ECEN 5013 Spring 2017 Project 3

Release

Ben Andre

Apr 17, 2017

Contents

Repository: https://bitbucket.org/bjandre/embedded-software-essentials

Tag of changeset: **project-3-rel**, **project-3.0.1-rel**

NOTE : The screen shots are readable, but you must zoom in several times to see the details.

1 Goal

- Updated logger output with time stamps and heartbeats.
- profile memory performance memmove and memset between the standard library, CPU based version, and DMA version (where applicable).
- Creation of a SPI library and high level application libray for driving the nRF24L01+

2 Logger output

Updated raw logger output is shown in figure Fig. 1. Processed logger output showing time stamps and heartbeat output is shown in Fig. 2 and Fig. 3.

main — -bash — 114×31

...jects/ecen5013/embedded-software-essentials — -bash ▸ Python | ...jects/ecen5013/embedded-software-essentials/src/main — -bash

```
HEARTBEAT(7) : 1492407798 : size = 0 : 2017-04-17 05:43:18 :
HEARTBEAT(7) : 1492407799 : size = 0 : 2017-04-17 05:43:19 :
HEARTBEAT(7) : 1492407800 : size = 0 : 2017-04-17 05:43:20 :
HEARTBEAT(7) : 1492407801 : size = 0 : 2017-04-17 05:43:21 :
non-hexadecimal number found in fromhex() arg at position 6
[(pyrite) main $ cat frdm-log.hex                                                                                  ]
774501040100000000000001e055f4580002e055f4580003e055f4580004e055f4580005e055f458000be055f458000ee055f458040d000000
0fe055f4580400000000010e055f458040d00000011e055f4580019e055f4580f6d656d6d6f76655f7374646c6962001ae055f4580451010000
1be055f4580f6d656d6d6f76655f7374646c69620019e055f4580c6d656d6d6f76655f637075001ae055f45804cc5b8f021be055f4580c6d65
6d6d6f76655f6370750019e055f4580c6d656d6d6f76655f646d61001ae055f45804bf0100001be055f4580c6d656d6d6f76655f646d610019
e055f4580e6d656d7365745f7374646c6962001ae055f45804b85a8f021be055f4580e6d656d7365745f7374646c69620019e055f4580b6d65
6d7365745f637075001ae055f45804fa0100001be055f4580b6d656d7365745f6370750019e055f4580b6d656d7365745f646d61001ae055f4
5804b40100001be055f4580b6d656d7365745f646d610019e055f4580f6d656d6d6f76655f7374646c6962001ae055f45804940400001be055
f4580f6d656d6d6f76655f7374646c69620019e055f4580c6d656d6d6f76655f637075001ae055f45804000e00001be055f4580c6d656d6d6f
76655f6370750019e055f4580c6d656d6d6f76655f646d61001ae055f458040c5c8f021be055f4580c6d656d6d6f76655f646d610019e055f4
580e6d656d7365745f7374646c6962001ae055f45804bd0300001be055f4580e6d656d7365745f7374646c69620019e055f4580b6d656d7365
745f637075001ae055f45804320a00001be055f4580b6d656d7365745f6370750019e055f4580b6d656d7365745f646d61001ae055f45804fe
5b8f021be055f4580b6d656d7365745f646d610019e055f4580f6d656d6d6f76655f7374646c6962001ae055f4580455240001be055f4580f
6d656d6d6f76655f7374646c69620019e055f4580c6d656d6d6f76655f637075001ae055f45804166700001be055f4580c6d656d6d6f76655f
6370750019e055f4580c6d656d6d6f76655f646d61001ae055f45804a40100001be055f4580c6d656d6d6f76655f646d610019e055f4580e6d
656d7365745f7374646c6962001ae055f45804781c00001be055f4580e6d656d7365745f7374646c69620019e055f4580b6d656d7365745f63
7075001ae055f458043d5b00001be055f4580b6d656d7365745f6370750019e055f4580b6d656d7365745f646d61001ae055f458042b040000
1be055f4580b6d656d7365745f646d610019e055f4580f6d656d6d6f76655f7374646c6962001ae055f4580452b100001be055f4580f6d656d
6d6f76655f7374646c69620019e055f4580c6d656d6d6f76655f637075001ae155f4580443fe01001be155f4580c6d656d6d6f76655f637075
0019e155f4580c6d656d6d6f76655f646d61001ae155f45804b90400001be155f4580c6d656d6d6f76655f646d610019e155f4580e6d656d73
65745f7374646c6962001ae155f458041c8a00001be155f4580e6d656d7365745f7374646c69620019e155f4580b6d656d7365745f63707500
1ae255f4580473c301001be255f4580b6d656d7365745f6370750019e255f4580b6d656d7365745f646d61001ae255f45804400700001be255
f4580b6d656d7365745f646d610013e255f4580007e255f4580007e355f4580007e455f4580007e555f4580007e655f4580007e755f4580007
e855f4580007e955f4580007ea55f4580007eb55f4580007ec55f4580007ed55f4580007ee55f4580007ef55f4580007f055f4580007f155f4
580007f255f4580007f355f4580007f455f4580007f555f4580007f655f4580007f755f4580007f855f4580007f955f4580007fa55f4
(pyrite) main $
```

