NC STATE UNIVERSITY

APPLICATION DESIGN BASICS

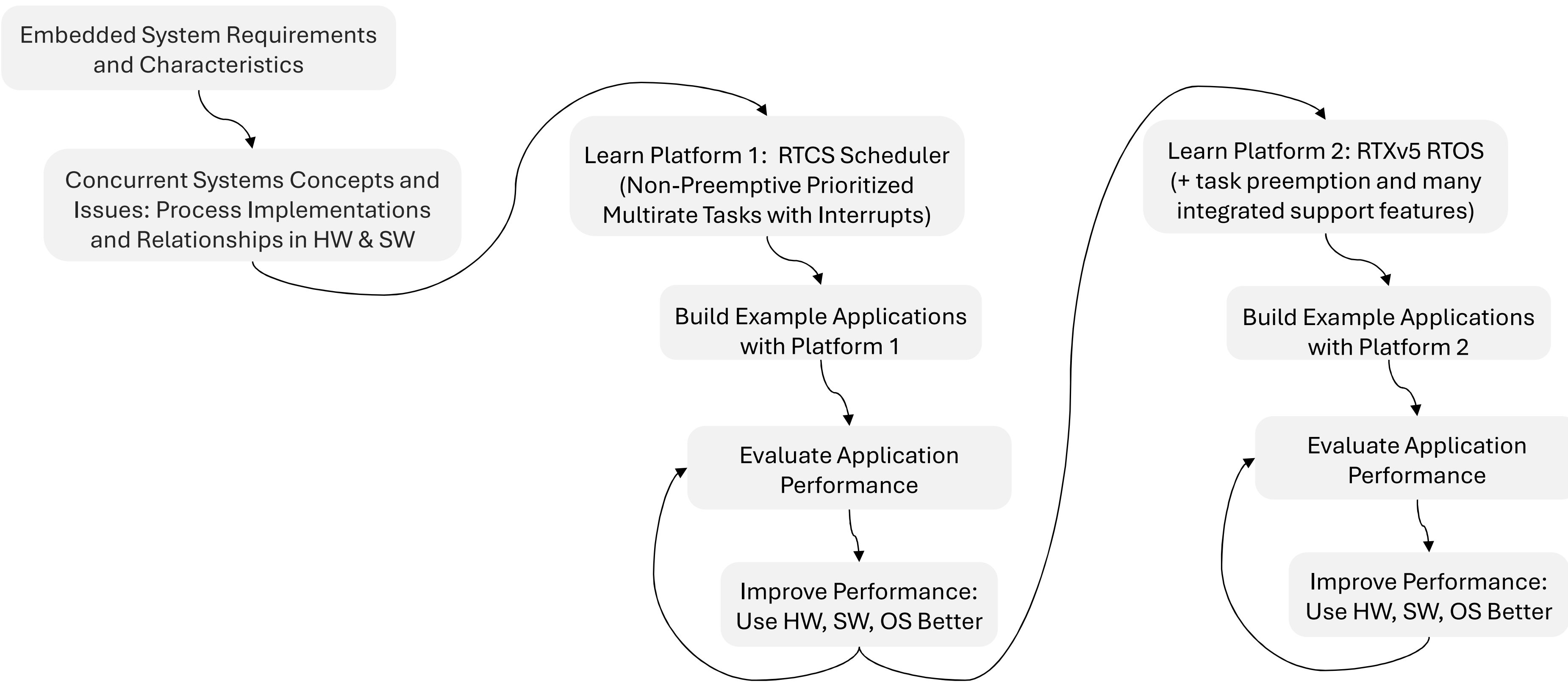
USING THE RTC SCHEDULER WITH INTERRUPTS

V2

9/14/2025

I

Where are we in the class?



Example Applications: Key Requirements and Challenges

Application	Providing Functionality					Meeting Performance Requirements			
	Digital Interfacing	Analog Interfacing	Sync and Do: Triggering	Sync and Don't: Sharing & Races	Inter-Process Comm.	Timing Stability	Responsiveness	Reducing Software Overhead	Tolerating Timing Mismatches
Blinky Control Panel	In, Out, PWM								
Quadrature Decoder	In								
Waveform Generator		Out							
Oscilloscope		In							
Serial Comms.	In, Out								
I2C Comms.	In, Out								
LCD Controller	Out								
Touchscreen		In							
SMPS Controller	Out	In							
µSD via SPI Comms.	In, Out								

Application Design Overview

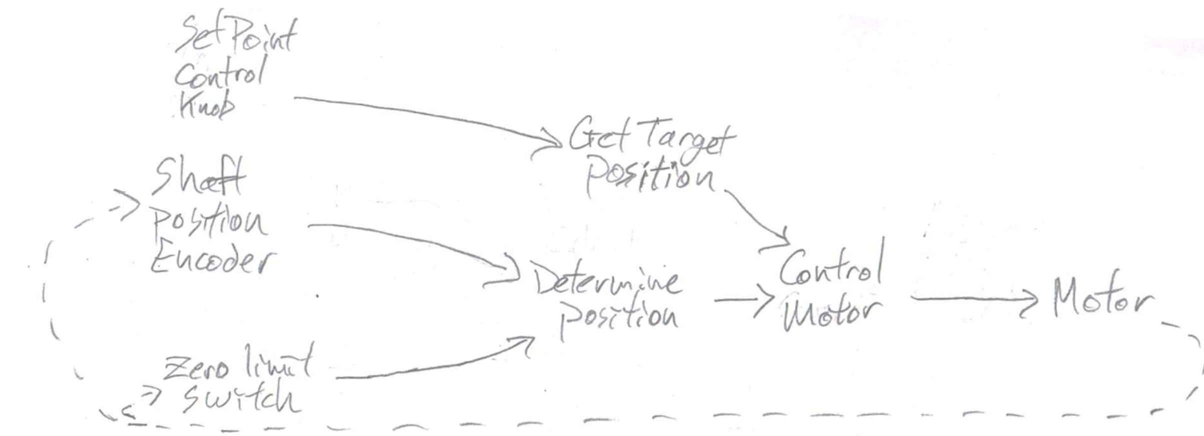
- Identify system's **inputs, outputs** and **processes**, and their key connections
- Identify key **hardware, software stages** in each process
  - Initially consider only peripheral's fundamental features (examine enhancements later as needed)
- Analyze each process to find key **synchronization requirements** (e.g. timing, events)
  - 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*)?
    - Do X, wait at least 3.2 ms but no more than 3.8 ms, do Y, Wait for event E, do Z.
- Analyze key **process interactions**
  - Identify and describe ***synchronization*** and ***communication*** **between** processes
    - Triggering, shared data and resources (one-way data flows, other data flows)
  - Identify implicit ***sync.*** required within the ***communication***.
    - Notification of new data? Buffer old data? How much? how to manage buffer? How to handle overrun condition?
- Choose mechanisms to support and implement these requirements and interactions
  - Program structure
  - Interrupt/Scheduler/OS support
  - Algorithms to implement in your code
- Continue with design and implementation
  - Functionality first, then performance
  - Iterate design to improve performance

