Project 2 Report Template

Date: 8th November 2018

Your Name –Saloni Shambhuwani

Unity Email Address-sdshambh@ncsu.edu

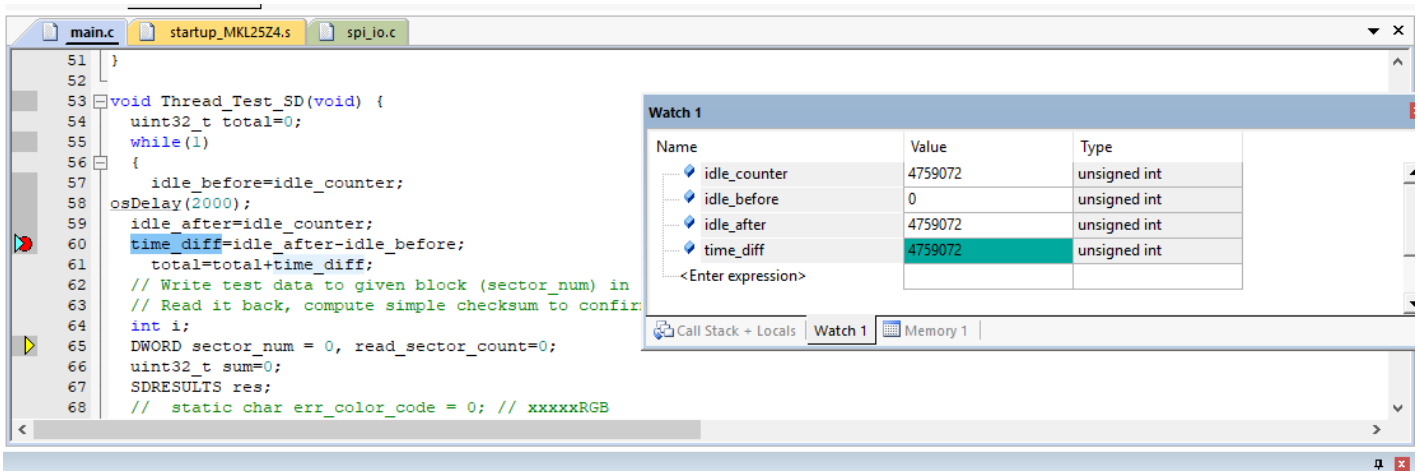
Note: For the requested diagrams, scans or photos of hand-drawn diagrams are sufficient for full credit.

Part B

Port to CMSIS-RTOS2

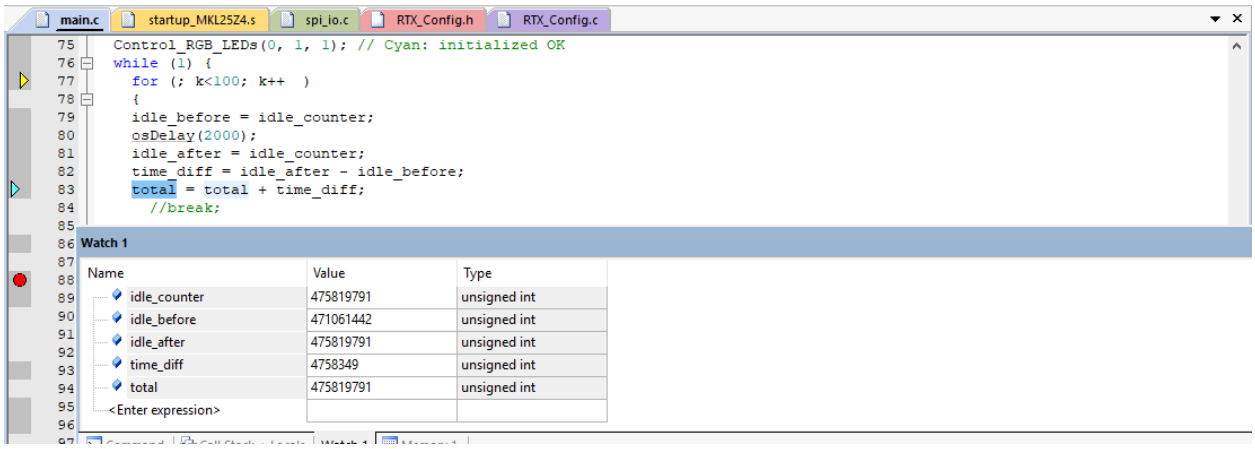
5. a. What is the "100% idle for one second" value for idle_counter?

Ans: 100% idle for one second is 4759072



6. Idle loop analysis

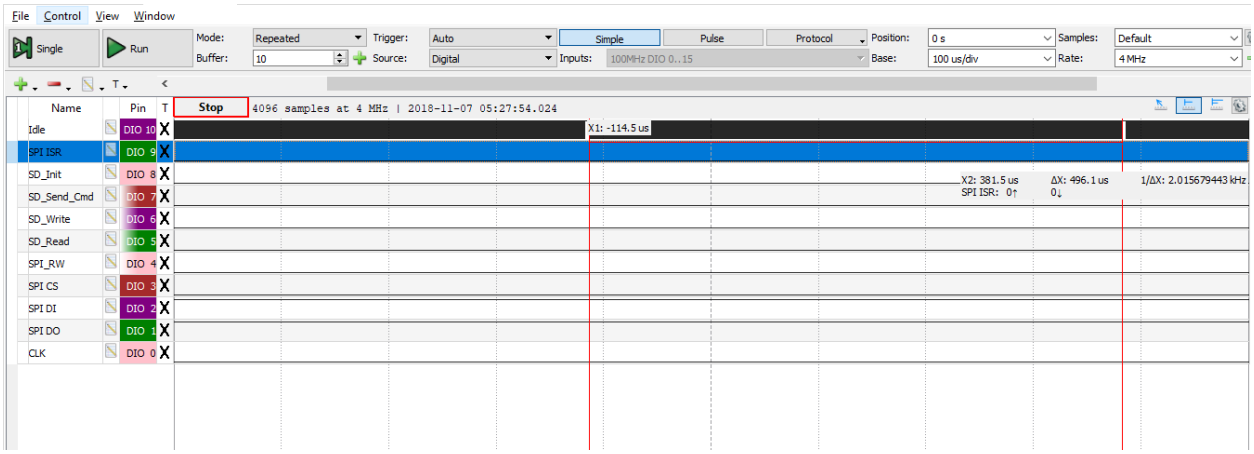
a. How long is an average idle loop iteration (based on 100 iterations)? Include a screenshot.