Fig. 1: Raw binary output capture from the UART logger comming off of the FRDM board.

main — -bash — 114×31

...jects/ecen5013/embedded-software-essentials — -bash ▸ Python | ...jects/ecen5013/embedded-software-essentials/src/main — -bash

```
HEARTBEAT(7) : 1491352433 : size = 0 : 2017-04-05 00:33:53 :
unpack requires a bytes object of length 4
[(pyrite) main $ ./ese-process-logs.py < frdm-log.hex                                                              ]
Byte 1: 77
Byte 2: 45
sizeof(ID): 1
sizeof(timestamp): 4
sizeof(size): 1
LOGGER_INITIALIZED(0) : 0 : size = 0 :  :
RTC_INITIALIZED(1) : 1492407776 : size = 0 :  :
GPIO_INITIALIZED(2) : 1492407776 : size = 0 :  :
DMA_INITIALIZED(3) : 1492407776 : size = 0 :  :
SPI_INITIALIZED(4) : 1492407776 : size = 0 :  :
SYSTEM_INITIALIZED(5) : 1492407776 : size = 0 :  :
POST_START(11) : 1492407776 : size = 0 :  :
POST_NUM_TESTS_RUN(14) : 1492407776 : size = 4 : 0d000000 :
POST_NUM_TESTS_SKIPPED(15) : 1492407776 : size = 4 : 00000000 :
POST_NUM_TESTS_PASSED(16) : 1492407776 : size = 4 : 0d000000 :
POST_COMPLETE(17) : 1492407776 : size = 0 :  :
PROFILING_START(25) : 1492407776 : size = 15 : 6d656d6d6f76655f7374646c696200 : memmove_stdlib
PROFILING_VALUE(26) : 1492407776 : size = 4 : 51010000 : 337
PROFILING_END(27) : 1492407776 : size = 15 : 6d656d6d6f76655f7374646c696200 : memmove_stdlib
PROFILING_START(25) : 1492407776 : size = 12 : 6d656d6d6f76655f63707500 : memmove_cpu
PROFILING_VALUE(26) : 1492407776 : size = 4 : cc5b8f02 : 42949580
PROFILING_END(27) : 1492407776 : size = 12 : 6d656d6d6f76655f63707500 : memmove_cpu
PROFILING_START(25) : 1492407776 : size = 12 : 6d656d6d6f76655f646d6100 : memmove_dma
PROFILING_VALUE(26) : 1492407776 : size = 4 : bf010000 : 447
PROFILING_END(27) : 1492407776 : size = 12 : 6d656d6d6f76655f646d6100 : memmove_dma
PROFILING_START(25) : 1492407776 : size = 14 : 6d656d7365745f7374646c696200 : memset_stdlib
PROFILING_VALUE(26) : 1492407776 : size = 4 : b85a8f02 : 42949304
PROFILING_END(27) : 1492407776 : size = 14 : 6d656d7365745f7374646c696200 : memset_stdlib
```

Fig. 2: Processed logger output showing the linux epoch timestamp on each log item.