Application Design Overview

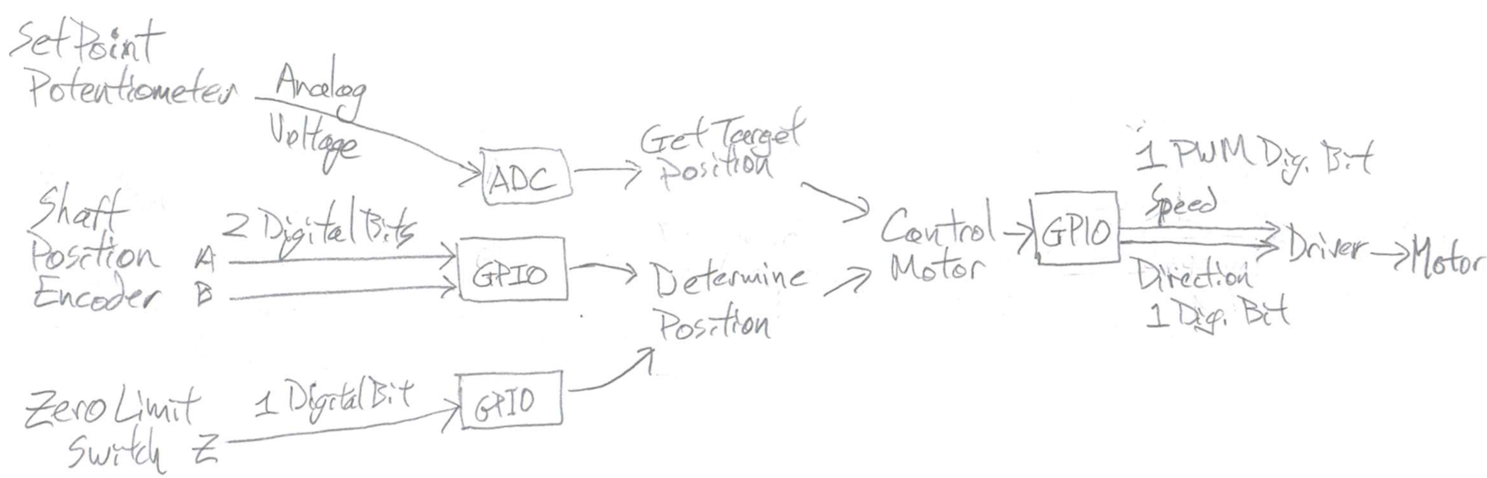
- Identify system fundamentals: **inputs, outputs** and **processes**, and their key connections
- Identify key **hardware, software stages** in each process
  - Initially consider only peripheral's fundamental features (examine enhancements later as needed)
- Analyze each process to find I/O-driven **synchronization requirements**
  - 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*)?
    - Do X, wait at least 3.2 ms but no more than 3.8 ms, do Y, Wait for event E, do Z.
- Analyze key **process interactions**
  - Identify and describe ***synchronization*** and ***communication*** **between** processes
    - Triggering, shared data and resources (one-way data flows, other data flows)
  - Identify implicit ***sync.*** required within the ***communication***.
    - Notification of new data? Buffer old data? How much? how to manage buffer? How to handle overrun condition?
- Choose mechanisms to support and implement these requirements and interactions
  - Program structure
  - Interrupt/Scheduler/OS support
  - Algorithms to implement in your code
- Continue with design and implementation
  - Functionality first, then performance
  - Iterate design to improve performance

Identify System Fundamentals

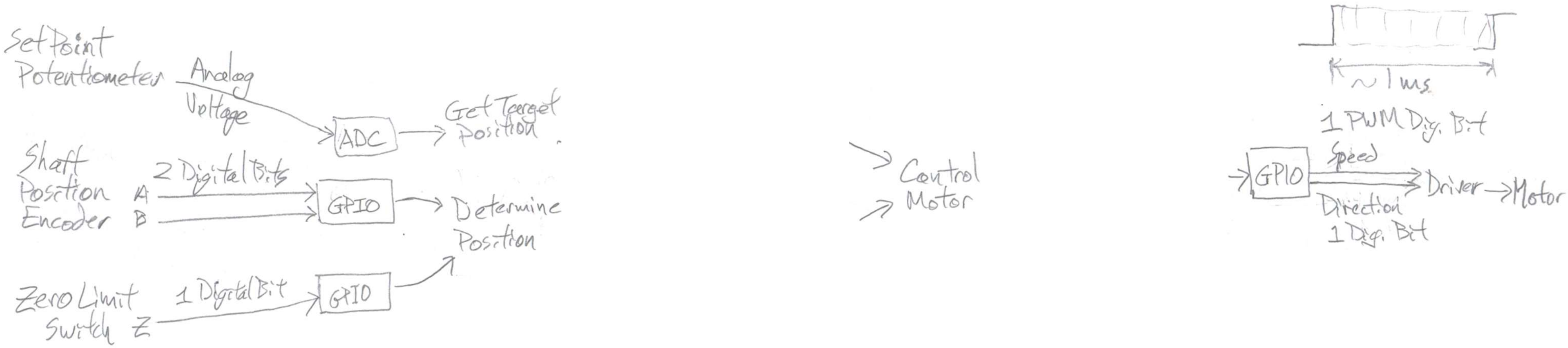
- Identify system's **inputs, outputs** and **processes**, and their key connections
  - Note feedback between output (motor) and some inputs