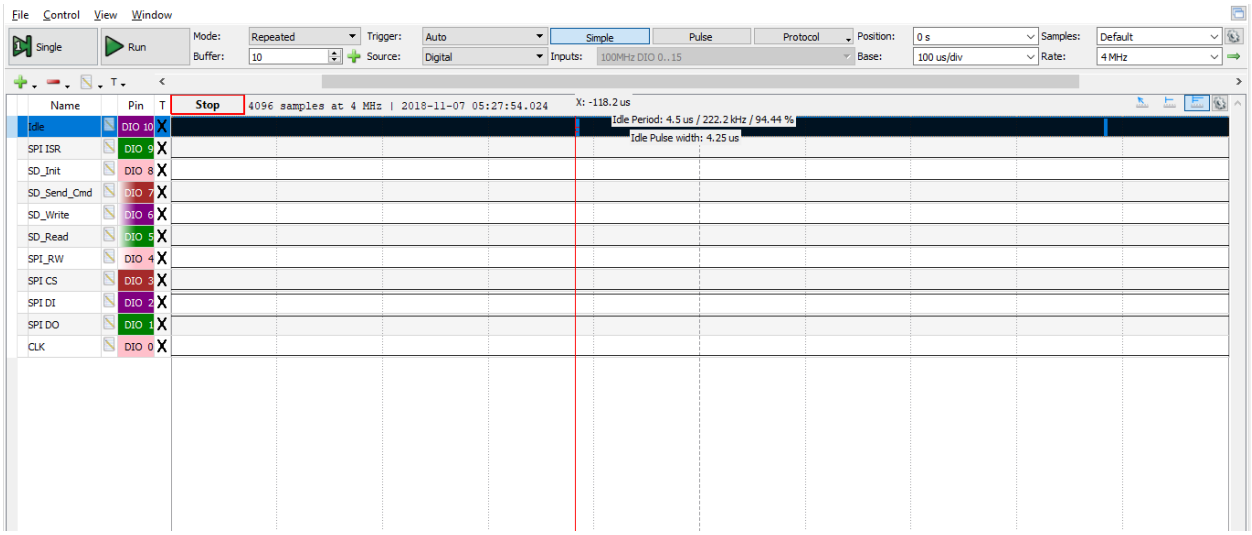
Average idle loop iteration is total/100=4758197 nearly 0.8 us

b. How often does the scheduler tick run, and how long does it take each time? Determine this using the gaps in the idle debug signal. What fraction of the processor's time does the timer tick use? Include a screenshot.

Ans: Scheduler tick runs every 0.5 ms=500 µs



It takes 4.25µs each time



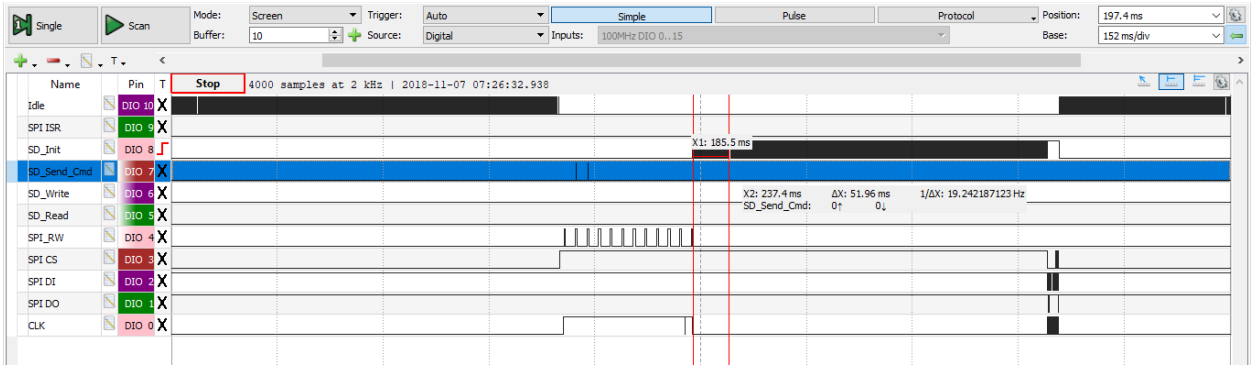
Timer tick uses [4.25/(500+4.25)]*100 =0.8428 fraction of processors time

7. **How long does each of the following functions take, and how long is each segment (computation, SPI comm. for polling, other SPI comm.)? The computation time is how long the function would take if the SD card and the SPI communication were infinitely fast (taking no time). Include a screenshot marked to indicate the busy waiting time.**
    a. **SD_Init**
    b. **SD_Read**
    c. **SD_Write**

