ECE3849
D-Term 2021

Real Time Embedded Systems

Module 4 Part 3

Module 4 Part 3 Overview

- Real-time debugging tools in CCS
  - ROV Classic
  - RTOS Analyzer
- Designer developed Software measurements
  - Latency
  - Execution time
  - CPU load
- Using lab equipment to take measurements

Real-time Debugging

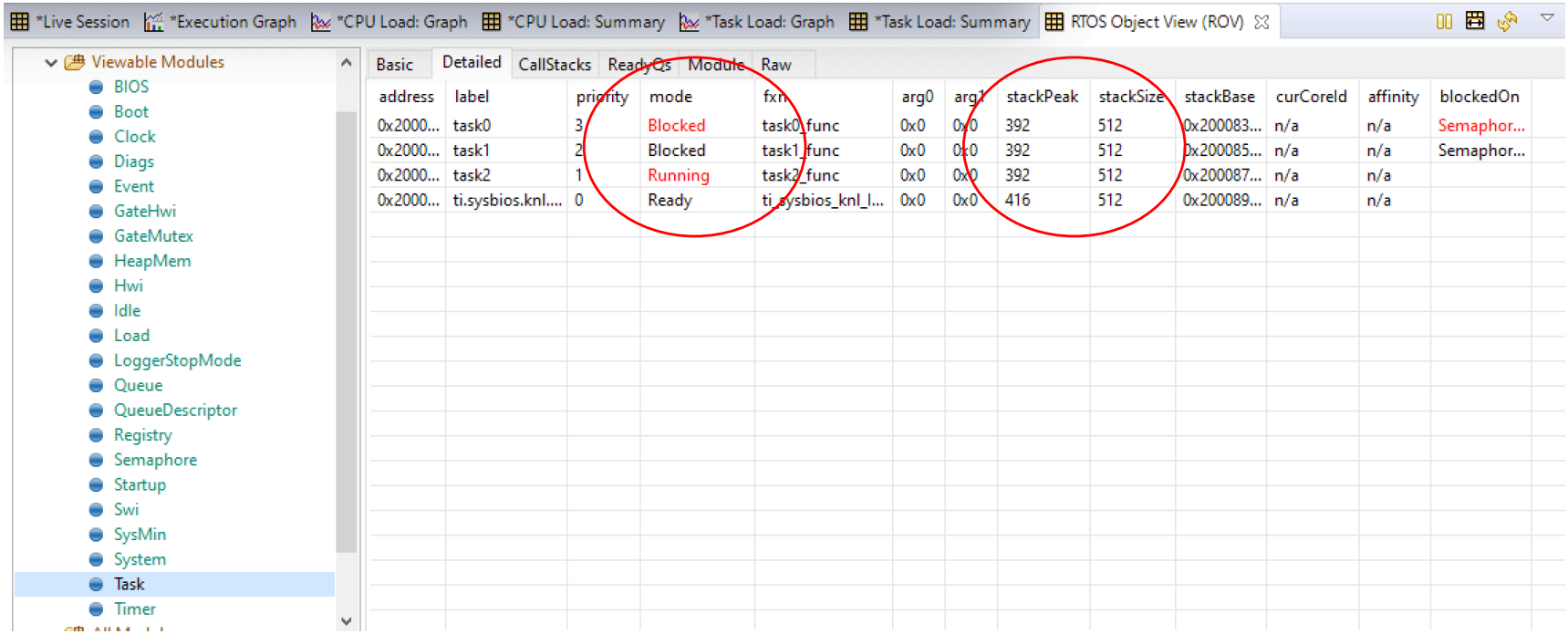
- Common debugging measurements.
  - Logging and observing program status in real time.
  - Measuring code execution time.
  - Measuring CPU load (utilization).
  - Measuring latency and response time.
- Real-time debugging methods must be nonintrusive or minimally intrusive.
  - They should have a negligible affect on the measurements taken.
  - To be completely nonintrusive, hardware debugging tools are needed.
  - Software debugging will always be intrusive.
    - Additional software instructions will always required to take the measurements, but the affect can be minimized with careful coding.

TI-RTOS Logging

- TI-RTOS can log events with timestamps.
  - Task context switches.
  - ISR entry and exit.
  - Semaphore posts and pends.
  - Task execution state and scheduling.
- It then reconstructs the system behavior from the logs and makes results accessible to the developer.
- TI-RTOS Tools.
  - ROV Classic.
    - Provides System Status for all TI-RTOS objects.
    - Useful in viewing the current state of tasks and their stack utilization.
  - RTOS Analyzer.
    - Provides time plots of semaphore, interrupt and task activity.
    - Provides CPU and task utilization information.