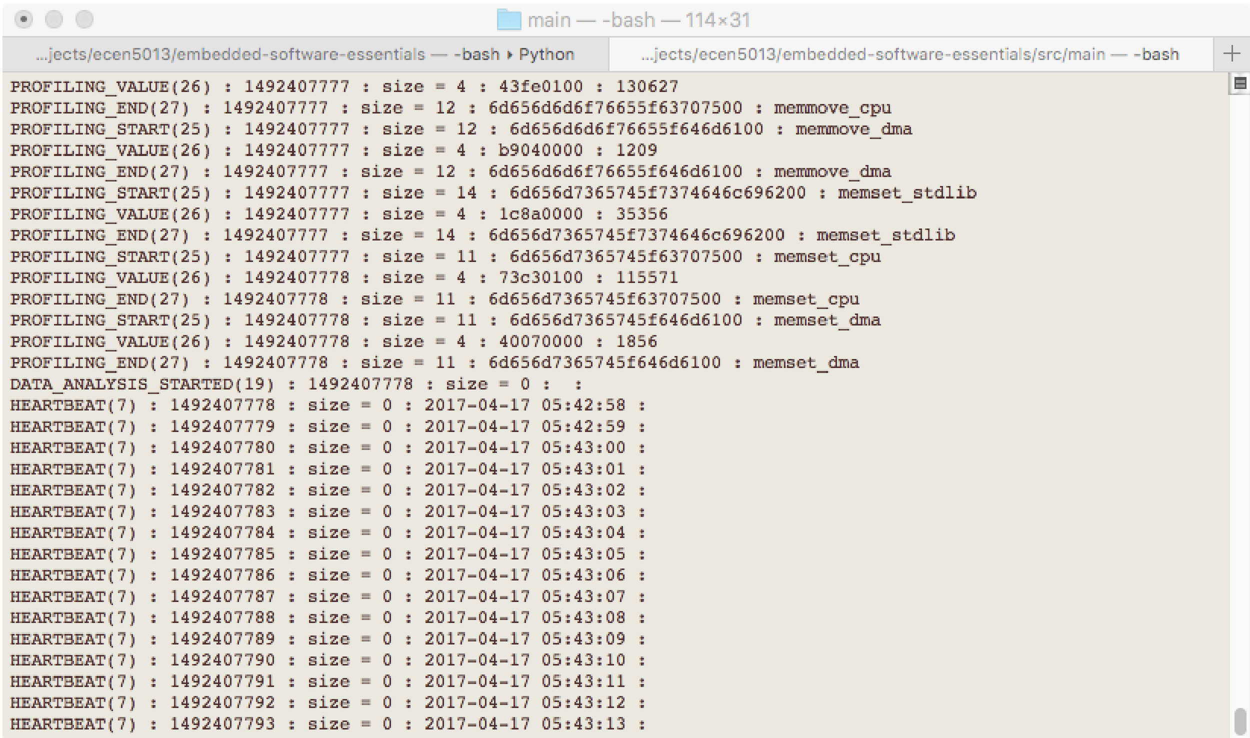

Fig. 3: Processed logger output showing the heartbeat log output with epoch timestamp.

3 Memory Profiling

Fig. 4 and Fig. 5 show an exerpt of the processed log output showing memory profiling results on the FRDM-KL25Z. Profiling was done by averaging the timing results for 100 samples for each memory size, memory function and platform.