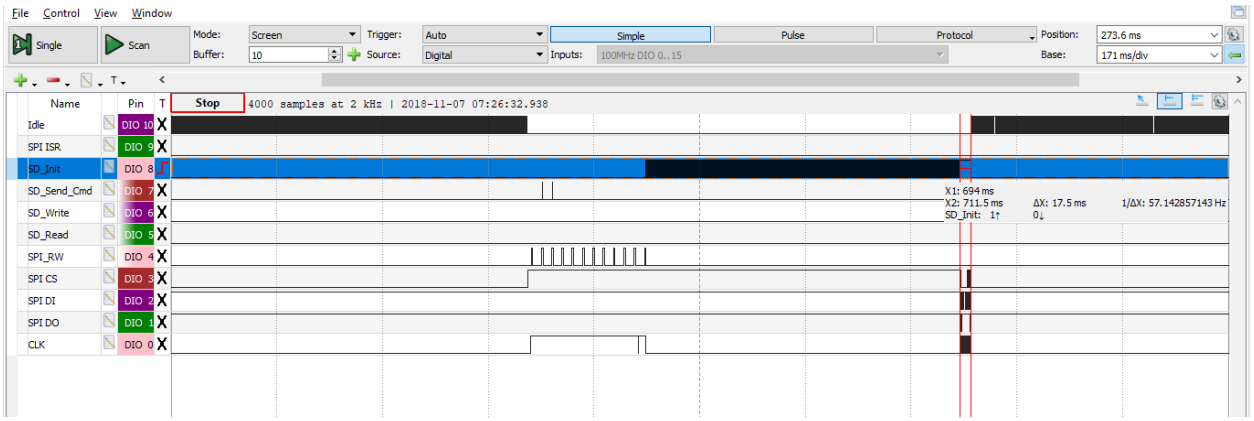
Ans:

SD_Init:

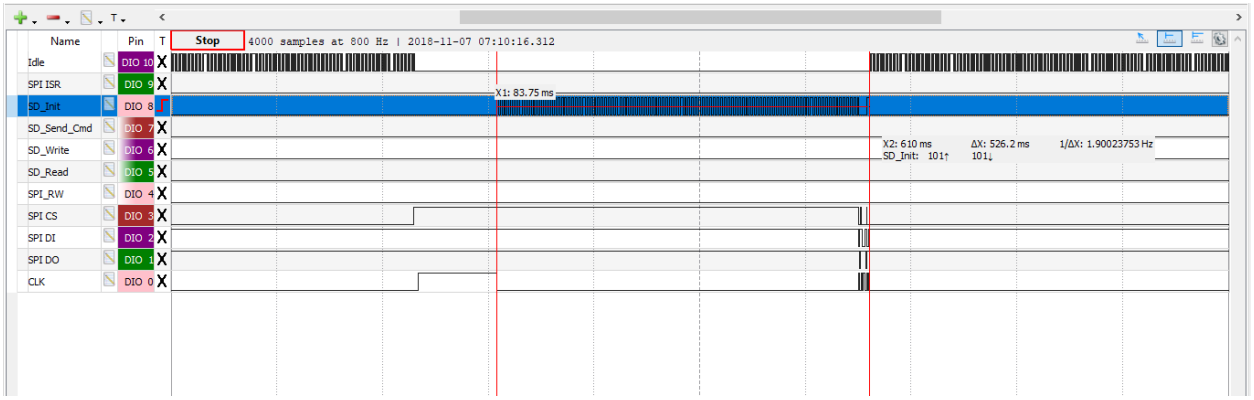
First busy wait loop=51.96 ms

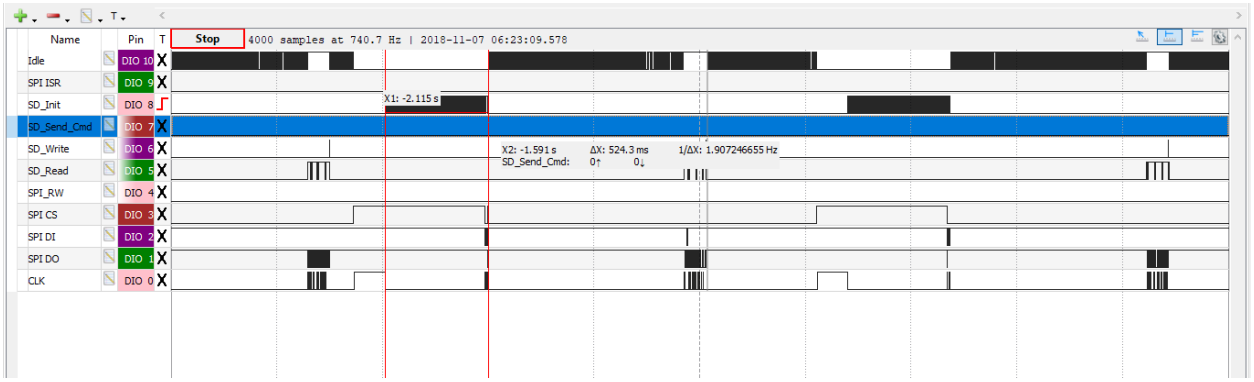


Computation time:17.5ms



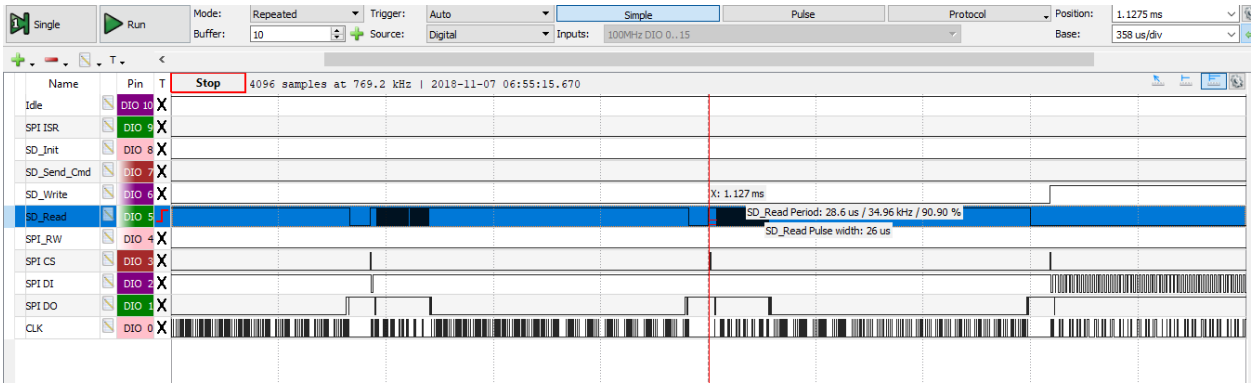
Total SD_Init time= 524.3ms



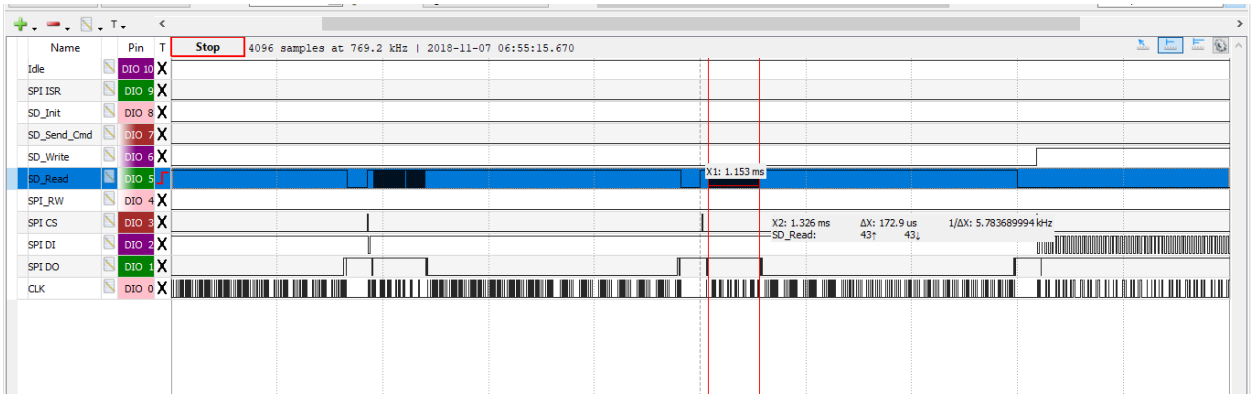


SD_Read:

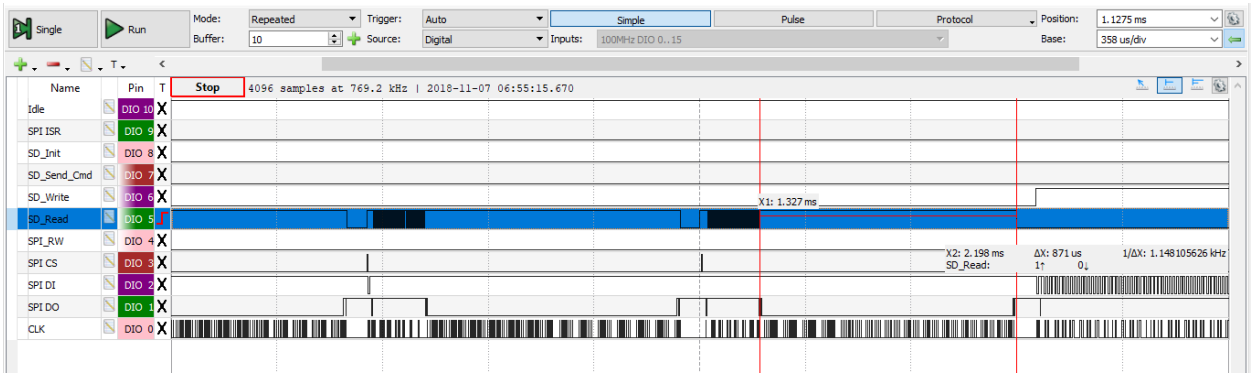
Initial computation time : 28.6 µs



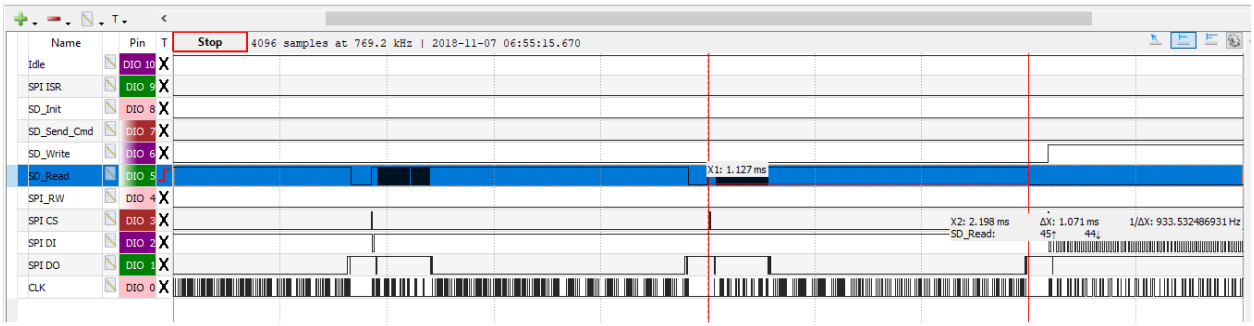
Wait time to get the packet: 172.9 µs



SPI communication and other computation: 871 µs

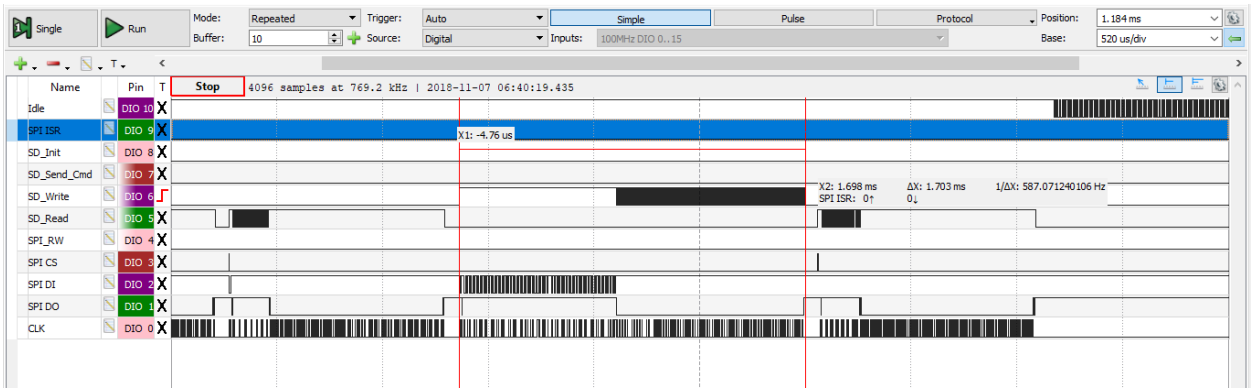


Total SD_Read time: 1.071ms

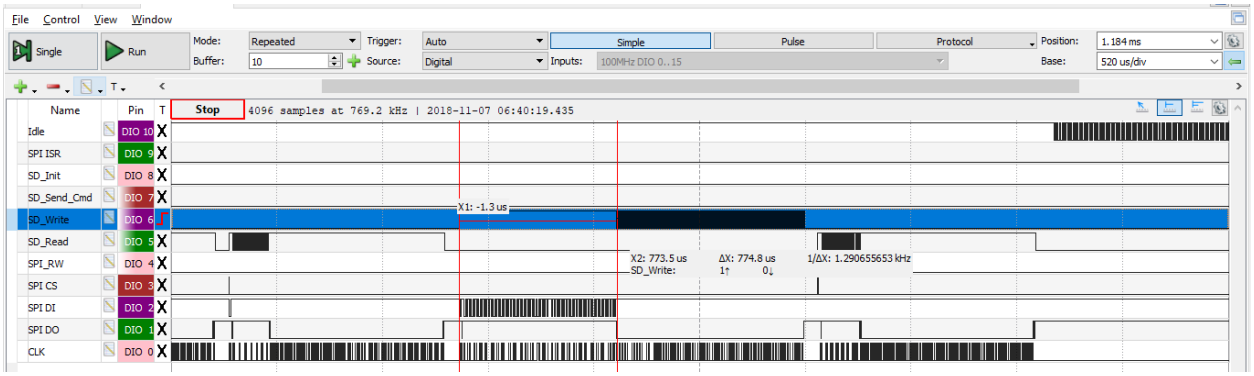


SD_Write :

