NC STATE UNIVERSITY

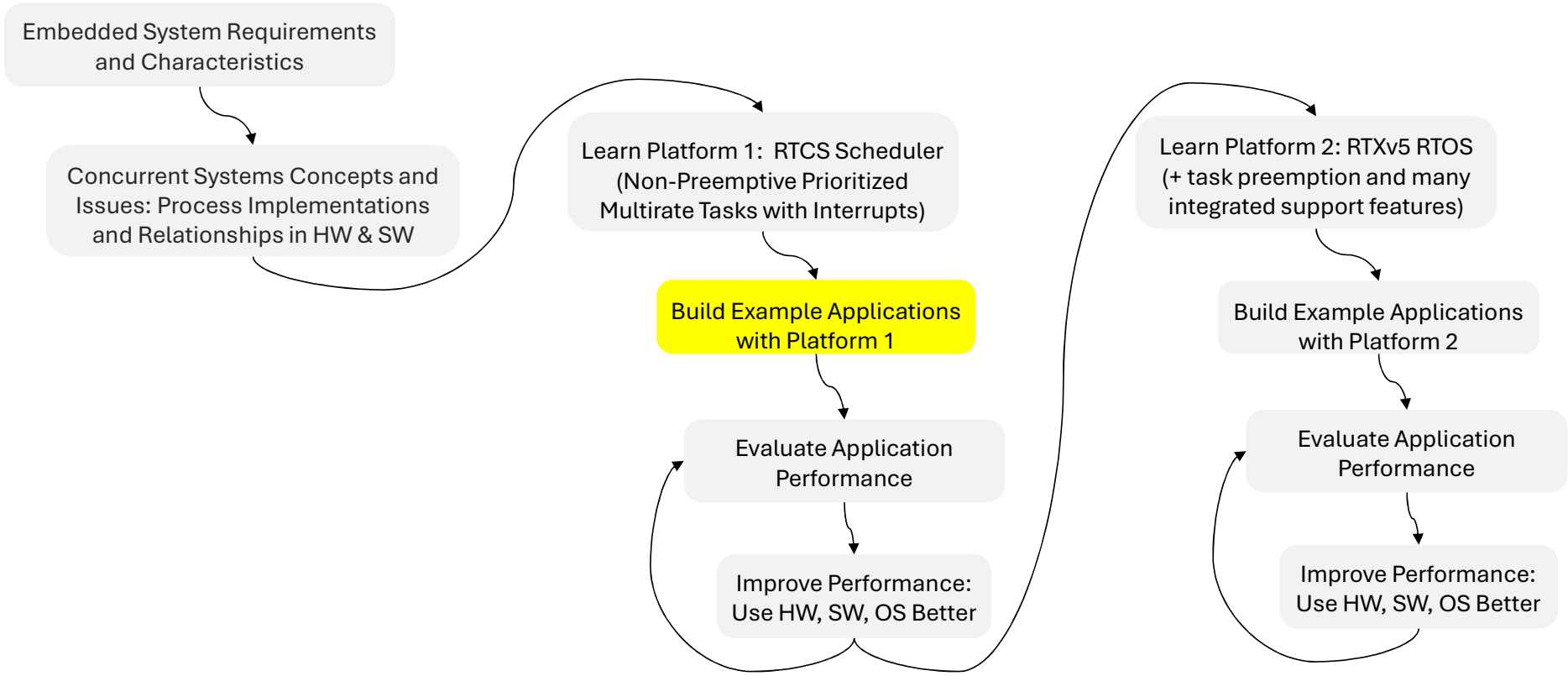
EXAMPLE APPLICATIONS ON PLATFORM 1 (RTC SCHEDULER WITH INTERRUPTS, BASIC PERIPHERAL USE)

V4

9/25/2025

NC STATE UNIVERSITY

Where are we in the class?



NC STATE UNIVERSITY

TOP-LEVEL VIEW OF APPLICATION PROCESSES

NC STATE UNIVERSITY

Example Application and Subsystem Processes

1. Quadrature Decoder w/Limit Switch
2. Waveform Generator
3. Blinky Control Panel
4. Touchscreen
5. Serial UART Communications
6. LCD Controller
7. Scope
8. SMPS Controller
9. More Comms
    1. SPI
    2. Higher protocol layers: I^2C, Secure Digital via SPI

NC STATE UNIVERSITY

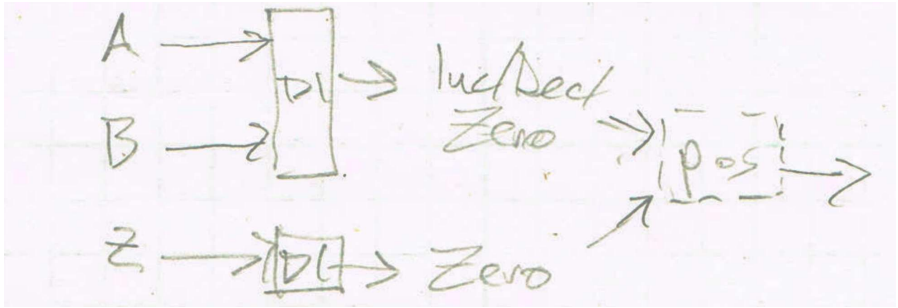
Applications: Functionality First, then Performance

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

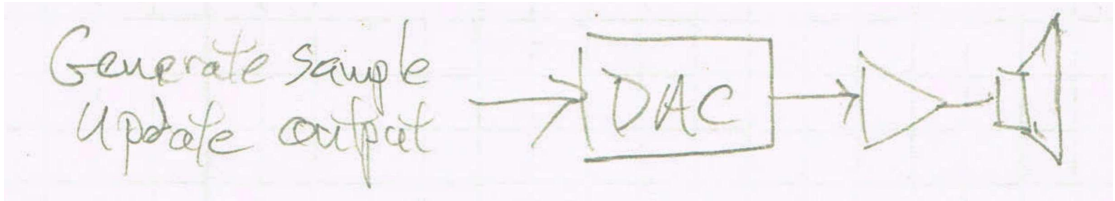
NC STATE UNIVERSITY

Quadrature Decoder with Zero Limit Switch Waveform Generator

- Quadrature Decoder with Zero Limit Switch



- Waveform Generator

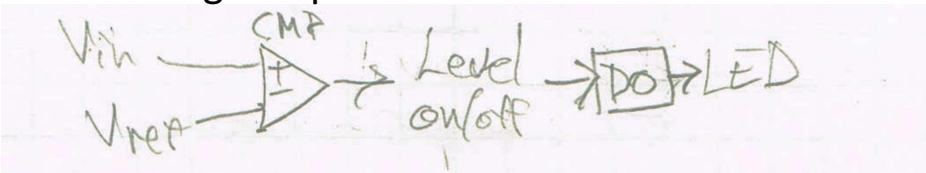


NC STATE UNIVERSITY

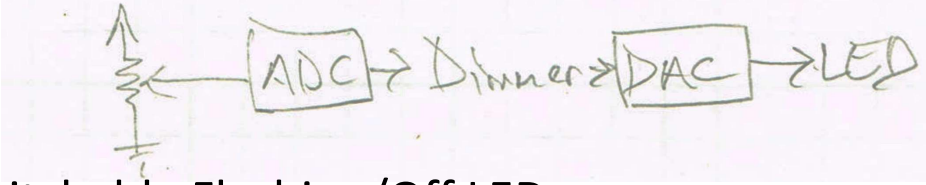
Blinky Control Panel

- Switchable On/Off LED

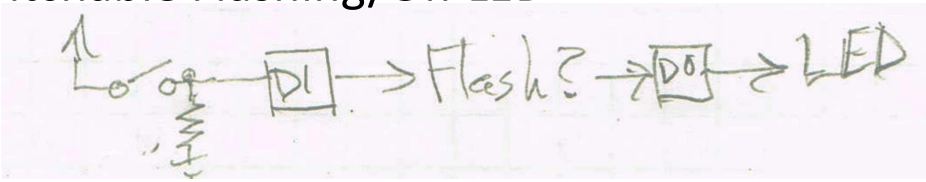
- Analog Level Alarm LED
  - Uses analog comparator