Timing results are shown in Fig. 3 and Fig. 3. Timing on the FRDM board shows that using DMA has more overhead that using the CPU, so it is only efficient above about 100 byte transfer sizes for both memmove and memset. Adove 100 bytes, it appears that the DMA is about 3-10 times faster than using the CPU for memset, and about 6-13 times faster for memmove. In both cases, the performance is better as the size of the transfers increases from one to four bytes. There are still some inconsistencies in the data, e.g. CPU faster that stdlib routine for memmove at 12 bytes, indicating that the profiler could be further debugged or improved. Using the nieve CPU implementation for memmove is significantly slower then the optimized assembly in the standard libary and DMA. For memset, where one byte is compied many times, using the nieve CPU implementation is approximately as fast as the standard library.

The timing on the Beagle Bone Black was done using the clock() function from time.h in the standard library. This has a resolution of 1 micro second, which was not fine enough to obtain accurate results. It should be replaced with the non-standard POSIX function clock_gettime() function, which has nanosecond resolution.

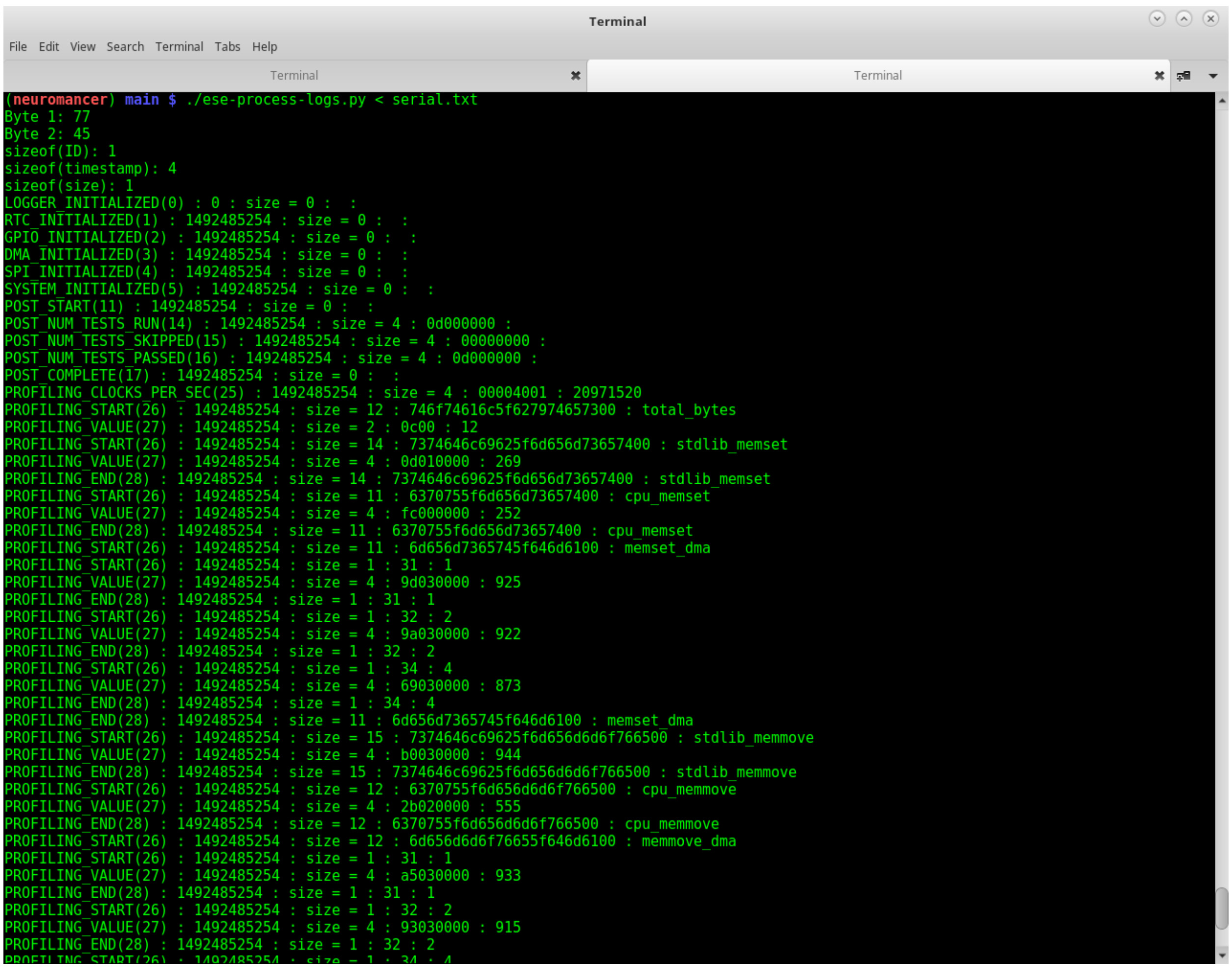

Fig. 4: Processed logger output showing the 12 byte memory profiling on the FRDM-KL25Z.