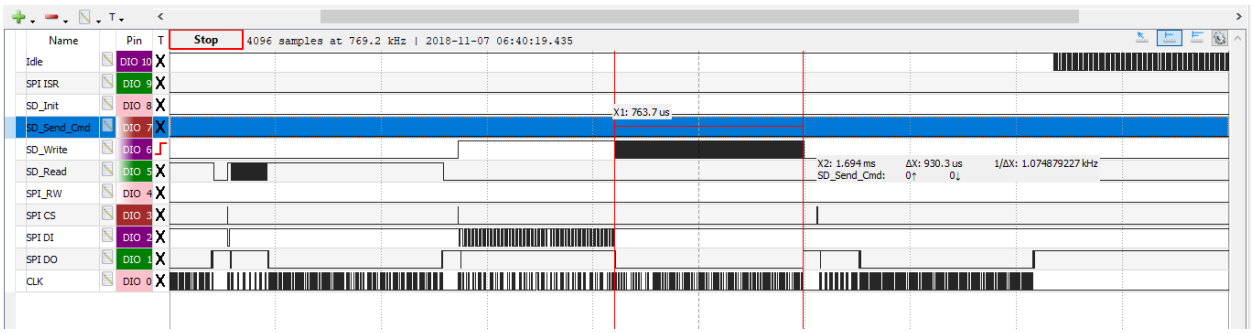
Total Write time:1.703 ms



Computation time:774.8 µs



Polling time and SPI Communication time: 930.3 µs

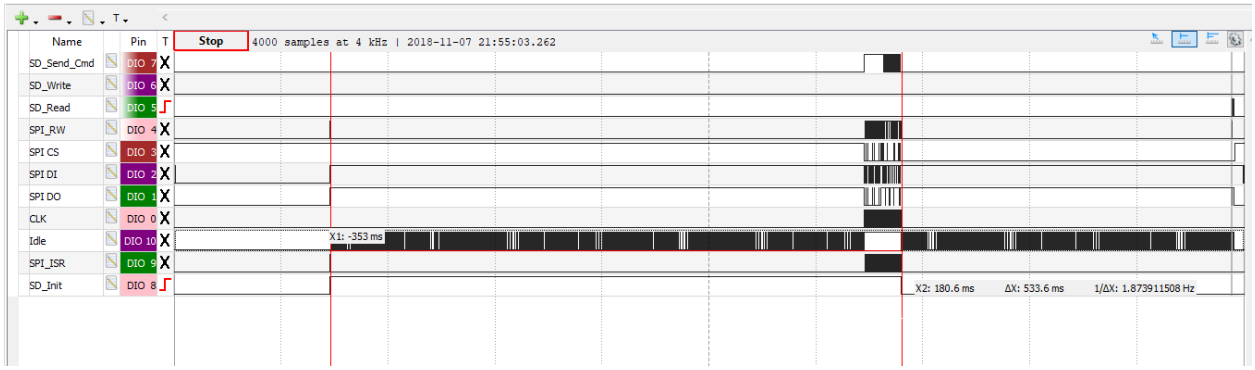


SD_Init

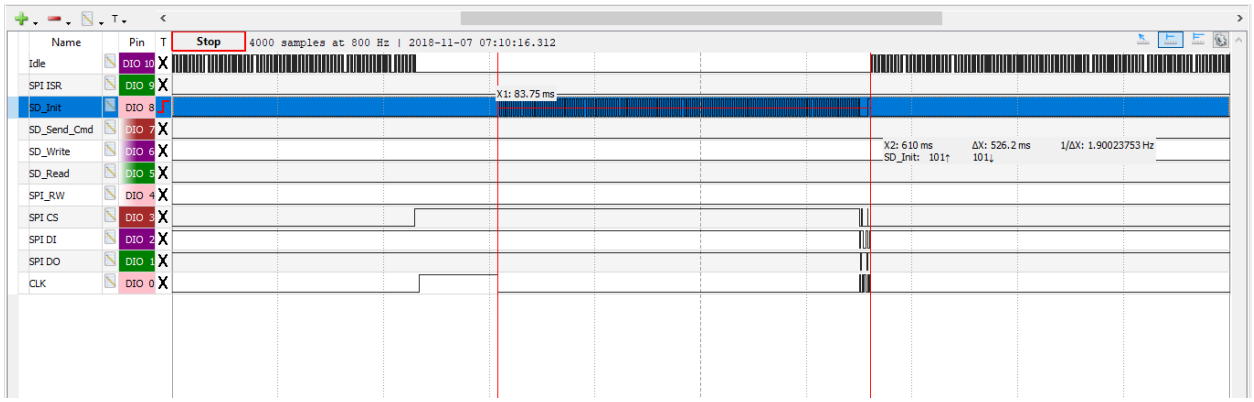
10. Analysis

a. How long does SD_Init take to execute? How long is each segment now? Include a screenshot.

Ans: SD Init segment takes 533.6ms to execute.



SD_init runs for 524.3ms



Init segment is thus taking 9.3ms

b. How much idle time is available now, based on idle_counter?

Ans: idle_counter =2377910

As the 1sec idle time is 4759072 thus, 2377910/4759072 =0.49965 which is nearly eaqual to 500 ms

c. Do these values differ from the previous implementation? Why or why not?

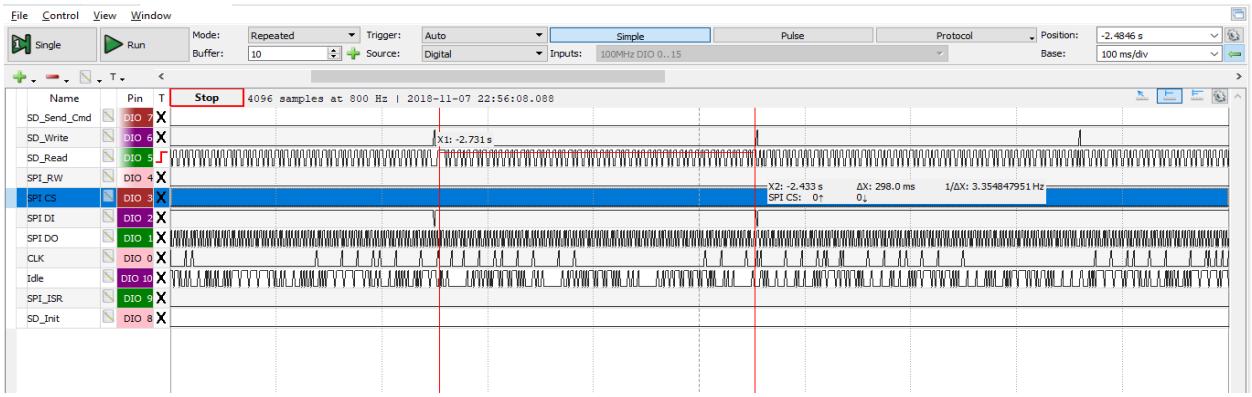
Ans: The total time difference doesn't change significantly, Since we have replaced busy wait loop with osDelay the control goes to the scheduler. Since, there's no task in the ready queue, the scheduler runs IDLE task.

SD_Read

12. Analysis

a. **How long does the SD_Read function take to complete? How long is each segment now? Include a screenshot.**

Ans: The read time for 100 sectors is 298.0 ms so the 1 sector read time after osDelay is 2.98 ms



b. **How much idle time is available now, based on idle_counter?**
Ans: The read_counter value is 14309. Dividing it by 1sec idle time we get 14309/4759072=3.066ms

Name	Value	Type
init_time_diff	0	unsigned int
WRITE_diff	0	unsigned int
READ_diff	14309	unsigned int
<Enter expression>		

c. **Do the times differ from the previous implementation? Why or why not?**
Ans: The total time difference doesn't change significantly, Since we have replaced busy wait loop with osDelay the control goes to the scheduler. Since, there's no task in the ready queue, the scheduler runs IDLE task.

