NC STATE UNIVERSITY

Class 03 - I/O, Timing and Synchronization (Rough)

A.G. Dean
agdean@ncsu.edu
https://sites.google.com/ncsu.edu/agdean/teaching

Class 03 – I/O, Timing and Synchronization

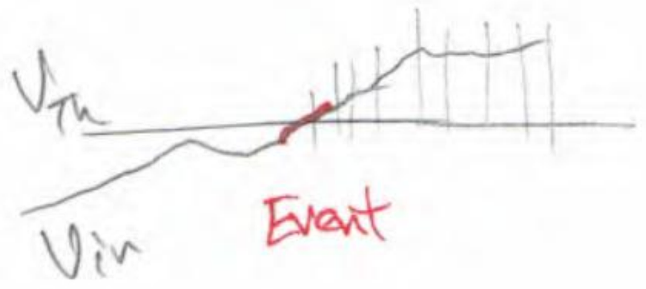
NC STATE UNIVERSITY

Outline

- OVERVIEW
  - Timing requirements for I/O activities are major driver for embedded system design decisions
  - May need to synchronize to event or time before doing the work (Sync and Do)
    - Scope trigger: detect input rising across threshold voltage, then can start sampling data
    - Quadrature decoder: detect input A rising, then sample input B, increment or decrement count
- UNDERSTANDING PROCESSING CHAIN FOR I/O ACTIVITIES
  - Synchronize with something
    - Types
      - Event-Triggered: Detect event
      - Time-Triggered: Await target time
  - Do processing in response
    - Timing requirements:
      - Simple deadline: within T_{DL} after event/time
      - Window deadline: Between T_{DL_Open} and T_{DL_Close} of event/time
  - Repeat?
    - May have burst or sequence of I/O activities, so next will sync (event or time) to next part or do it immediately/ASAP
    - Examples inputs:
      - Quadrature decoder,
      - UART receive data
- HOW TO SYNCHRONIZE?
  - All Hardware
    - Easy: Dedicated signals
  - Some Software
    - HW/SW allocation and processing chain. SW polls hardware (input peripheral)
    - Hard, since software timing is sloppy, gets even harder when sharing CPU
      - Timing variation diagram (ramp), sync to stabilize/cut timing variation
    - Start simple: Not sharing CPU
      - Detect with blocking SW loop polling (busy-waiting)
      - Responsiveness
      - Greedy
    - Share CPU with software scheduling method
      - Round-Robin Loop/Cyclic Exec.
        - Detector doesn't block, but take turns with other code (possibly multiple detectors)
        - Responsiveness
        - Not so greedy
      - Many other sharing options. Prioritization, preemption …
        - + Schedule, dispatch.
  - HW Event Detection
    - Hardware peripheral detects event
    - HW/SW allocation and processing chain. SW polls event detector
  - HW Event Detection + Interrupt System
    - HW/SW allocation and processing chain
    - Handler runs
- BASIC TIMING ANALYSIS
  - Approaches
    - Slack time
      - How late can process start and meet deadline?
    - Response time
      - When will this process finish, considering effects of other processes in system
  - Complications from scheduler sharing the CPU among SW processes
    - Basic: static fixed schedule
    - Dynamic scheduling – different orders possible
      - Prioritize SW procs
        - Static or dynamic?
        - Timing-based or other?
    - Preemption of SW proc
      - By interrupt service routines
      - By other SW processes
    - Results: timing delays
      - Interference by same, higher-priority SW processes
      - Blocking
        - Non-preemptive scheduler
        - by lower-priority SW processes sharing resource with this process

1. Overview, Scope Trigger Example

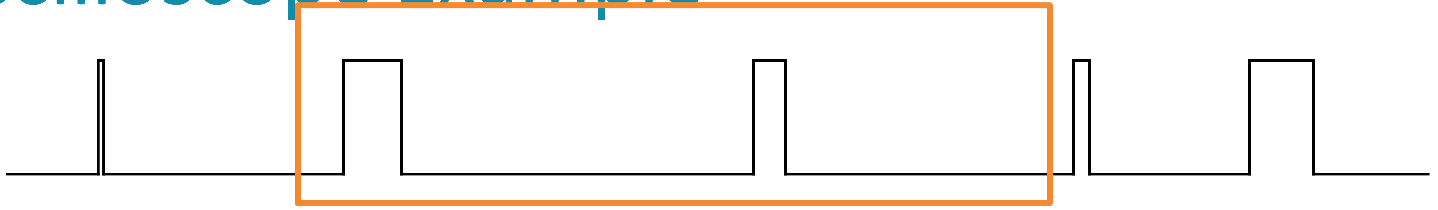
I. **Overview**
   A. **Timing requirements for I/O activities are major driver for embedded system design decisions**
   B. **May need to synchronize to event or time before doing the work (Sync and Do)**
      1. Scope trigger: detect input rising across threshold voltage, then can start sampling data
      2. Quadrature decoder: detect input A rising, then sample input B, increment or decrement count



