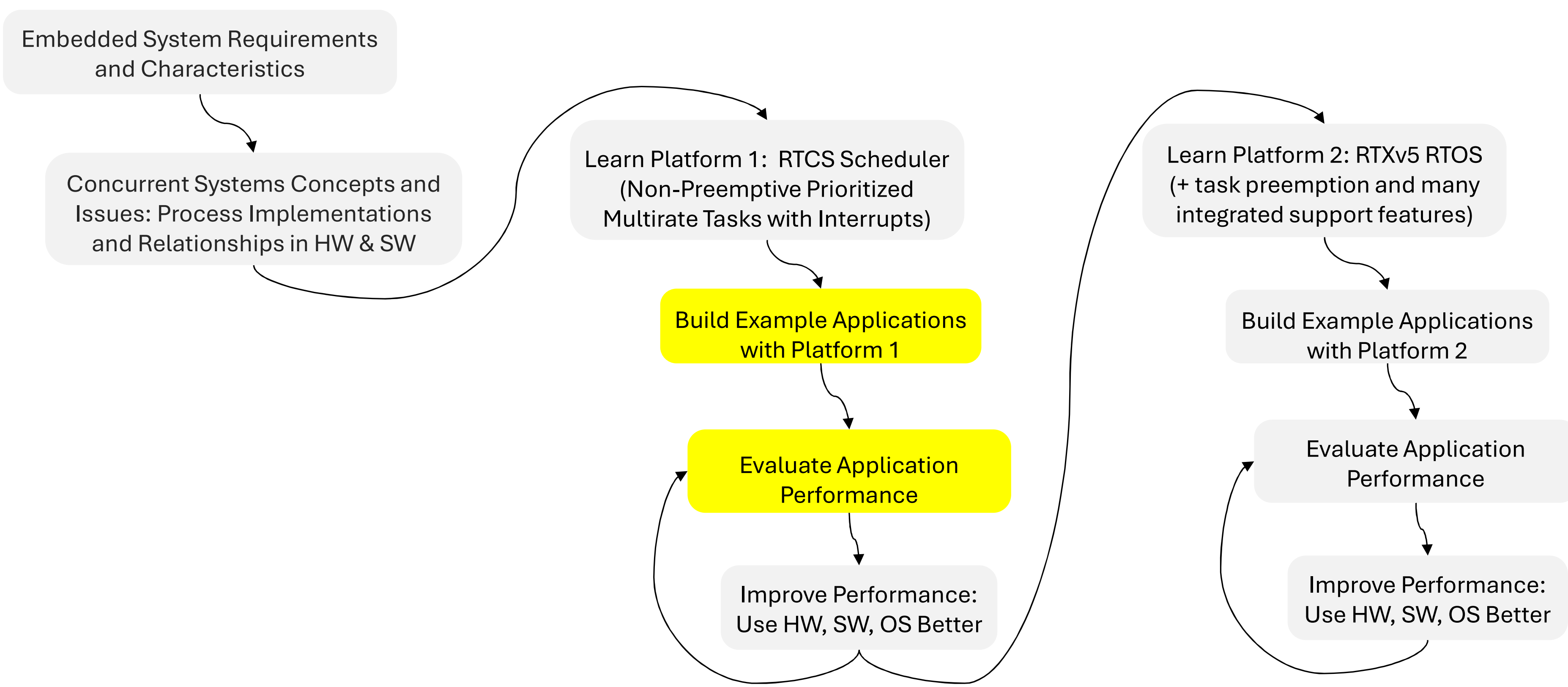
I/O-DRIVEN SYNCHRONIZATION IN EXAMPLE APPLICATIONS

PREVIOUSLY CALLED
"PERFORMANCE ANALYSIS OF EXAMPLE APPLICATIONS ON PLATFORM 1 (RTC SCHEDULER WITH INTERRUPTS, BASIC PERIPHERAL USE)"

V3

9/30/2025

Where are we in the class?