- Identify key **hardware** and **software stages** in each process
  - Initially consider only peripheral's fundamental features (later will examine enhancements as needed)

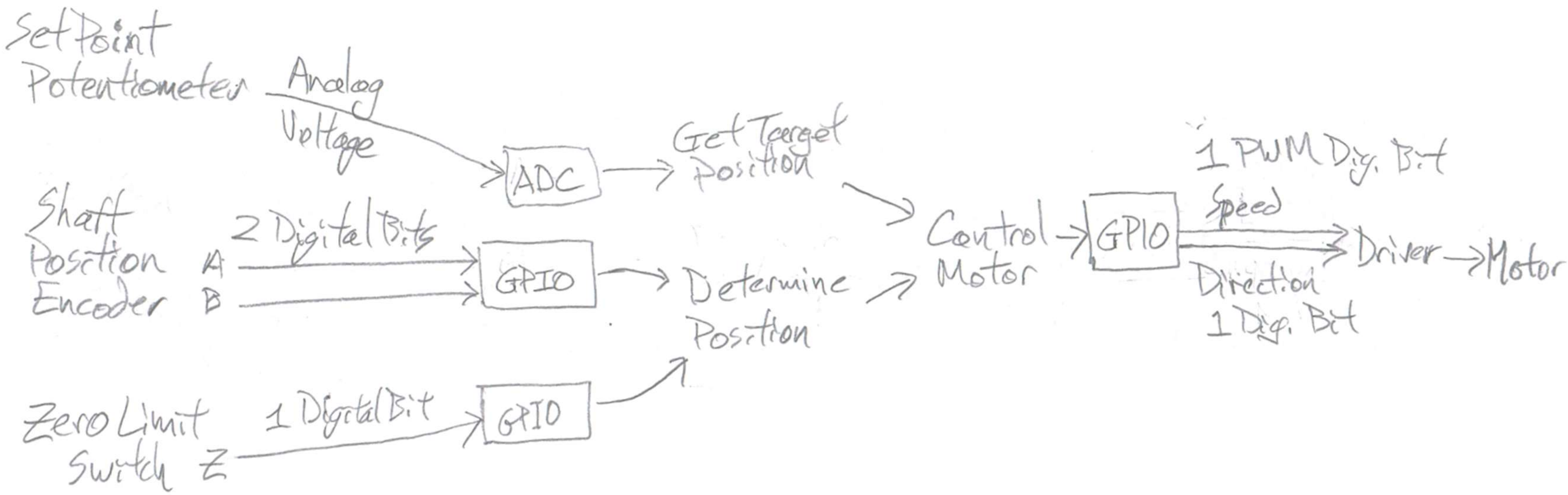


Synchronization Requirements for Inputs and Outputs



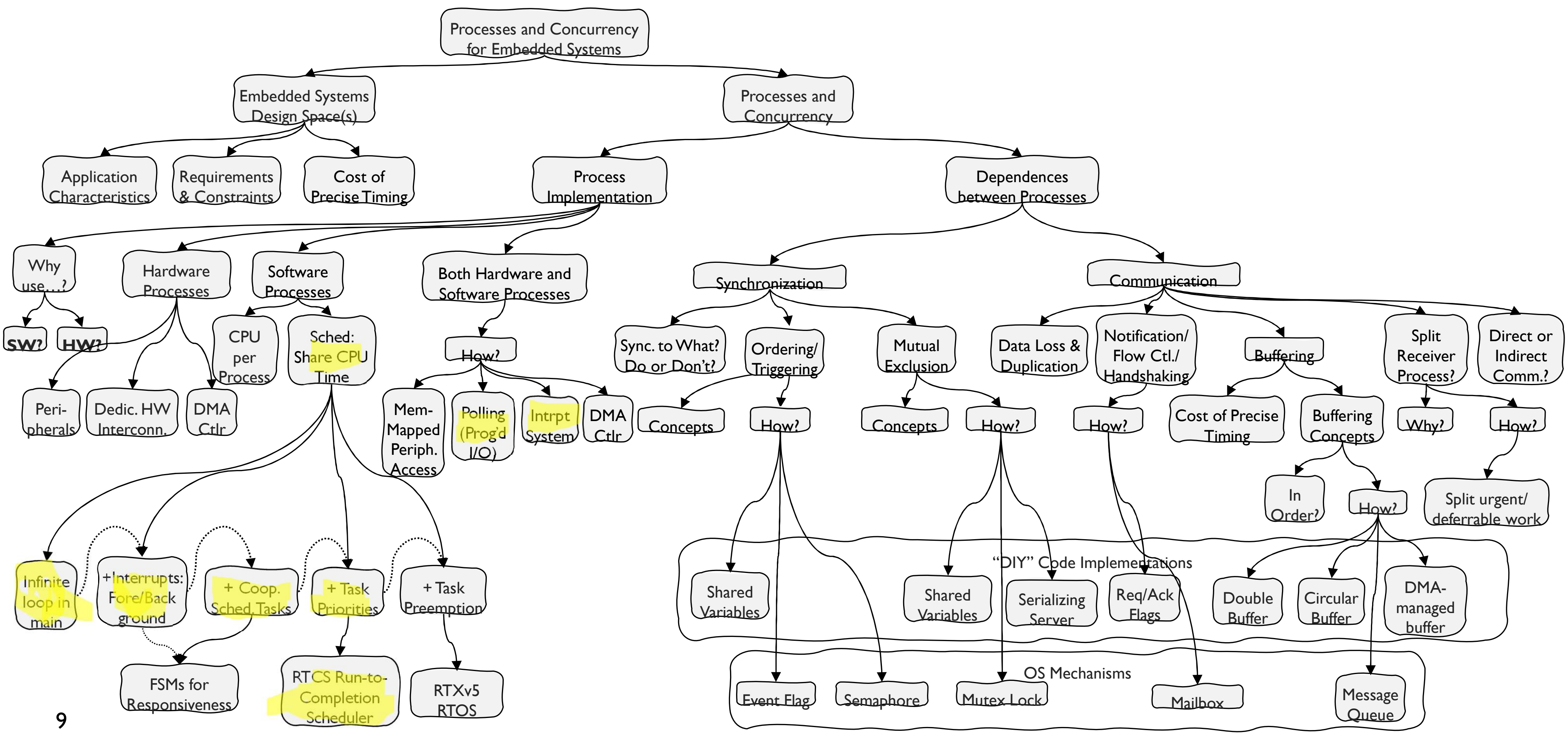
- Analysis may drive early design decisions: using interrupts, hardware peripheral features, etc.
- Initial synchronization: What triggers/releases/allows process (e.g. thread's work code) to start execution? ***Event or time?***
  - Get Target Position: Time-based (periodic): run at least 5 times /second (at most every 200 ms)
  - Detect Position: Event: run on rising edge of A or rising edge of Z.
    - Separate ISRs or single ISR?
  - Control Motor: Time-based (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 (*wait until*)? ***Event or time?***
  - Control Motor: Time-based (delay). Clear Speed to 0 after 0 to 1 ms, depending on data for drive level

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? Buffer old data? How much? how to manage buffer? How to handle overrun condition? …

Extended Topic Map: Class 08



NC STATE UNIVERSITY

APPLICATION DESIGN USING RTCS

Checklist for Using RTC Scheduler

1. Add RTCS folder to project directory
2. Add RTCS folder to include search path
3. Link in scheduler tick routine
   1. Add call to tick_timer_intr() to a periodic interrupt.
   2. Add #include “rtcs.h” to that source file too.
4. Create code for your tasks in tasks.[ch]
5. Modify main code to
   1. Add your tasks to scheduler task table
