NC STATE UNIVERSITY

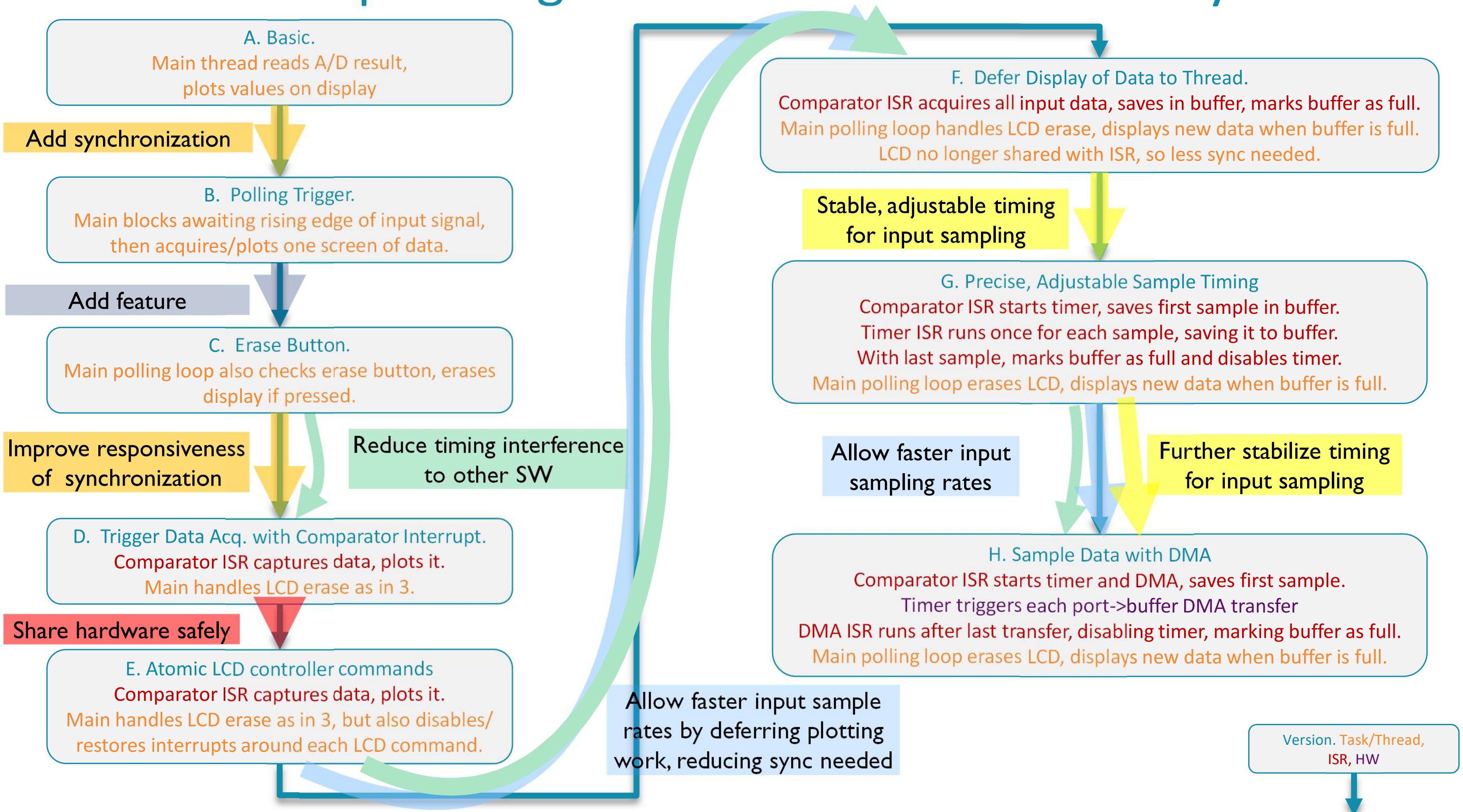
# 18: Scope Designs on Platform 1

v3

NC STATE UNIVERSITY

# Analog Scope Example

# Overview of Scope Design Evolution: What and Why



NC STATE UNIVERSITY

# Detailed Overview of Scope Designs: What and Why

**A. Basic.**
Main thread reads A/D result, plots values on display

*Main loop starts scope work (data acquisition and display) immediately (no sync).*

*Input signal displayed*
*Unstable timing of signal on display*

*Add Triggering: Synchronize data acq/display to input signal event*

**B. Polling Trigger.**
Main blocks in polling loop until detecting rising edge of input signal, then acquires/plots one screen of data.

*Main loop syncs scope work to signal edge.*

*Stable display*
*Want to erase display manually*

*Add erase button as digital port input bit.*

**C. Erase Button.**
Main polling loop also checks erase button, erases display if pressed.

*Main loop syncs scope work to signal edge, erase work to button press.*

*On-demand erasing*
*Slow trigger response while erasing*

*Use analog comparator interrupt to detect trigger condition*

**D. Trigger Data Acq. with Comparator Interrupt.**
Comparator ISR captures data, plots it.
Main handles LCD erase as in 3.

*Interrupt system & Comparator sync scope work/ISR (data acq. & plot) to signal edge.*
*Main loop syncs erase work to button press*

*Very fast trigger response*
*Pixels lost if triggered while erase code is sending an LCD command*

*Protect LCD commands for mutually exclusive execution*

**E. Atomic LCD controller commands**
Comparator ISR captures data, plots it.
Main handles LCD erase as in 3, but also disables/ restores interrupts around each LCD command.

*(As in 4) Int. Sys. & Comparator sync scope work/ISR (data acq. & plot) to signal edge.*
*Main loop syncs erase work to button press, prevents interrupt during LCD command.*

*Trigger during erase erases part of new signal*
*Max. sample rate limited by LCD update speed*

*Defer non-urgent work until after ISR using buffer.*

**F. Defer Display of Data to Thread.**
Comparator ISR acquires screen-full of input data, saves in buffer, marks buffer as full.
Main polling loop handles LCD erase, displays new data when buffer is full.
LCD no longer shared with ISR, so don't need to disable interrupt per LCD command.

*Int. Sys. & Comparator sync scope work (ISR (data acq. & save) to signal edge.*
*Main loop syncs display update to full buffer, erase work to button press.*

*Much faster sample rate*
*Want to control sample rate*

*After port edge trigger, use timer peripheral to trigger each data sample.*

**G. Precise, Adjustable Sample Timing**
Comparator ISR starts timer, saves first sample in buffer.
Timer ISR runs once for each sample, saving it to buffer. With last sample, marks buffer as full and disables timer.
Main polling loop handles LCD erase, displays new data when buffer is full.

*Int. Sys. & Comparator sync start of data acquisition to signal edge.*
*Int. Sys. & Timer sync each data sample and end of acq. to timer.*
*Main loop syncs display update to full buffer, erase work to button.*

*Stable, HW-controlled sample rate*
*CPU interrupt overhead delays other processing, limits max. sample rate*

*Offload sampling/buffering from CPU software (ISR) to hardware (DMA)*

**H. Sample Data with DMA**
Comparator ISR starts timer and DMA, saves first sample.
Timer triggers each port->buffer DMA transfer
DMA ISR runs after last transfer, disabling timer and marking buffer as full.
Main polling loop handles LCD erase, displays new data when buffer is full.