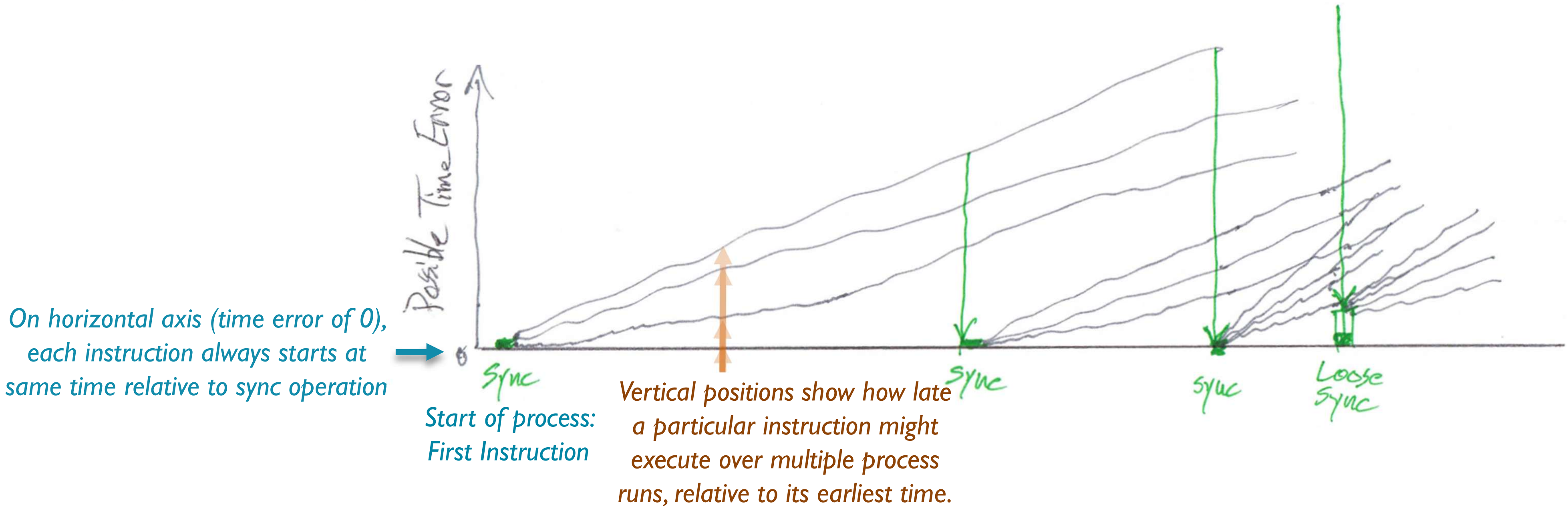


LN12 – I/O-Driven Sync: Concepts, In Example Applications

- I/O-driven synchronization
  - Why, what, where and how
- Examine performance limits of Platform 1
- Summarize I/O-driven sync of first implementations of example applications

PROCESS SYNCHRONIZATION: WHY, WHERE AND HOW?

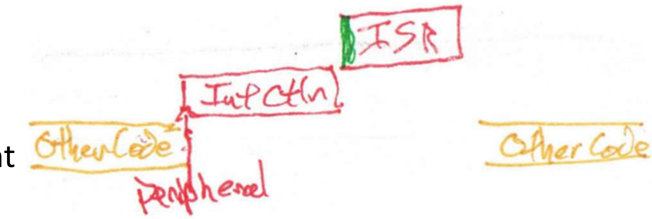
Why Do We Synchronize Processes?



- Sometimes need to sync to external events or timing requirements
  - Timing variability (-> error) accumulates as process executes
  - Synchronization removes accumulated timing error in process since last sync
- Note
  - Synchronizing code C to event E means code C will not run **before** E, but **after** E (if at all).
  - How long after? It depends….

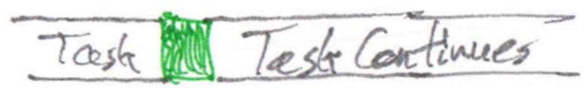
Where and How Can a Process Synchronize?

- At **start** of process
  - Start of interrupt handler
    - **Interrupt system** syncs start of handler (ISR) to **after** trigger event
      - 15 cycles after for Cortex-M0+ CPU

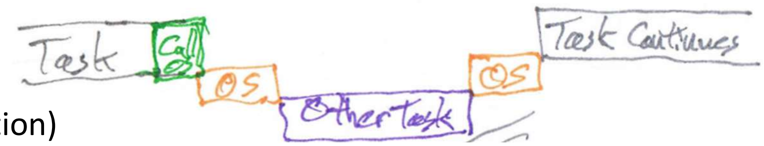


  - Start of task function
    - **Task scheduler** syncs start of task function to **after** trigger event