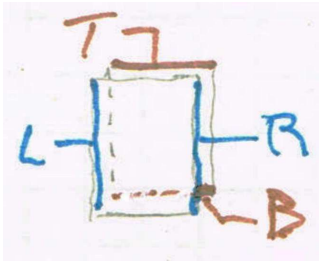
- Dimmable LED



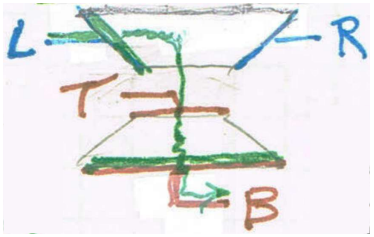
- Switchable Flashing/Off LED



Touch Screen Interface

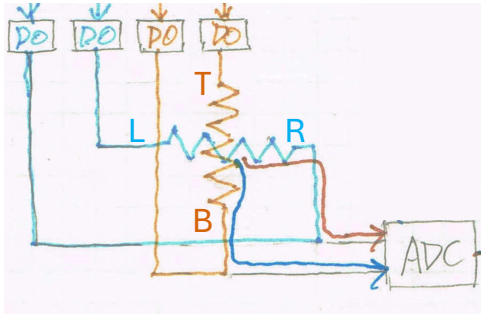


- Resistive touchscreen structure
  - Sandwich of two partially conductive clear plastic sheets
  - Sheet has terminal along two edges: L and R, T and B
  - Example sheet resistances: L to R: 300 Ω, T to B: 400 Ω

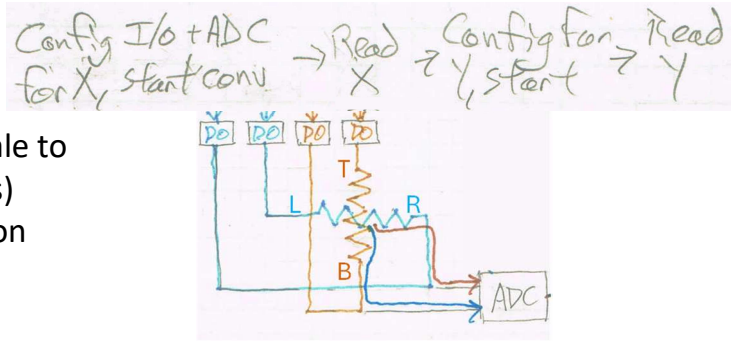


- Behavior
  - Sheets are normally separated, so infinite resistance between L (or R) and T (or B)
  - Pressing screen connects sheets at point of contact, creating conductive path between L (or R) and T (or B)

- Make two voltage dividers
  - Use digital outputs to drive a terminal (L) with V_{supply}, other terminal of that sheet (R) with 0 V
  - Use ADC to measure voltage on either terminal of other sheet (T, B)
  - Disable digital outputs driving L, R
  - Repeat using digital outputs to drive T, B, read L or R with ADC

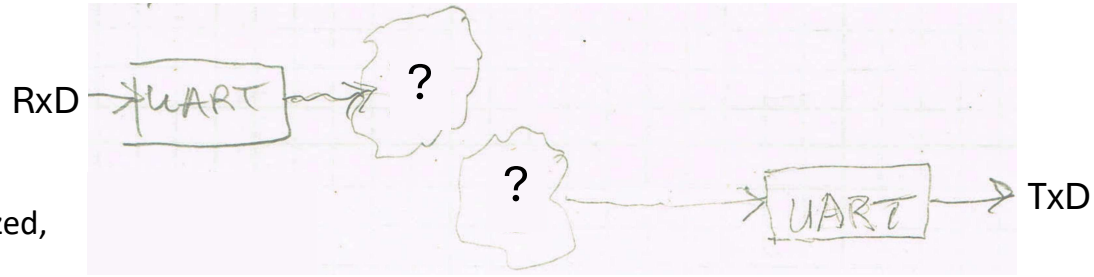


- Driver Operation
  - Read X position (between L and R), remove offsets, scale to output units (pixels)
  - Repeat for Y position (between T and B)



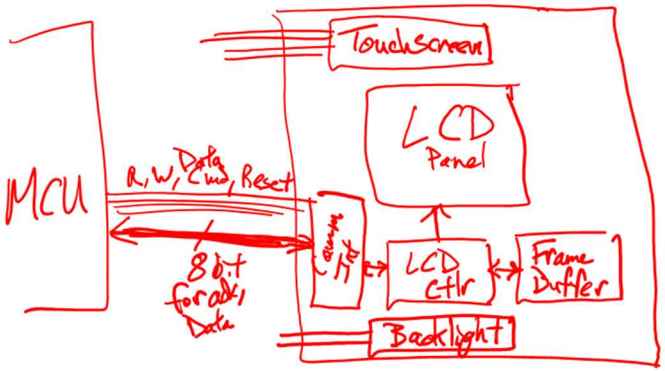
Serial Communications

- Use UART peripheral's receiver and transmitter
  - Convert between serial and parallel formats
  - Provide event notifications: transmit, receive, error, etc.
- Two processes: receive data, transmit data
  - Receive and transmit operations probably not synchronized, so Rx and Tx process must be able to make independent progress

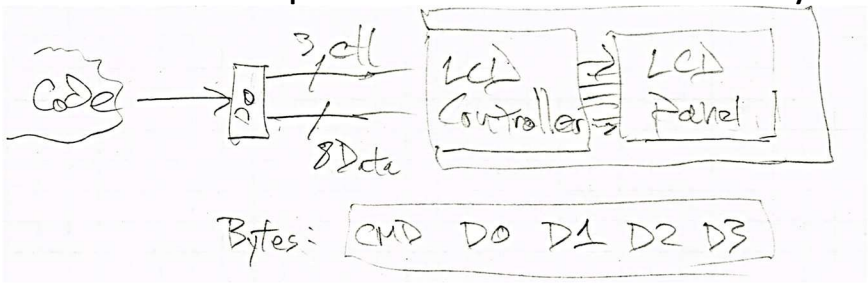


- Why the ***?***
  - Not defining those process details yet, since multiple approaches possible
  - Approaches depend on how and when that process synchronizes with the UART
    - After receiving a byte?
    - After receiving a specific byte?
    - After receiving N bytes
  - Note: last two approaches probably need to buffer intermediate data

