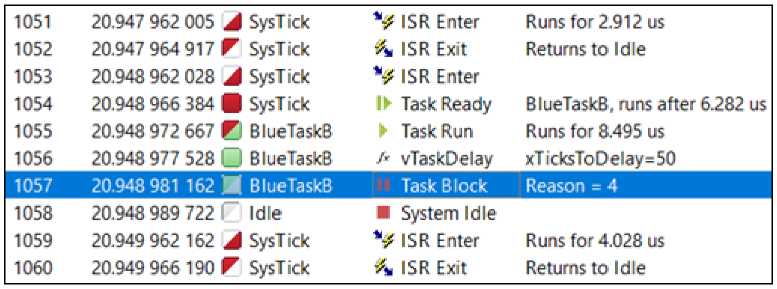
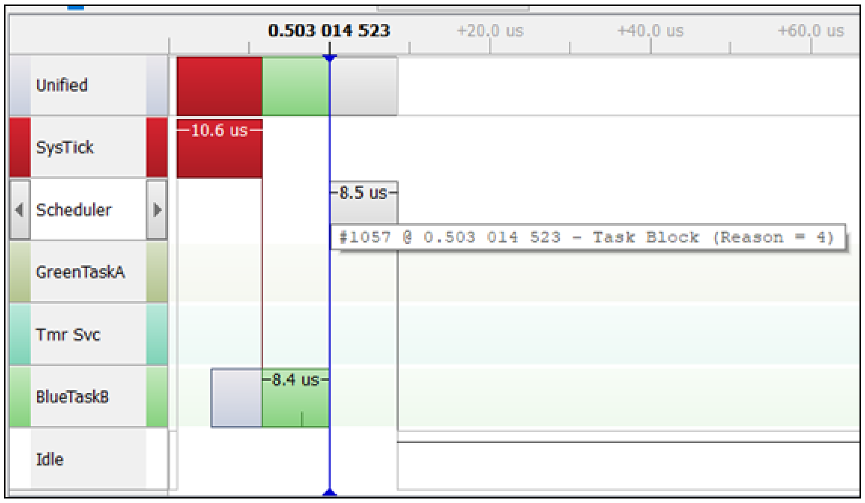


1051	20.947 962 005	SysTick	ISR Enter	Runs for 2.912 us
1052	20.947 964 917	SysTick	ISR Exit	Returns to Idle
1053	20.948 962 028	SysTick	ISR Enter	
1054	20.948 966 384	SysTick	Task Ready	BlueTaskB, runs after 6.282 us
1055	20.948 972 667	BlueTaskB	Task Run	Runs for 8.495 us
1056	20.948 977 528	BlueTaskB	vTaskDelay	xTicksToDelay=50
1057	20.948 981 162	BlueTaskB	Task Block	Reason = 4
1058	20.948 989 722	Idle	System Idle	
1059	20.949 962 162	SysTick	ISR Enter	Runs for 4.028 us
1060	20.949 966 190	SysTick	ISR Exit	Returns to Idle



#1057 @ 0.503 014 523 – Task Block (Reason = 4)

- Idle task:
  - The Idle task runs when no other tasks are running.
  - For `GreenTaskA` and `BlueTaskB`, their priority is set relative to the `Idle` task, and higher than the `Idle` task.
  - For `mainSemExample.c`, the `Idle` task runs for relatively long periods. During that time, the `SysTick ISR` is run every 1 ms (shown in events 1058 and 1059, above)

- Scheduler
  - *"...FreeRTOS doesn't even have a real scheduler. It maintains a list of runnable tasks, and at every scheduling point (return from interrupt or explicit yield), it takes the highest priority task from that list."*
    - https://stackoverflow.com/questions/7506461/implementing-scheduler-in-free-rtos
  - That Stack Overflow post seems plausible, but I haven't confirmed it.
  - From the screen-shots above, it appears that when `BlueTaskB` runs `vTaskDelay, it` results in `scheduler-code` being run (shown in the row `Scheduler`.) And, the scheduler-code changes `BlueTaskB` to the `Blocked` state, and it starts the `Idle` task (event 1057 and blue vertical-line, and event 1058).

- **Bug** in book and code (`mainSemExample.c`), page 182f
  - Problem:
    - The operations on the variable "`flag`" here need to be atomic, but they are not.
    - `flag` is a global variable. It's set and referenced by the two concurrent tasks `GreenTaskA` and `BlueTaskB`.
    - In general, for two tasks to use a variable like `flag` in this manner, atomic instructions are needed, for the code to work properly.
      - If this code does not require atomic instructions to work properly, determining that with certainty would be difficult and impractical, in my estimation.
    - It appears that the generated assembly language does not use atomic instructions.
      - The generated assembly language uses `LDR` and `STR`. It's shown below.
      - In my Internet searches, I didn't see mention of `LDR` and `STR` being atomic. But, my search wasn't exhaustive.
  - Possible solutions, for concurrent operations on the `flag` variable:
    - A mutex could be used, when accessing `flag`