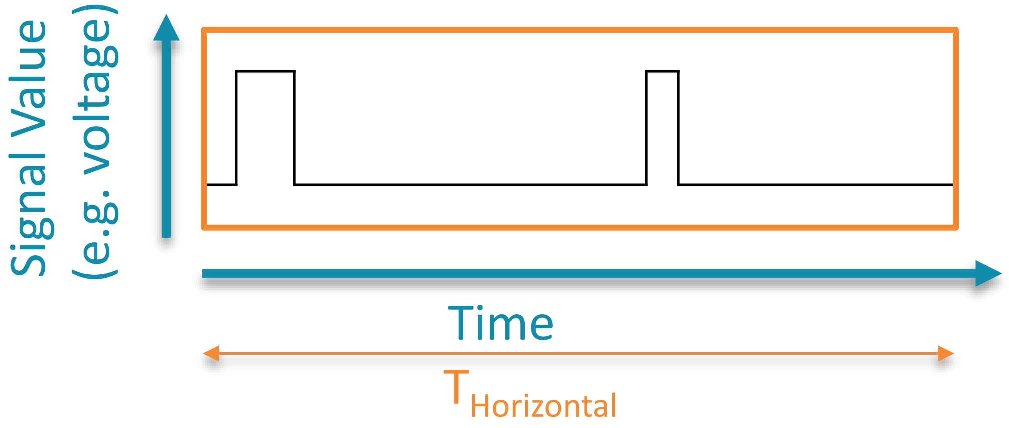
CW
CCW
A
B
pos++
pos--

NC STATE UNIVERSITY

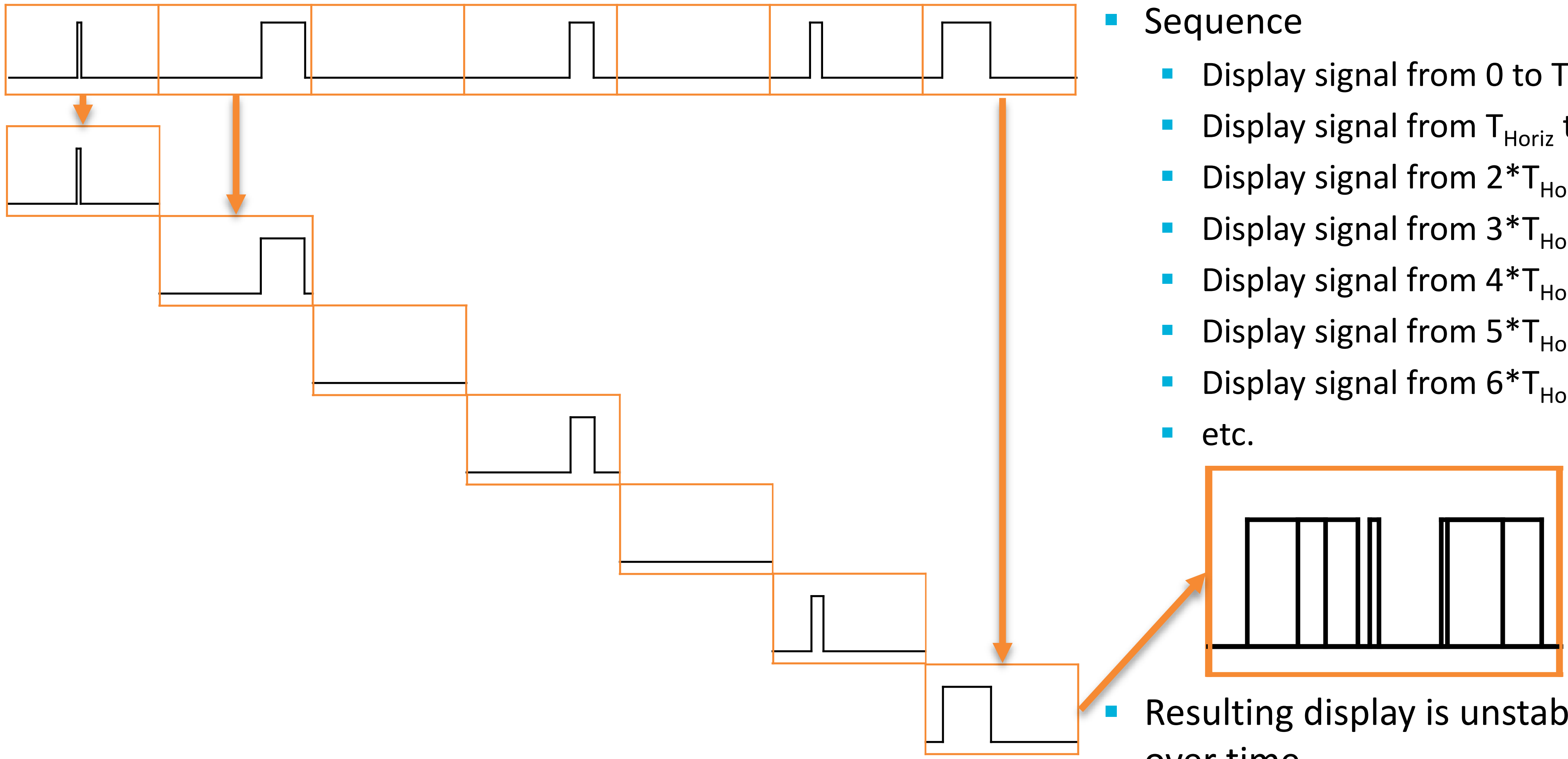
Synchronization: Simple Oscilloscope Example



- Input signal
  - Start with simple one-bit digital signal (do analog later)
  - Pulses have irregular start times, changing pulse widths
- Displaying the signal
  - Oscilloscope (“scope”) plots signal value (e.g. voltage) vertically vs. time horizontally
  - Horizontal time base determines amount of time (T_{Horiz}) represented on scope display
  - Display stability depends **timing relationship** between when scope starts displaying the signal, and when the signal changes
  - “Infinite persistence” accumulates all acquired traces on display until erase button is pressed



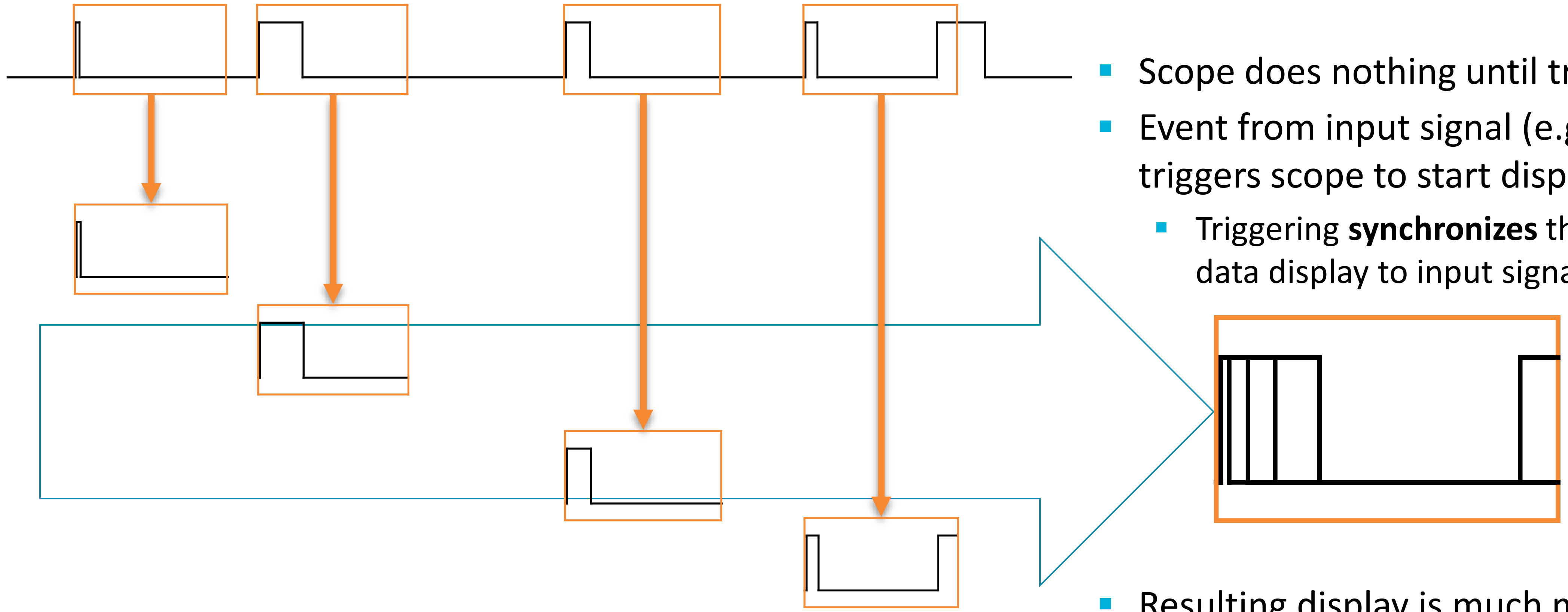
Simple Method: Display Signal Continuously



- Sequence
  - Display signal from 0 to T_{Horiz}
  - Display signal from T_{Horiz} to $2*T_{Horiz}$
  - Display signal from $2*T_{Horiz}$ to $3*T_{Horiz}$
  - Display signal from $3*T_{Horiz}$ to $4*T_{Horiz}$
  - Display signal from $4*T_{Horiz}$ to $5*T_{Horiz}$
  - Display signal from $5*T_{Horiz}$ to $6*T_{Horiz}$
  - Display signal from $6*T_{Horiz}$ to $7*T_{Horiz}$
  - etc.
- Resulting display is unstable, jumps around over time.