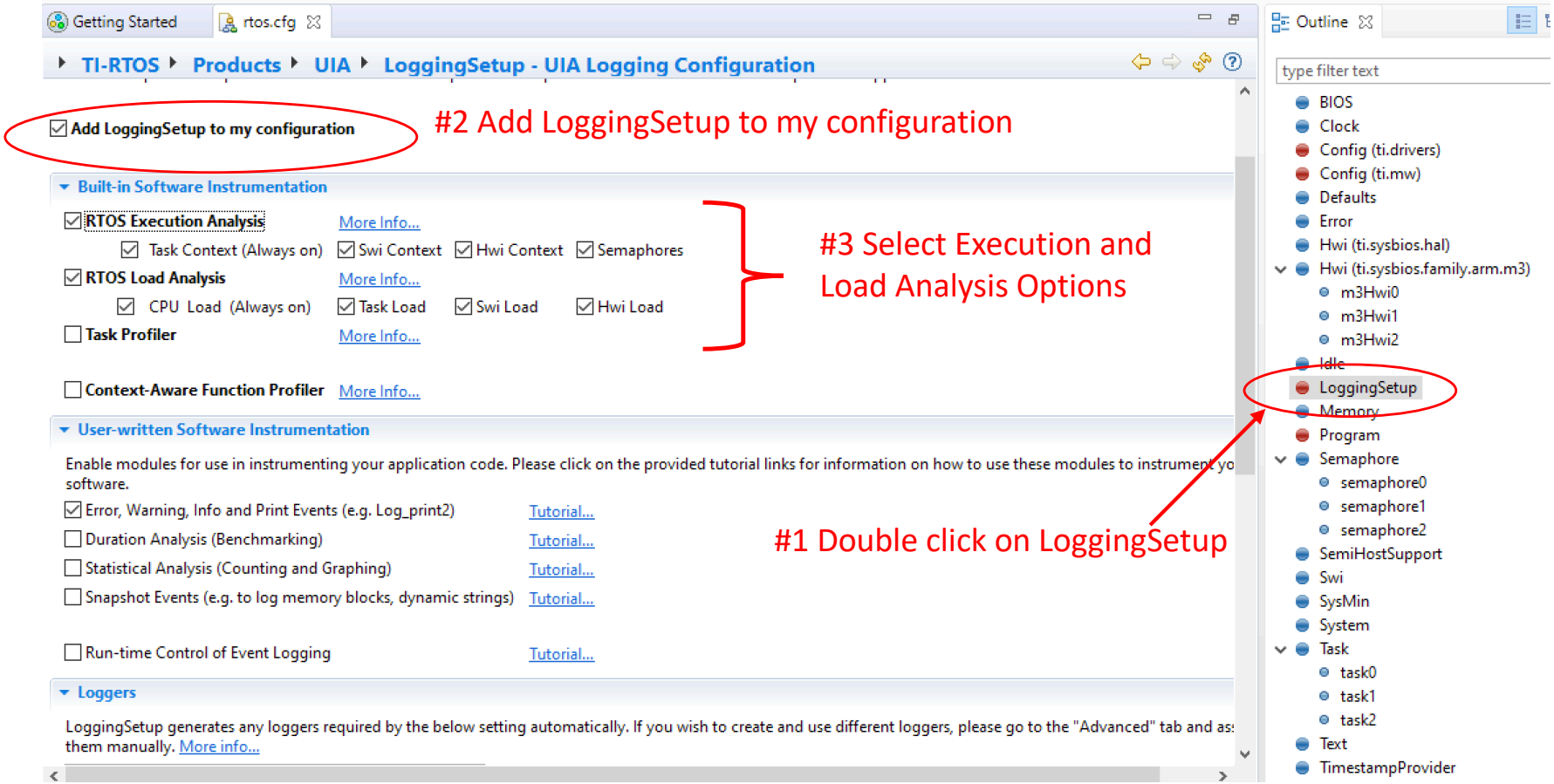
ROV Classic

- RTOS Object View (ROV)
  - Start the debugger.
  - In the debugger tool, select Tools -> ROV Classic.
  - Run and then pause debugger to view current information.
- On the left it shows all the viewable RTOS objects.
  - Clicking on the object type will show the latest status.
  - For tasks, the Detailed view shows the state of each task and how much of its stack it is using.

