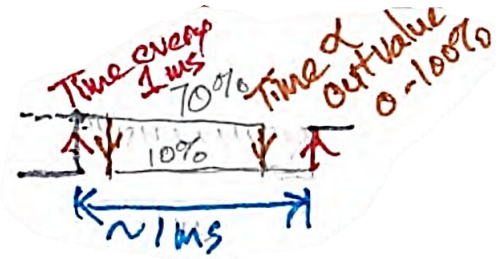
Examine Sync. Requirements from Inputs and Outputs

```
ControlMotor() {
  error = target_pos - pos
  effort = calculated of error
  on_time = absval(effort) * period
  MDir = sign of effort
}
```

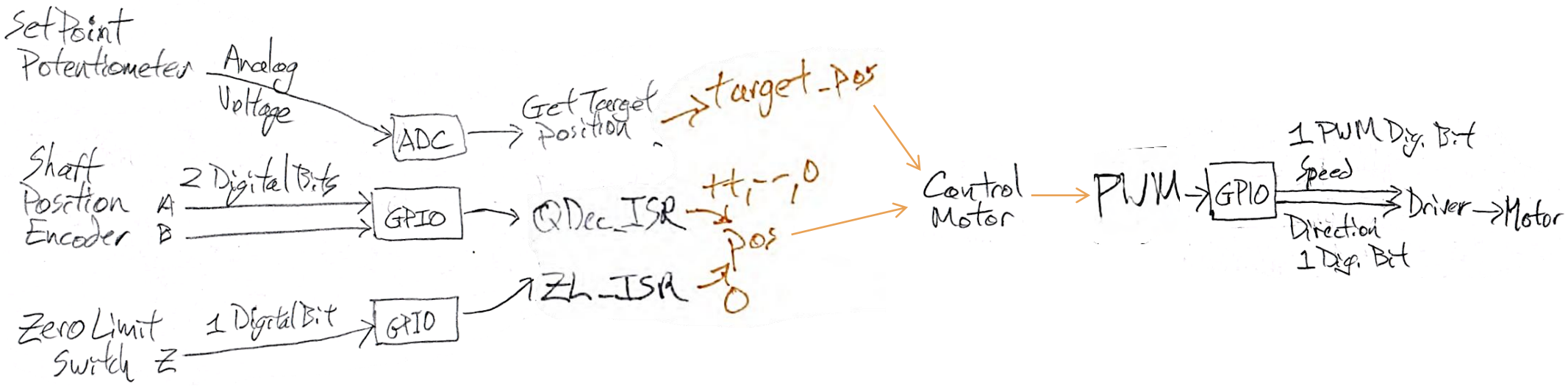
- What triggers ControlMotor?
  - Run periodically to simplify design and stability/response analysis of controller
    - Alternative: Whenever new data is received
  - How often? Every 1 ms?
    - Does it make sense to run faster than PWM frequency?
    - How slow can we go?
- Consequences of timing errors?
  - Control loop stability

Process	Urgency	Synchronization	
		Trigger	Processing
Control Motor	Moderate	Periodic: 1 kHz? Simplify Control Theory	Measure position error, calculate new motor effort and direction. Update MDir.