*Int. Sys. & Comparator sync **start** of data acquisition to signal edge.*
*DMA system & Timer sync each data sample to timer overflow.*
*Int. Sys. & Comparator sync **end** of data acq. to last sample transfer.*
*Main loop syncs display update to full buffer, erase work to button.*

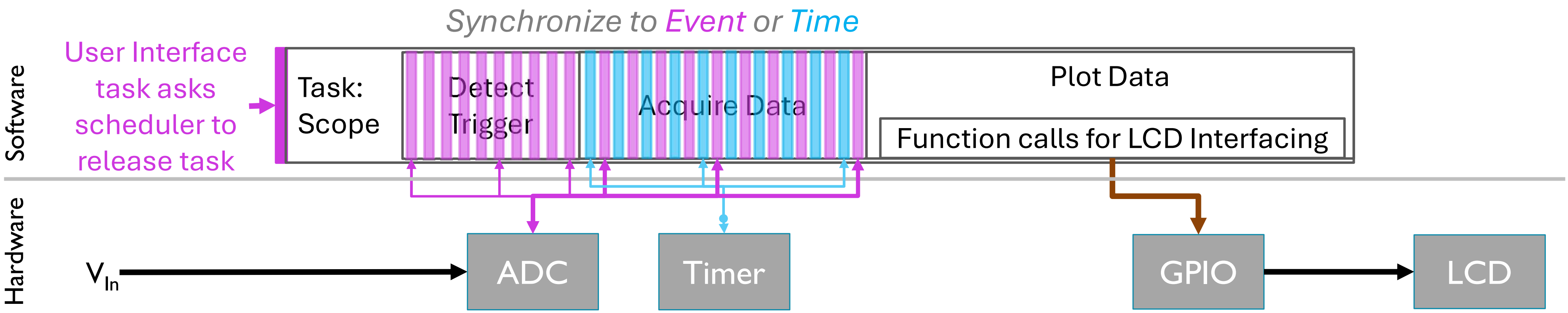
*Main limitation? TBD*

Version. Task/Thread, ISR, HW

*Process Scheduling, Synchronization and Communication Highlights*

*Performance, behavior.*
*Problem(s).*
*Changes to solve problem(s).*

# I/O-Driven Synchronization Requirements for Scope



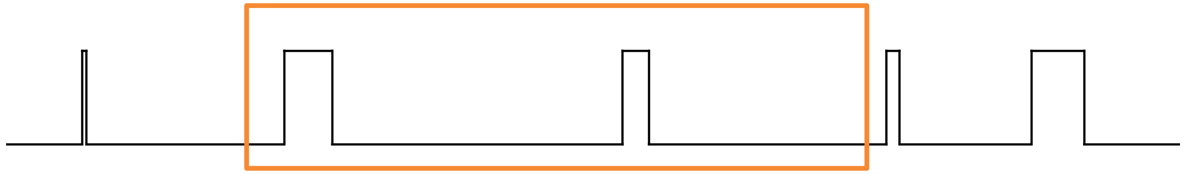

- First synchronization
  - Task starts after UI asks scheduler to release it
- Detect Trigger: 2nd set of synchronizations
  - Start A/D conversion
  - Sync to A/D conversion complete, read result, analyze, decide if trigger condition met or need to repeat
- Acquire Data: 3rd set of synchronizations
  - Sync to time (periodic) for next sample, start A/D conversion
  - Sync to A/D conversion complete, read result, analyze, decide if trigger condition met or need to repeat

- Plot data on LCD: No sync needed
  - LCD controller faster than software & GPIO interface
- Simplify for design examples A-G
  - Eliminate sync to A/D conversion by assuming fast ADC: one A/D conversion per CPU instruction, non-stop conversions.

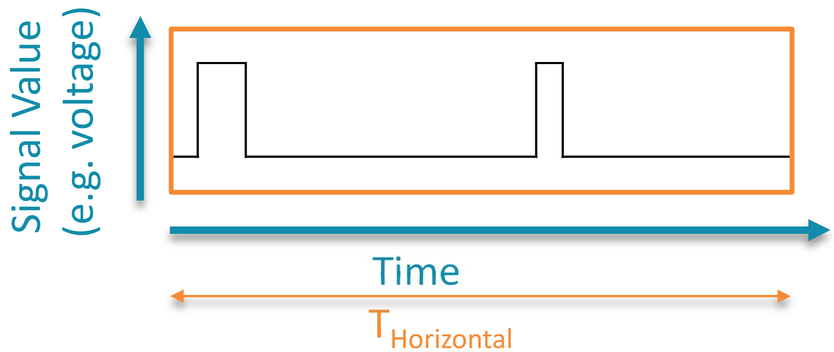
NC STATE UNIVERSITY

# Scope Triggering: Synchronizing **Data Acquisition** to **Input Signal Activity**

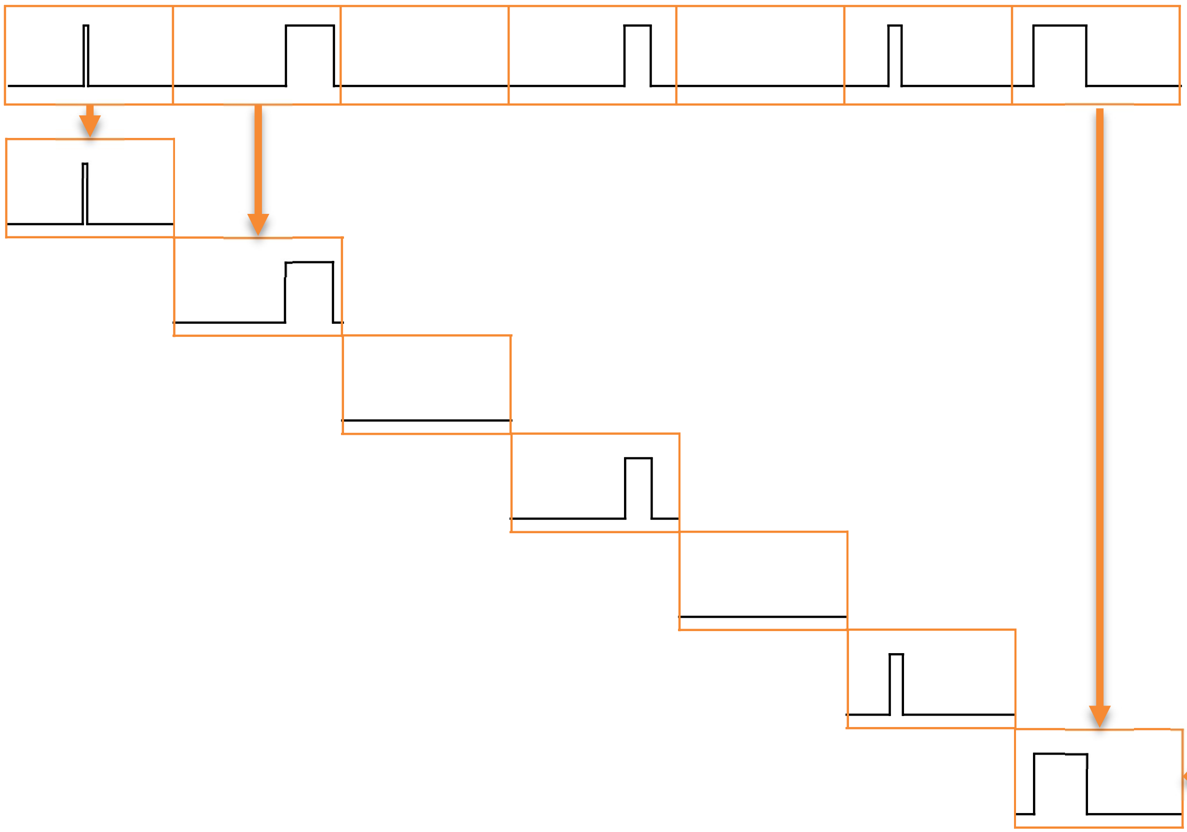
# Synchronization: Simple Oscilloscope Example



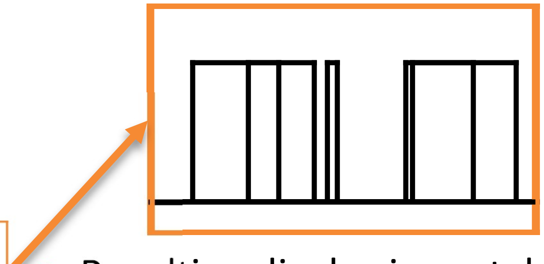
- Input signal
  - Pulses with irregular start times, changing pulse widths
- Displaying the signal
  - Oscilloscope ("scope") plots signal value (e.g. voltage) vertically vs. time horizontally
  - Horizontal time base determines amount of time ($T_{Horiz}$) represented on scope display
  - Display stability 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