NC STATE UNIVERSITY

Stabilize Display with Triggering



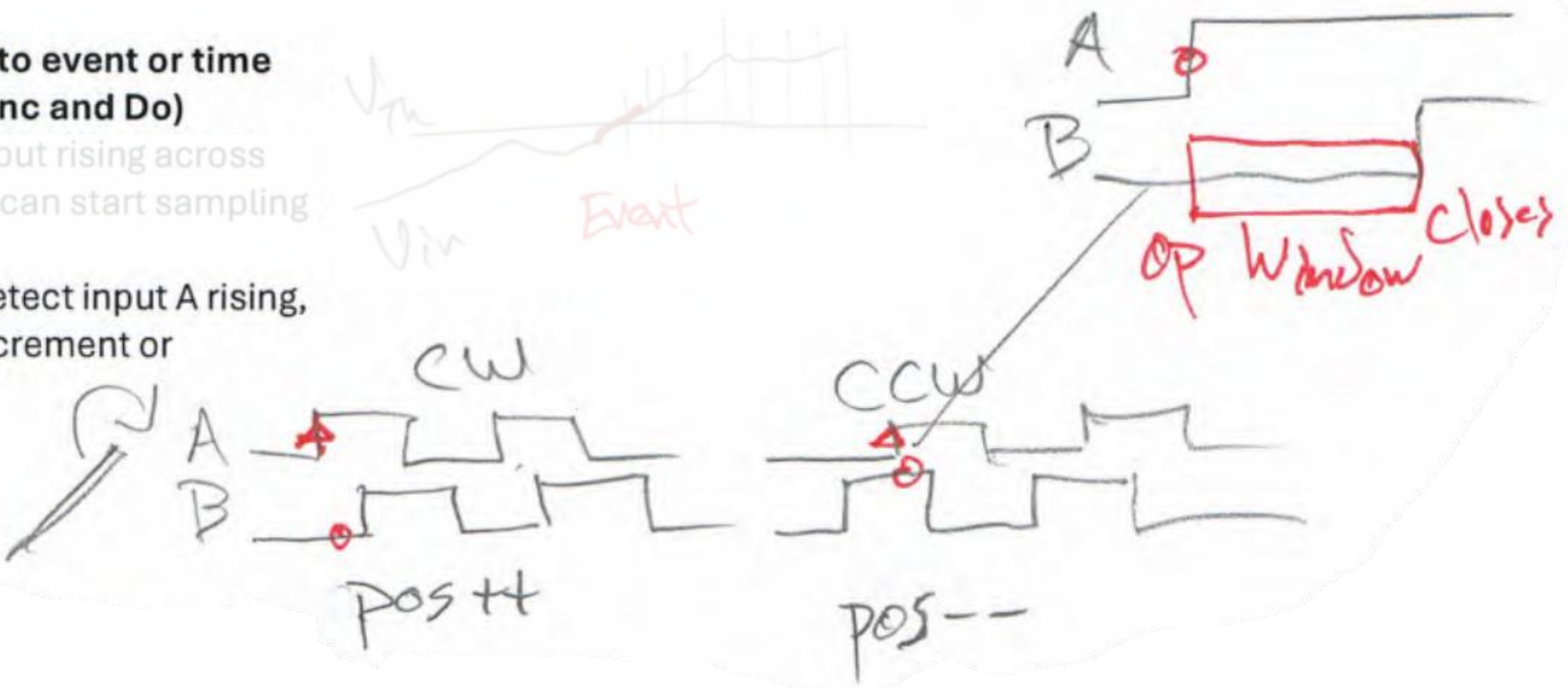
- Scope does nothing until triggered
- Event from input signal (e.g. 0 to 1 edge) triggers scope to start displaying signal
  - Triggering **synchronizes** the scope's start of data display to input signal event
- Resulting display is much more stable
  - Rising edge of signal is stable
    - Except for last acquisition, where time between rising edges < T_{horiz}
  - Falling edge is not stable, because pulse width varies

NC STATE UNIVERSITY

Quadrature Decoder Example

I. **Overview**
   A. Timing requirements for I/O activities are major driver for embedded system design decisions
   B. **May need to synchronize to event or time before doing the work (Sync and Do)**
      1. Scope trigger: detect input rising across threshold voltage, then can start sampling data
      2. Quadrature decoder: detect input A rising, then sample input B, increment or decrement count

V_{Th}
Event
Vin



Process Chain for I/O Activities

II. **Understanding Process Chain for I/O Activities**

A. **Synchronize with something**

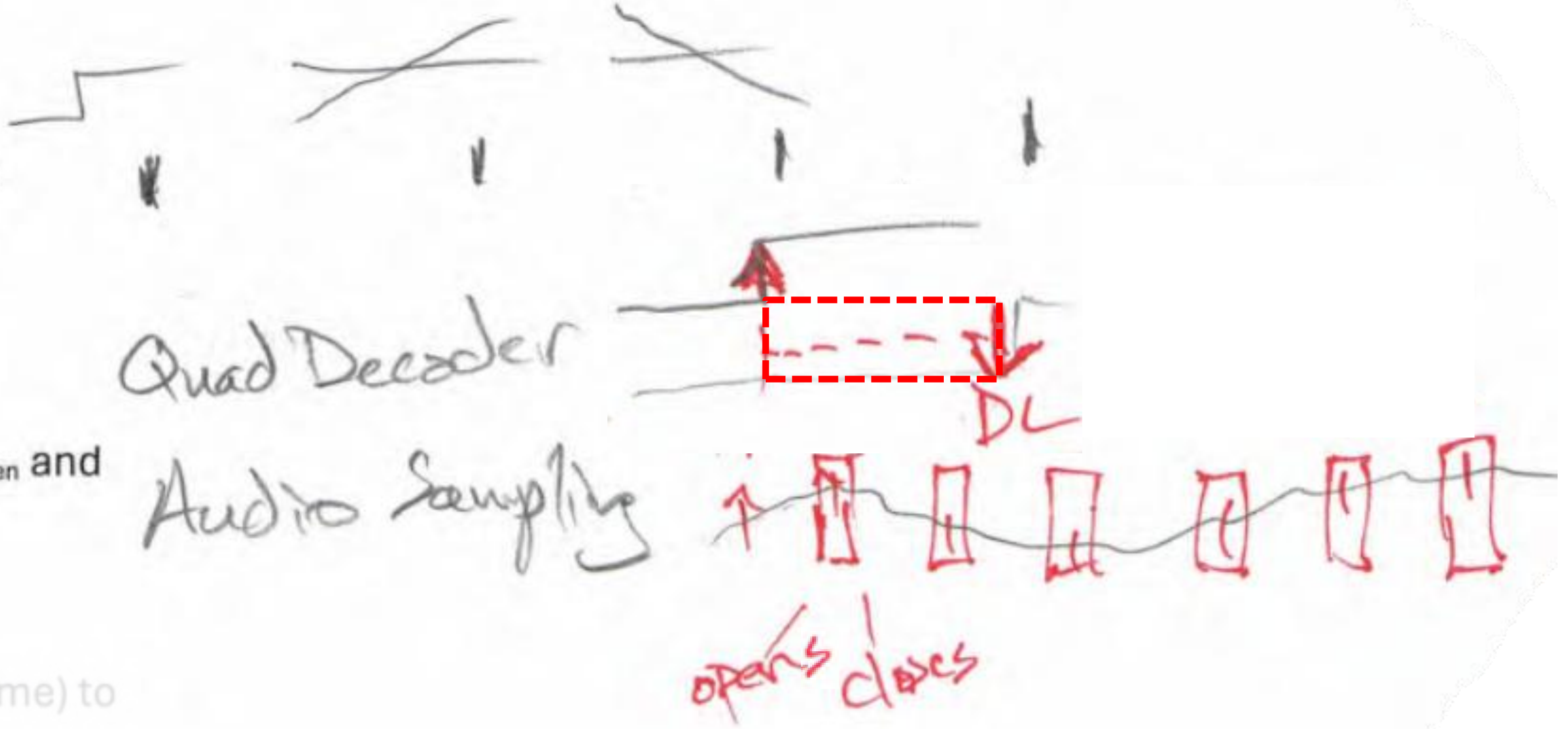
1. Types
   a. Event-Triggered: Detect event
   b. Time-Triggered: Await target time