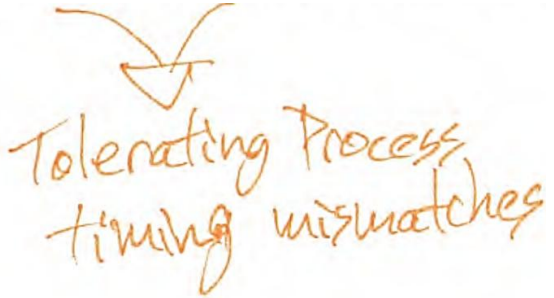




Sync. and Comm. from Key Interactions between Processes

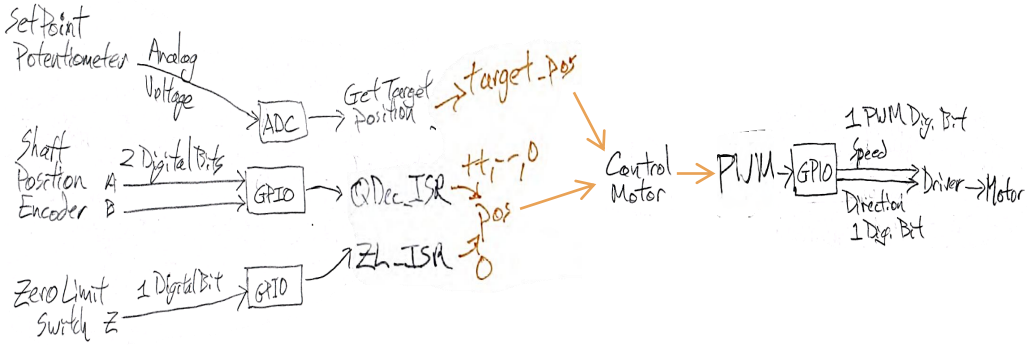


- Identify and describe ***synchronization*** and ***communication*** **between** processes
  - Additional triggering, shared data and resources (one-way data flows, other data flows, …)
- Identify implicit ***sync.*** required within the ***communication***.
  - Notification of new data? Handle over-run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

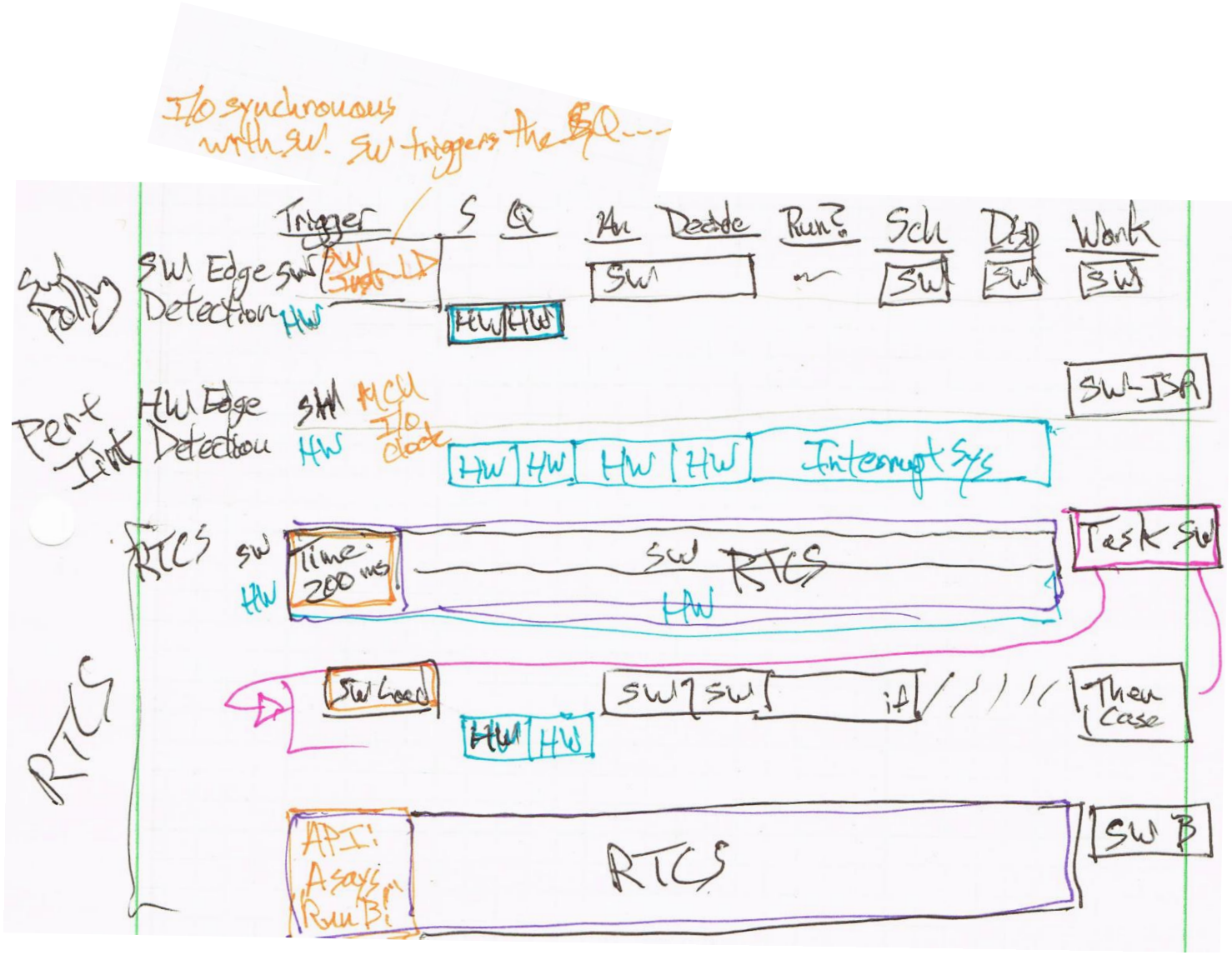


Define Architecture (High-Level Design)