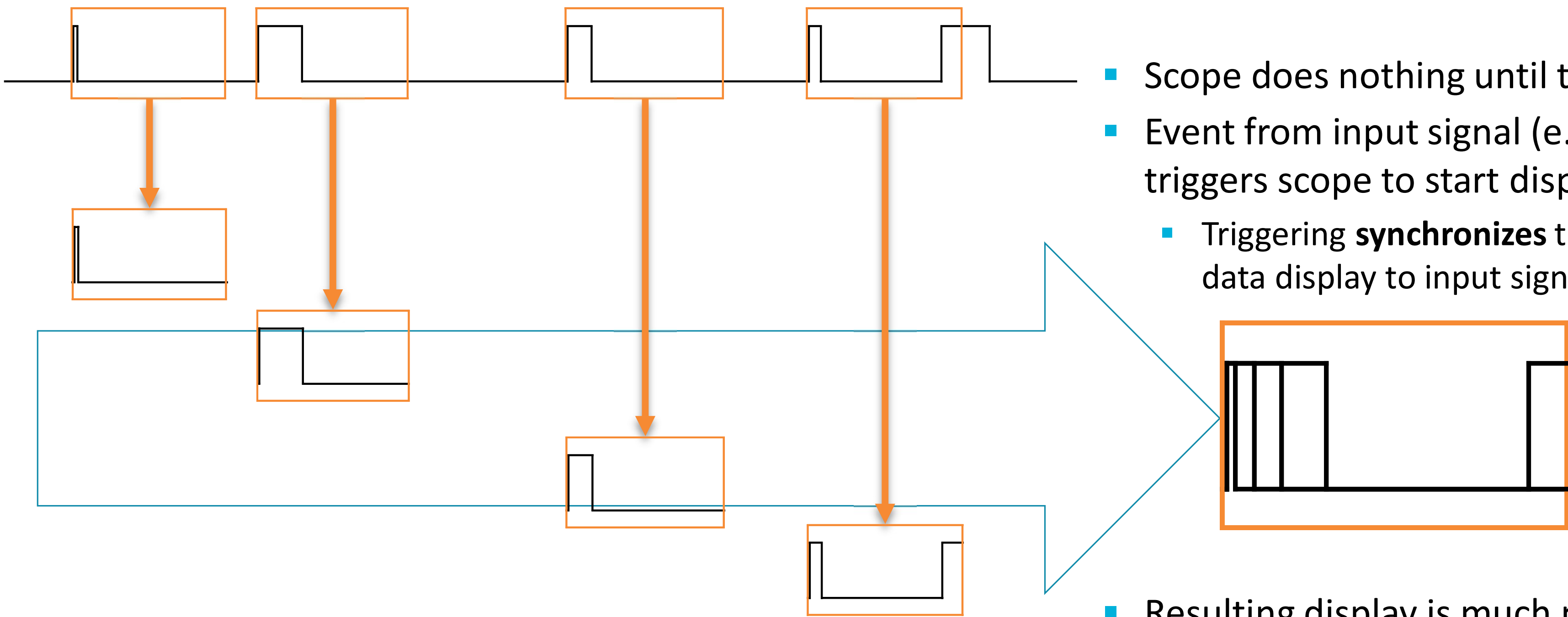


- Sequence
  - Display signal from 0 to $T_{Horiz}$
  - Display signal from $T_{Horiz}$ to $2*T_{Horiz}$
  - Display signal from $2*T_{Horiz}$ to $3*T_{Horiz}$
  - Display signal from $3*T_{Horiz}$ to $4*T_{Horiz}$
  - Display signal from $4*T_{Horiz}$ to $5*T_{Horiz}$
  - Display signal from $5*T_{Horiz}$ to $6*T_{Horiz}$
  - Display signal from $6*T_{Horiz}$ to $7*T_{Horiz}$
  - etc.



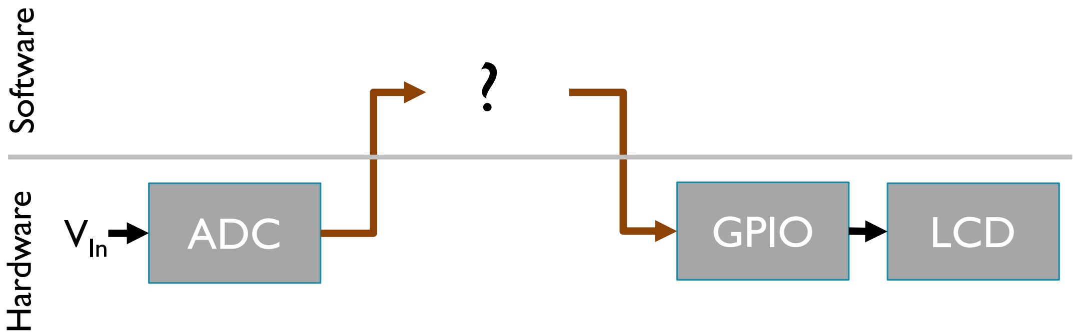
- Resulting display is unstable, jumps around over time.

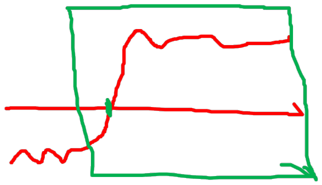
# Stabilize Display with Triggering



- 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

# Implementation of Scope with Triggering



- Behavior
  - Display signal voltage on LCD, synchronized to signal's rising edge through threshold
  - Simplify: Ignore erasing previous acquisition from LCD. Ignore user controls
- Hardware Components
  - Analog to Digital converter: Fast enough to keep up with MCU: 48 Msamples/second. (NOT the ADC on the KL25Z)
  - Microcontroller: KL25Z
  - LCD: 320x240 display with controller
- Basic flow of operations
  - **Wait for/detect** $V_{in}$ rising through $V_{threshold}$
    - Rising Edge: $V_{in}$(previous sample) < $V_{threshold}$ AND $V_{in}$(current sample) >= $V_{threshold}$
  - Loop to **Acquire** N data samples and **display** them
    - Sample ADC: acquisition[n] = ADC_data
    - Scale acquisition[n], plot on LCD until reaching end of display

NC STATE UNIVERSITY

# Scope Design Version: Process and Synchronization Summary

| Process | Erase Display | | Acquire and Plot Data | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Design Version** | **Sync: Trigger Erase** | **Erase Display** | **Sync: Trigger Scope** | **Sync: Schedule A/D Conversion Start** | **Start A/D Conversion** | **Read ADC Data** | **Sync: Indicate Data Acq. Done** | **Plot Data** |
| A | - | | - | - | Main thread | | | |
| B | - | | Main thread | - | Main thread | | | |
| C | Main thread | | Main thread | - | Main thread | | | |
| D, E | Main thread | | Comparator | - | Comparator ISR | | | |
| F | Main thread | | Comparator | - | Comparator ISR | | | Main thread |
| G | Main thread | | Comparator | Timer | Timer ISR | | | Main thread |
| H | Main thread | | Comparator | Timer | Timer | DMA | DMA ISR | Main thread |

| |
|---|
| Software: Thread |
| Software: ISR / IRQ Handler |
| Hardware: Peripheral |

# Scope Example Processes: Alternate View

| Ver. | Software | | | | Hardware | | |
|---|---|---|---|---|---|---|---|
| | Threads | ISRs | | | | | |
| | Main | Comparator | Timer | DMA | Comparator | Timer | DMA |
| A | Sample, Display | | | | | | |
| B | Detect Trigger Condition, Sample, Display | | | | | | |
| C | Detect Trigger Condition, Sample, Update Display, Erase Display | | | | | | |
| D, E | Erase Display | Take Sample, Update Display | | | Detect Trigger Condition | | |
| F | Erase  Display, Update Display | Take Sample | | | Detect Trigger Condition | | |
| G | Erase Display, Update Display | | Take Sample | | Detect Trigger Condition | Schedule Sample | |
| H | Erase Display, Update Display | | | | Detect Trigger Condition | Schedule Sample | Take Sample |