B. **Do processing in response**

1. Timing requirements:
   a. Simple deadline: within T_{DL} after event/time
   b. Window deadline: Between T_{DL_Open} and T_{DL_Close} of event/time

C. Repeat?

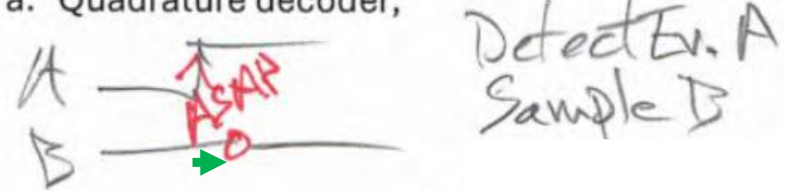
1. May have burst or sequence of I/O activities, so next will sync (event or time) to next part or do it immediately/ASAP
2. Examples inputs:

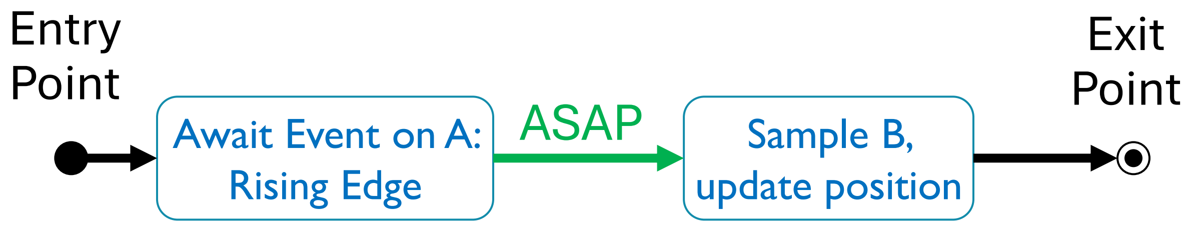


NC STATE UNIVERSITY

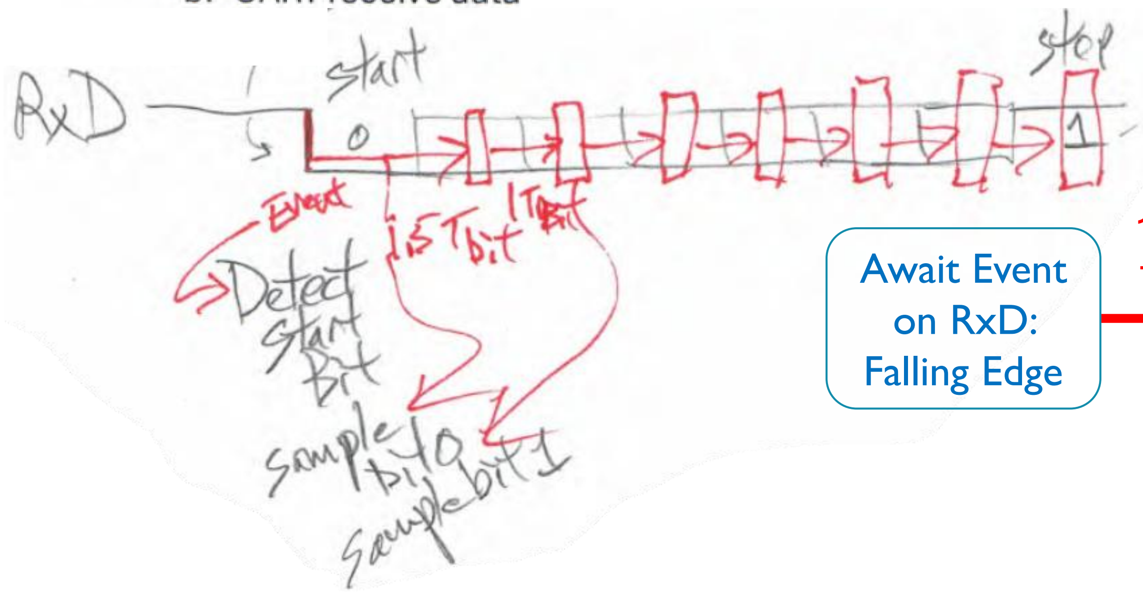
Sequence of I/O Activities

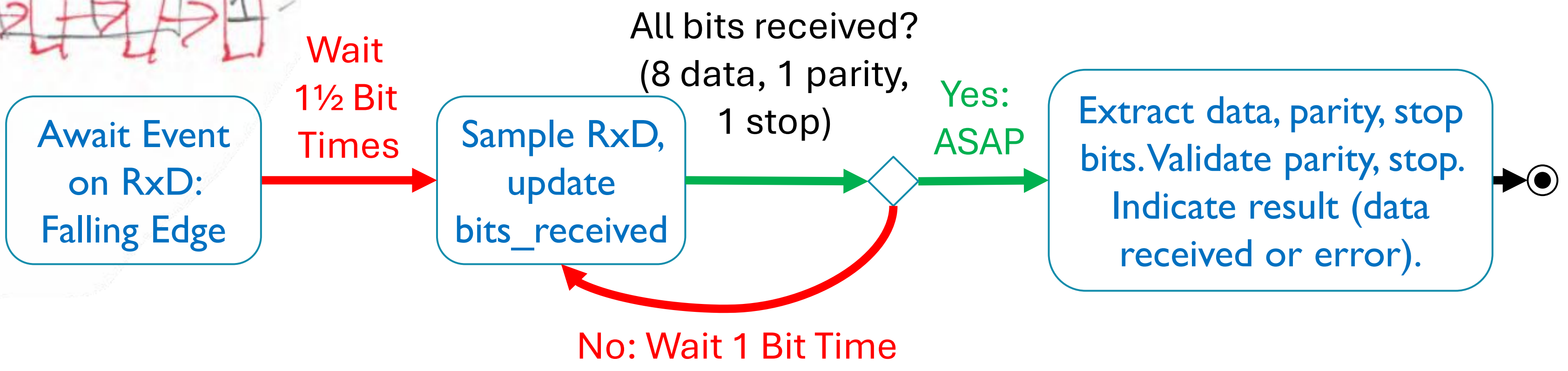
C. **Repeat?**
1. May have burst or sequence of I/O activities, so next will sync (event or time) to next part or do it immediately/ASAP
2. Examples inputs:
   a. Quadrature decoder,





   b. UART receive data




Await Event on RxD: Falling Edge
Wait 1½ Bit Times
Sample RxD, update bits_received
All bits received? (8 data, 1 parity, 1 stop)
Yes: ASAP
No: Wait 1 Bit Time
Extract data, parity, stop bits. Validate parity, stop. Indicate result (data received or error).