- 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



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

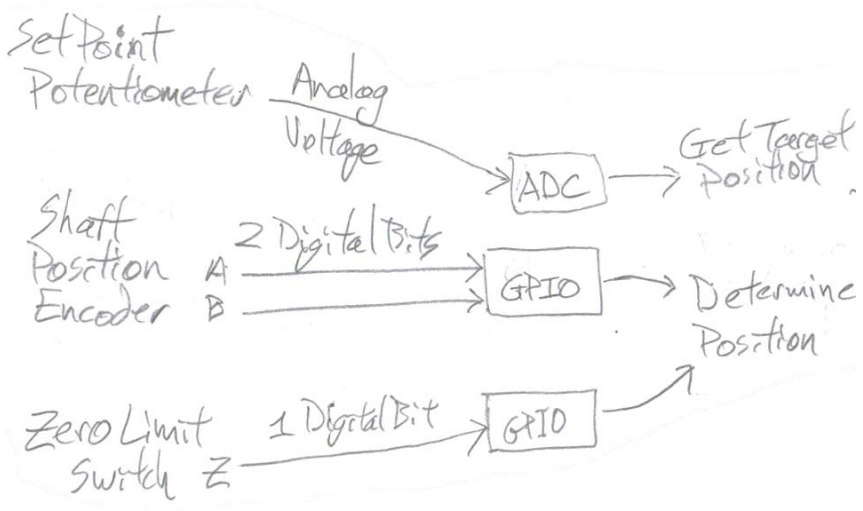
SLIDES FOR LATER

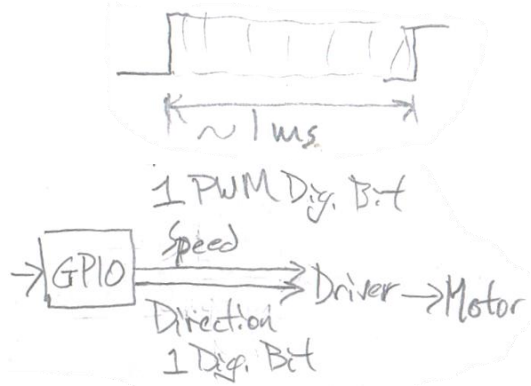
Processing Chain Stage Options

Process Analysis Template

	Input Devices and Signals			Processes			Output Devices and Signals
		Hardware Stages		Software Stages		Hardware Stages	
Add I/O-driven Sync and Comm							
Add inter-process-driven Sync and Comm							

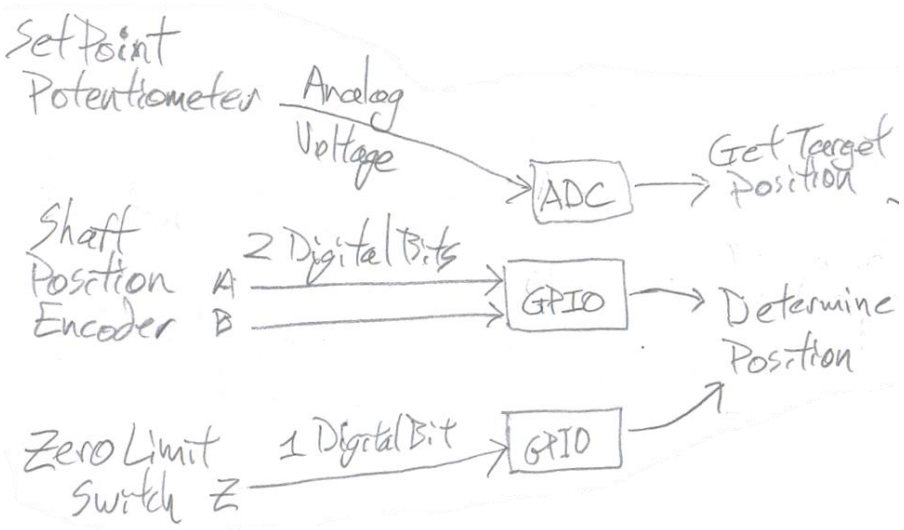
Synchronization Requirements for Inputs and Outputs