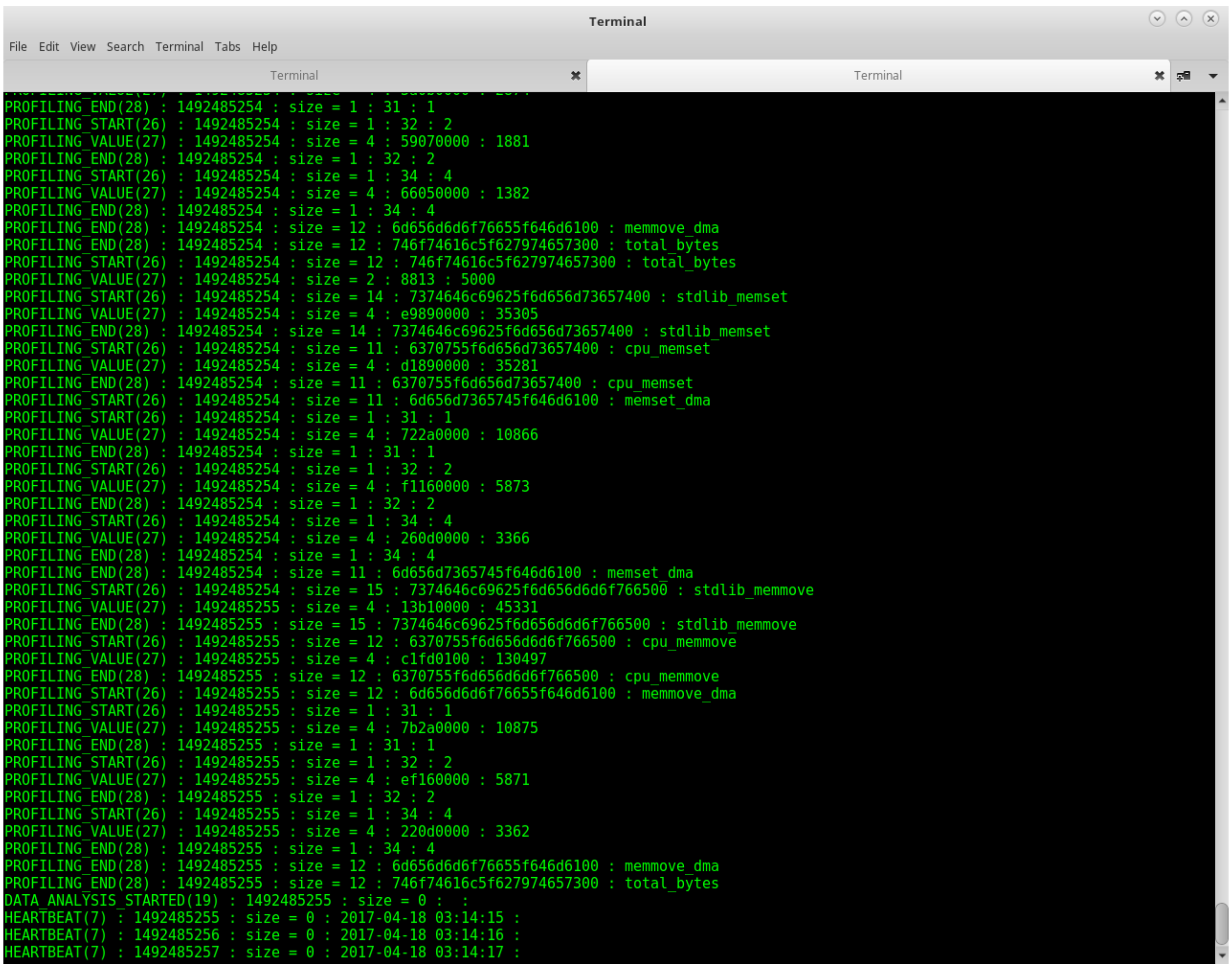

Fig. 5: Processed logger output showing the 5000 byte memory profiling on the FRDM-KL25z.

FRDM KL25Z	Total bytes 12	100	1000	5000
memset				
stdlib	269	918	7257	35305
cpu	252	900	7228	35281
dma 1 byte	925	1107	2865	10866
dma 2 byte	922	992	1888	5873
dma 4 byte	873	952	1386	3366
memmove				
stdlib	944	1128	9263	45331
cpu	555	2871	26321	130497
dma 1 byte	933	1115	2874	10875
dma 2 byte	915	987	1881	5871
dma 4 byte	867	952	1382	3362

units: clock cycles

FRDM KL25Z	Total bytes 12	100	1000	5000
memset				
stdlib	0	0	0	0
cpu	0	0	0	0
				s
memmove				
stdlib	0	0	0	0
cpu	0	100	100	300

units: clock cycles

4 nRF24L01+

The required demo commands for the nRF24L01 library were implemented as a set of power-on-self-tests, POST. These are run every time the board resets to ensure the MCU and radia are communicating properly. Since the nRF24L01 does not appear to have a software reset, it must be physically power cycled everytime the board is reset to reset the internal registers to a known condition. This limitation can be fixed by using an additional GPIO pin on the MCU to control power to the radio, but that is beyond the scope of the class. Fig. 6 shows an example screen shot of nRF24L01 SPI commands on the logic analyzer