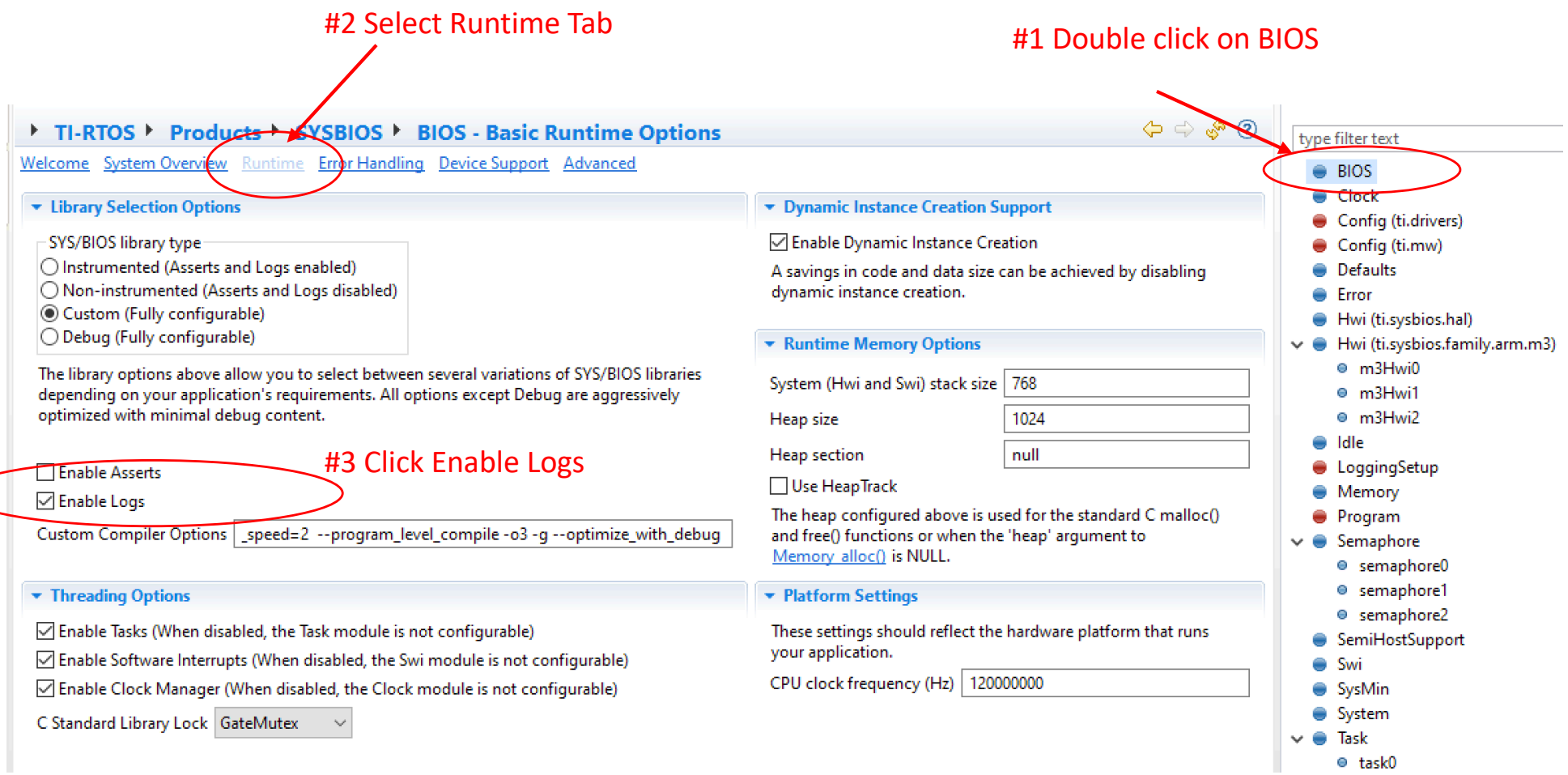


RTOS Analyzer Setup: Add LoggingSetup

- RTOS Analyzer Setup.
  - Requires LoggingSetup to be enabled and configured.
  - Requires BIOS -> Runtime settings to enable logs.
  - Save rtos.cfg after changes.