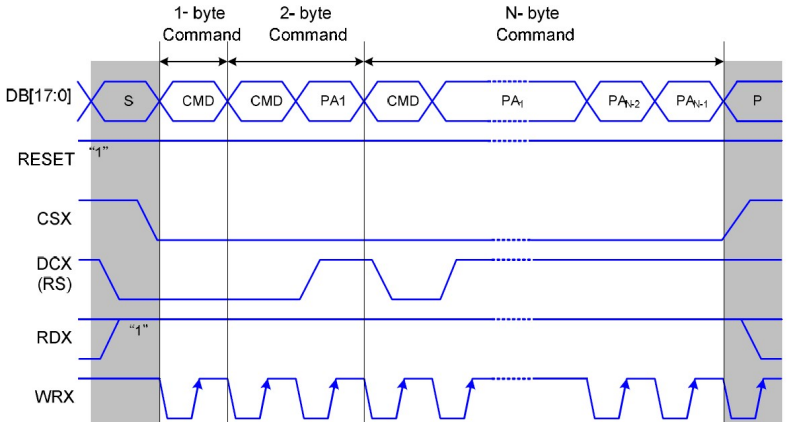
LCD Controller



- MCU emulates parallel bus interface used by LCD Controller



- MCU sends sequence of bytes (1 command + N data parameters) to LCD Controller

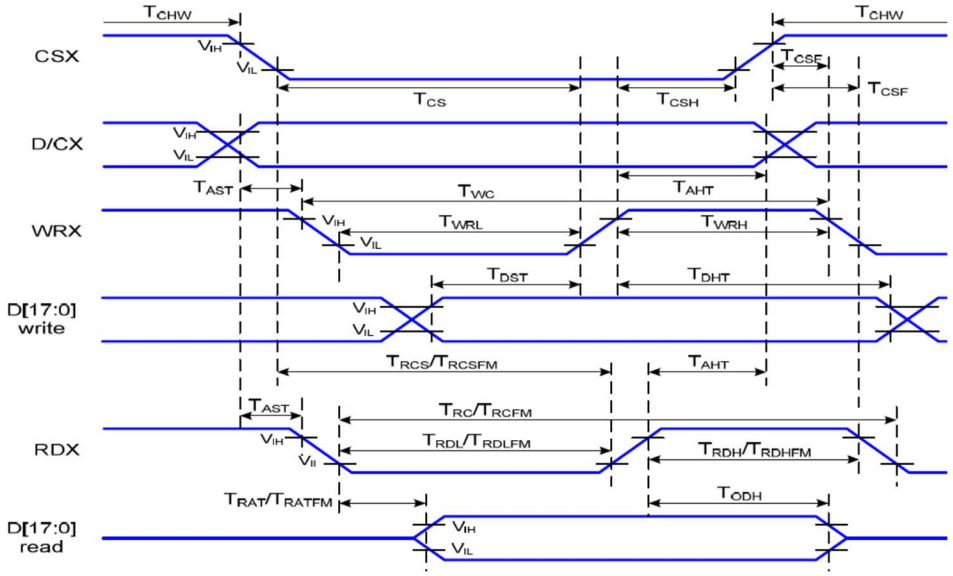


9.1.22 RAMWR (2Ch): Memory Write

2CH	RAMWR (Memory Write)												
Inst / Para	D/CX	WRX	RDX	D17-8	D7	D6	D5	D4	D3	D2	D1	D0	HEX
RAMWR	0	↑	1	-	0	0	1	0	1	1	0	0	(2Ch)
1st parameter	1	↑	1	D1[17]-1[8]	D1[7]	D1[6]	D1[5]	D1[4]	D1[3]	D1[2]	D1[1]	D1[0]	
...	1	↑	1	Dx[17]-x[8]	Dx[7]	Dx[6]	Dx[5]	Dx[4]	Dx[3]	Dx[2]	Dx[1]	Dx[0]	
N parameter	1	↑	1	Dn[17]-n[8]	Dn[7]	Dn[6]	Dn[5]	Dn[4]	Dn[3]	Dn[2]	Dn[1]	Dn[0]	

NC STATE UNIVERSITY

Any Synchronization Needed with LCD Controller?

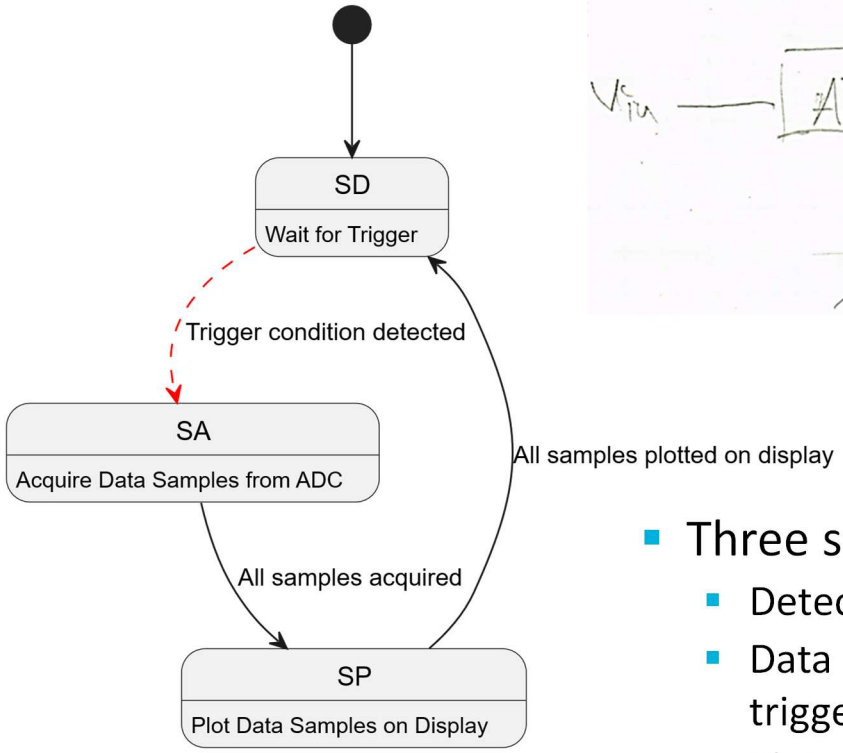


- Could MCU have to wait for LCD controller?
  - Data sheet shows timing requirements
- MCU's emulated bus interface is slower than max speed for LCDC, so don't need to synchronize
- Upcoming: SecureDigital flash memory controller is much slower …

VDDI=1.65 to 3.3V, VDD=2.4 to 3.3V, AGND=DGND=0V, Ta= -30 to 70 ℃

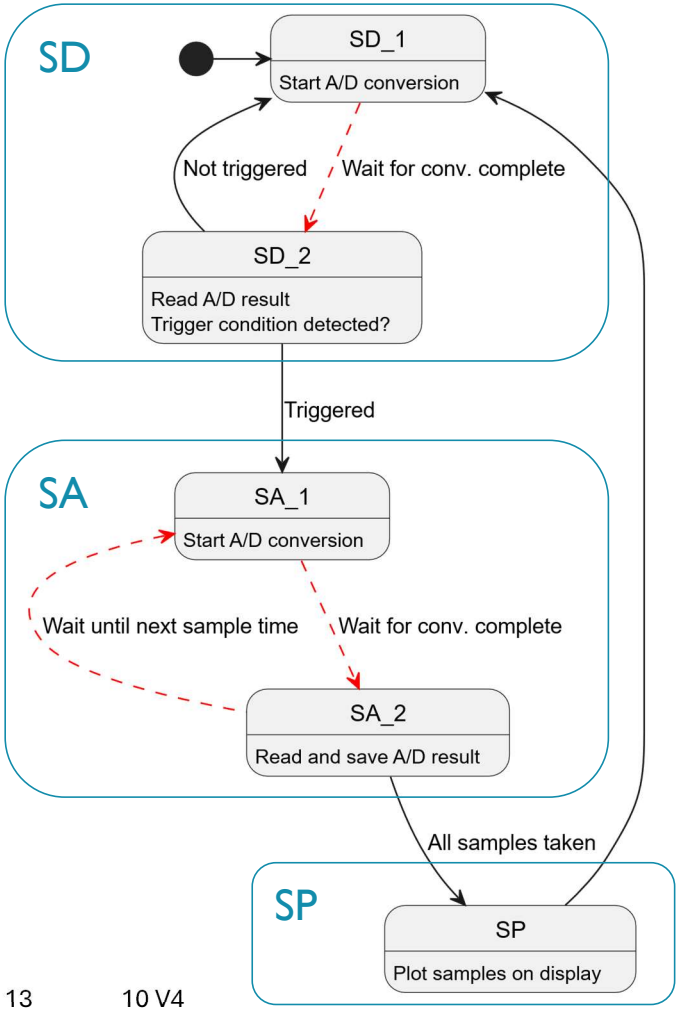
Signal	Symbol	Parameter	Min	Max	Unit	Description
D/CX	T_{AST}	Address setup time	0		ns	-
	T_{AHT}	Address hold time (Write/Read)	10		ns	
CSX	T_{CHW}	Chip select "H" pulse width	0		ns	-
	T_{CS}	Chip select setup time (Write)	15		ns	
	T_{RCS}	Chip select setup time (Read ID)	45		ns	
	T_{RCSFM}	Chip select setup time (Read FM)	355		ns	
	T_{CSF}	Chip select wait time (Write/Read)	10		ns	
	T_{CSH}	Chip select hold time	10		ns	
WRX	T_{WC}	Write cycle	66		ns	
	T_{WRH}	Control pulse "H" duration	15		ns	
	T_{WRL}	Control pulse "L" duration	15		ns	
RDX (ID)	T_{RC}	Read cycle (ID)	160		ns	When read ID data
	T_{RDH}	Control pulse "H" duration (ID)	90		ns	
	T_{RDL}	Control pulse "L" duration (ID)	45		ns	
RDX (FM)	T_{RCFM}	Read cycle (FM)	450		ns	When read from frame memory
	T_{RDHFM}	Control pulse "H" duration (FM)	90		ns	
	T_{RDLFM}	Control pulse "L" duration (FM)	355		ns	
D[17:0]	T_{DST}	Data setup time	10		ns	For CL=30pF

Oscilloscope Behavior: High-Level View



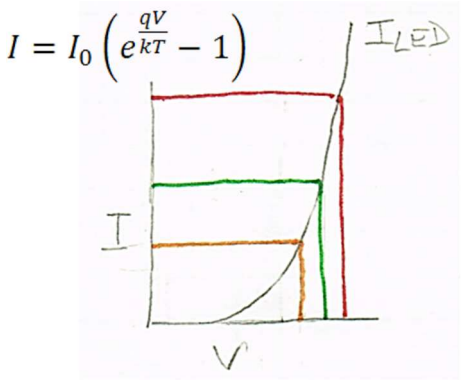
- Three sequential states (phases) in scope process
  - Detect trigger condition (wait for trigger)
  - Data acquisition: Start running (sync) after event detected (valid trigger). Periodically sample with ADC, save results in buffer.
  - Plot data from buffer.