How to Synchronize?

- All Hardware
  - HW sync with signals
- Some Software, Some Hardware
  - Don’t share CPU
  - Share CPU with software scheduling
  - HW event detection
  - HW event detection + Interrupt system

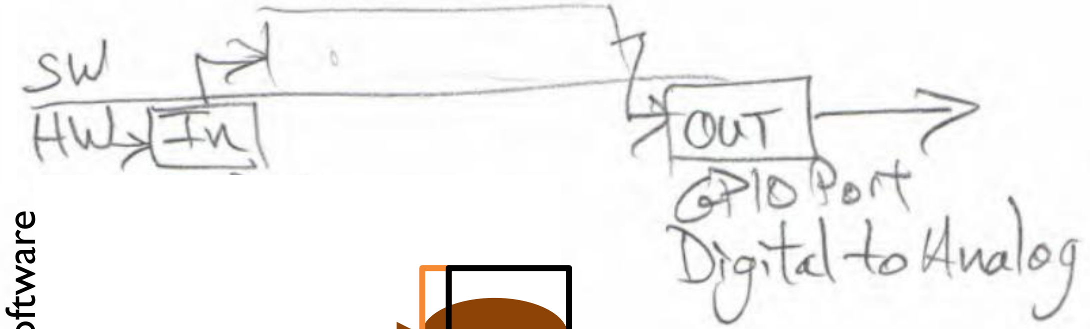
III. How to Synchronize?