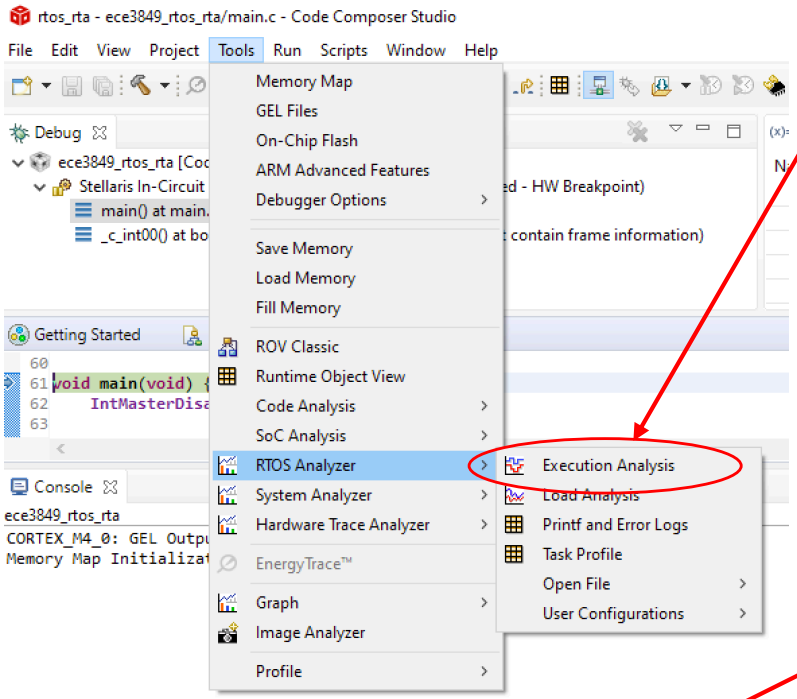
RTOS Analyzer Setup: Enable Logs

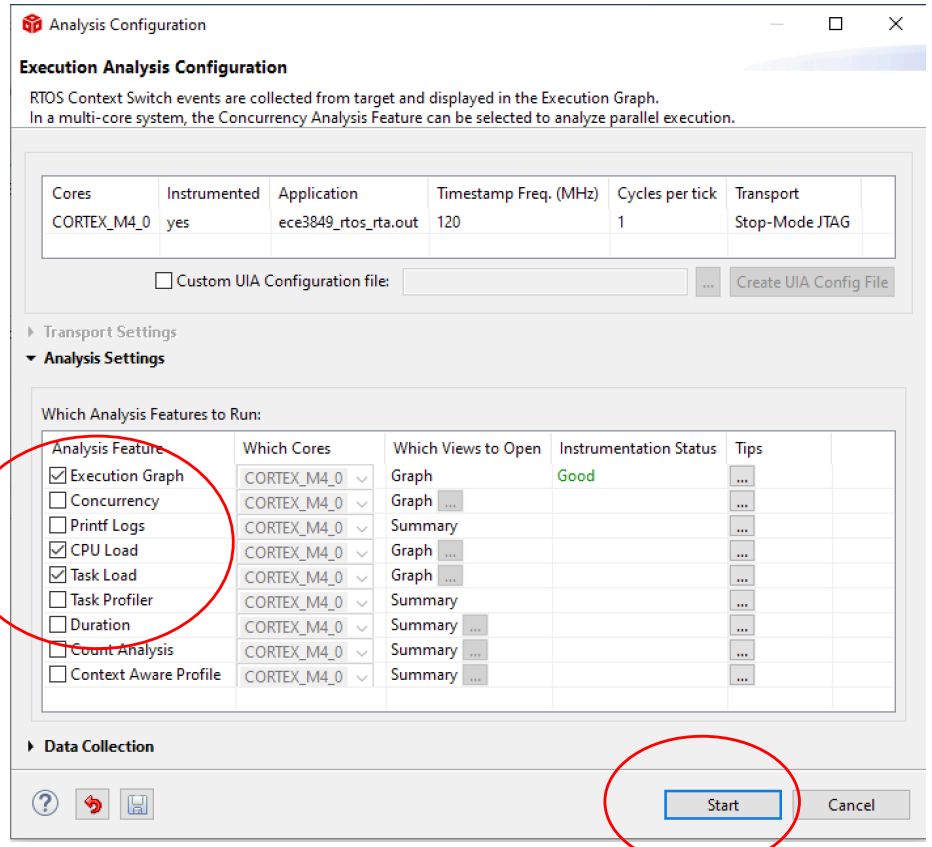


Running RTOS Analyzer

- Start the debugger.
- In the debugger tool, select Tools -> RTOS Analyzer -> Execution Analysis.
- Check the Execution Graph, CPU Load and Task Load Options.
- Click Start.

#1 Select Execution Analysis



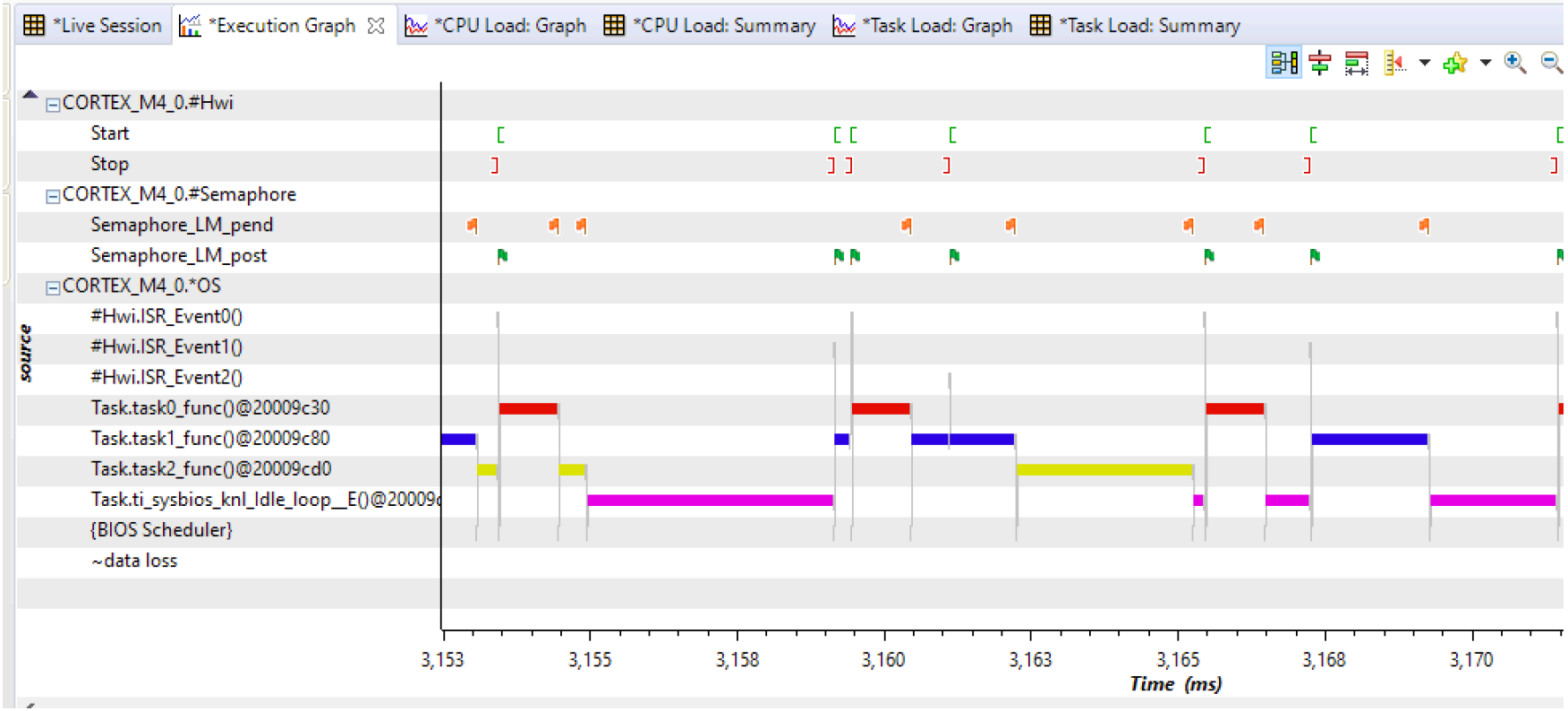


#2 Enable
Execution Graph
CPU Load
Task Load

#3 Start the tool

RTOS Analyzer: Execution Graph

- Run the debugger to start gathering data.
- Pause the debugger to view the data.
  - Press the pause button.
  - Breakpoints can be used to stop at a certain point in the code.
- View your data.
  - The Execution Graph shows a graph of RTOS scheduling activity, task execution, semaphore activity, and interrupt status over time.