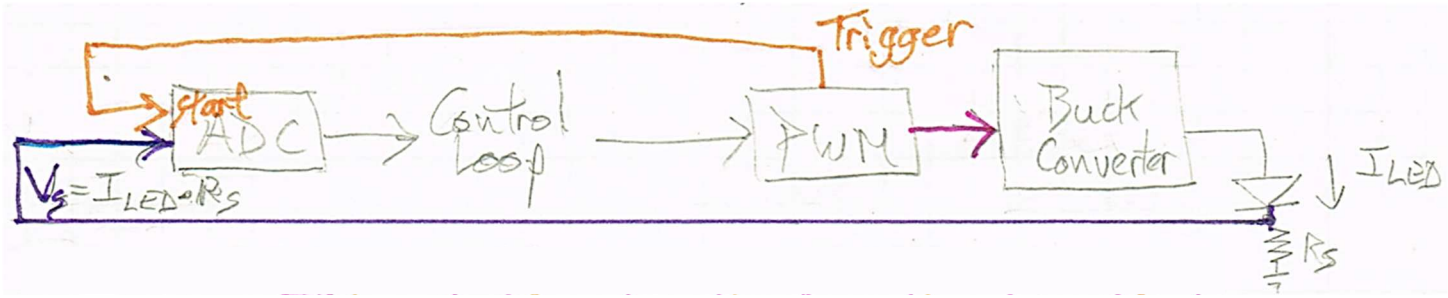
Oscilloscope Behavior: Internal Synchronization?

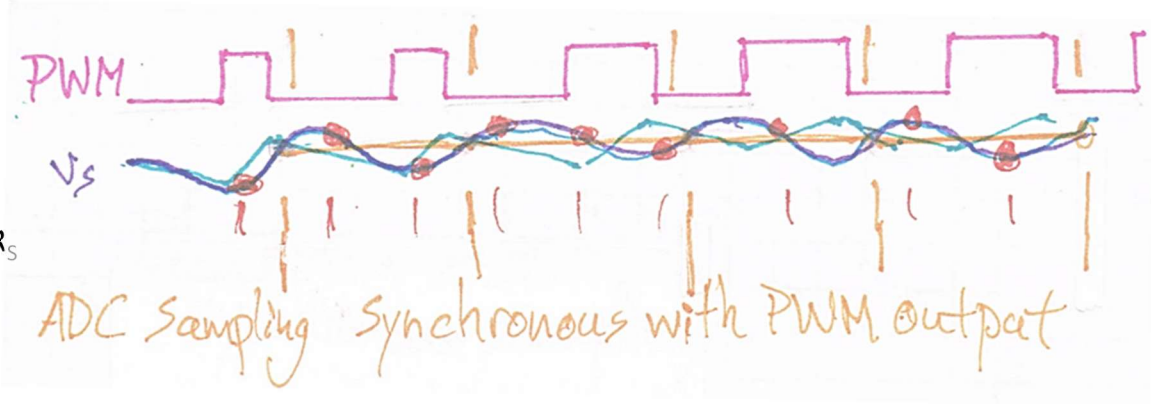


- Can a state's code run non-stop after it starts, or is internal synchronization needed after a state starts?
  - Break down states for Detect Trigger, Acquire Data where synchronization is needed
- Detect trigger - No
  - Convert input with ADC, analyze, decide until valid trigger detected.
  - Can't run non-stop. Need to sync to each ADC conversion complete event
- Acquire data - No
  - Uses ADC, and time of starting conversion matters
  - Can't run non-stop, needs to synchronize **twice** internally for each sample:
    - Sync start of conversion to correct **time** (periodic at sampling rate)
    - Sync reading A/D result to **conversion completion**
- Plot data – Yes
  - LCD Controller is fast enough to keep up with CPU at full speed

Constant-Current LED Driver with Buck Converter







- LED current rises exponentially with voltage
- Use control system to regulate current
  - Use switching power converter (buck) to efficiently reduce voltage
  - Measure current through LED using sense resistor R_s
  - Switching causes voltage ripple on sense voltage V_s
- When to sample?
  - Red: not synchronized with switching, lots of varying error due to ripple
  - Orange: synchronized with switching, constant error or no error from ripple.
- How to synchronize ADC sampling with power converter's switching?
  - Timer generating PWM signal also triggers ADC sampling (followed by A/D conversion)

NC STATE UNIVERSITY

FIRST VERSIONS OF APPLICATIONS WITH RTC SCHEDULER AND INTERRUPTS

NC STATE UNIVERSITY

LN10 – First pass through apps

- Quadrature Decoder with Zero Limit Switch
  - Uses GPIO Port inputs
  - Design 1: Periodic task, as RTCS doesn't have edge events
    - One Task
      - Poll GPIO Z for 1, else poll A for edge, then sample B.
    - Two Tasks
      - Poll GPIO Z, zero pos if Z==1
      - Poll A to detect edge (can be blocking or non-blocking (RTCS will share CPU)), then sample B, update pos.
    - Problems: max event rate is limited unless greedy scheduling
  - Design 2: Event-triggered:
    - Use port interrupts for A, Z rising edges to trigger ISR(s).
    - ISR(s) can do all work. Otherwise could release task(s) to finish.
    - Possible data race condition for pos.
- Waveform Generator
  - Design
    - Uses DAC output
    - One periodic task to calculate new value, update DAC.
  - Problems
    - Max rate is limited by scheduler tick frequency.
    - Timing stability suffers from task interference
- Blinky Control Panel
  - Independent tasks
  - Use GPIO, ADC, DAC
  - Block until ADC done
  - Flash LED with periodic task
  - Introduces timing interference with multiple tasks
- Touch Screen Interface
  - Periodic task: Check for touch, then config for X, do ADC (block), read, then repeat for Y.
  - Interrupt driven? Could do, not needed yet. Do later (low power operation to wake up MCU)
- Serial Comms
  - Uses UART
  - Sync and comm between ISRs and tasks
    - ISRs release tasks when?
      - Rx: Received new character? N more characters? Special character (buffering)? (FIFO full?)
      - Tx: Tx buffer ready to accept new character? Done sending character?
  - Evaluate performance limits: RX or TX time vs. interference. → will lead to buffering multiple data entries
- LCD Controller
  - Parallel bus interface, bit-banging it
  - Higher-level protocol on top of bytes
- Scope
  - Sync for triggering, Then stable timing for data capture
- SMPS Controller
  - Synchronous sampling by ADC
  - Closed-loop control
  - PWM

Building Blocks in First Pass (V1) for Sync and Do/Don't

- Main function/thread
- Basic peripheral operation
- MCU's Built-in Scheduler: Interrupt system
  - When interrupt is requested, interrupt system forces CPU to run that interrupt's service routine
  - Use interrupt requests from peripherals (event detectors)
- Task Scheduler: RTCS
  - RTC = run to completion
    - Tasks run to completion = tasks run sequentially, not concurrently.
    - Good fit for some systems, not for others.
  - RTCS Task Synchronization Support: releasing a task to run
    - Time-triggered: On every timer tick, RTCS examines task scheduling table to see if it's time to release any tasks. Releases tasks as appropriate.
    - Event-triggered: Not supported natively in scheduler. Task or ISR can detect event, then request scheduler to run a service task in response. Use API call (RTCS_Release_Task())
- Preemption
  - Tasks may not preempt tasks
  - ISRs may preempt tasks
  - ISRs may preempt ISRs