NC STATE UNIVERSITY

# Diagram Syntax

**Boxes: Processes, code, and data objects**

Do Work Process (thread or handler (interrupt service routine))

Synchronization Code (includes scheduling, dispatching at this level)

Interrupt System (includes scheduling, dispatching at this level)

Synchronization data object

Data buffer in architecturally-visible memory,available to software and hardware

**Arrows: Data and Synchronization (Control) Flows**

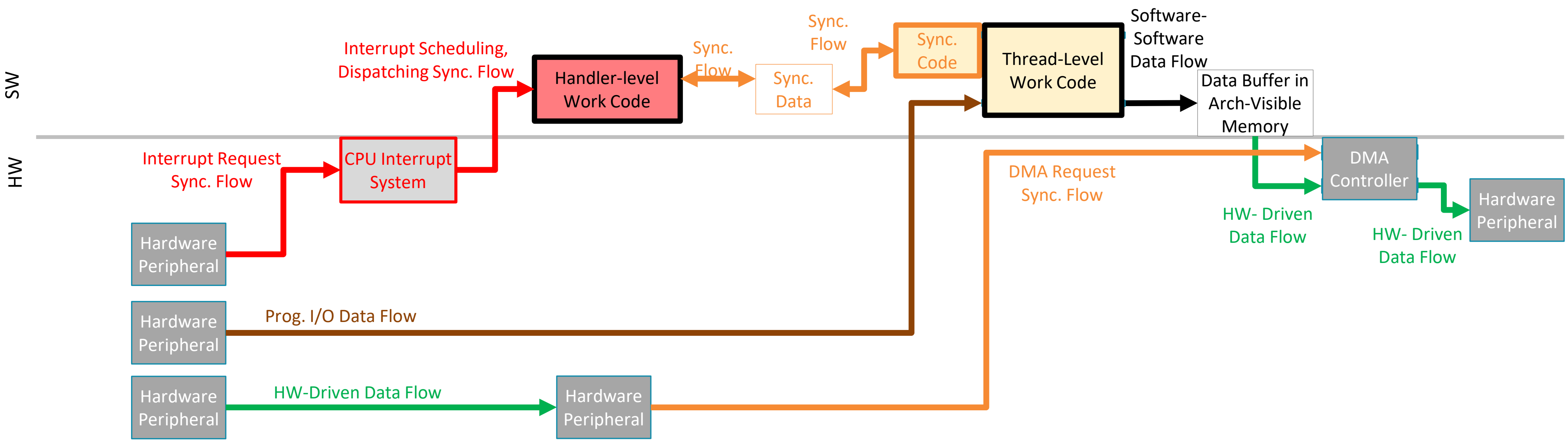
Software activity data flow

Programmed I/O data flow (software-driven)

Hardware activity data flow (hardware-driven)