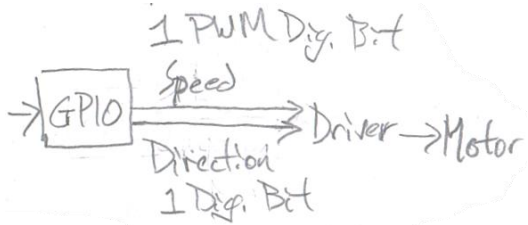




- Analysis may drive early decisions: using interrupts, HW peripheral features, etc. Covered in **Define Architecture** soon.
- First synchronization: What triggers/releases/allows process (e.g. thread's work code) to start execution? ***Event or time?***
  - Get Target Position: Time-triggered, periodic: 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.

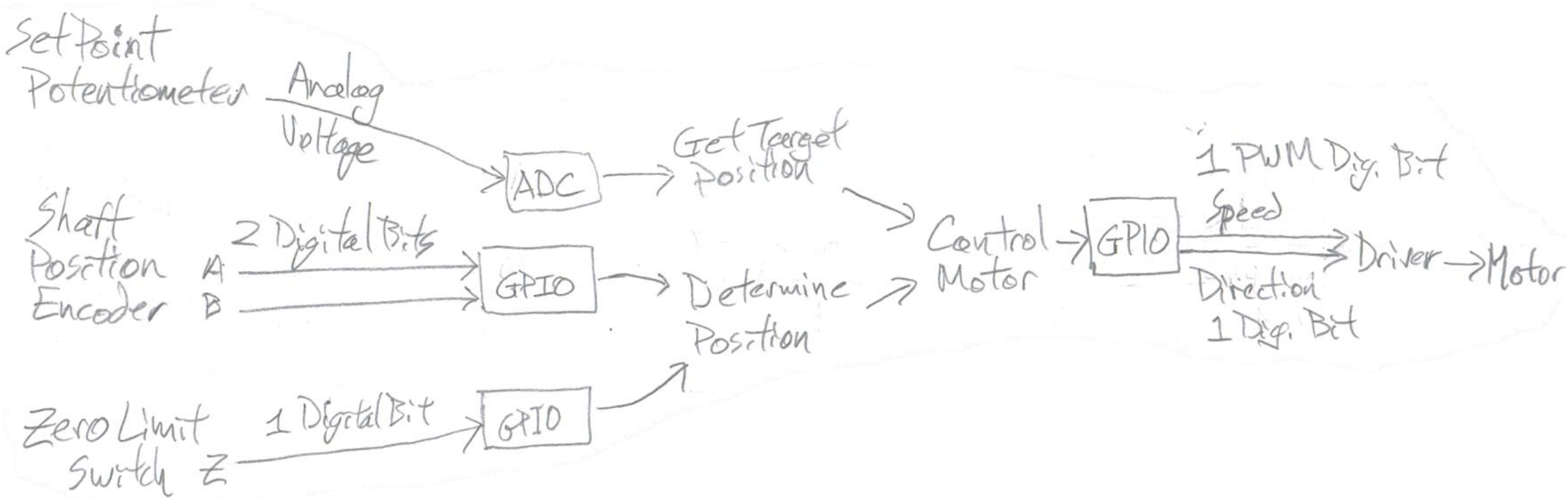
Process Analysis Template

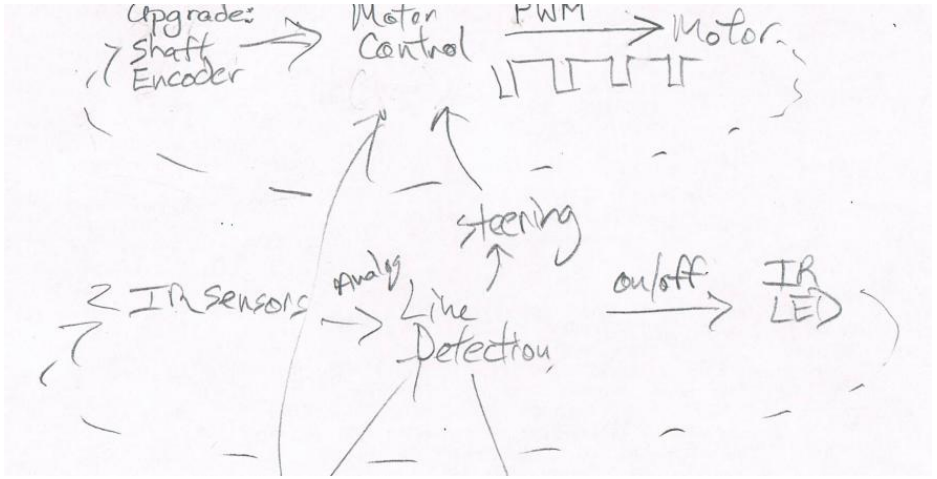
1. Inputs, processes, outputs, key connections
2. Key hardware and software stages in processes
3. Process synchronization and communication
    1. From system inputs and outputs