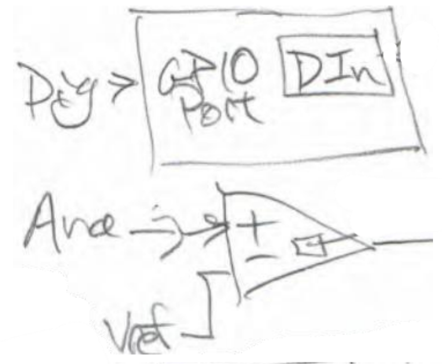
A. All Hardware

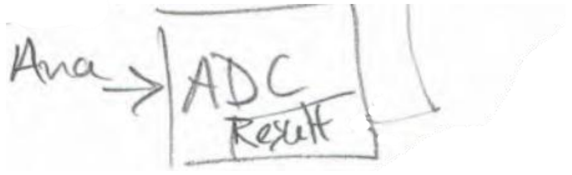
1. Easy: Dedicated signals

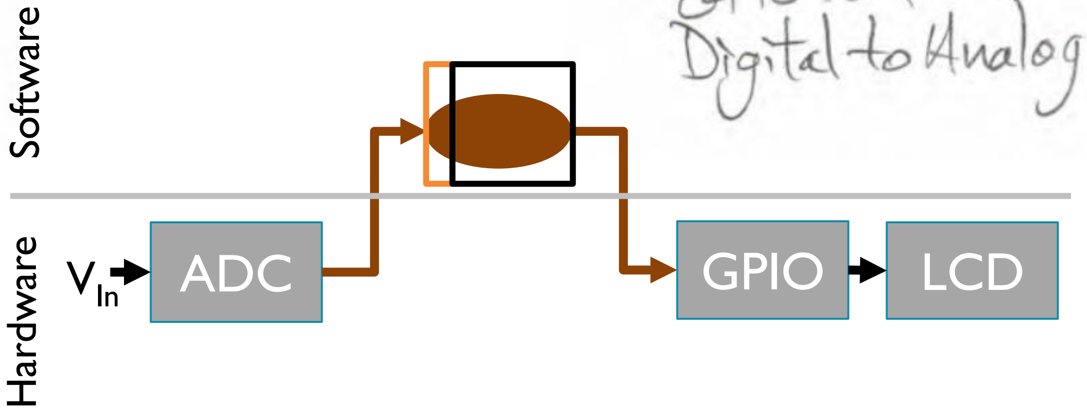
B. Some Software

1. HW/SW allocation and processing chain. SW polls hardware (input peripheral)

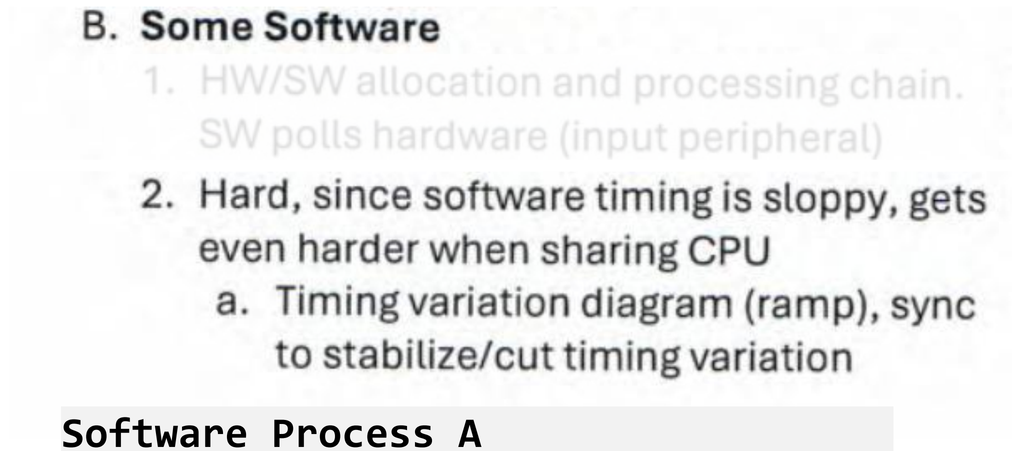








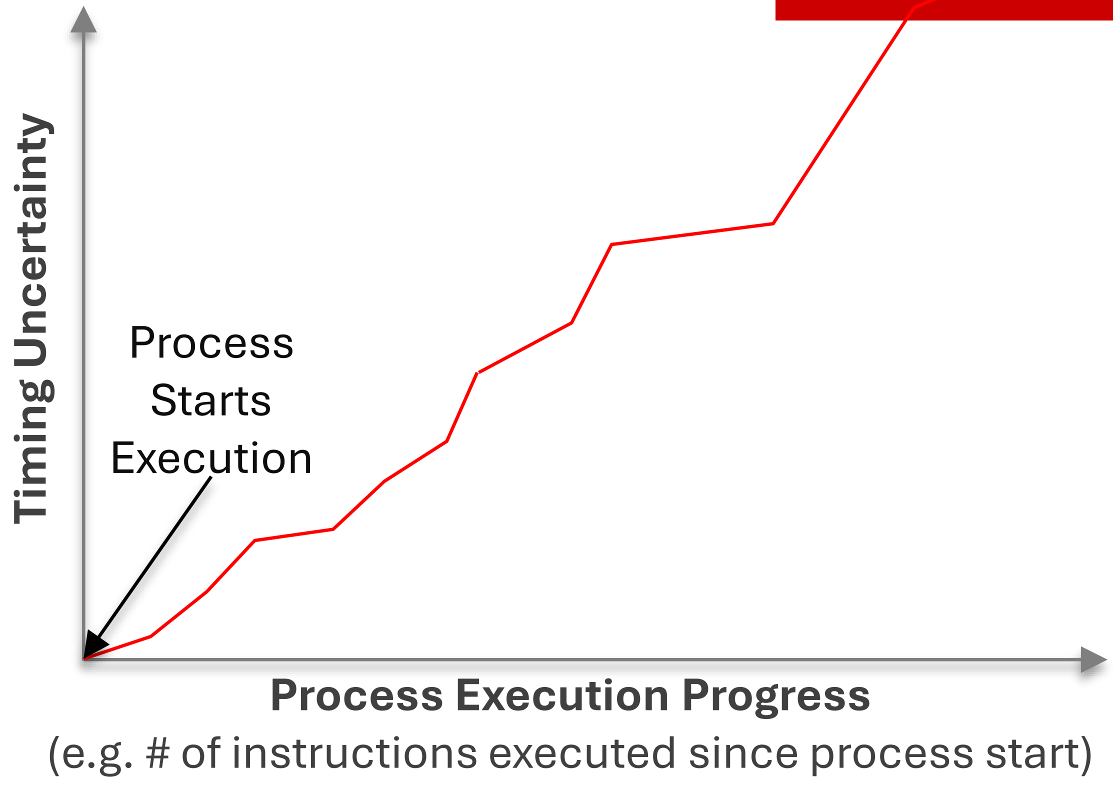
```
Software Process A
// Detector/Synchronizer
while (ADC->Result < V_Threshold)
  ;
// Handler part of process goes here
```



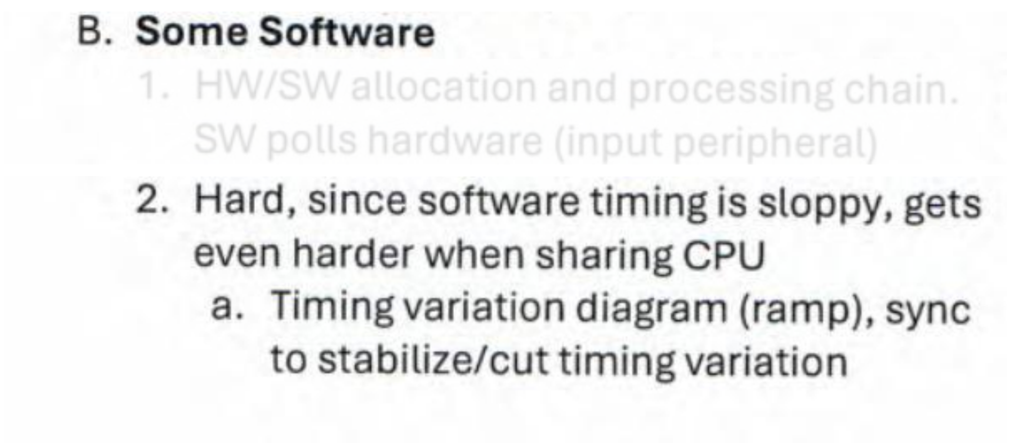
B. **Some Software**

1. HW/SW allocation and processing chain. SW polls hardware (input peripheral)
2. Hard, since software timing is sloppy, gets even harder when sharing CPU
   a. Timing variation diagram (ramp), sync to stabilize/cut timing variation

```
Software Process A
…
// Handler part of process
x = 0;
for (n=0; n<NS; n++) {
  r = ADC->Result; // When does
    this instruction first execute?
  y = scale(r);
  LCD_Plot(x++,y);
}
```

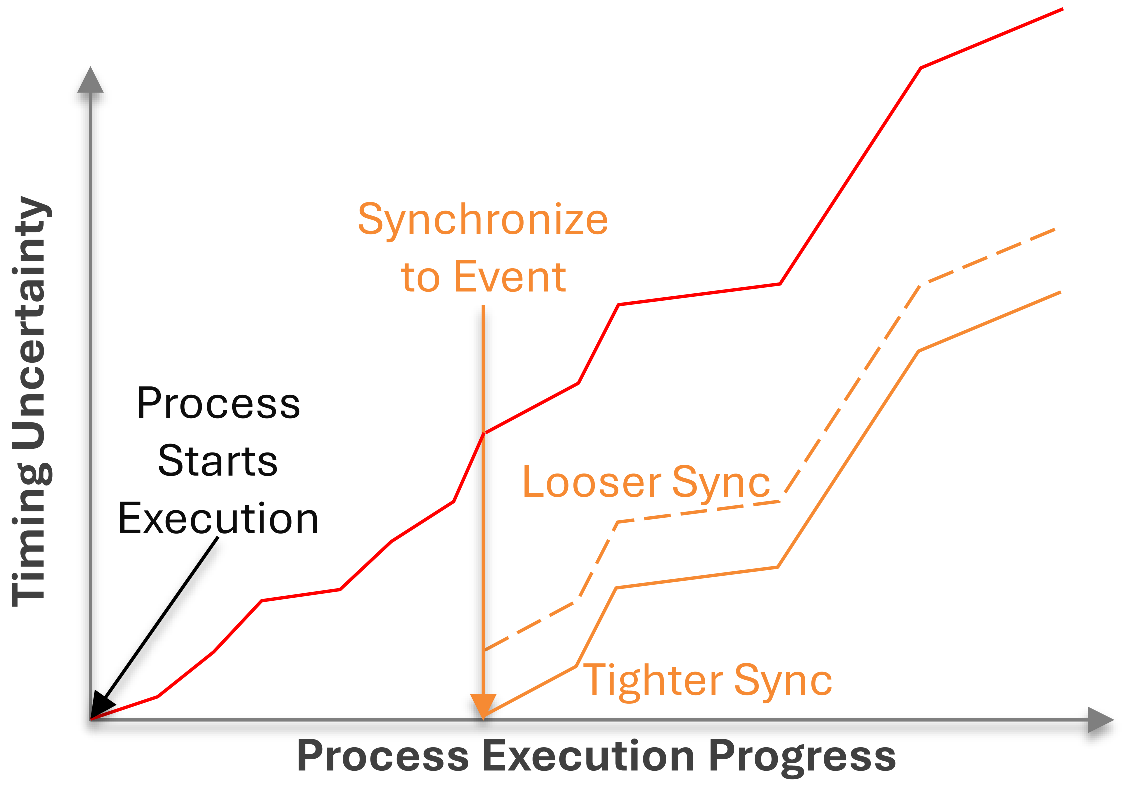


- **When** a specific instruction executes (relative to start of process execution) depends on
  - Past behavior: conditional control flow (ifs, loops, etc.), interrupts, other software processes, scheduling approach
  - Time process started relative to to relevant event (e.g. voltage crosses trigger level for scope)
- **Timing uncertainty** for an instruction is latest exec. time – earliest exec. time