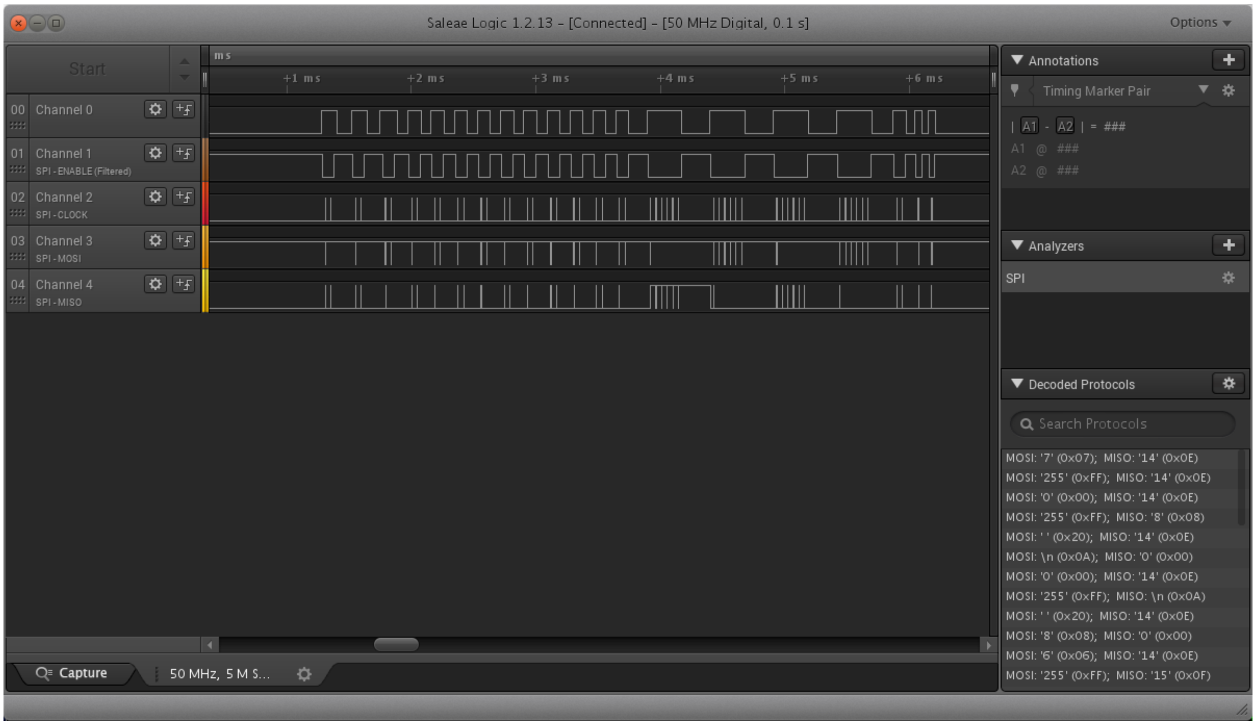

Fig. 6: Overview of SPI command on logic analyzer during POST of nRF24L01 library. The top graph is the radio enable line (active high). The second line is the chip select line (active low). The middle line is the clock, and the bottom two lines are the MOSI and MISO respectively. Bytes send and received can be seen on the right under 'decoded protocols'.

4.1 Status register

Fig. 7 and Fig. 8 shows a zoomed view of the logic analyzer when performing an read of the status register. This is done as a two byte sequence, explicitly reqesting a read of the status register, then sending a no-op to receive the result. This could also be done by sending a single no-op command and returning the first MISO byte.

4.2 Config register

Fig. 9 through Fig. 14 show the read-write-read sequence get the intial state of the config register, modify the register, and verify the change.