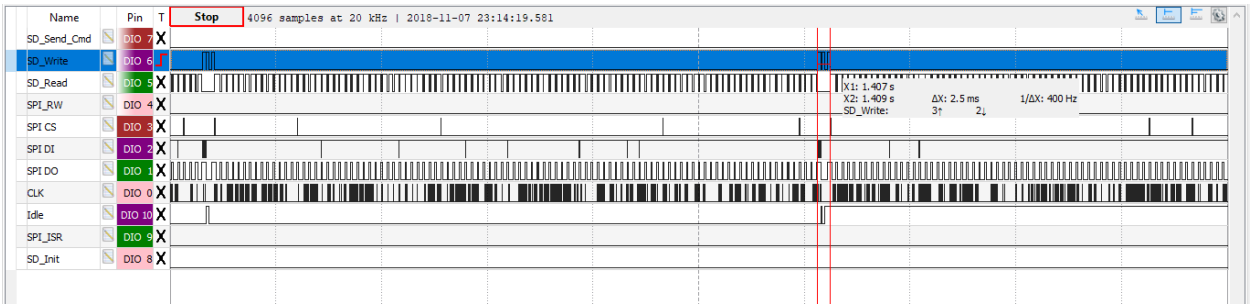
SD_Write

13. Change this code to use an osDelay() call to reduce the polling frequency to once per timer tick. Include a screenshot from the logic analyzer.

14. Analysis

a. **How long does the SD_Write function take to complete? How long is each segment now? Include a screenshot.**

Ans: Write takes 2.5ms to complete .



b. **How much idle time is available now, based on idle_counter?**

time_diff	0	unsigned int
total	<cannot evaluate>	uchar
init_time_diff	0	unsigned int
WRITE_diff	11722	unsigned int
<Enter expression>		

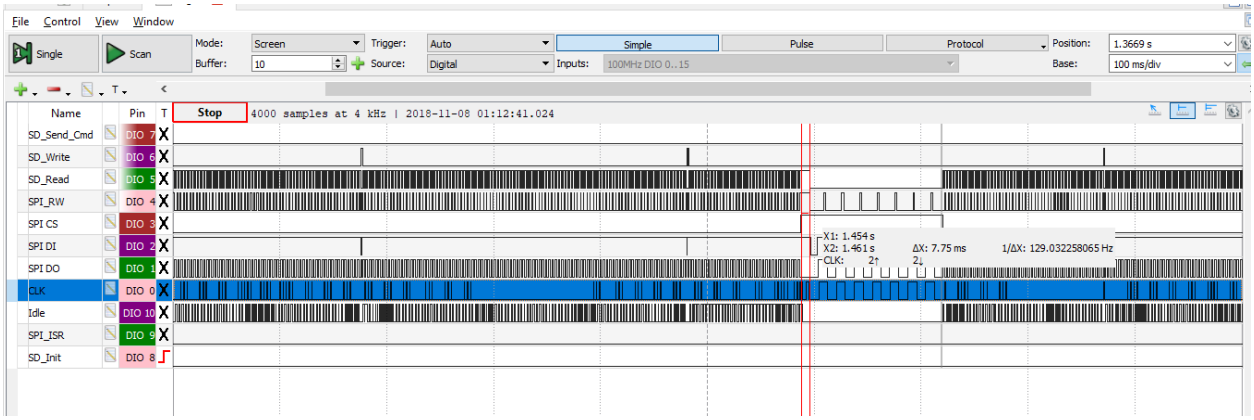
The counter value is 11722 thus, 11722/4759072 =2.4631ms

c. **Do the times differ from the previous implementation? Why or why not?**
The total time difference doesn't change significantly, Since we have replaced busy wait loop with osDelay the control goes to the scheduler. Since, there's no task in the ready queue, the scheduler runs IDLE task.

SPI_RW

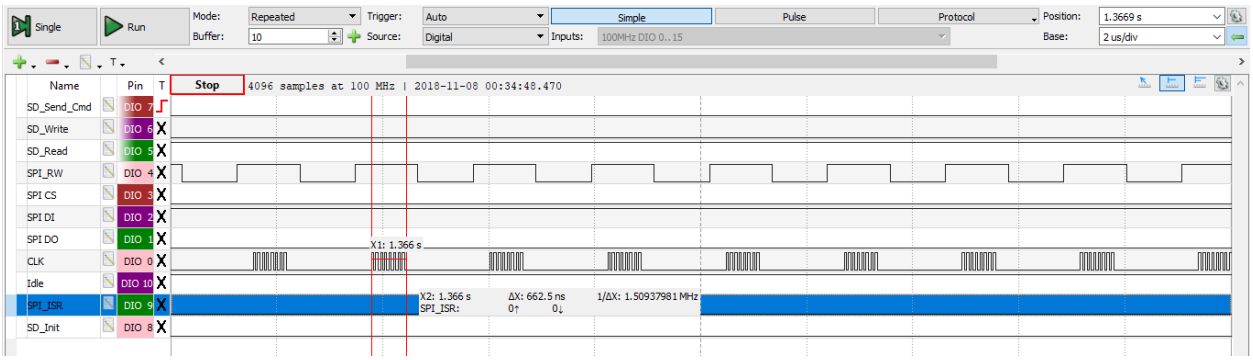
18. SPI Processing Analysis. Consider the operations occurring when executing SPI_RW() once (assuming SPI_Freq_Low() has been called). Mark these events and times on a screenshot.

a. **How long does SPI_RW() take to execute (DBG_SPI_RW rising edge to falling edge)?**
Total SPI_RW time is 7.75ms



b. How much of this time is used for SPI communications (observe SPI Clock)?