- **Anywhere within** process
  - **Explicit synchronization code** within task code
    - Test for condition.
      - If true, then continue and execute following code
      - Else repeat test for condition. (Or do something else…)

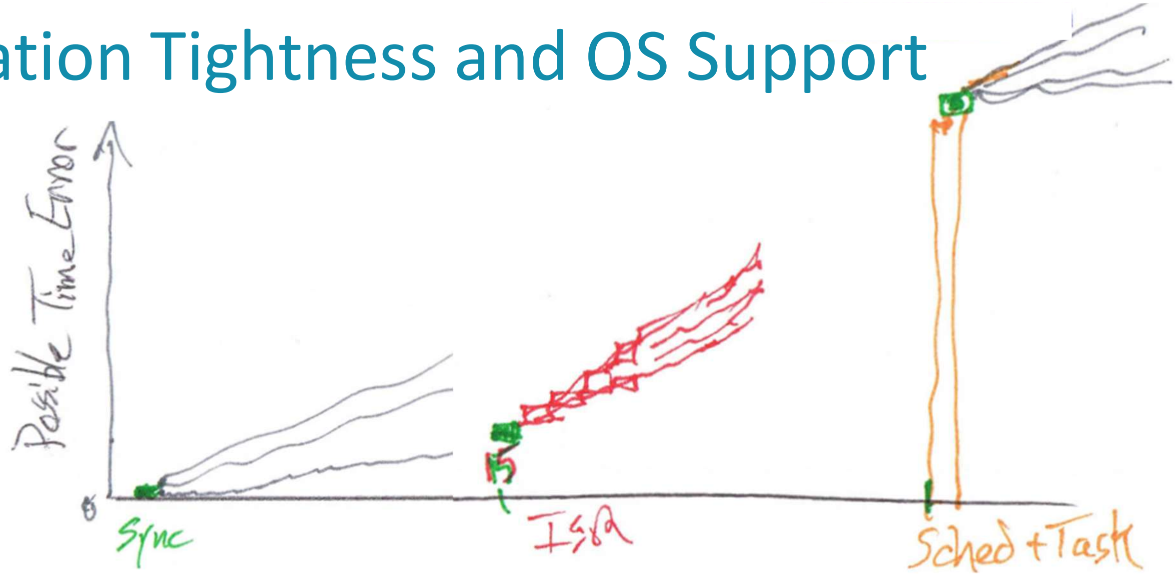


  - **Synchronization call to OS** within task code
    - Available? Depends on task scheduling
      - Probably not supported if non-preemptive (e.g. run-to-completion)
      - Probably supported if preemptive (e.g. RTXv5, FreeRTOS, etc.)



    - How to do it?
      - Task calls an OS sync. function, which doesn't return (it *blocks* the process from continuing) until after sync. event occurs.
      - **Scheduler** synchronizes **resumption** of task to **after** trigger event