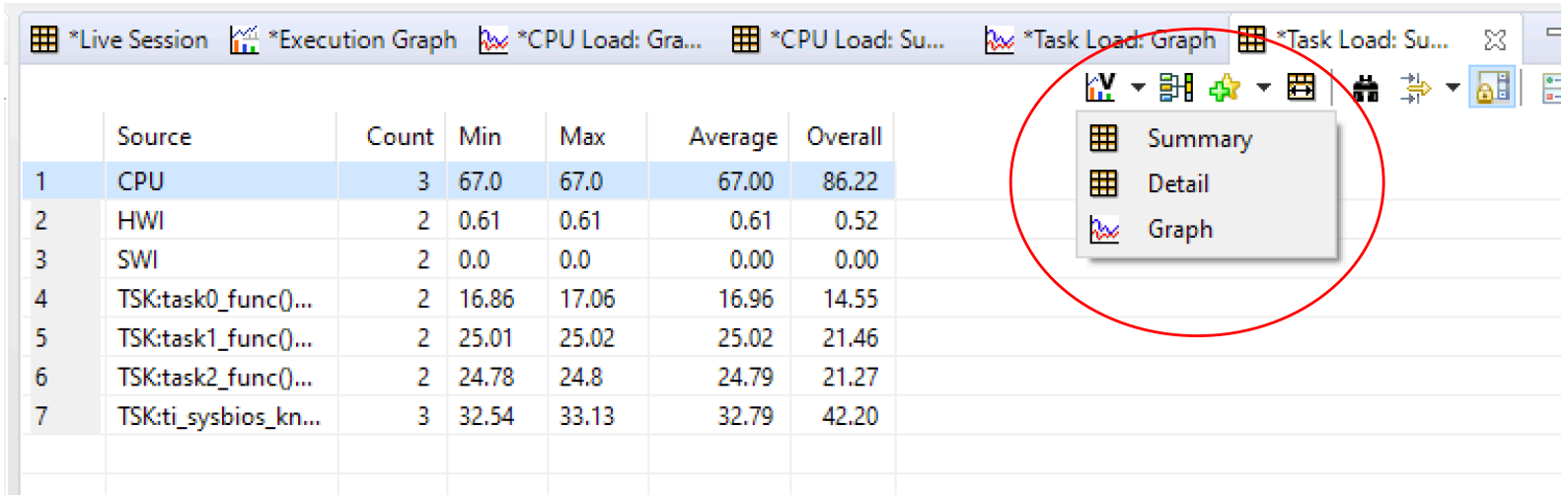


CPU Load Measurements

- In lab1, we took CPU load measurements by instrumenting our code.
  - Using an incrementing counter in a while loop, we counted for a fixed period of time.
  - We took an unloaded measurement with interrupts disabled.
  - We took a loaded measurement with interrupts enabled over a fixed time interval.
  - The fraction of loaded/unloaded*100 was the % of time spent doing all real time tasks.
- We then displayed the value to the LCD to observe continual real-time values.
  - Advantage: Allowed real-time in system monitoring of results.
- It is a good measurement but had limits.
  - Disadvantage: We could not break down utilization on a per task basis.
  - Disadvantage: The measurement itself took CPU time and delayed execution of other tasks like drawing the LCD screen.

RTOS Analyzer: CPU Load Measurement

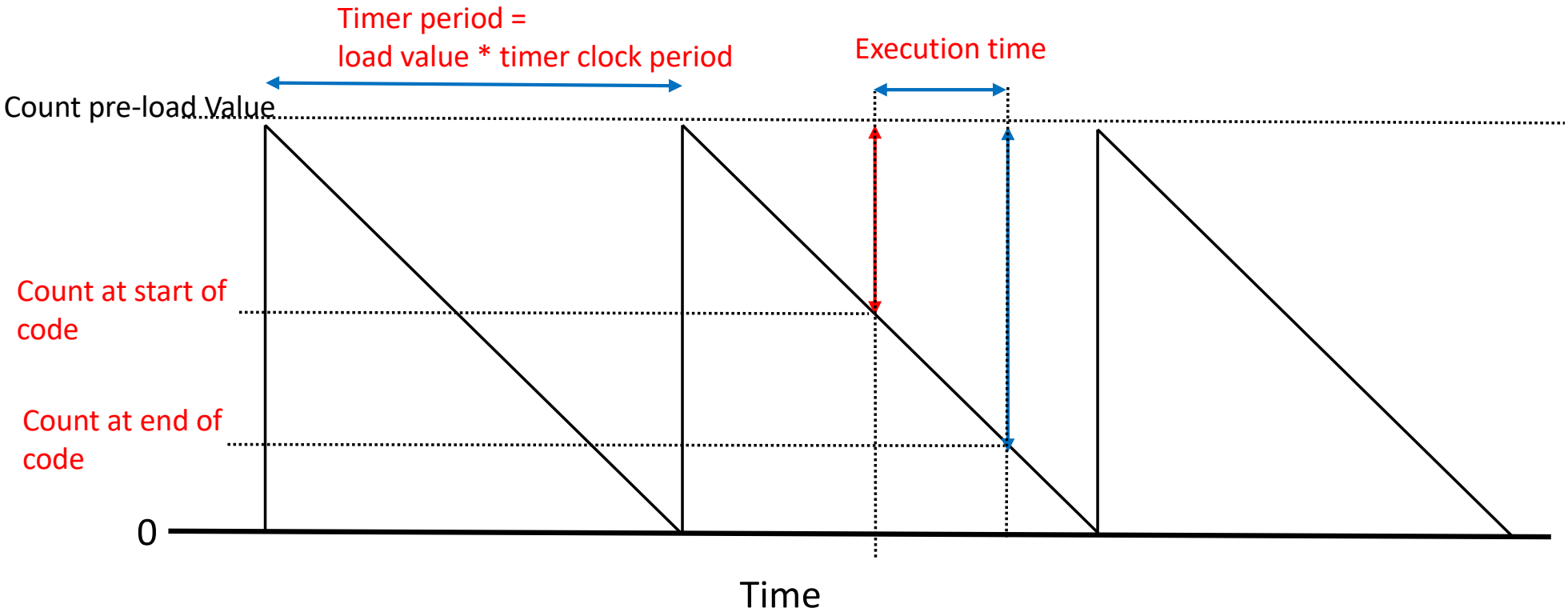
- RTOS Analyzer has several benefits.
  - It breaks down utilization for each thread.
  - It provides the results in an easy to read GUI interface.
  - It is already debugged and ready to use.
  - Over the monitoring period several load measurements are taken.
    - The Task Load Summary gives a clear snapshot of CPU utilization and how much each task is taking (min, max and average).
- Limitations of RTOS Analyzer.
  - Specific to TI-RTOS and CCS tools support = not portable.
  - Results can only be accessed in a debug environment, not live in system.
  - Logs of all events are taken and retrieved. This can have significant overhead, increasing latency and CPU load.
  - Does not always show the measurements you want. Neither the CPU load or the Execution Graph calculate the maximum response time.