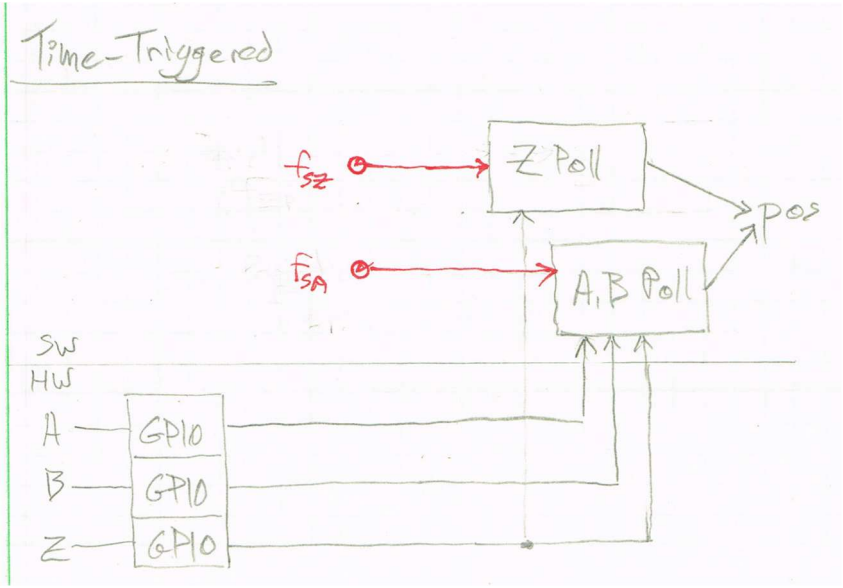
RTC Scheduler API

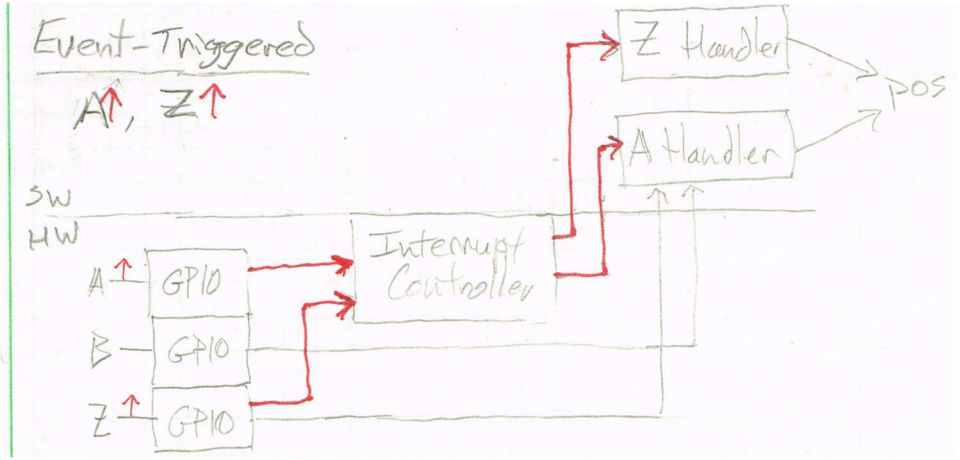
```
/* Initializes task table. Configures LPTimer to generate
interrupt at freq Hz. Maximum freq is 500 Hz using
LPTimer. Can get higher tick frequency by using a
different timer (e.g TPM). */
void RTCS_Init(uint32_t freq);

/* Updates TicksToNextRelease (and possibly
ReleasesPending) for each enabled, non-null task with
TicksToNextRelease ("TTNR")> 0. For each such task,
decrement TTNR. If TTNR reaches 0, then increment
ReleasesPending, and copy Period to TTNR. Must be called
from ISR for tick timer. */
void RTCS_Timer_Tick(void);

/* Runs the scheduler and never returns. Scheduler
searches RTCS_Task_Table from [0] (highest priority)
looking for the first enabled, non-null task entry with
ReleasesPending > 0. When it is found, decrement
ReleasesPending, and run task. Then repeat, starting at
[0] again. Must call RTCS_Add_Task at least once before
this call. */
void RTCS_Run_Scheduler(void);

/* Look up priority/position in table of given task.
Returns -1 for error. */
int RTCS_Find_Task_Priority(void (*task)(void));
```

```
/* Adds task to table at entry number “priority”,
overwriting what was there previously. Sets
TicksToNextRelease to period, ReleasesPending to 1
(request first run ASAP), and enables task. */
int RTCS_Add_Task(void (*task)(void), uint32_t priority,
uint32_t period);

/* Request task run by incrementing ReleasesPending
(number of unfulfilled run requests) for given task.
Should be changed to protect against race conditions. */
int RTCS_Release_Task_i(int i);
int RTCS_Release_Task(void (*task)(void));

/* Enable or disable task by updating its Enable flag
based on enable parameter (1 = enable, 0 = disable) */
int RTCS_Enable_Task(void (*task)(void), uint32_t enable);
int RTCS_Enable_Task_i(int i, uint32_t enable);

/* Write period to Period, and reload TicksToNextRelease
with period. If release_now is > 0, increment
ReleasesPending to trigger run ASAP. */
int RTCS_Set_Task_Period(void (*task)(void), uint16_t
period, int release_now);
int RTCS_Set_Task_Period_i(int i, uint16_t period, int
release_now);
```

NC STATE UNIVERSITY

Quadrature Decoder with Zero Limit Switch – V1





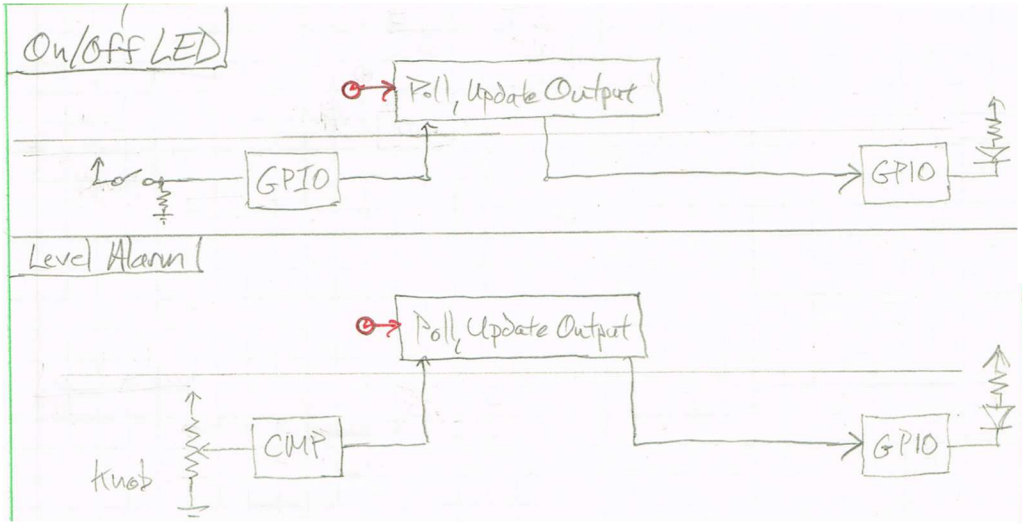
- RTCS doesn't support these events, so use time-triggered sampling approach instead
  - Periodically sample Z to detect 1
  - Periodically sample A to detect 0->1 transition
- Port/GPIO and interrupt system do support events
  - Use interrupts to detect A, Z events
  - ISR can do work, or ask RTCS to release task to do work.

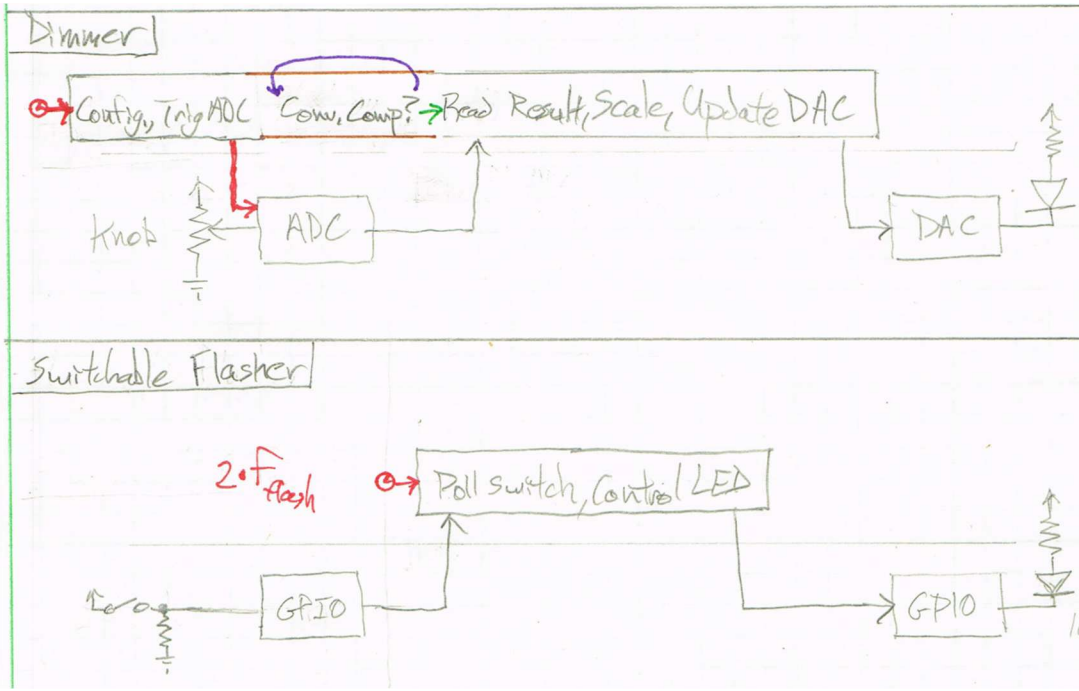
Waveform Generator – V1

- Simple – Periodic task! If scheduler can release task …
  - at the frequency we want
  - with timing which is stable enough