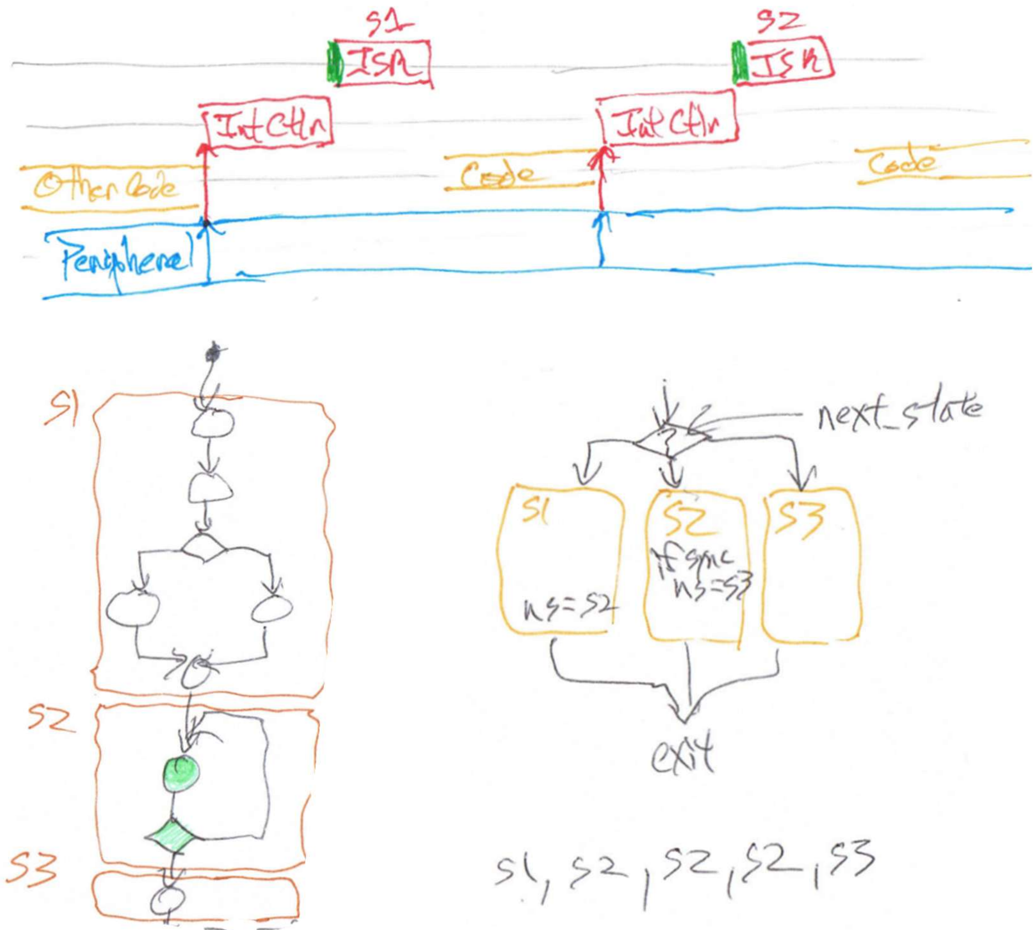
Synchronization Tightness and OS Support



- Tighter synchronization = less time between event and synchronized code starting to run
- Range of tightness for synchronization methods
  - Ideal case: delay of 0
  - ISR has delay of 15 CPU cycles on KL25Z's Cortex-M0+ if…
    - Interrupts are enabled, no higher-priority interrupt/exception handler is executing
  - Polling loop taking $T_{PollTime}$ running every $T_{PollPeriod}$ has sync delay between $T_{PollTime}$ and $T_{PollPeriod}$
  - Task Scheduler takes time to…
    - Select task to run, dispatch it (maybe switch contexts, then run or resume it)
- Why does or doesn't an OS support synchronization within a process?
  - Not that easy for a task to pause itself for synchronization
    - Something must save/restore all CPU core registers (including PC, stack pointer) to switch between tasks
    - Each task needs memory for its function call stack
  - To support task preemption, scheduler **must** be able to preempt a task partway through, resume it later.
    - Not much more work to offer synchronization (which may require blocking) partway through task, but offers major benefits

How to Sync Within a Run-to-Completion Process?

- Background
  - Applies to all ISRs (even with task preemption), all tasks with RTC scheduler
  - Interrupt system, RTC scheduler can only run the process from its first instruction
- Convert process to finite state machine
  - Each call to process executes code for one state, allowing this process to block while sharing CPU with other processes
  - Example: I^2C ISR in KL25Z Reference Manual



