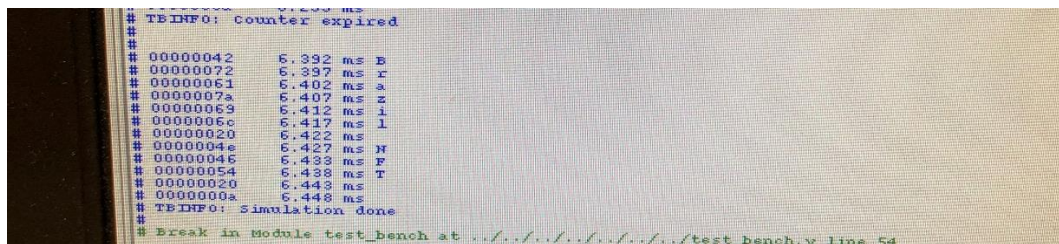


## COE 4DS4 – LAB 5 REPORT

1. The picture below shows our transcript of after counter is expired “Brail NFT”. First letter of Brazil NFT, which is B in hex 42, is displayed at 6.392ms and the last letter in printf is at 6.448ms (displayed in transcript as exact time). From observation we see it took each printf approximately 0.005 ms to finish and go to the next. From the waveform (jtag uart av irq) the actual start of the interrupt is 6.36663ms and completion time is 6.45361 ms. In total interrupt service routine time was  $6.45361 - 6.392 = 0.08698$  ms.



```
# TBDF0: counter expired
##
##
# 00000042 6.392 ms B
# 00000072 6.397 ms r
# 00000061 6.402 ms a
# 0000007a 6.407 ms z
# 00000069 6.412 ms i
# 0000006c 6.417 ms l
# 00000020 6.422 ms
# 0000004e 6.427 ms N
# 00000046 6.433 ms F
# 00000054 6.438 ms T
# 00000020 6.443 ms
# 0000000a 6.448 ms
# TBDF0: Simulation done
##
# Break in Module test_bench at ...../test_bench.v line 54
```

2. For this exercise, we used the lab 3 exercise 3 maximum subsequence to verify and test our design. The offset in hardware were set as in the criteria in counter component where we instantiate the dual port ram along with the counter sequence. When the hardware sequence search is initialed, a write to the MSB of offset 3 occurs. On the hardware side, when we get a chipselect and write signal, depending on the address we either write the values for the address and data to be stored in the DP RAM for offset 1, or we check the MSB and LSBs of offset register 3 to determine whether we are starting a search or clearing an interrupt. We had to debug our largest element manually from c from print statements to see if the criteria was satisfied as array is written to dp ram and read back. Once When checking the software generated sequence with hardware, we can to account for the fact dual port has 1 clock cycle latency so we had a dumbey state to make up for it.
3. We implemented two Boolean functions with help of a reference, one for error checking and the other for little endian criteria. From the experiment see see that since the sd\_card\_read function returns a short int, which is 2 bytes, we are shifting by 8 so the top byte should not contain anything. Also we acknowledge the fact that the sd\_card\_read function returns a -1 which is basically FFFF in hex, Then since with little endian we want to reverse the order, we shift the least significant hex value 24 values to the left, then the 2<sup>nd</sup> least to 16 values etc. When writing to the display, we write the first quarter of the lines to be black since we need to center the 320x240 image in the 640x480 display