	Source	Count	Min	Max	Average	Overall
1	CPU	3	67.0	67.0	67.00	86.22
2	HWI	2	0.61	0.61	0.61	0.52
3	SWI	2	0.0	0.0	0.00	0.00
4	TSK:task0_func()...	2	16.86	17.06	16.96	14.55
5	TSK:task1_func()...	2	25.01	25.02	25.02	21.46
6	TSK:task2_func()...	2	24.78	24.8	24.79	21.27
7	TSK:ti_sysbios_kn...	3	32.54	33.13	32.79	42.20

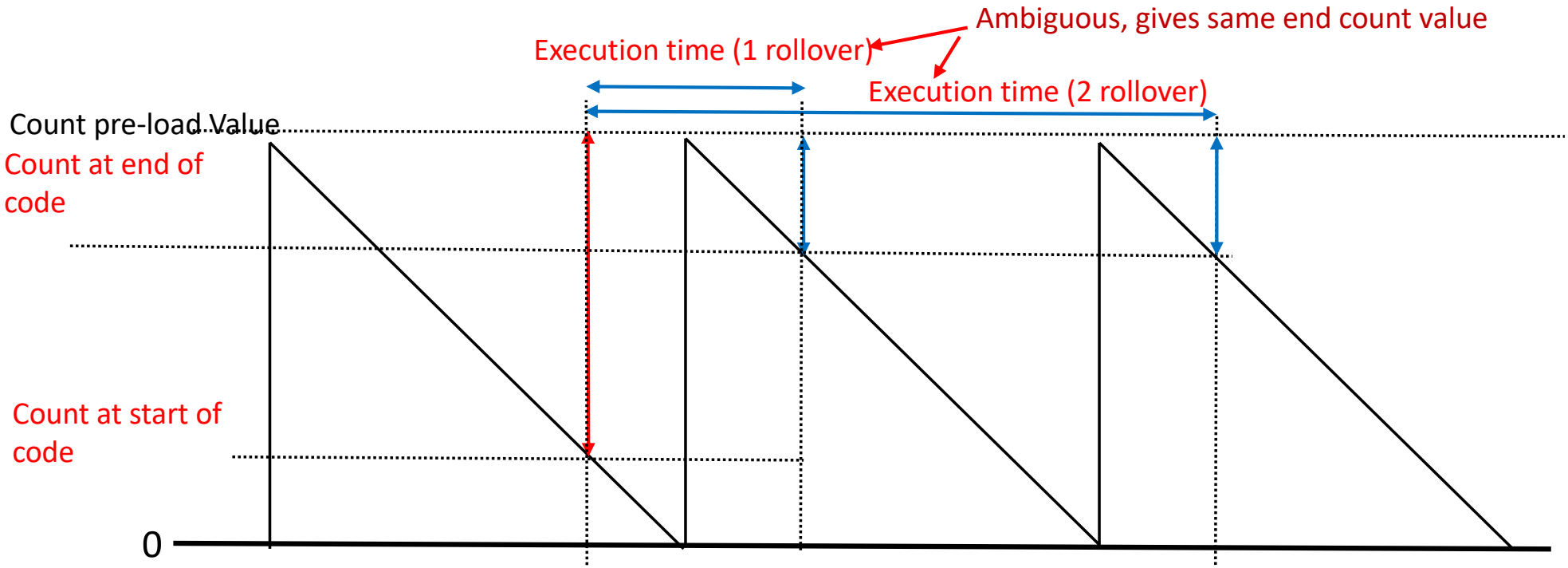
Measuring Execution Time: Timers

- A common way to measure execution time is to use a timer.
  - You will first need to setup your timer.
    - Timer will start at a preloaded value and count up or down on every timer clock tick.
    - The timer period = load value * timer clock period.
  - Then instrument your code to read the timer count value before and after executing the code of interest.
  - Devise experiments to exercise the worst case path, yielding the maximum execution time.
  - Convert your count values to actual time, time = delta count value * usec per count.



Execution Time: Timer Setup

- Setup your timer / know your timer settings.
  - Is it counting up or down?
  - What is the input clock frequency of the timer?
  - If counting down, a rollover occurs when the counter reaches 0 and reloads the original count value. If counting up, it resets to 0 after reaching its maximum count.
    - Using load values of 2^N– 1, simplifies the roll over calculation.
    - Example if using a 24-bit counter:
      - execution count = (end count – start count) & 0xFFFFFF ;
  - Set your timer period to be greater than the maximum time you want to measure in a continuous mode.
    - This will guarantee only one rollover occurs in a measurement period and avoids ambiguous measurements.