Example: I2C Interrupt Routine for KL25Z

- Determine current state, select its code
- Run that state's code

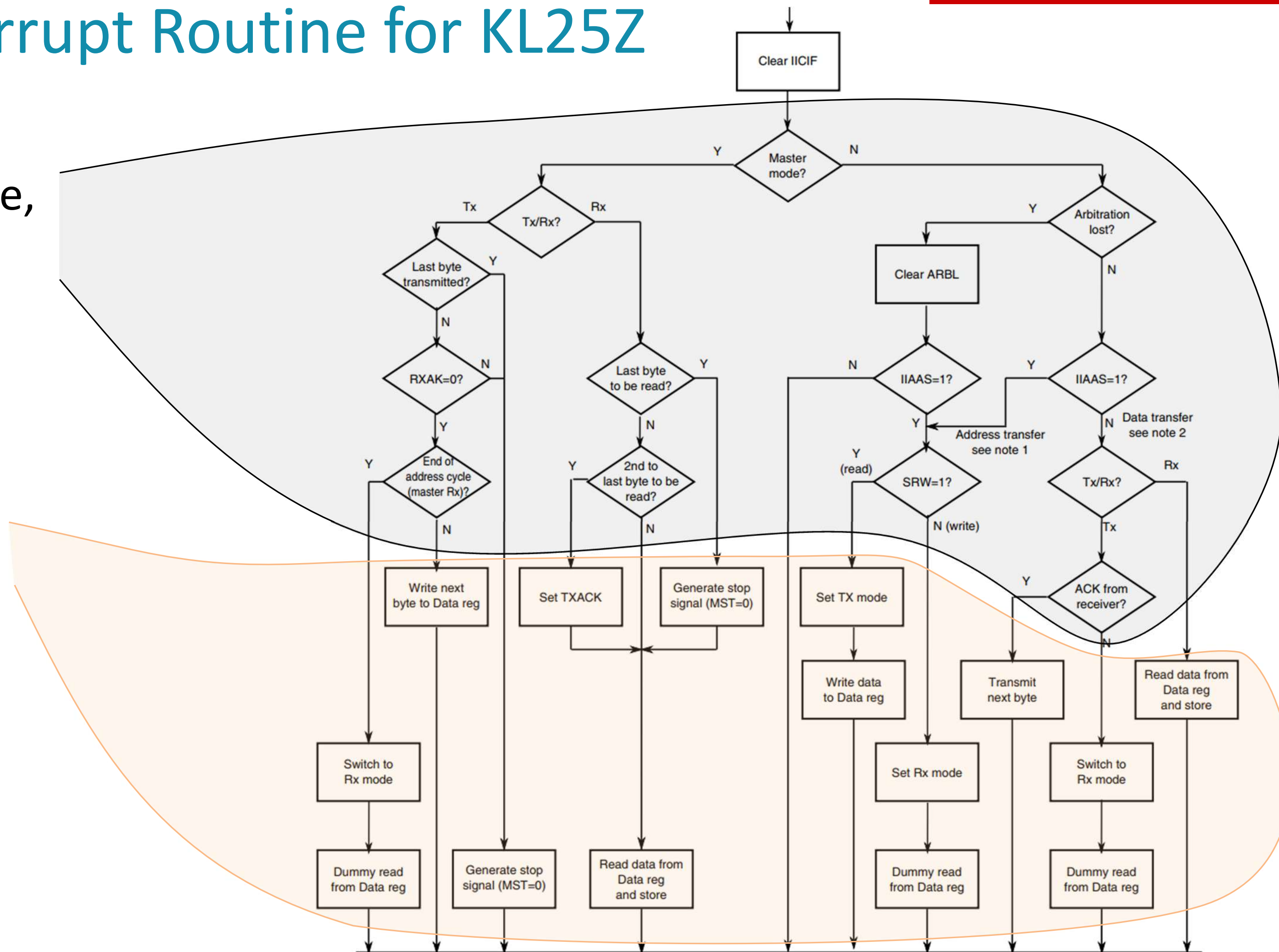


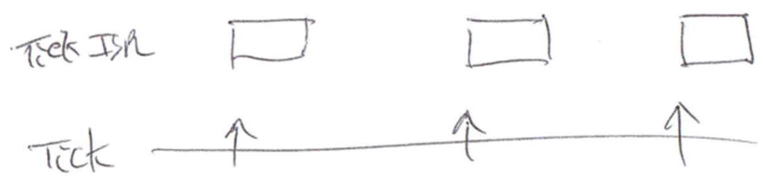
Figure 38-42. Typical I2C interrupt routine

NC STATE UNIVERSITY

PLATFORM LIMITS

Platform 1 Performance Limits

- Characterize basic performance limits of Platform 1 (48 MHz Cortex-M0+)
  - Interrupt System
    - Latency: 15 clock cycles
  - RTCS
    - Time-triggered scheduling resolution: 1 tick



f_{tick}	(48 MHz) CPU cycles
1 kHz	48,000
10 kHz	4,800
100 kHz	480
1 MHz	48

    - Scheduler latencies: time to examine table, find next ready task, call task function