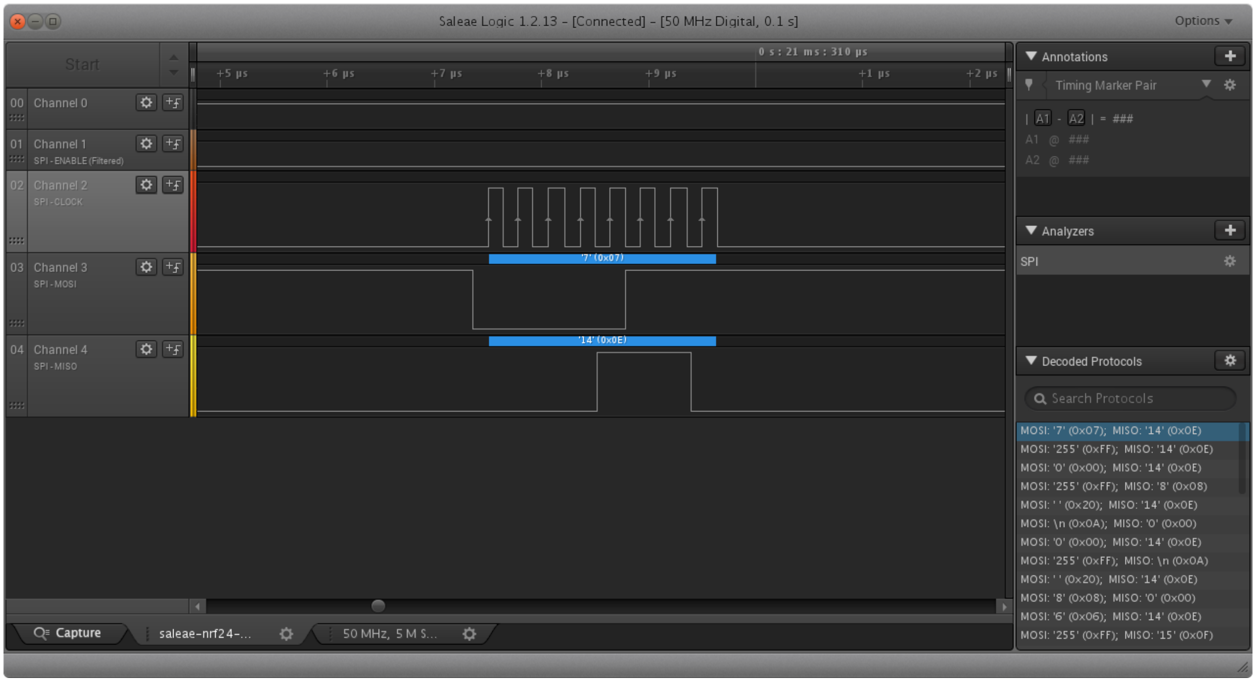

Fig. 7: Sending the command to explicitly read the status register, *0x00 | 0x07* on MOSI, and the default status return on MISO, power up value of *0x0E*.