    2. From interactions between processes (IPC)

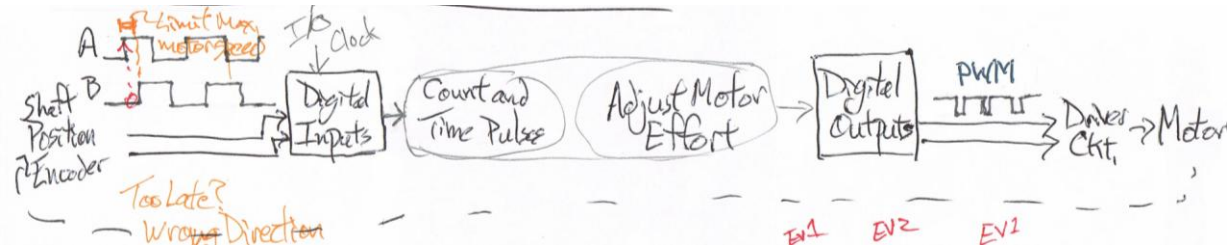
1		Input Devices and Signals			Processes			Output Devices and Signals
2			Hardware Stages		Software Stages		Hardware Stages	
3	Add I/O-driven Sync and Comm							
	Add inter-							

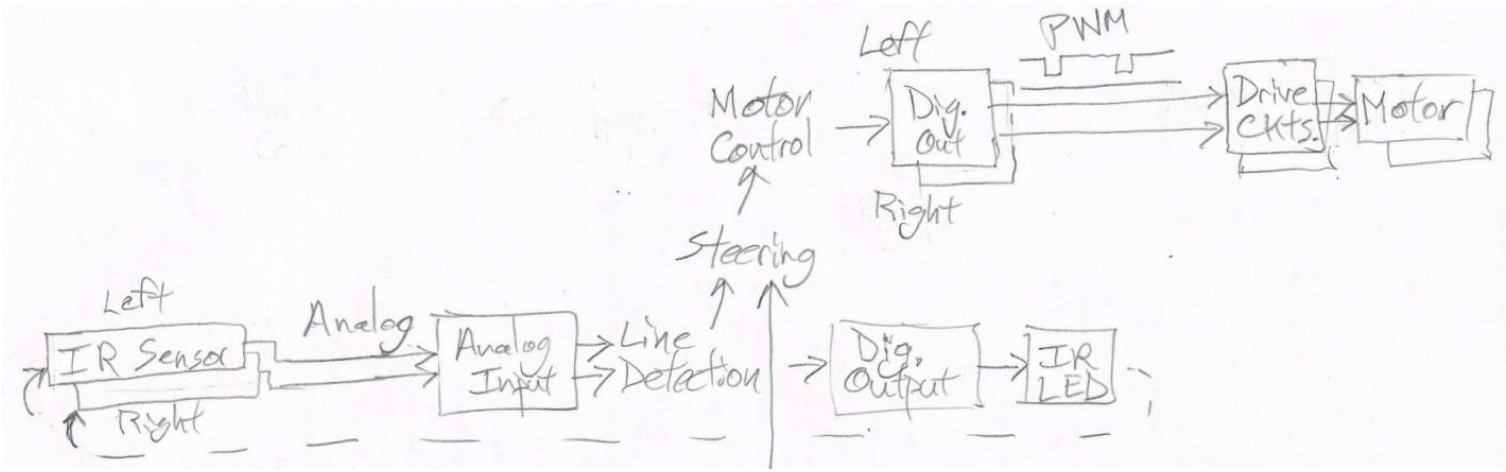
Slide Completion Plan

- #7. Update hand diagrams to match phases 1, 2 (adding PWM generator?), scan, integrate
- #8, #9, #10. Sync for I/O
  - Add pseudocode listings
  - Diagram for timing of control loop and PWM output
- #12: Arch:
  - Add bullet: Is trigger synchronized with software? Sync or async?
  - Synch methods: plot time overhead, fairness
  - Processing chain stage diagram: cleaner, blocks (table?)