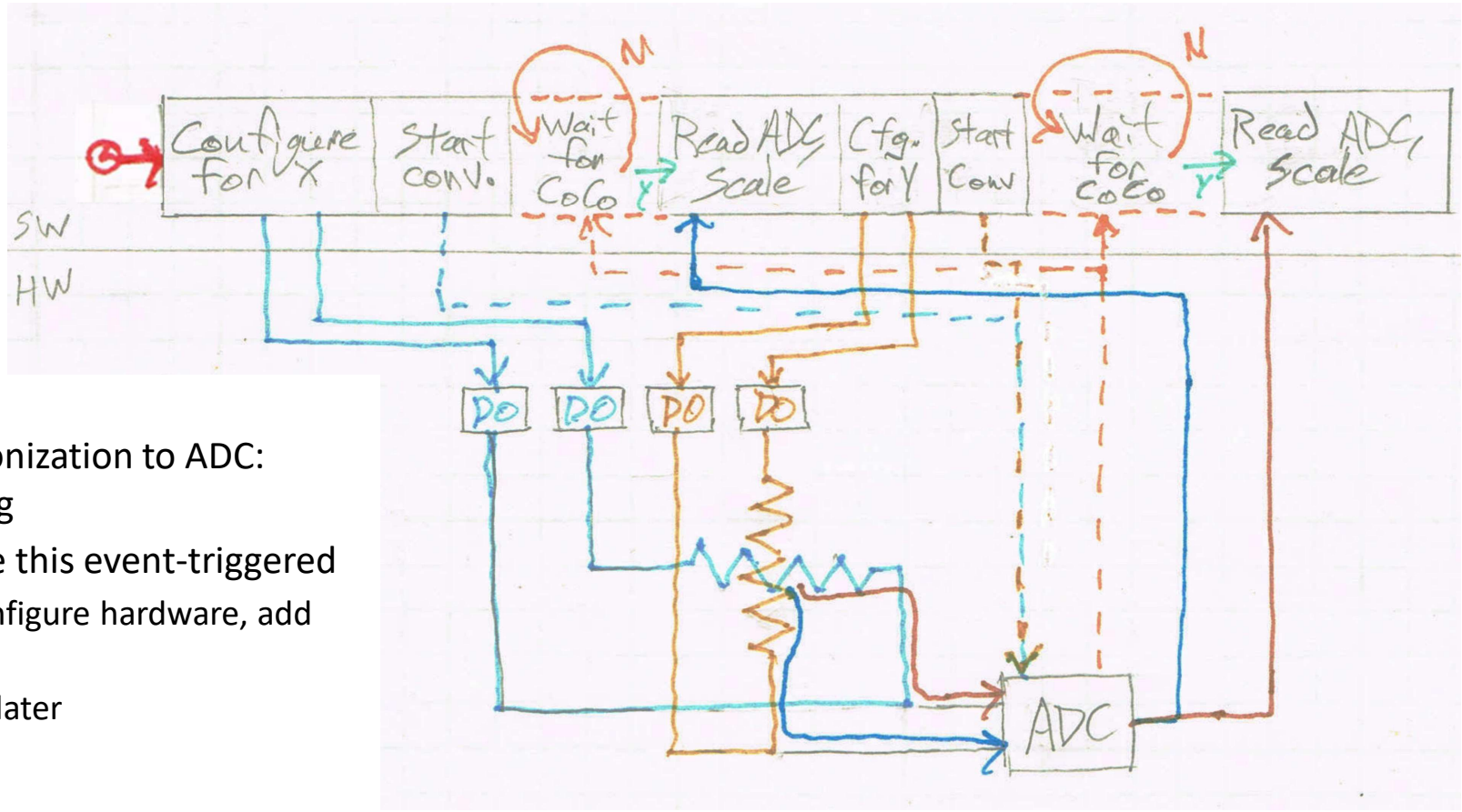
NC STATE UNIVERSITY

Blinky Control Panel – V1





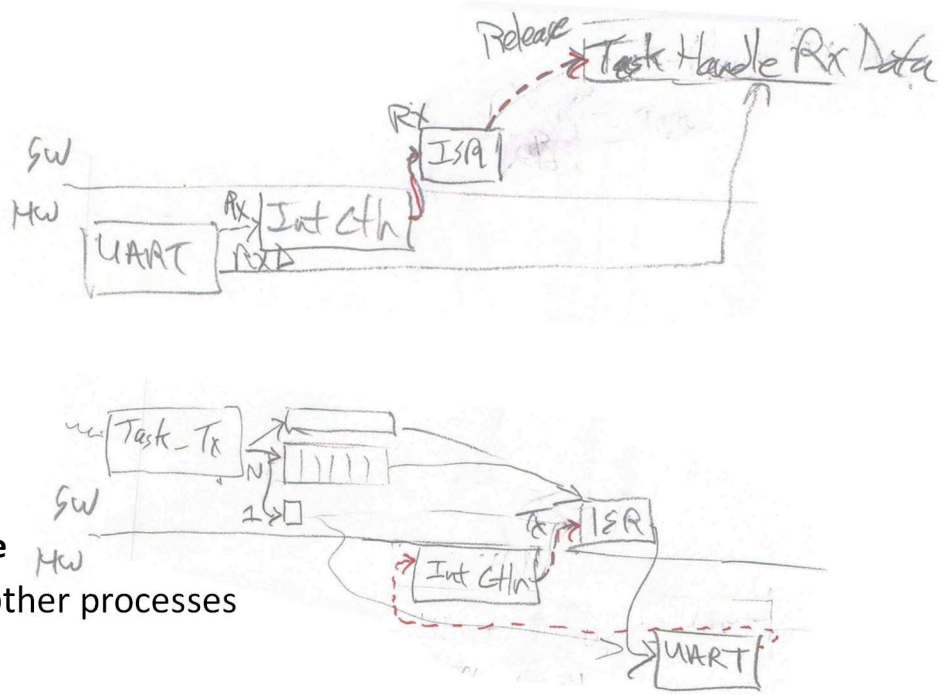
Touchscreen Interface - V1



- Poll periodically
- Internal synchronization to ADC: use busy-waiting
- Could also make this event-triggered
  - Need to reconfigure hardware, add software
  - Will examine later