   2. Start up the scheduler

RTCS_DEMO PROGRAM:
INDEPENDENT R, G, B LED FLASHING

Example Application (RTCS_Demo)

- Toggle red LED every 500 ms
- Toggle green LED every 490 ms
- Toggle blue LED every 480 ms
- How would you code this without a scheduler?
- With a periodic scheduler, consider greatest common divisor (GCD) of periods

Demo Code for a Task

```
void Task_R(void) {
  static uint8_t LED_On=0;

  PTB->PSOR = MASK(DEBUG_RED_POS);

  if (LED_On)
       PTB->PCOR = MASK(RED_LED_POS);
  else
       PTB->PSOR = MASK(RED_LED_POS);

  LED_On = 1 - LED_On;

  PTB->PCOR = MASK(DEBUG_RED_POS);
}
```

Set (1) a debug output bit to see on scope/logic analyzer when task starts running

Clear (0) the debug output bit to see when task stops running

Demo Scheduler Start-Up

```
int main (void) {
  Init_Debug_Signals();
  Init_RGB_LEDs();

  RTCS_Init(100); // 100 Hz timer ticks
  RTCS_Add_Task(Task_R, 0, 50);
  RTCS_Add_Task(Task_G, 1, 49);
  RTCS_Add_Task(Task_B, 2, 48);

  RTCS_Run_Scheduler();  // This call never returns
}
```

NC STATE UNIVERSITY

RGB/FLASHER PROGRAM WITH RTCS

RGB/Flasher Upgrade to RTCS

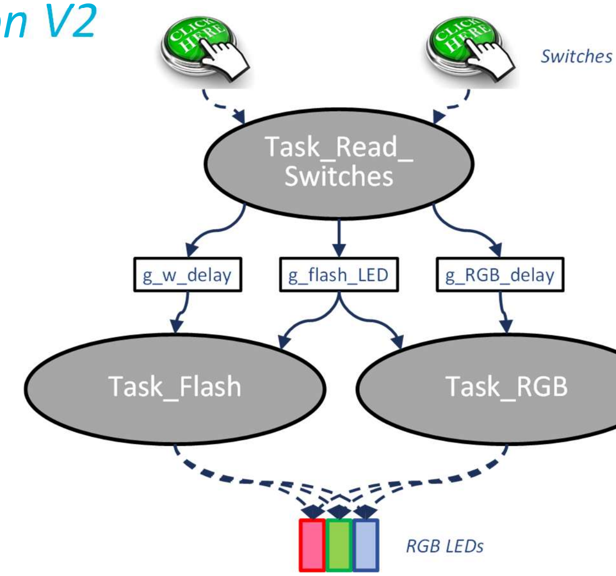
- **ESF** textbook Chapter 3 example
  - LED behavior: flash (White/Off) or sequence (R/G/B)
  - LED timing: slow or fast
- How to port the program to use the RTCS?
- Version 1
  - Simple port, start with switch polling version (V2)
  - Uses a few RTCS features
- Version 2
  - Better port, start with switch interrupt version (V3)
  - Takes advantage of more RTCS features
- Code for both is in course repository in RTCS folder