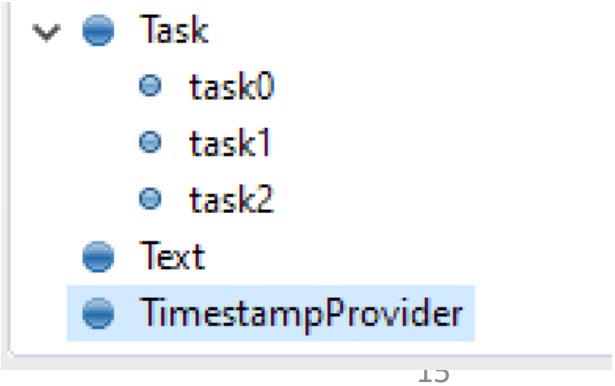


Measuring Execution Time: Timers

- Using a timer to measure execution time.
  - Setup your timer.
  - Disable interrupts during the measurement time.
    - We do not want any preemption during our measurement.
  - Adjust for your measurement bias.
    - It takes time to read the count values, this will give a bias to your measurements.
    - For accurate measurements you will want to adjust for the bias.
    - Do two reads in a row without code between them. The count difference is your measurement bias.
  - Read the timer count before and after executing the code.
  - Calculate Execution time in number of clocks.
    - Execution time in clock ticks = difference between the start and stop count - bias.
  - Save the maximum execution time value.
    - Sometimes it is interesting to record the minimum, maximum, average times but it increases measurement time and we only really care about the maximum.
  - Repeat the measurement, exercising all alternative code paths to get the worst case.
  - Convert clock counts to time delay.
    - Execution time (usec) = Execution time (clock ticks) * usec per clock tick.

Timer Options

- Hardware timer (similar to ece3849_rtos_latency example).
  - Configure hardware timer using Timer*() function calls.
  - Read hardware timer registers directly TIMER0_TAR_R.
  - Considerations.
    - Uses an additional dedicated hardware resource, fussy to setup.
    - Minimizes measurement times reading directly from registers.
    - Can configure clock inputs and load values to customize period.
- SysTick Timer (see ece3849_profiling example).
  - Uses simple 24-bit timer integrated into the NVIC controller for the Cortex-M processor. Runs off of CPU clock, 120 MHz.
  - Requires #include “driverlib/systick.h”.
  - Simple setup, independent of BIOS.
    - SysTickPeriodSet(1<<24); //full 24 bit value.
    - SysTickEnable(); //start counter.
    - countValue = SysTickValueGet(); // get counter value.
- BIOS Timestamp Module (see BIOS user guide).
  - Double click, TImestampProvider in rtos.cfg GUI at very bottom of Outline window.
    - In the configuration window add the module the project.
    - Click use clock’s timer.
  - In main, include
    - #include <xdc/runtime/Types.h>
    - #include <xdc/runtime/Timestamp.h>
  - No other configuration is needed.
  - countValue = Timestamp_get32(); //read 32-bit count value.



Example ece3849_profiling

```
IntMasterDisable();

// Enable the Floating Point Unit, and permit ISRs to use it
FPUEnable();
FPULazyStackingEnable();

// Initialize the system clock to 120 MHz
gSystemClock = SysCtlClockFreqSet(SYSCTL_XTAL_25MHZ | SYSCTL_OSC

// initialize SysTick timer for the purpose of profiling
SysTickPeriodSet(1 << 24); // full 24-bit counter
SysTickEnable();

for (i = 0; i < N_32BYTE_BLOCKS * 8; i++)
    src[i] = i;

// dry run to determine how long it takes to read the timer
start = SysTickValueGet(); // read SysTick timer value
finish = SysTickValueGet();
bias = (start - finish) & 0xffffff;

// profile the assembly function 4-byte blocks
start = SysTickValueGet();
block_copy4(dst, src, N_32BYTE_BLOCKS * 8);
finish = SysTickValueGet();
asm4_cycles = ((start - finish) & 0xffffff) - bias;
```