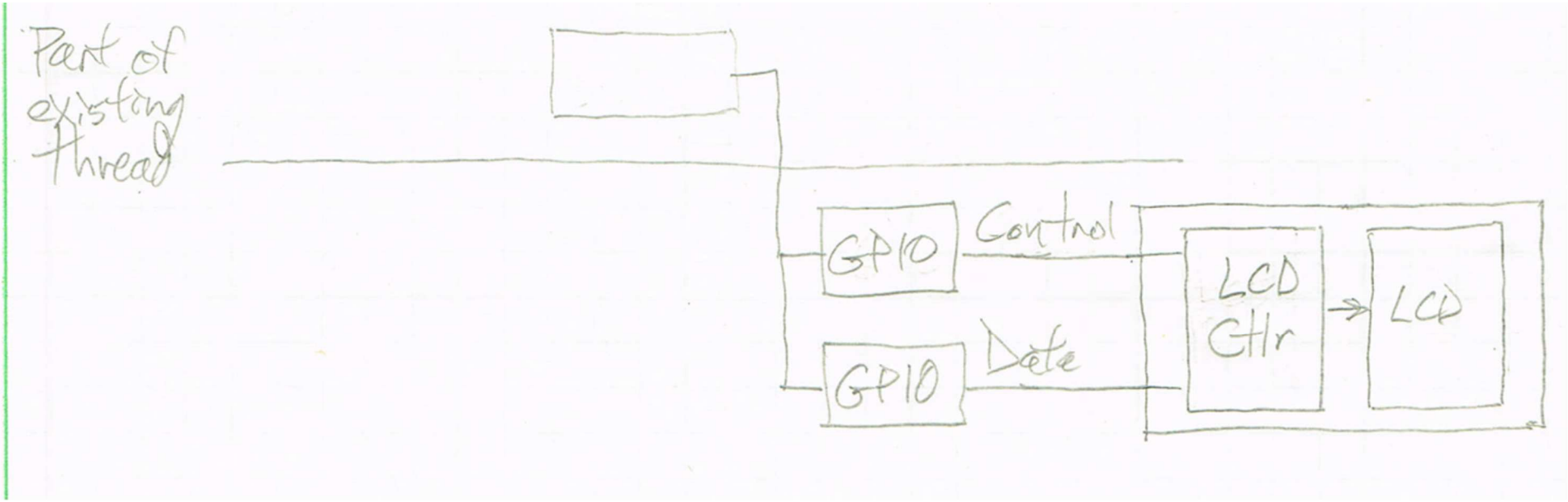
Serial Communications - V1

- *Many* types of implementations possible
- Requirements for all
  - Receiver handler sync with UART
    - Synchronize receive handler code to data reception event
    - Read data from UART before next item **overwrites** it
  - Transmitter handler sync with UART
    - Synchronize transmit handler code to transmitter event
      - Events: Ready for new data, or transmission complete
    - Write data to UART preferably right after event, for **performance**
  - Sync transmitter and receiver handlers with each other or other processes
    - Need to notify of new data and communicate that data
- Event-driven implementation
  - Use UART ISR for Tx, Rx events.
  - ISR asks scheduler to run handler task for event with RTCS_Release_Task
  - How much work to do in ISR depends on response time for handler tasks



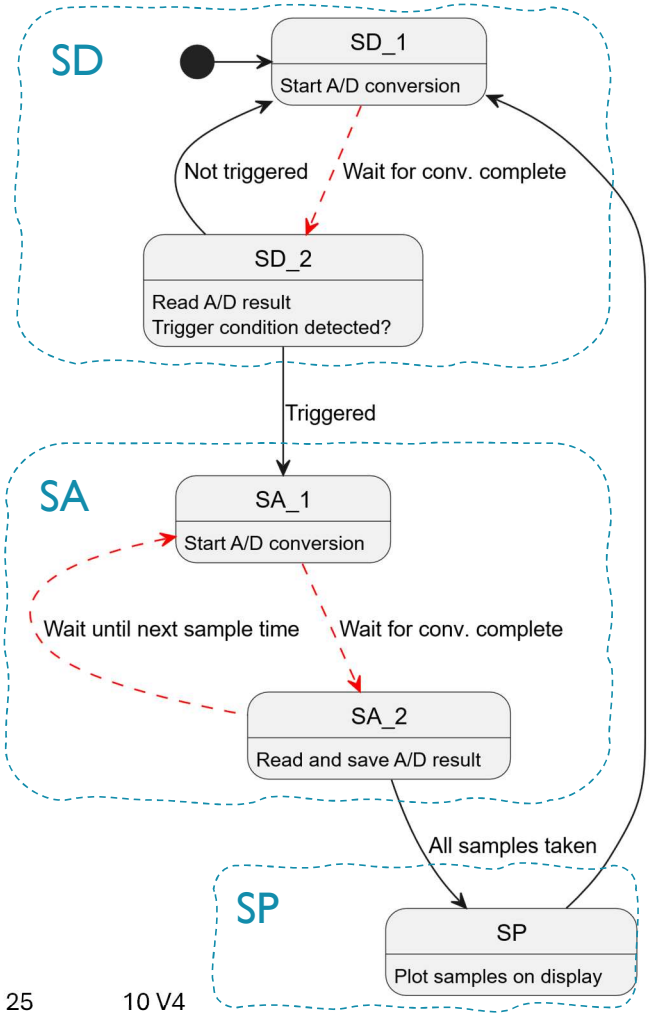
NC STATE UNIVERSITY

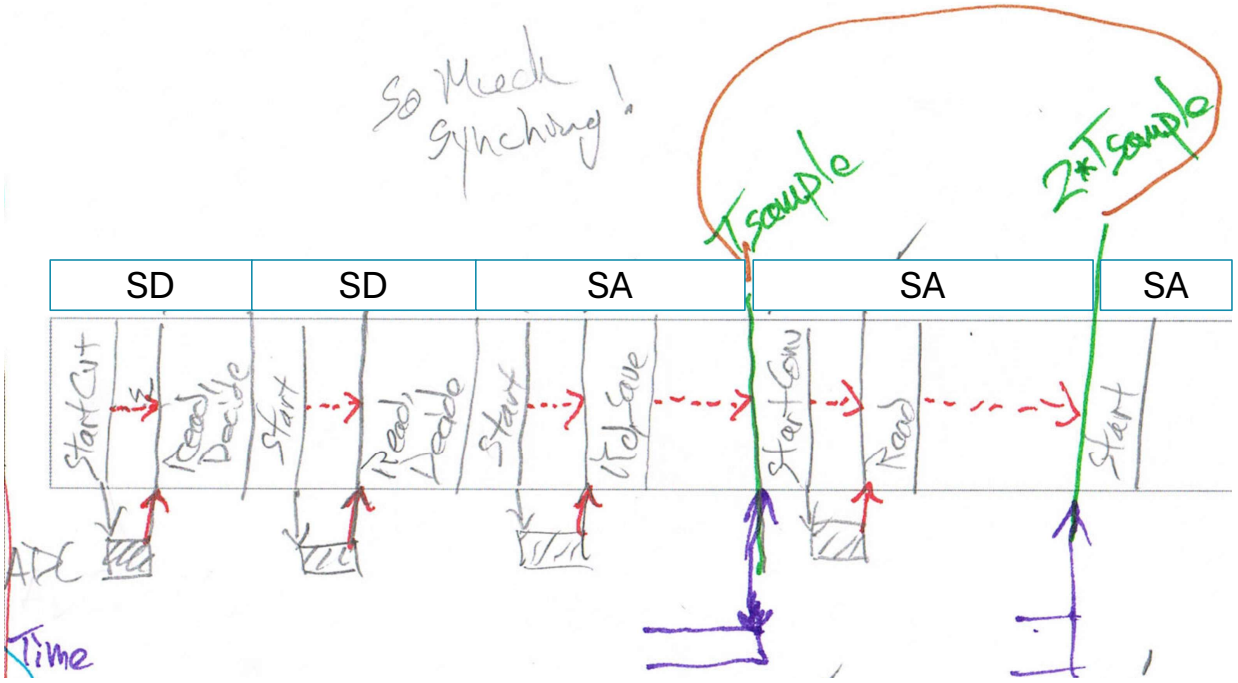
LCD Controller Interface - V1



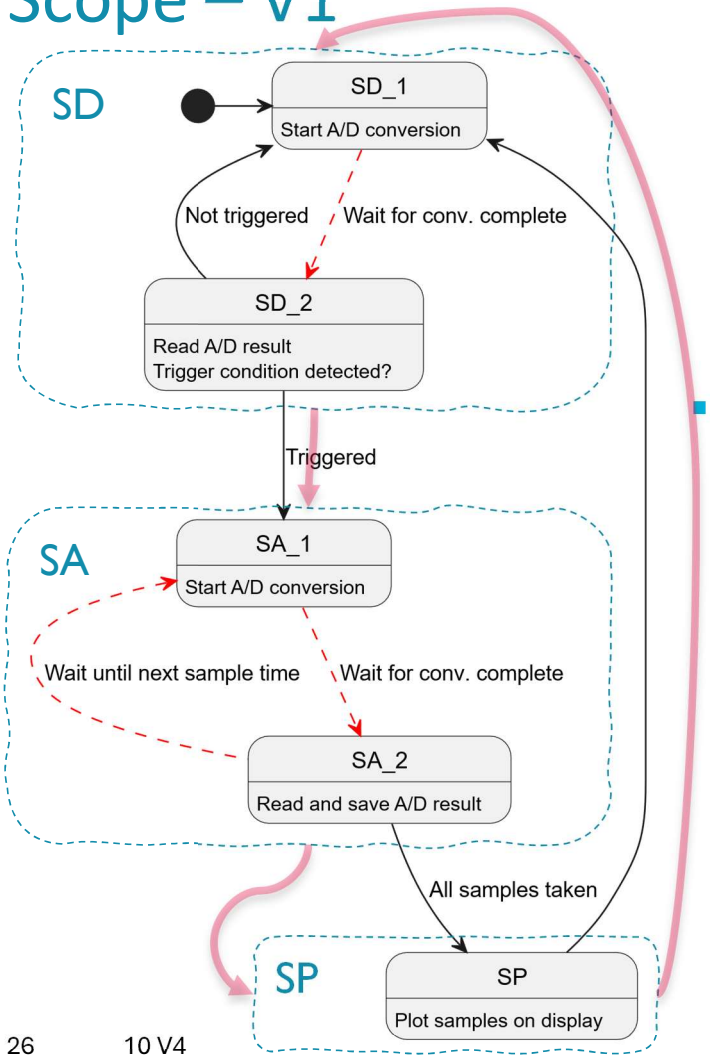
- Controller interface can be part of existing thread (called as subroutine) because no internal sync required