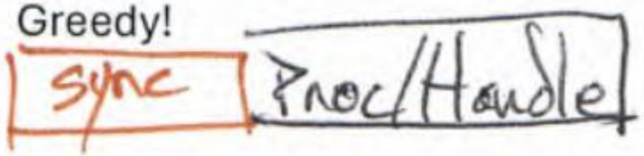


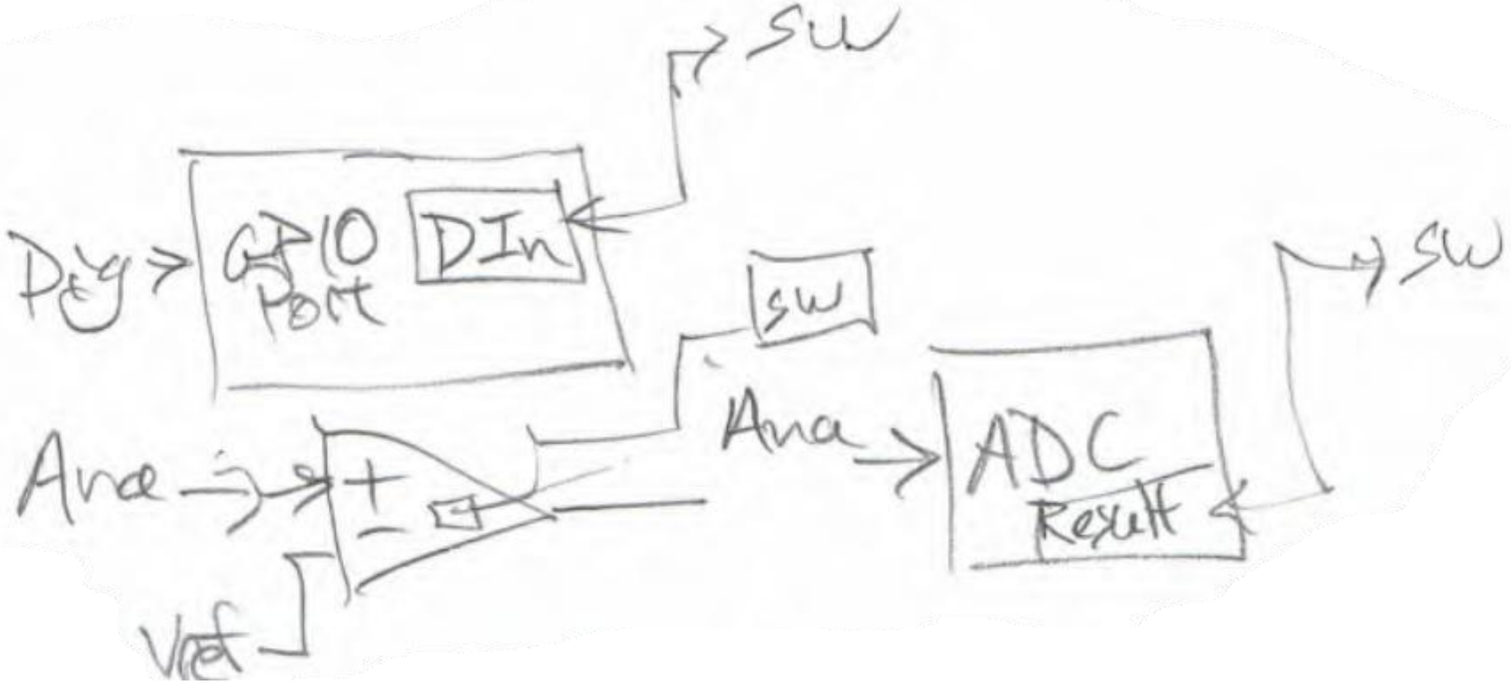
B. **Some Software**
   1. HW/SW allocation and processing chain. SW polls hardware (input peripheral)
   2. Hard, since software timing is sloppy, gets even harder when sharing CPU
      a. Timing variation diagram (ramp), sync to stabilize/cut timing variation

```
Software Process A
…
// Detector/Synchronizer built into process
while (ADC->Result < V_Threshold)
  ;
// Handler part of process
x = 0;
for (n=0; n<NS; n++) {
  r = ADC->Result; // When does this
                instruction first execute?
  y = scale(r);
  LCD_Plot(x++,y);
}
```



3. Start simple: Not sharing CPU
   a. Detect with blocking SW loop polling (busy-waiting)
   b. Responsiveness
   c. Greedy!

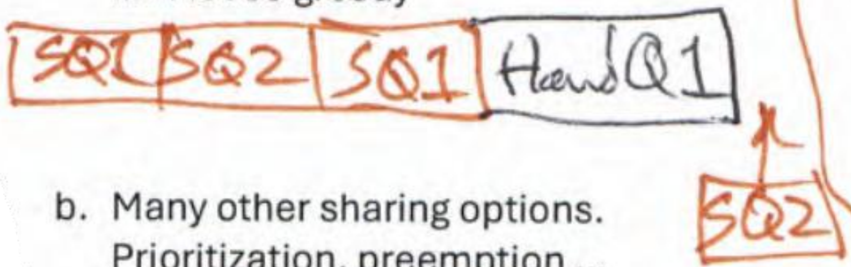




4. Share CPU with software scheduling method
   a. Round-Robin Loop/Cyclic Exec.
      i. Detector doesn't block, but take turns with other code (possibly multiple detectors)

non-blocking

      ii. Responsiveness
      iii. Not so greedy



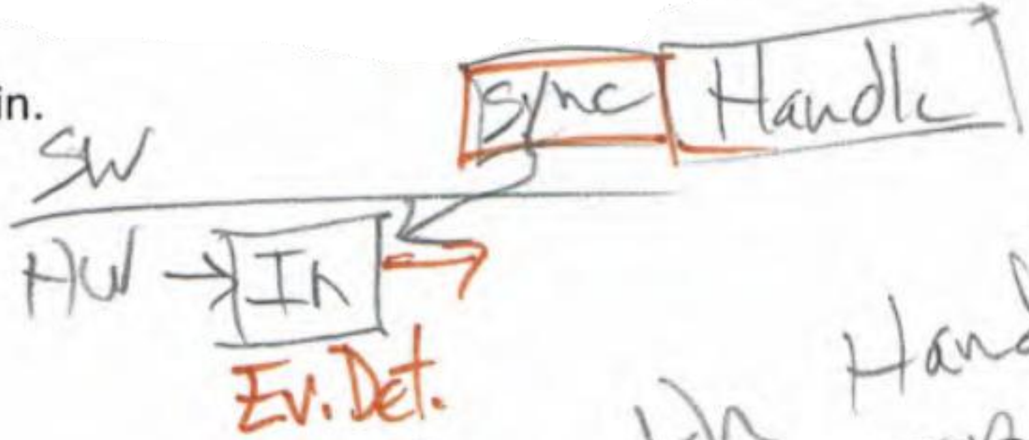
```
if (QA1==1)&&(QA1_prev==0)    <- Q1A..
    Handle Q1
if (QA2==1)&&(QA2_prev==0)    <- Q2A.
    Handle Q2
```

   b. Many other sharing options. Prioritization, preemption ...
      i. + Schedule, dispatch.