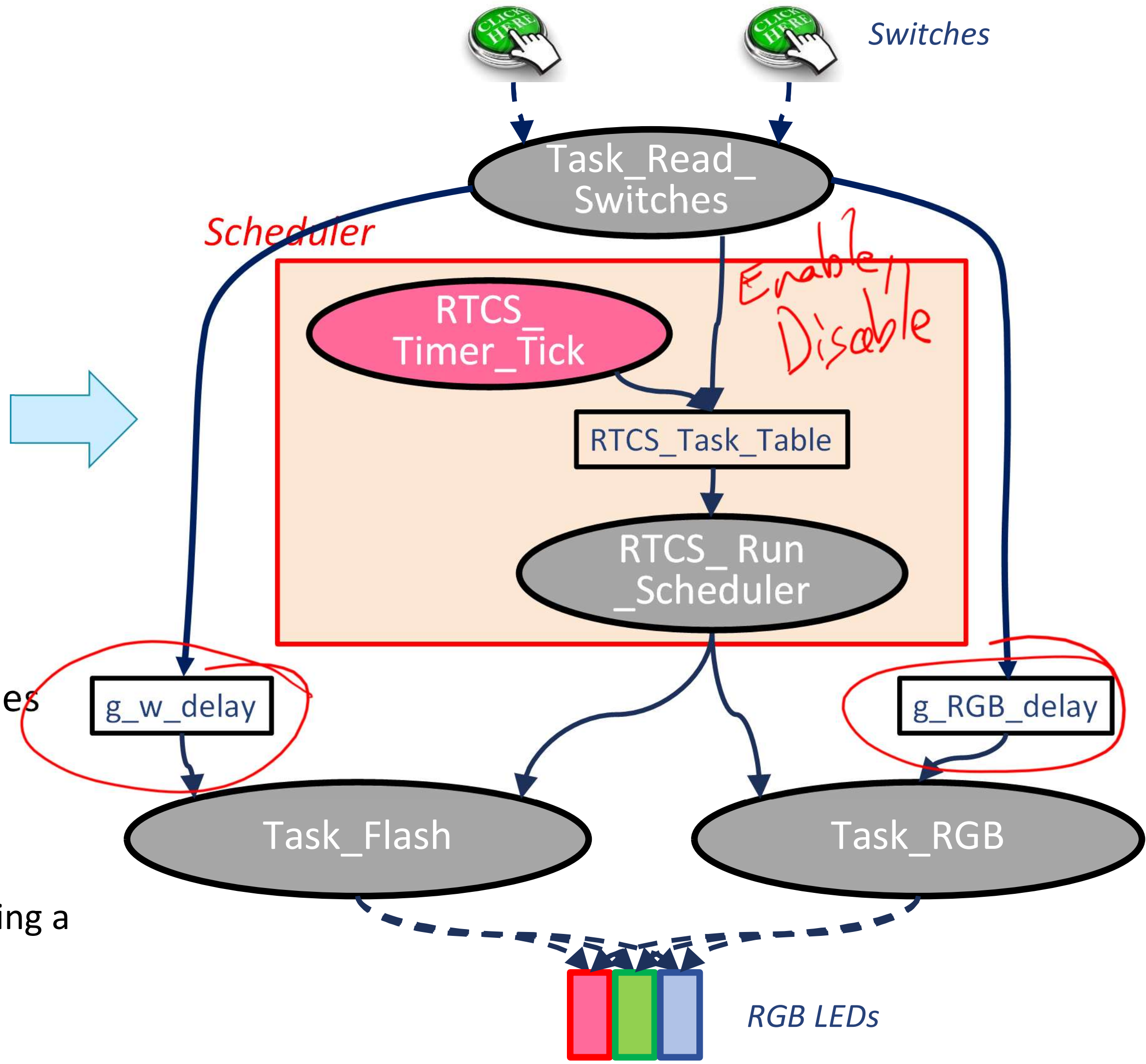
NC STATE UNIVERSITY

RGB/FLASHER PROGRAM WITH RTCS

RTCS_RGB_Flasher_1

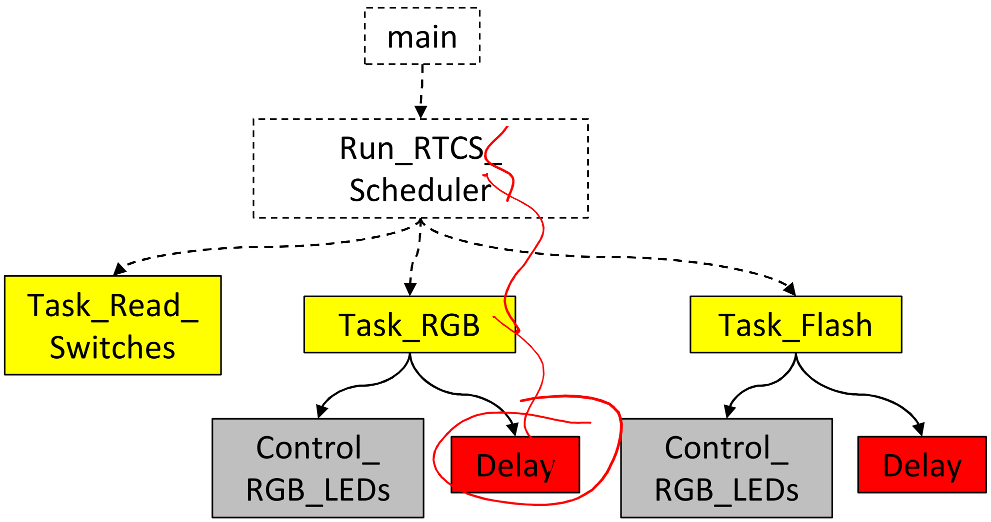
Based on V2





- Switches polled by periodic Task_Read_Switches
  - ***Enables*** and ***disables*** tasks
  - Modifies global delay variables
- LEDs
  - Task_RGB and Task_Flash control LED timing using a busy-wait Delay function

Where is the Idle Time?