Disable Interrupts

Configure SysTick TImer

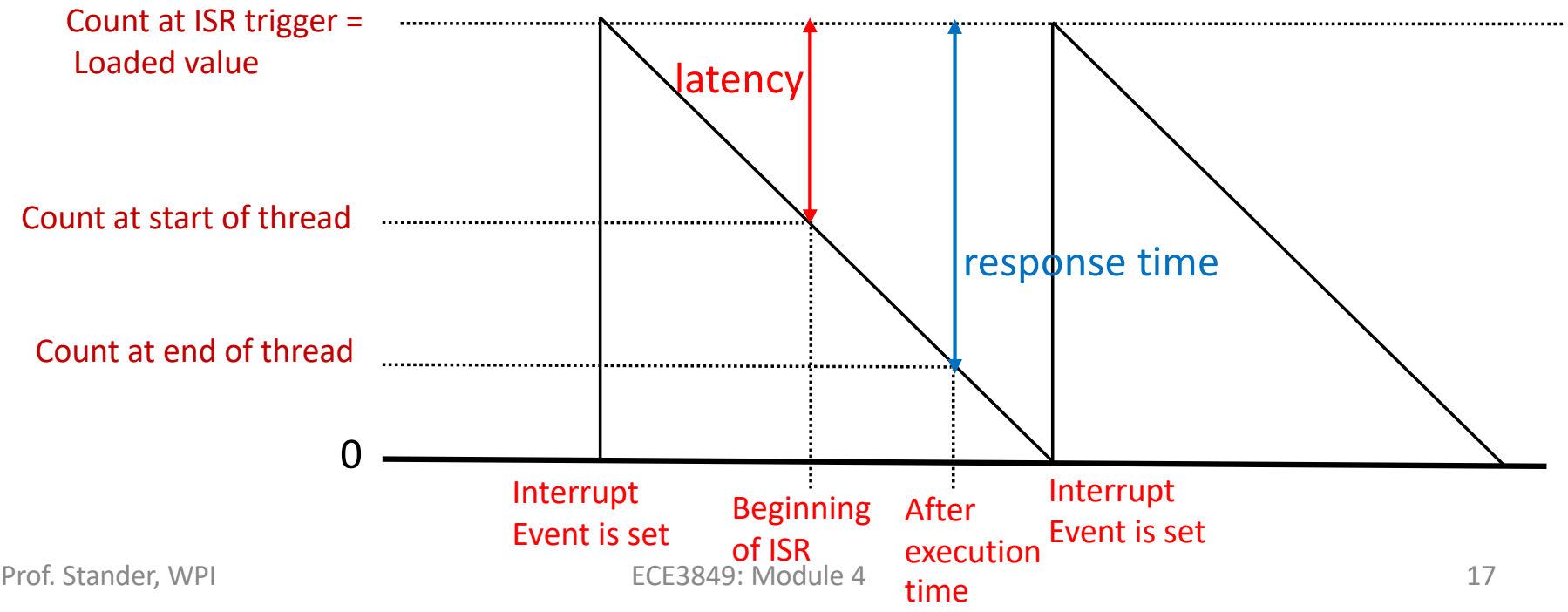
Initialize data values

Calculate measurement bias

- Measure start count
- Run code
- Measure end count
- Calculate execution time in clock counts

Measuring Latency: Using a timer

- To measure latency, a timer can be configured to trigger an interrupt.
  - If measuring the latency of a task the interrupt would then call the task.
  - In class examples: ece3849_int_latency and ece3849_rtos_latency.
  - When the interrupt is triggered the counter value equals its load value.
  - The count value is then measured at the beginning of the interrupt or task.
    - Latency (in timer counts) = Counter Load Value - measured count value.
  - Store the maximum value of the latency.
  - Exercise all alternative code paths to guarantee the worst case conditions have been tested.

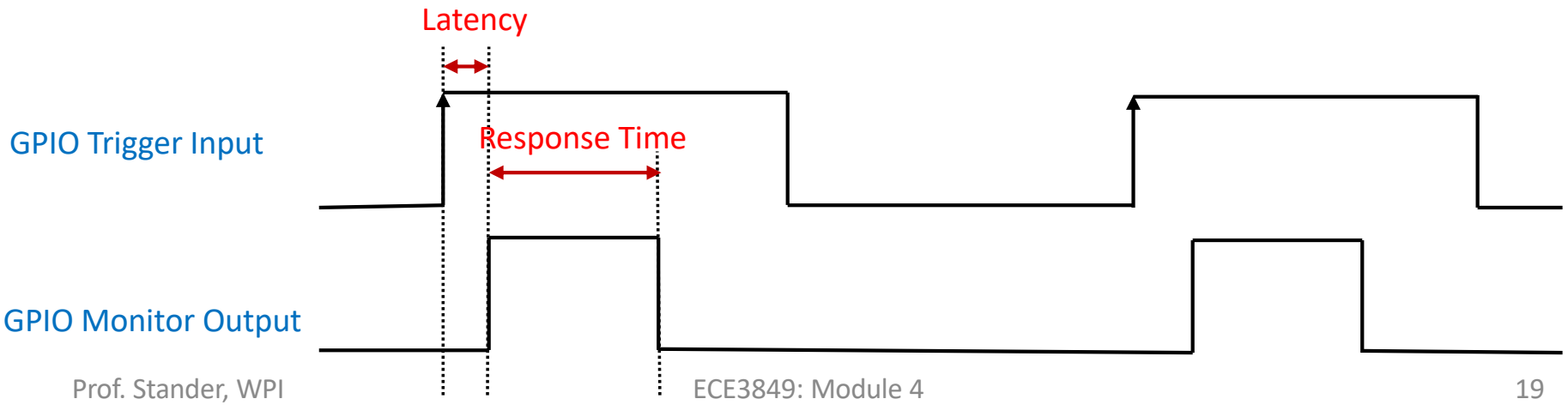


External Measurement Techniques

- The TI provided software integrated tools can only analyze the system performance over relatively short periods of time.
  - They have limited memory resources to store logs and increase CPU load.
  - Some require that the debugger be paused to retrieve data.
- Using external equipment such as signal generators, logic analyzers, and oscilloscopes…
  - Allow the for real-time debugging over longer periods of time.
  - Allow the developer to see the distribution of measurements if the equipment has a persistent viewing mode.
- Both latency and response time can be measured using lab equipment.

Lab Measurement: Latency & Response Time

- Trigger generation
  - Instead of triggering the interrupt off of an internal timer, you can trigger it off the rising edge of a GPIO input.
  - This trigger could be generated using a signal generator.
  - Or by looping back a GPIO output to its input like we did in our labs with the PWM signal.
- Observing Latency and Execution time.
  - Configure a GPIO as an output to monitor the thread state.
  - When the thread is entered drive the output high.
  - When the thread is exited drive the output low.
  - Connect both the trigger input and the GPIO output to an oscilloscope or logic analyzer for viewing.
- The latency is the delay between the rising edge of the GPIO trigger input and the rising edge of the GPIO monitor output.
- The response time is the length of time the GPIO monitor output is high.
- To generate an execution time graph similar to the RTOS analyzer each thread could have its own GPIO monitor output.



Lab Equipment: Measuring Maximum Values

- Some oscilloscopes can setup measurements to measure delays and pulse times.
  - These measurements can keep statistics and report min, max and average times.
- Oscilloscopes can be placed in a persistence mode.
  - Triggering the oscilloscope on the input trigger rising edge, will allow all the triggers to display at the same location on the screen.
  - Persistence will overlap all the output results and cursors can be used to measure desired values.