Synchronization flow

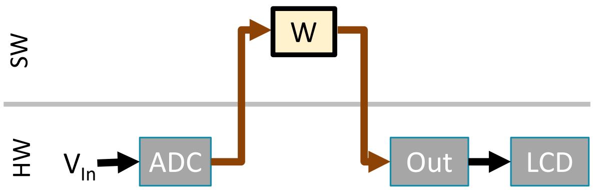
Interrupt flow



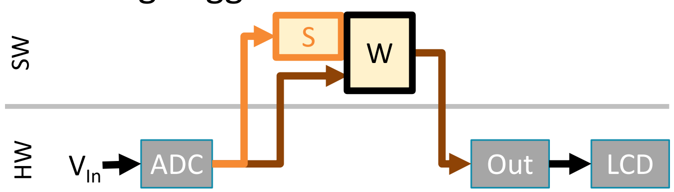
# Scope Evolution

*Synchronous Input for Trigger Detection*

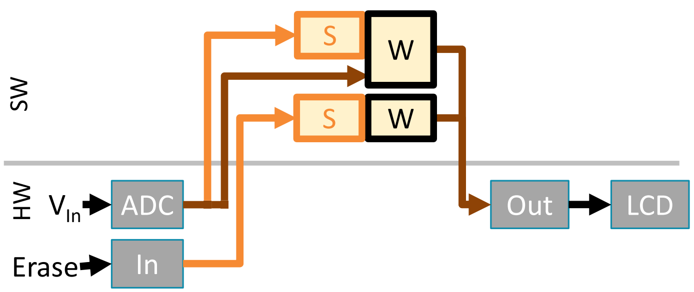
## A. Basic



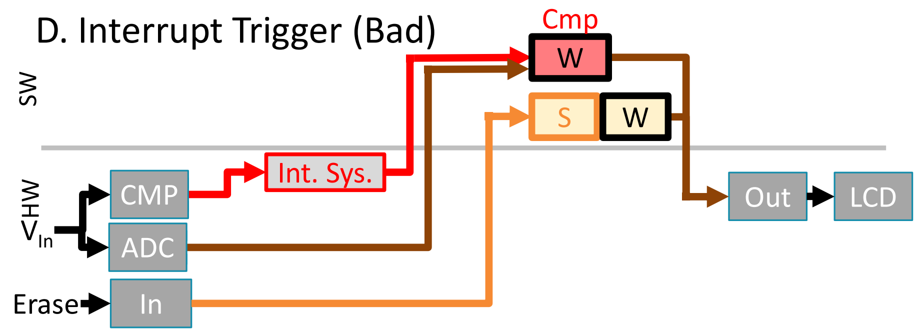

## B. Polling Trigger



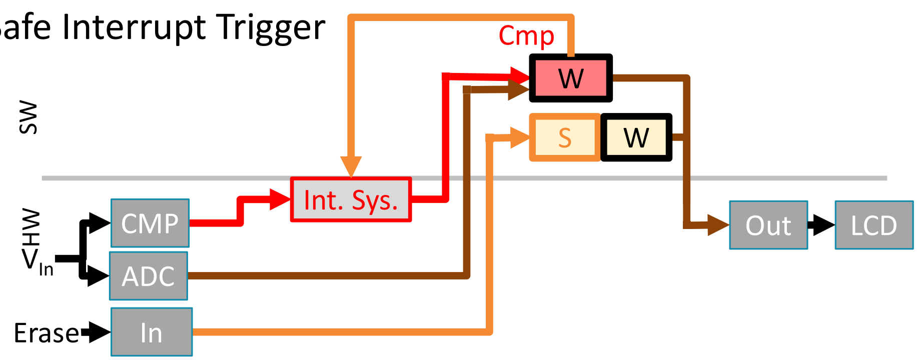

## C. Erase Button



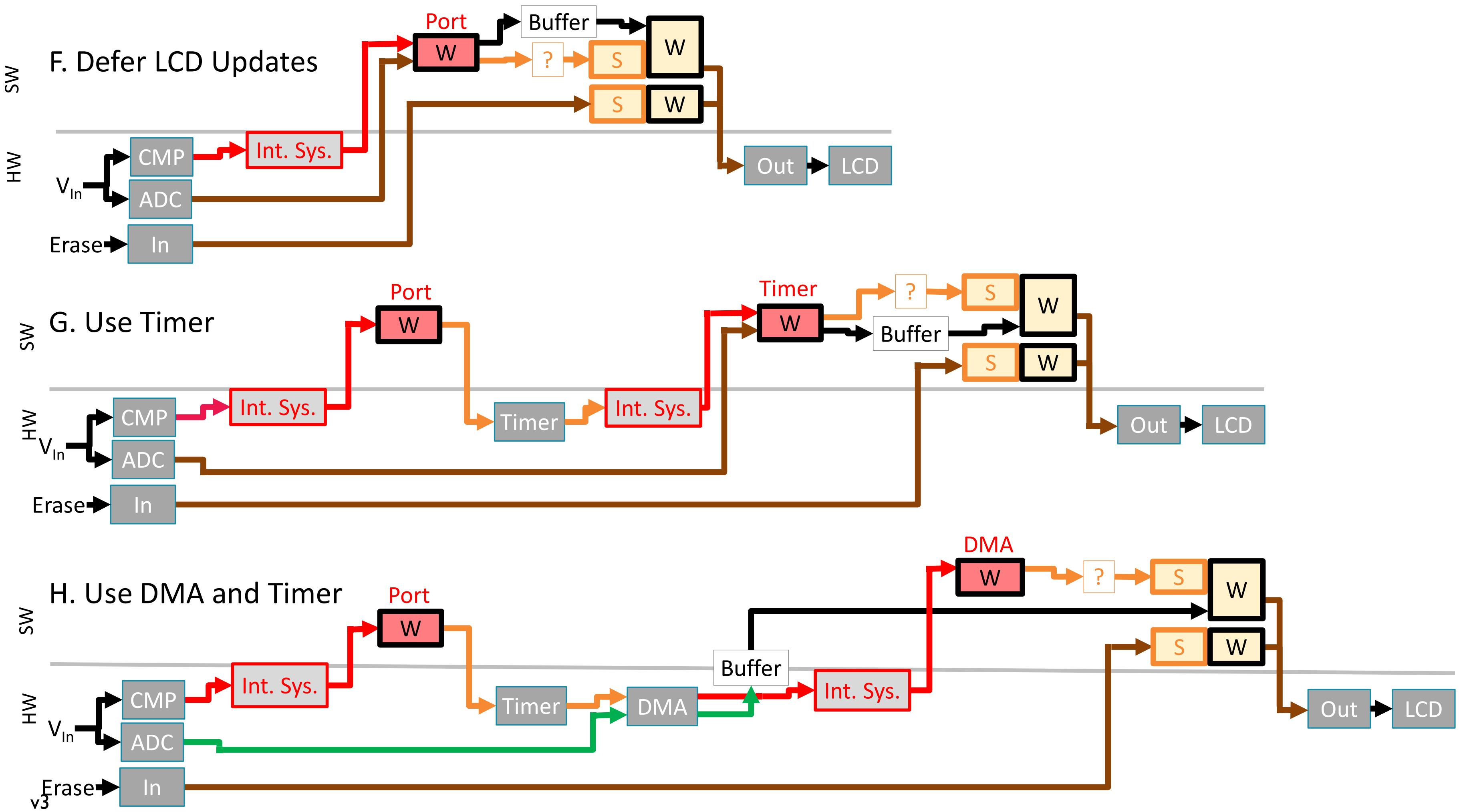

## D. Interrupt Trigger (Bad)



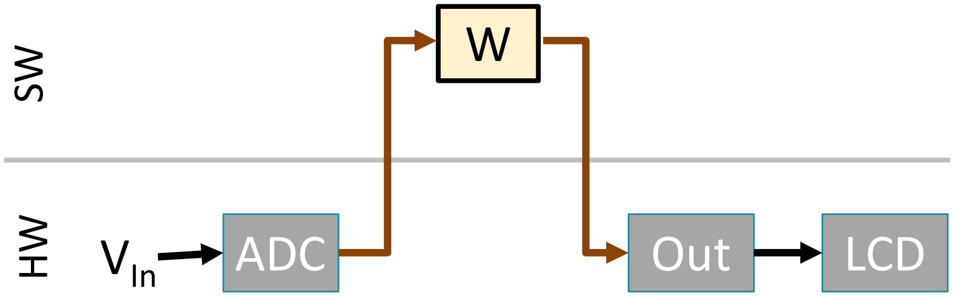

## E. Safe Interrupt Trigger



# Scope Evolution



# A. No Trigger Detection

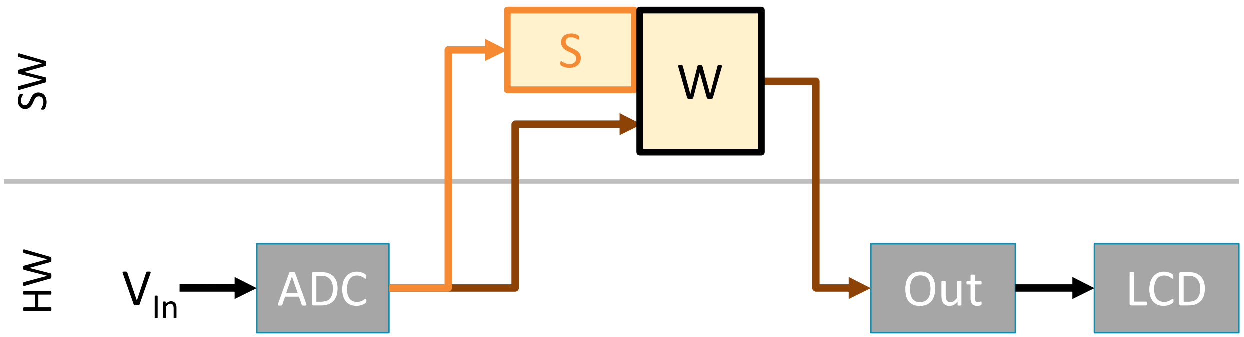



- Synchronize to trigger condition: None
- Schedule: Implicit
- Dispatch: Implicit

```
Thread
…
// No Detector/Synchronizer
// No Scheduler
// No Dispatcher
// Handler process
for (x=n=0; n<NS; n++) {
  r = ADC->Result;
  y = scale(r);
  LCD_Plot(x++, y, tr_clr);
}
```

```
LCD_Plot(px, py, pcolor){
  Draw rectangle from (px, py)
    to (px, py) with pcolor
}
```

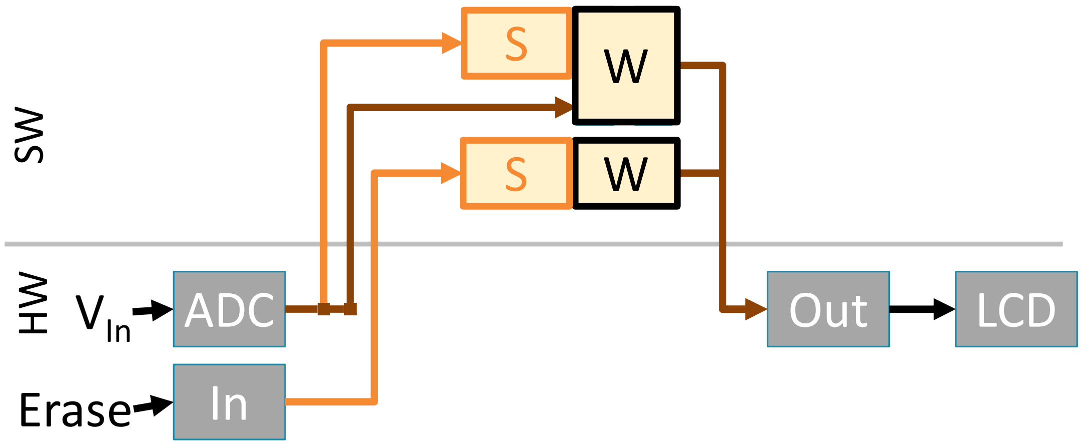
# B. Polling Trigger Detection with Busy-Wait Loop




- Synchronize to trigger condition: Loop in thread analyzes ADC results
- Schedule: Implicit SW in thread
- Dispatch: Implicit SW in thread

```
Thread
…
// Detector/Synchronizer
while (ADC->Result < V_Threshold)
  ;
// No Scheduler
// No Dispatcher
// Handler process
for (x=n=0; n<NS; n++) {
  r = ADC->Result;
  y = scale(r);
  LCD_Plot(x++, y, tr_clr);
}
```