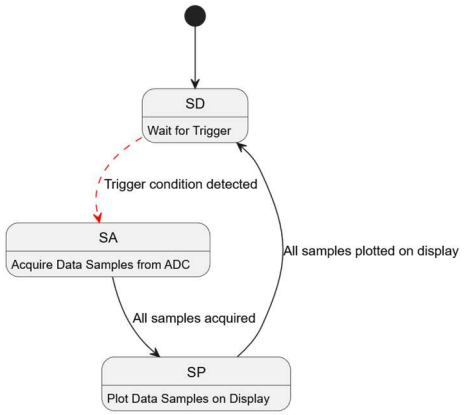
Scope – V1 – Closer look at Internal Synchronization





Scope – V1

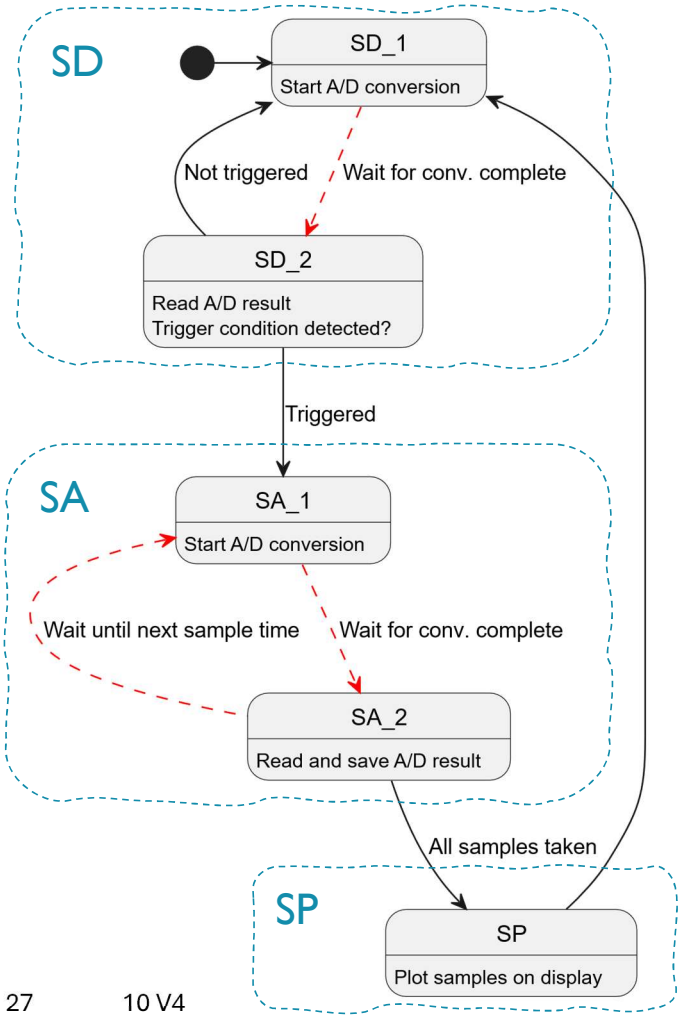




- How to synchronize these parts (using RTCS, ADC and polling)?
  - RTCS doesn't let a task pause partway through and let scheduler run a different task
  - So, use three tasks to implement scope: Task_SD, Task_SA, Task_SP. Enable and disable them to control their execution.
  - Could also restructure scope task into a finite state machine (see FSM tasks in **ESF** Chapter 3)

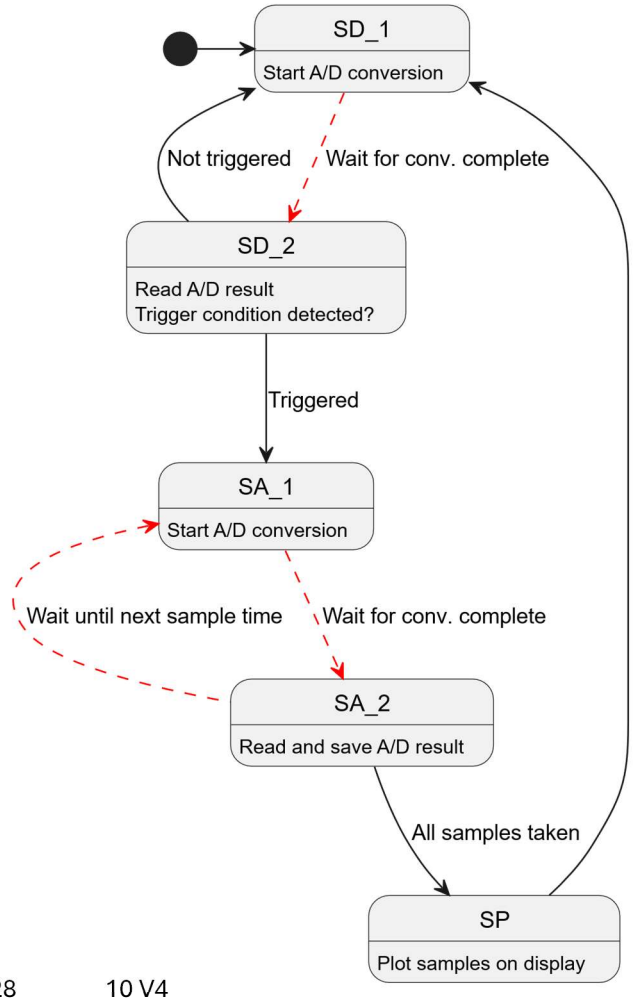
- Task_SD: Detect trigger
  - Task is event triggered: run when user interface says so (calls RTCS_Release_Task())
  - Internal synchronization
    - Start A/D conversion
    - Wait for (sync to) event: greedy polling of ADC Conversion Complete flag (CoCo)
    - Analyze, decide if trigger condition was met.
  - If trigger detected, Task_SD…
    - Disables self (RTCS_Enable_Task(…,0)
    - Releases Task_SA (RTCS_Release_Task())

Scope – V1



- Task_SA: Acquire data samples
  - Wait until (sync to) time n*T_{Sample} (except 1st time, to simplify design). How?
    - RTCS only schedules tasks, so would have to split Task_SA's SA_1 code into a **separate task**, or convert Task_SA into an **FSM** with states SA_1, SA_2.
    - For initial implementation (V1), just get it working. Start conversions as fast as possible, accept no control of timing initially.
  - SA_1: Start A/D conversion
  - Wait for (sync to) event: ADC Conversion Complete*. How?
    - Use scheduler? Split into another task, convert Task_SA into FSM
    - Simple version: use greedy polling loop
  - * Could remove by overlapping with next wait until time n*T_{sample}
  - SA_2: Read and save A/D result
    - Read A/D result, save
  - Repeat if more samples needed, else Task_SA …
    - Disables self (RTCS_Enable_Task(…,0)
    - Releases Task_SP (RTCS_Release_Task())
- Task_SP: Convert and plot data on LCD
  - Can run non-stop, no sync needed
  - When done plotting, Task_SP …
    - Disables self (RTCS_Enable_Task(…,0)
    - Releases Task_SA (RTCS_Release_Task())

Scope – Future Version Possibilities



- Maybe later implement Scope function with internal FSM?
  - Each call to Scope function executes code for one state
- Would allow use of **synchronization** and **scheduling** by
  - **RTCS task scheduler** at a finer grain than with just three states (SD, SA, SP)
  - **MCU's interrupt system,** using **peripheral state and events** (CoCo, timer overflow)