Time

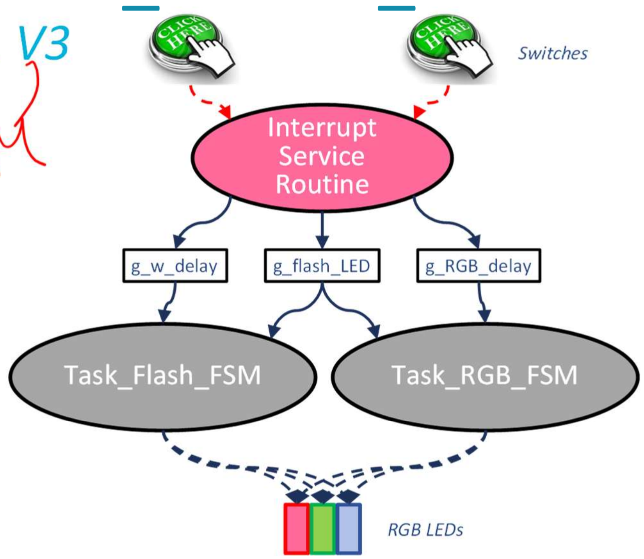
Run_RTCS_Scheduler	
Task_Read_Switches	
Task_RGB	
Delay	
Task_Flash	
Delay	

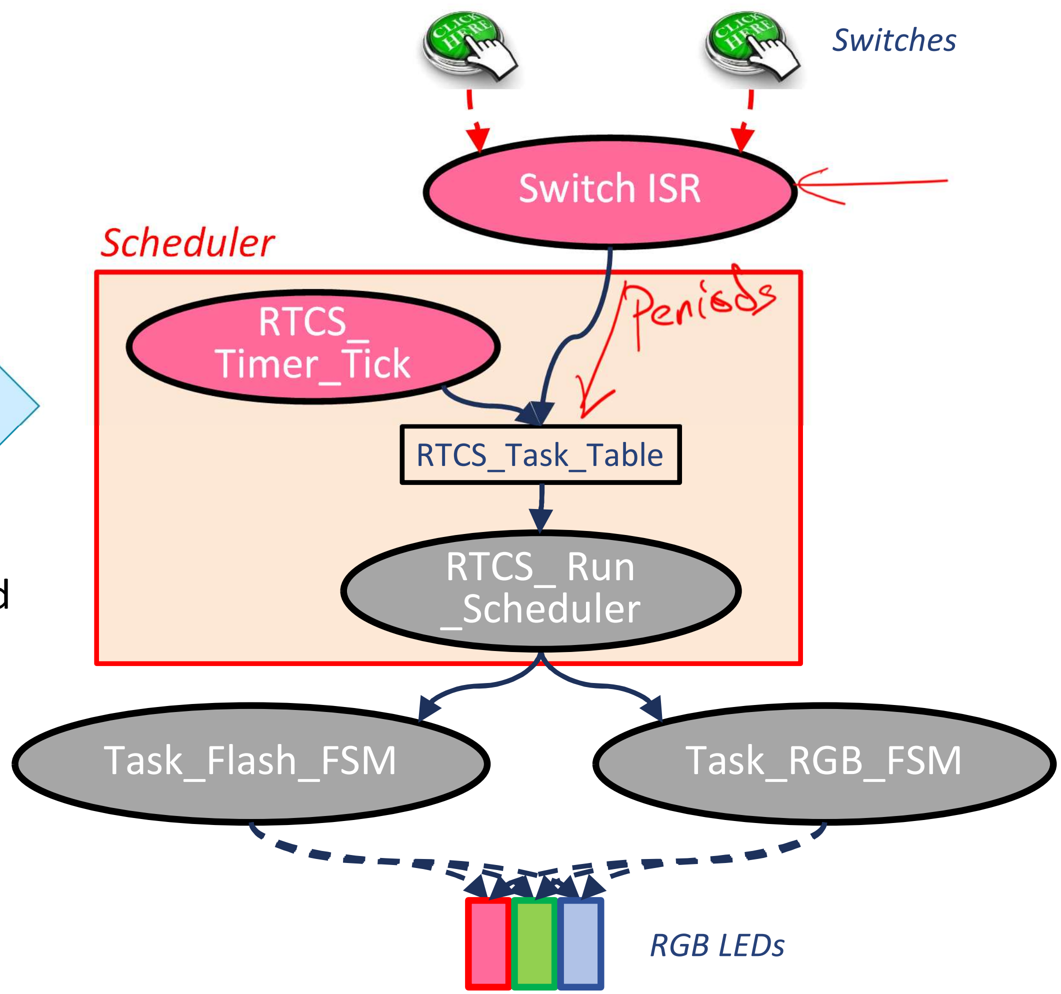
- Analysis
  - Examine call graph
  - Consider code execution timeline
- Idle time is buried within task call to in busy-wait Delay function
  - Not available to scheduler