# C. Add Erase Button

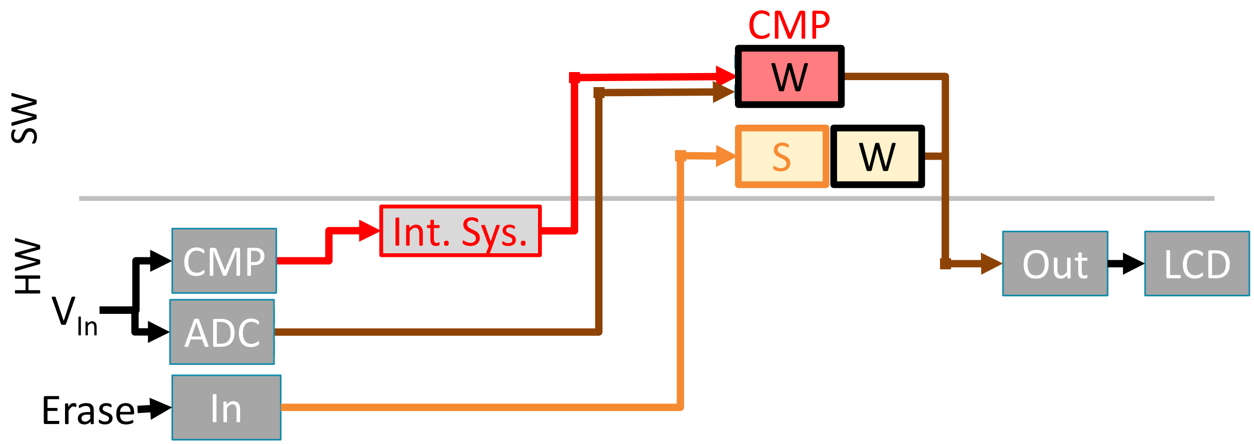



```
Thread(s)
…
// Detector/Synchronizer
while (ADC->Result < V_Threshold) {
   if (Erase_Pressed)
      LCD_Erase();
}
// Do Work: Acquire data
for (x=n=0; n<NS; n++) {
   r = ADC->Result;
   y = scale(r);
   LCD_Plot(x++, y, tr_clr);
}
```

```
LCD_Plot(px, py, pcolor){
   Draw rectangle from (px, py)
      to (px, py) with pcolor
}
LCD_Erase() {
   Draw rectangle from (XMIN, YMIN)
   to (XMAX, YMAX) filled with
   background bg_clr
}
```

- Trigger
  - Synchronize: Polling/analysis loop in thread
  - Schedule: SW in thread
  - Dispatch: Subroutine call
- Erase
  - Synchronize: Polling/analysis test in thread
  - Schedule: SW test
  - Dispatch: Subroutine call

# D. Comparator Interrupt Triggers Data Acquisition




```
CMP Interrupt Handler
for (n=0; n<NS; n++) {
  r = ADC->Result;
  y = scale(r);
  LCD_Plot(x++, y, tr_clr);
}
```

```
Thread
…
// Detector/Synchronizer
if (Erase_Pressed)
  LCD_Erase();
…
```

- Trigger
  - Synchronize: Comparator detects trigger, requests interrupt
  - Schedule: Interrupt System
  - Dispatch: Interrupt System
- Erase
  - Synchronize: Polling/analysis test in thread
  - Schedule: SW test
  - Dispatch: Subroutine call

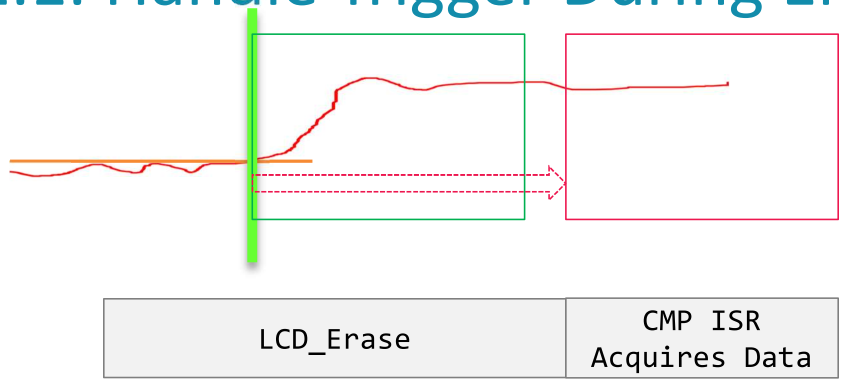
# Problem #1 with Approach D

```
CMP Interrupt Handler
for (n=0; n<NS; n++) {
   r = ADC->Result;
   y = scale(r);
   LCD_Plot(x++, y, tr_clr);
}
```

```
Thread
…
// Detector/Synchronizer
if (Erase_Pressed)
   LCD_Erase();
…
```


- What if scope is **triggered** during an erase?
  - Start to erase screen, get partway through
  - Plot trace
  - Erase rest of screen, maybe erasing part of new trace just plotted
- Observation
  - LCD_Erase is not executed atomically, can run concurrently with CMP ISR
- Should LCD_Erase be executed atomically?

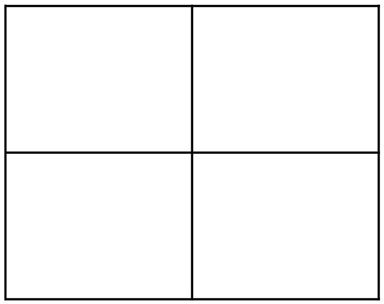
# E.1. Handle Trigger During Erase




```
Thread
…
// Detector/Synchronizer
if (Erase_Pressed)
  save CMP interrupt enabled state;
  disable CMP interrupt;
  LCD_Erase();
  restore CMP interrupt enabled state
…
```

- Solution (?) 1.0
  - Don’t continue erasing after interrupt?
  - May leave old unerased trace in part of screen
  - Not clear yet how to make LCD_Erase() give up partway through.
- Solution (?) 1.1
  - Disable interrupts (or just comparator interrupt) during LCD_Erase?
  - Long synchronization delay possible from trigger condition to CMP ISR execution

# E.1. Handle Trigger During Erase

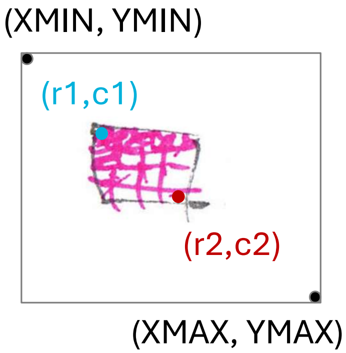


```
CMP Interrupt Handler
  for (n=0; n<NS; n++) {
    r = ADC->Result;
    y = scale(r);
    LCD_Plot(x++,y);
  }
  NewDAcqDone = 1
```

```
Thread
…
// Detector/Synchronizer
if (Erase_Pressed)
  LCD_Erase();
…
void LCD_Erase(void) {
for each tile {
  if (NewDAcqDone) {
     NewDAcqDone = 0;
    break; // exit for loop
  }
  save CMP interrupt enabled state;
  disable CMP interrupt;
  draw background color rect. for tile
  restore CMP interrupt enabled state
}
```

- Solution (?) 1.2
  - Break screen into tiles
  - Erase each tile atomically, allow interrupts (triggering) between tiles
  - Don’t finish erasing if triggered during erase.
  - Interrupt handler sets flag,  LCD_Erase tests flag
  - What about tiles that aren’t erased? Similar to 1.0, but reduces trigger detection latency
- Solution(?) 1.3
  - When ISR runs, **acquire** data but **don’t plot it yet**.
  - Need to save it, trigger other code to run and plot it.
    - More items on the **Sync and Comm To Do List**
    - More complex, will examine deferred plotting in design F

# Background for Problem #2: LCD Controller Interface (ST7789VI)




- MCU writes Command ID byte, then parameters (data bytes)
  - Received Command ID terminates previous command, starts new command
  - Received parameters interpreted based on most recent command
- Screen coordinates: columns (x) and rows (y)
- Draw rectangle between corners (c1,r1) and (c2,r2)
  - Set first and last columns (CASET)
  - Set first and last rows (RASET)
  - Send pixels to fill in rectangle (RAMWR)
- Use LCD_Plot and LCD_Erase