SPI communication time is 662.5ns which is about 10% of SPI Clock.



c. How much idle time is available during an execution of SPI_RW()?
Ans: Idle time will be the difference of total SPI_RW() time and SPI communication time=7.75ms-662.5ns=7.0875ms

d. What is the delay from SPI communication completion to the start of SPI1_IRQHandler()? How much of this is the Cortex-M0+ CPU's delay in responding to an interrupt (see ESF chapter 4 and KL25 Reference Manual)?

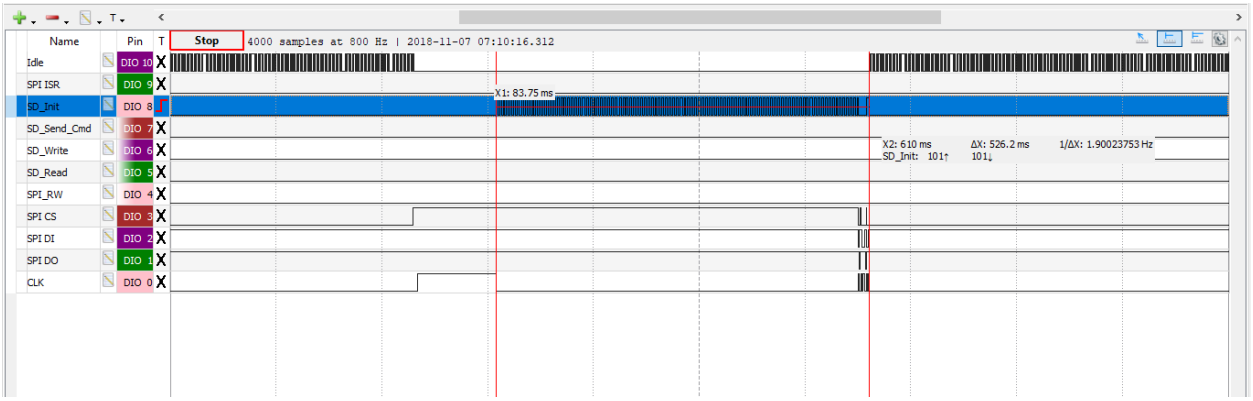
Ans: The delay between the last SPI_CLOCK tick used for SPI Communication and the following ISR Handler is 2.44 micro-seconds. The worst case Interrupt Latency for ARM-Coretx M0+ processors is around 15 clock cycles. With the system core clock running at 48 MHz , it takes around 0.312 micro-seconds to respond to an interrupt.

e. How long does SPI1_IRQHandler() take to execute?
Ans: About 14 micros seconds

f. What is the delay from the SPI1_IRQHandler() completing to SPI_RW() completing? How much of this is the Cortex-M0+ CPU's delay in responding to an interrupt (see ESF chapter 4 and KL25 Reference Manual)? What else causes this delay?

Ans: The worst case Interrupt Latency for ARM-Coretx M0+ processors is around 15 clock cycles**.** With the system core clock running at 48 MHz **,** it takes around 0.312 micro-seconds to respond to an interrupt.

19. **ECE-560 Only: Alternatives**
    a. **Why are we using a message queue instead of simply inserting osDelay(1) calls in the SPI_RW() blocking loops?**
    Ans: To hold the ISR acquired byte for the SPI_RW to be used. If we don't use the queue, we might loose the data. And, if osDelay expires, there's no guarantee that the task will run immediately. There might be a higher priority task running in the background
    b. **Why didn't we change the first blocking loop in SPI_RW?**
    Ans: We could have enabled Interrupts on the SPI Transmit Buffer Empty flag to eliminate the busy wait loop. But, the SPI_IRQHandler() ISR gets more complex that it has to recognize which of the two flags generated the interrupt which might increase the ISR execution time.
    c. **Estimate how long it would take for SPI_RW() to execute if we raised the SPI bit rate to 1 MHz.**
    Ans: It takes nearly same time to execute**.**
20. **Thread Analysis**
    a. **How long does SD_Init take to execute? How long is each segment now? Include a screenshot.**
    Ans: It takes same 536.2ms



    b. **How much idle time is available now, based on idle_counter?**

Ans: It is the same with idle_counter =2377910 .As the 1sec idle time is 4759072 thus, 2377910/4759072 =0.49965 which is nearly eaqual to 500 ms

    c. **Do these values differ from the previous implementation? Why or why not?**
    Although we increase SPI bit rate the idle time difference in the thread remains the same**.**

ECE 560-Only: Eliminating the use of the SPI Timer

21. **Find all of the loops with time-outs in the functions (SD_Read, SD_Write, SD_Init) and explain what each loop does and what the timeout period is.**

Ans:

SD_Read:

1.

```
 do {
data = SPI_RW(0xff);
if ((byte_num >= ofs) && (byte_num < ofs+cnt)) {
            *(BYTE*)dat = data;
            ((BYTE *) dat)++;
             DEBUG_TOGGLE(DBG_2);
} // else discard bytes before and after data
       } while(++byte_num < SD_BLK_SIZE + 2 ); // 512 byte block + 2 byte CRC
```

SD_Init:

1.
```
SPI_Timer_On(500);
      while ((__SD_Send_Cmd(CMD0, 0) != 1)&&(SPI_Timer_Status()==TRUE)) {}
```

2.
```
SPI_Timer_On(250);
            while((SPI_Timer_Status()==TRUE)&&(__SD_Send_Cmd(cmd, 0))) {}
```

3.
```
 SPI_Timer_On(1000);
   while ((SPI_Timer_Status()==TRUE)&&(__SD_Send_Cmd(ACMD41, 1UL << 30))) {}
```

SDWrite:

1.
```
do {
                                                line = SPI_RW(0xFF);
                                    } while((line==0)&&(SPI_Timer_Status()==TRUE));
```