RTCS_RGB_Flasher_2

Based on V3

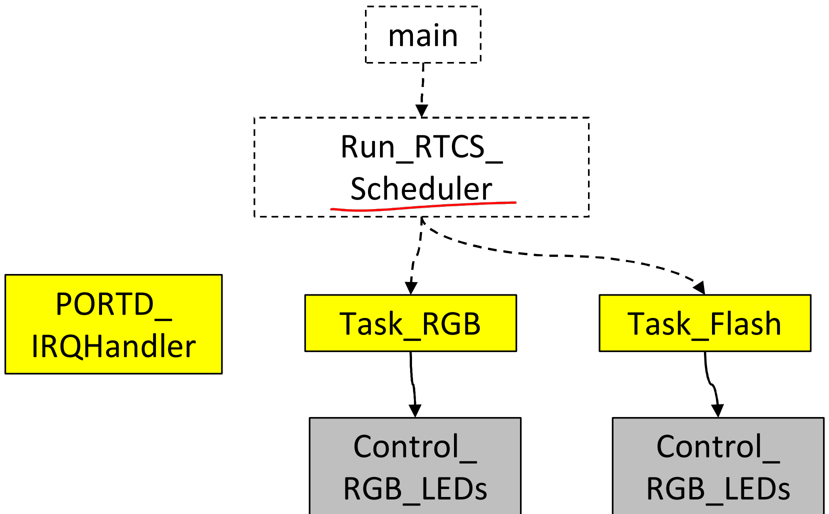
+ FSM

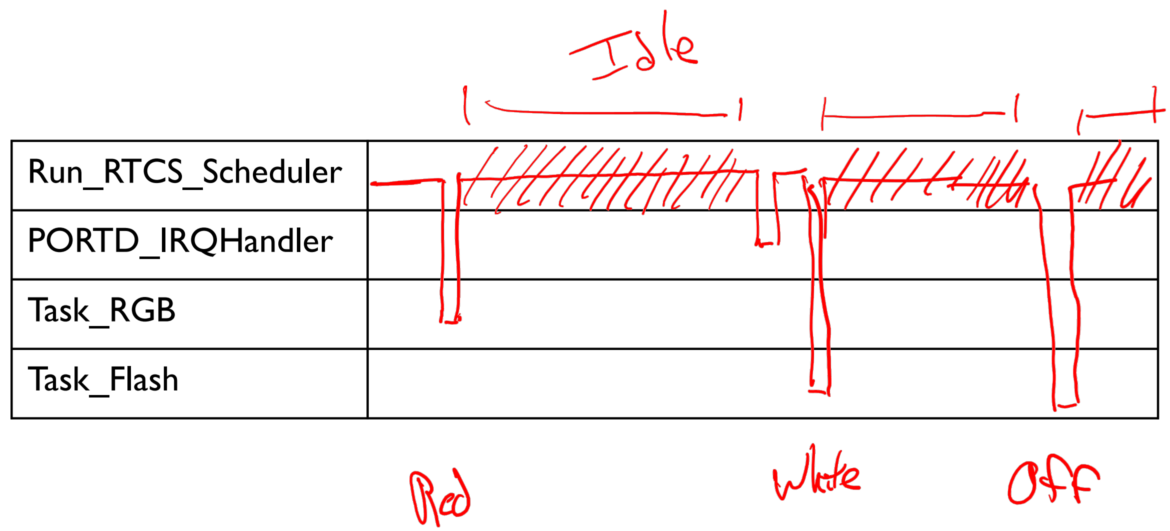




- Switches trigger interrupts when pressed or released
  - Handled by ISR PORTD_IRQHandler
  - ISR enables, disables tasks, and changes task periods. No more g_*_delay global variables!
- LEDs
  - Task_RGB and Task_Flash broken into FSMs (one state per color)
  - Scheduler controls LED timing (via task periods, changed by PORTD_IRQHandler)

Where is the Idle Time?





- Analysis
  - Examine call graph
  - Consider code execution timeline
- Idle time
  - Available to scheduler!

NC STATE UNIVERSITY

How to Change Program Components?

- Tasks
  - Get rid deciding whether to run and busy wait delay
- Switch ISR
  - Update scheduler's task table (not g_*) based on desired behavior
- Variables
  - g_w_delay, g_flash_LED, g_RGB_delay aren't needed any more since scheduler will take care of them

Task Changes

```
void Task_Flash_FSM(void) {
    static enum {ST_WHITE, ST_BLACK} next_state = ST_WHITE;

    if (g_flash_LED == 1) {       // Only run task when in flash mode
        switch (next_state) {
            case ST_WHITE:
                Control_RGB_LEDs(1, 1, 1);
                Delay(g_w_delay);
                next_state = ST_BLACK;
                break;
            case ST_BLACK:
                Control_RGB_LEDs(0, 0, 0);
                Delay(g_w_delay);
                next_state = ST_WHITE;
                break;
            default:
                next_state = ST_WHITE;
                break;
        }
    }
}
```

- Eliminate run test
- Eliminate delay loop calls, as scheduler will provide delays
- Similar changes for Task_RGB_FSM

Switch ISR Changes

- Variables g_w_delay, g_flash_LED, g_RGB_delay aren't needed any more since scheduler will take care of their function
  - Use RTCS interface functions to update scheduling information
  - Code will update scheduler's data to "Make it so"
- If SW1 pressed
  - Enable Task_Flash_FSM, request to run it and disable Task_RGB_FSM
  - Else do opposite
- Update task periods Task_Flash_FSM and Task_RGB_FSM based on if SW2 is pressed
- Set, clear debug bit to show start and end of ISR on scope/logic analyzer

```
void PORTD_IRQHandler(void) {
  PTB->PSOR = MASK(DEBUG_ISR_POS);
  // Read switches
  if ((PORTD->ISFR & MASK(SW1_POS))) {
    if (SWITCH_PRESSED(SW1_POS)) { // flash white
      RTCS_Enable_Task(Task_Flash_FSM, 1);
      RTCS_Release_Task(Task_Flash_FSM);
      RTCS_Enable_Task(Task_RGB_FSM, 0);
    } else {
      RTCS_Enable_Task(Task_Flash_FSM, 0);
      RTCS_Enable_Task(Task_RGB_FSM, 1);
      RTCS_Release_Task(Task_RGB_FSM);
    }
  }
  if ((PORTD->ISFR & MASK(SW2_POS))) {
    if (SWITCH_PRESSED(SW2_POS)) {
      RTCS_Set_Task_Period(Task_Flash_FSM, W_DELAY_FAST);
      RTCS_Set_Task_Period(Task_RGB_FSM, RGB_DELAY_FAST);
    } else {
      RTCS_Set_Task_Period(Task_Flash_FSM, W_DELAY_SLOW);
      RTCS_Set_Task_Period(Task_RGB_FSM, RGB_DELAY_SLOW);
    }
  }
  // clear status flags
  PORTD->ISFR = 0xffffffff;
  PTB->PCOR = MASK(DEBUG_ISR_POS);
}
```