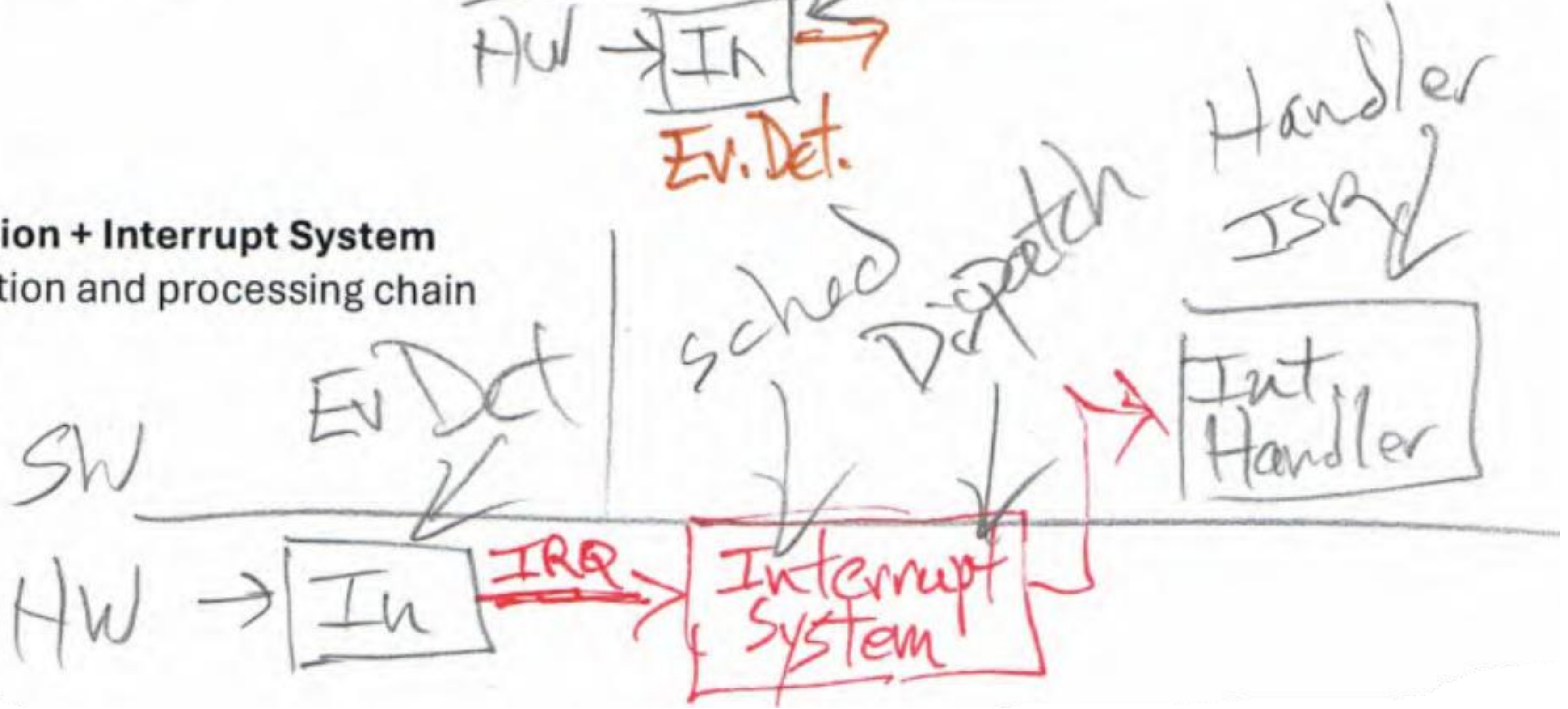
Run what SWProc?

Run it

C. **HW Event Detection**
1. Hardware peripheral detects event
2. HW/SW allocation and processing chain. SW polls event detector



D. **HW Event Detection + Interrupt System**
1. HW/SW allocation and processing chain
2. Handler runs



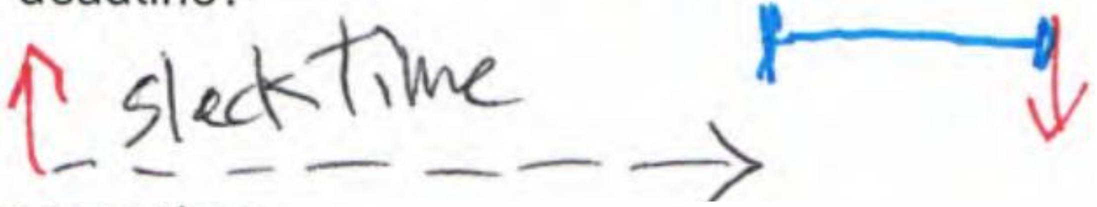
IV. Basic Timing Analysis

- Approaches
- Complications from sharing CPU

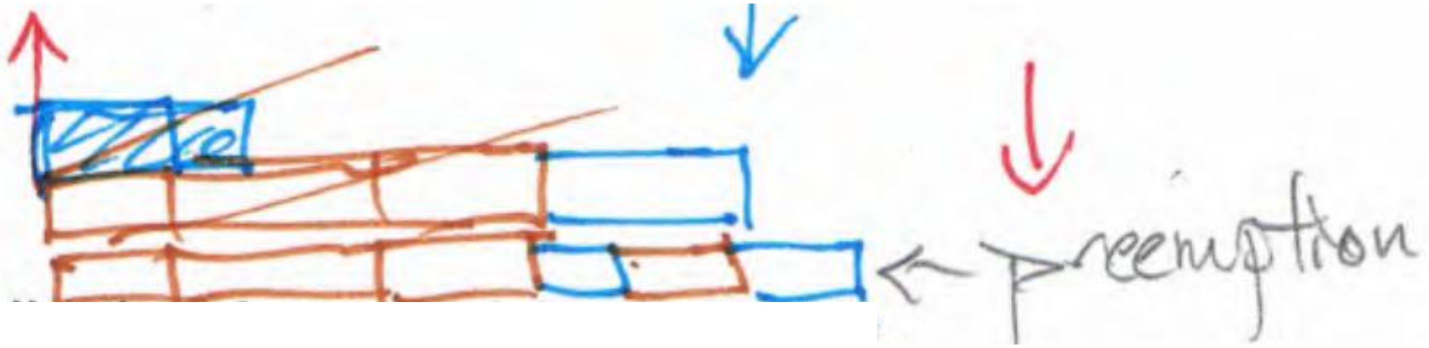
IV. Basic Timing Analysis

A. Approaches

1. Slack time
   a. How late can process start and meet deadline?



2. Response time
   a. When will this process finish, considering effects of other processes in system

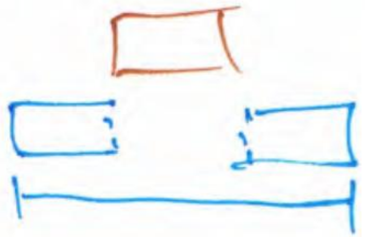


B. **Complications from scheduler sharing the CPU among SW processes scheduler**

1. Basic: static fixed schedule
2. Dynamic scheduling – different orders possible
   a. Prioritize SW procs
      i. Static or dynamic?
      ii. Timing-based or other?
3. Preemption of SW proc
   a. By interrupt service routines
   b. By other SW processes
4. Results: timing delays
   a. Interference by same, higher-priority SW processes
   b. Blocking
      i. Non-preemptive scheduler
      ii. by lower-priority SW processes sharing resource with this process

A B C D A B C D

A B A C A D A B A C A D



Priority H V

H Prio Task - Interference

Task

Blocking Low-Prio Task - Shared Resource

Schedules (Not preemptive)