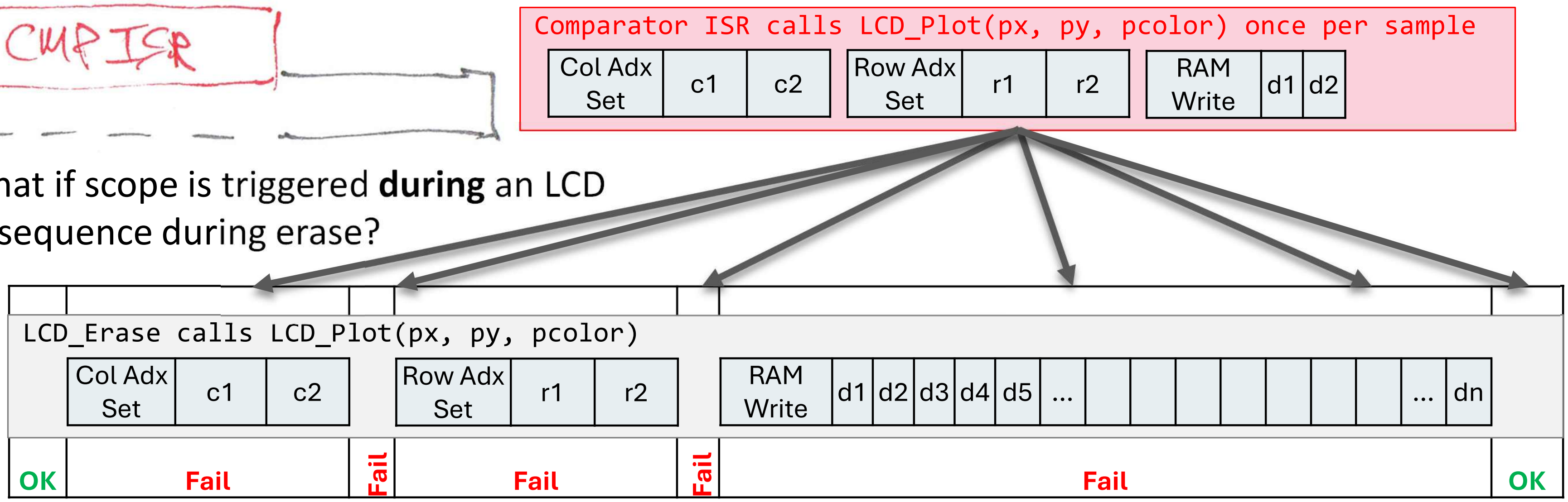
| Command ID (byte) | Parameters (data bytes) | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | ... |
| Column Address Set (CASET) | c1 | | c2 | | | |
| Row Address Set (RASET) | r1 | | r2 | | | |
| RAM Write (RAMWR) | d1 | d2 | d3 | d4 | d5 | ... |

```
LCD_Plot(px, py, pcolor){
  // Draw rectangle
  LCD_Command(CASET, px, px)
  LCD_Command(RASET, py, py)
  LCD_Command(RAMWR, pcolor, 1)
}
```

```
LCD_Erase() {
  // Draw rectangle
  LCD_Command(CASET, XMIN, XMAX)
  LCD_Command(RASET, YMIN, YMAX)
  n = (XMAX-XMIN + 1) * (YMAX-YMIN + 1)
  LCD_Command(RAMWR, bg_color, n)
}
```

# Problem #2 with Approach D


- Issue 2. What if scope is triggered **during** an LCD command sequence during erase?




- Result depends on **when** trigger occurs **relative to** LCD_Erase's **command sequence**
  - Before or after complete command sequence (CASET + data, RASET + data, RAMWR + data)? OK
  - Before or after a command and its data: Fail
  - Between a command and its data: Fail

# E.2. Trigger during LCD Command Sequence. Make LCD Controller Commands Atomic?

```
LCD_Erase calls LCD_Plot(px, py, pcolor)
```

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Col Adx Set | c1 | c2 | Row Adx Set | r1 | r2 | RAM Write | d1 | d2 | d3 | … | dn |

Interrupt disabled

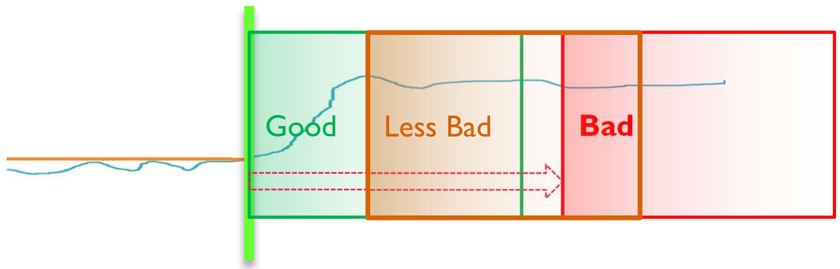
- Solution (?) 2.1
  - Don't allow triggering within LCD command sequences
  - For each command + data sequence (CASET+data, RASET +data, RAMWR +data) within preemptable code (e.g. thread)…
    - Disable comparator interrupt
    - Send LCD command + data sequence
    - Restore comparator interrupt state
  - Display may still be corrupted by erasing newly plotted pixels
  - *As with 1.0, detection latency can be as long as erase time. Break screen into tiles? Still have problems*

```
Thread
…
// Detector/Synchronizer
if (Erase_Pressed)
  LCD_Erase();
…
void LCD_Erase_Atomic() {
  save CMP interrupt enabled state;
  disable CMP interrupt;
  // Draw rectangle
  LCD_Command(CASET, XMIN, XMAX)
  LCD_Command(RASET, YMIN, YMAX)
  n = (XMAX-XMIN + 1) * (YMAX-YMIN + 1)
  LCD_Command(RAMWR, bg_color, n)
  restore CMP interrupt enabled state
}
```

# Problems with Designs E.1, E.2

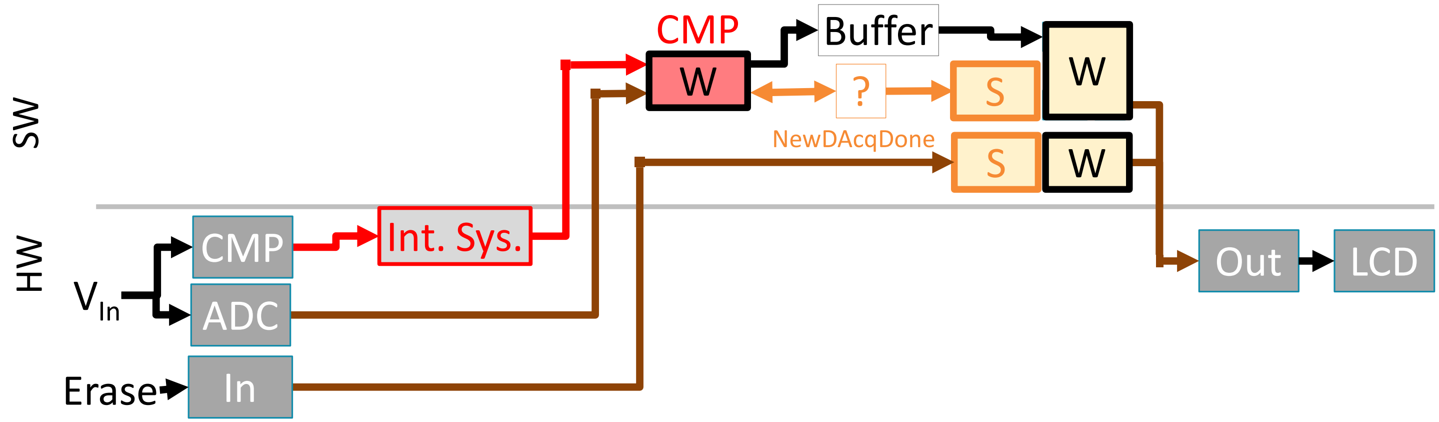
- LCD is shared by main thread and comparator interrupt handler
- Shared resource bugs for LCD_Plot during LCD_Erase
  - High-level sharing (independent of LCD communication)
    - Problem: want to erase everything but the new trace
    - Critical sections cover up to entire screen
      - LCD_Erase: all pixels in each column
      - LCD_Plot: one pixel in each column
  - Low-level sharing (LCD controller interface)
    - Critical sections: Complete command sequence (CASET + data, RASET + data, RAMWR + data)