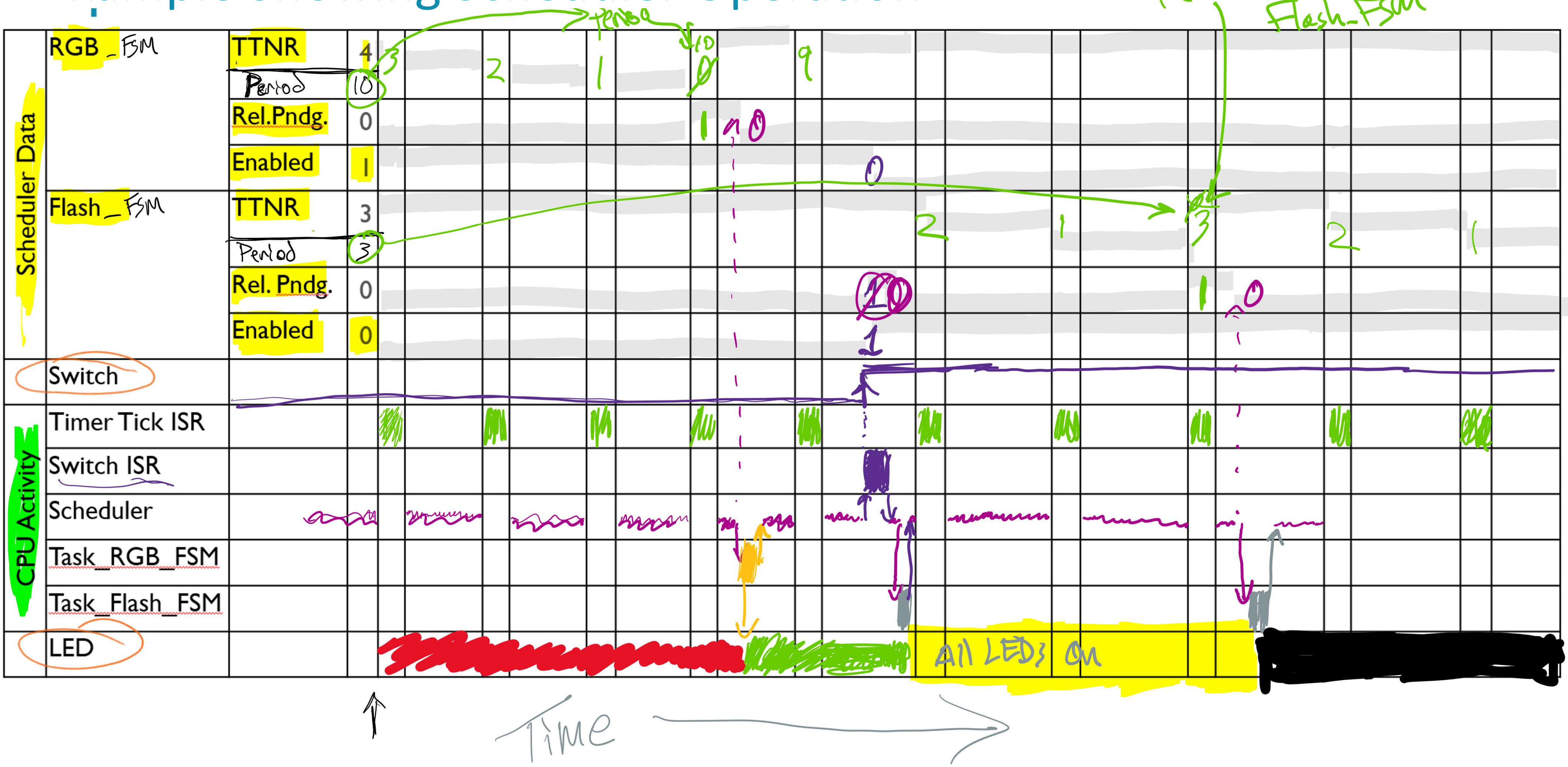
Scheduler Set-Up

```
int main (void) {
  Init_Debug_Signals();
  Init_RGB_LEDs();
  Init_Interrupts();

  RTCS_Init(100); // 100 Hz timer ticks

  RTCS_Add_Task(Task_Flash_FSM, 0, W_DELAY_SLOW);
  RTCS_Enable_Task(Task_Flash_FSM, 0);

  RTCS_Add_Task(Task_RGB_FSM, 1, RGB_DELAY_SLOW);
  RTCS_Enable_Task(Task_RGB_FSM, 1);

  RTCS_Run_Scheduler();  // This call never returns
}
```

priority

NC STATE UNIVERSITY

Example Showing Scheduler Operation



Analyze Resulting Performance with Logic Analyzer

- Measure the responsiveness…
  - For switch 1 (Flash/RGB)
  - For switch 2 (Fast/Slow)
- How much idle time?

EXAMPLE APPLICATIONS: FIRST VERSIONS WITH RTC SCHEDULER AND INTERRUPTS

Example Applications