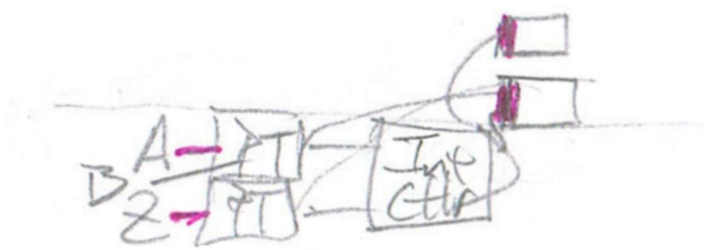
NC STATE UNIVERSITY

SUMMARY AND EVALUATION OF INITIAL APPLICATION DESIGN APPROACHES

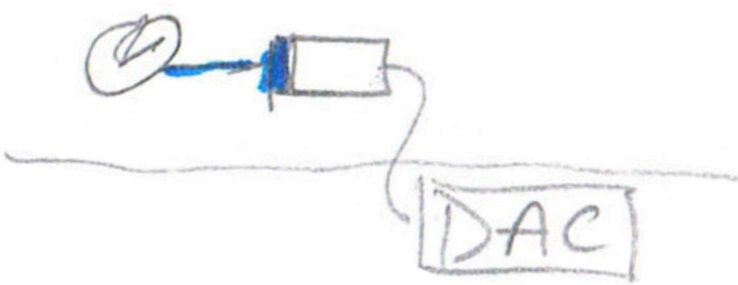
Examples

Event Trig.
Time Trig.

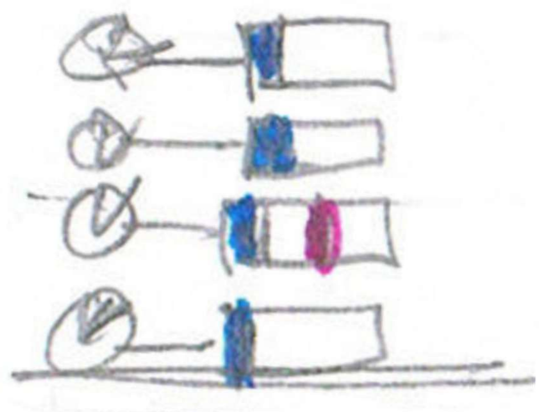
- Quadrature Decoder w/Limit Switch



- Waveform Generator

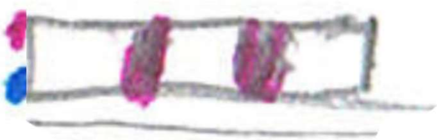


- Blinky Control Panel

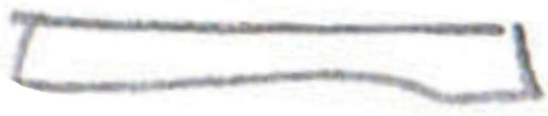


Examples

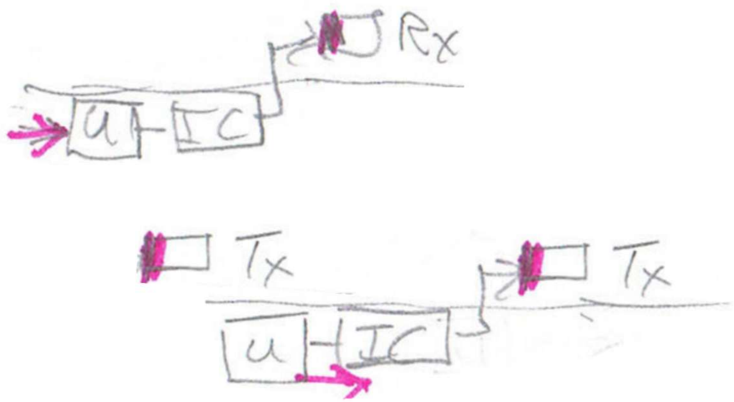
- Touchscreen



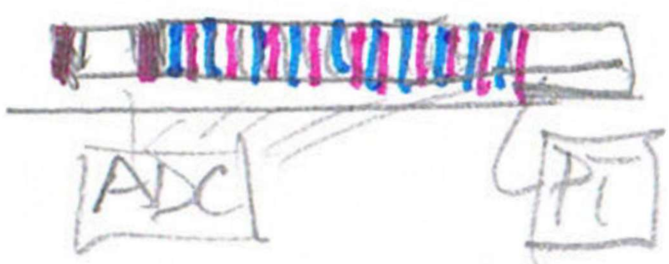
- LCD Controller



- Serial UART Communications



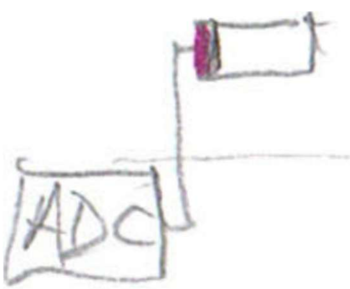
- Scope



NC STATE UNIVERSITY

Examples

- SMPS Controller



- PWM