- Both designs prevent corruption of LCD communication and display contents
- But probably not good enough for scope
  - Scope triggering is **time-critical** (real-time system)




- So try Solution(?) 1.3 despite complexity
  - When ISR runs, **acquire** data but **don't plot it yet**.
  - Need to save data, trigger other code to run and plot it => More for the **Sync and Comm To Do List**

# F: Hardware & OS Help: Comparator Interrupt, Deferred Plotting



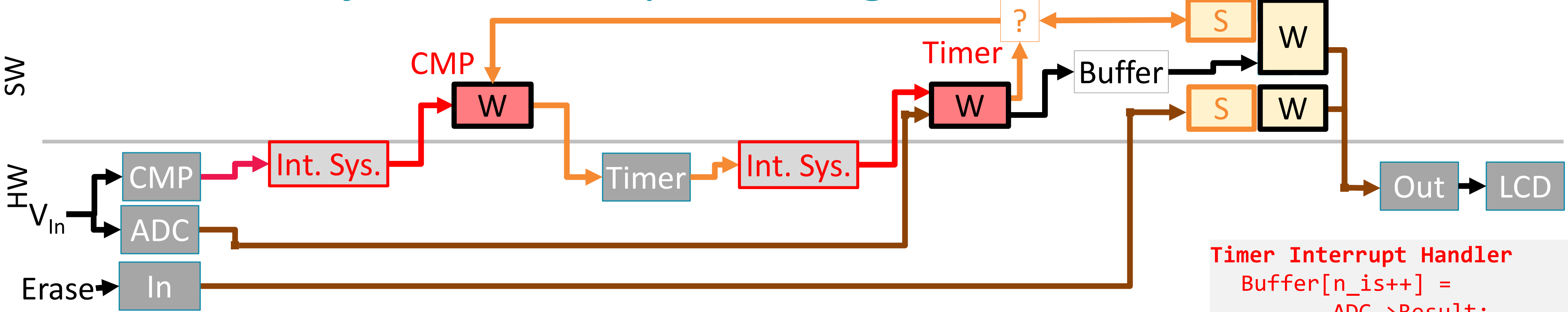

```
volatile int NewDAcqDone=0;

CMP Interrupt Handler
  if (NewDAcqDone==0) {
    for (n=0; n<NS; n++) {
      Buffer[n] = ADC->Result;
    }
    NewDAcqDone = 1;
  }
```

```
Thread
…
  if (NewDAcqDone) {
    for (x=n=0; n<NS; n++) {
      y = scale(Buffer[n]);
      LCD_Plot(x++, y);
    }
    NewDAcqDone = 0
  }
  if (Erase_Pressed)
    LCD_Erase();
```

- Trigger
  - Synchronize: Comparator detects trigger, requests interrupt
  - Schedule & Dispatch: Interrupt System
- Plot Data
  - Synchronize: Test integrated w/ thread
  - Schedule & Dispatch: Integrated w/ thread
- Erase
  - Synchronize: Test integrated w/ thread
  - Schedule & Dispatch: Integrated w/ thread
- What about retriggering while plotting?
  - Synchronize by having CMP ISR do work if thread is done plotting data (NewDAcqDone)
- Consequences
  - Skip disabling interrupts for safe LCD interfacing
  - Skip dividing up LCD_Erase for better responsiveness
  - More delay to plot trace on LCD; only after all samples are acquired

# G: Precise, Adjustable Sample Timing




- Trigger: Comparator Interrupt Handler
  - Synchronize: Comparator detects trigger, requests interrupt
- Sample Data: Timer Interrupt Handler
  - Synchronize: Timer requests interrupt
- Plot Data
  - Synchronize: Integrated w/ thread
- Erase
  - Synchronize: Integrated w/thread
- What about retriggering while plotting?
  - Test NewDAcqDone in trigger detection (CMP) interrupt handler

- Consequences
  - No need to disable interrupts for LCD safety or divide LCD_Erase for responsiveness
  - More delay to plot trace on LCD; only after all samples are acquired

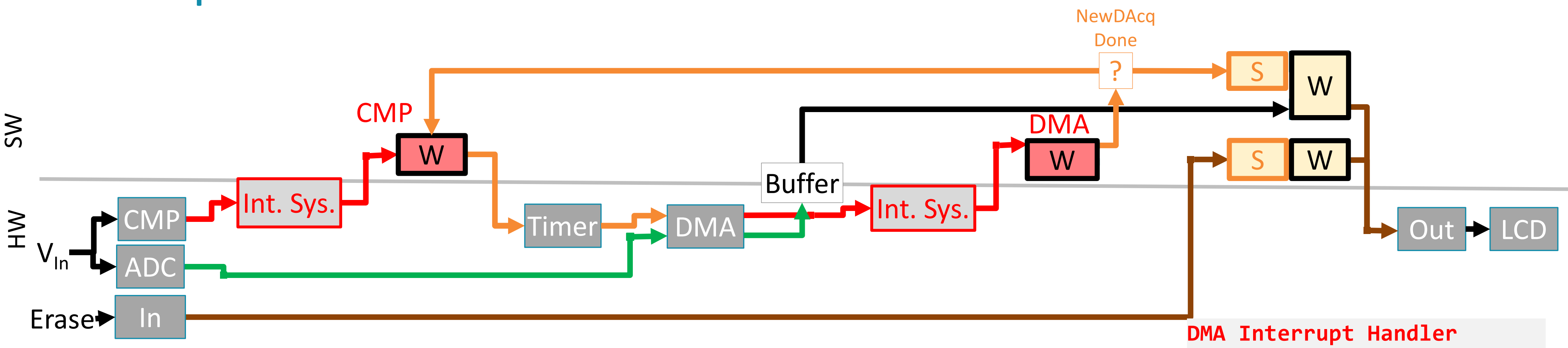
```
volatile int NewDAcqDone=0;
// input sample number
volatile int n_is=0;

CMP Interrupt Handler
  if (NewDAcqDone == 0) {
    n_is = 0
    Start timer
  }
}
```

```
Timer Interrupt Handler
  Buffer[n_is++] =
        ADC->Result;
  if (n_is == NS) {
    Stop timer
    NewDAcqDone = 1;
  }
```

```
Thread
…
  if (NewDAcqDone) {
   for (x=n=0; n<NS; n++) {
      y = scale(Buffer[n]);
      LCD_Plot(x++, y);
   }
   NewDAcqDone = 0
  }
  if (Erase_Pressed)
    LCD_Erase();
```

# H: Sample Data with Timer-Driven DMA




- Trigger Detection
  - Synchronize: Comparator detects trigger, requests interrupt
- Sample Data
  - HW: DMA transfers from ADC to Buffer
- Plot Data
  - Synchronize: Integrated with thread
- Erase
  - Synchronize: Integrated with thread
- What about retriggering during plotting?
  - Same as F, G

- Consequences
  - Responsiveness from trigger condition to start of sampling is limited by interrupt latency
  - In affordable systems, sample rate is limited by ADC

```
volatile int NewDAcqDone=0;

CMP Interrupt Handler
  if (NewDAcqDone==0){
    Start timer
  }
```

```
DMA Interrupt Handler
  Stop timer
  NewDAcqDone = 1;
```

```
Thread
…
  if (NewDAcqDone) {
   for (x=n=0; n<NS; n++) {
     y = scale(Buffer[n]);
     LCD_Plot(x++, y);
   }
   NewDAcqDone = 0
  }
  if (Erase_Pressed)
    LCD_Erase();
```