Set Point
Potentiometer
Analog
Voltage
ADC
Get Target
Position
Shaft
Position
Encoder
A
B
2 Digital Bits
GPIO
Determine
Position
Control
Motor
GPIO
1 PWM Dig. Bit
Speed
Direction
1 Dig. Bit
Driver
Motor
Zero Limit
Switch Z
1 Digital Bit
GPIO


Upgrade: Shaft Encoder
Motor Control
PWM
Motor
steering
IR Sensors
analog
Line Detection
on/off
IR LED


A
B
Shaft Position Encoder
Digital Inputs
Clock
Count and Time Pulses
Adjust Motor Effort
Digital Outputs
PWM
Driver Ckt.
Motor
Too Late?
Wrong Direction
EV1
EV2
EV1


Left
PWM
Motor Control
Dig. Out
Drive Ckts.
Motor
Right
Steering
Left
IR Sensor
Analog
Analog Input
Line Detection
Dig. Output
IR LED
Right

Scheduler Feature Summary

- To be added…

1. Triggering

- Two basic triggering options
  - Time-triggered: periodic thread runs every X ms. (Could have aperiodic time triggers too)
    - Structure the thread as an infinite loop with an OS ***delay*** call or wait for next interval
  - Event-triggered: Thread can run after event Y happens
    - Use an OS-provided synchronization primitive (event, semaphore etc.) to signal the event has occurred
    - Structure the thread as an infinite loop with an OS ***wait*** call
    - Will need the triggering thread(s) to use the OS ***signal*** call
- Allow multiple pending trigger requests?
  - What if periodic 2 ms thread hasn't gotten to run for 7 ms?
  - What if 5 events have occurred but thread hasn't run yet?

2. Messages

- Does the thread pass data to another thread (one-way, only one writer)?
  - If so, use OS-provided message primitives
- How much information is to be passed?
  - Event: something (implicitly defined) has happened
  - Data + event = message: something has happened, and here are the explicit details
- Does handshaking matter?
  - Does sender care if receiver has gotten the notification?
- Allow multiple pending messages, or just use the last one?
  - How much (if any) information buffering is needed? *Depends on how many events can occur before other thread can service them.*
  - Single item:
    - OS: use event or mailbox
  - Multiple items:
    - OS: use queue

3. Shared Data and Resources

- Do multiple threads need to update shared data or a common resource?
- ***Sharing data*** in a ***preemptive system*** (threads, ISRs) introduces risks
  - If **data updates are not atomic** (are divisible and can be interrupted)
    - Anything which takes multiple instructions to modify (anything in memory!)
    - Multiword variable (long int, float, double)
    - Multiword structure or object
  - If **multiple threads can write** to the same data variable

- Protect the object
  - Reduce or disable preemption manually
  - Use algorithm for protection: access flag, double buffering, etc.
  - Apply architectural solution (e.g. server)
  - Use OS-provided support: mutex

POWER AND ENERGY MANAGEMENT

Automatically Saving Power & Energy when Idle

- Scheduler **knows** if system is idle (no tasks ready to execute)
  - So it can put processor into low-power mode
- Any enabled IRQ will wake up MCU, run ISR
- After ISR, scheduler resumes running (and perhaps sleeping!)

- Special case: multiple tasks may be ready to run
  - Break out of for loop after completing one task in order to restart at top of priority table
  - There may still be other tasks (lower priority) with runs requested
  - Add variable to count how many tasks were run in this while loop iteration, use this to sleep

```
void RTCS_Run_Scheduler(void) {
/* Loop forever */
  while (1) {
    tasks_run = 0
    /* Check each task */
    for (i=0 ; i<RTCS_MAX_TASKS ; i++) {
       if task i is ready
              run task i
              tasks_run++;
              break;
    } // at end of for loop
    if tasks_run == 0
       // go to sleep
       __wfi()
  } // end of while loop
}
```