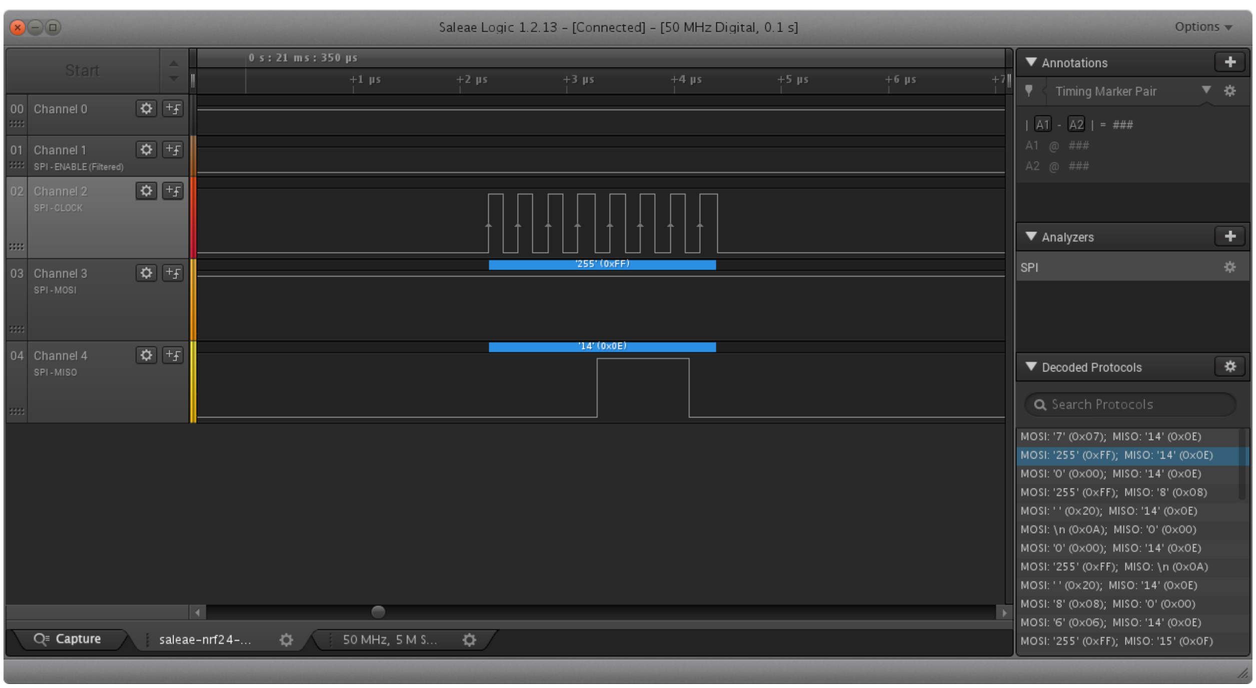

Fig. 8: Sending the second of the two byte sequence, a no-op command, *0xFF*, to receive the status register again. The default on power-up is *0x0E*.

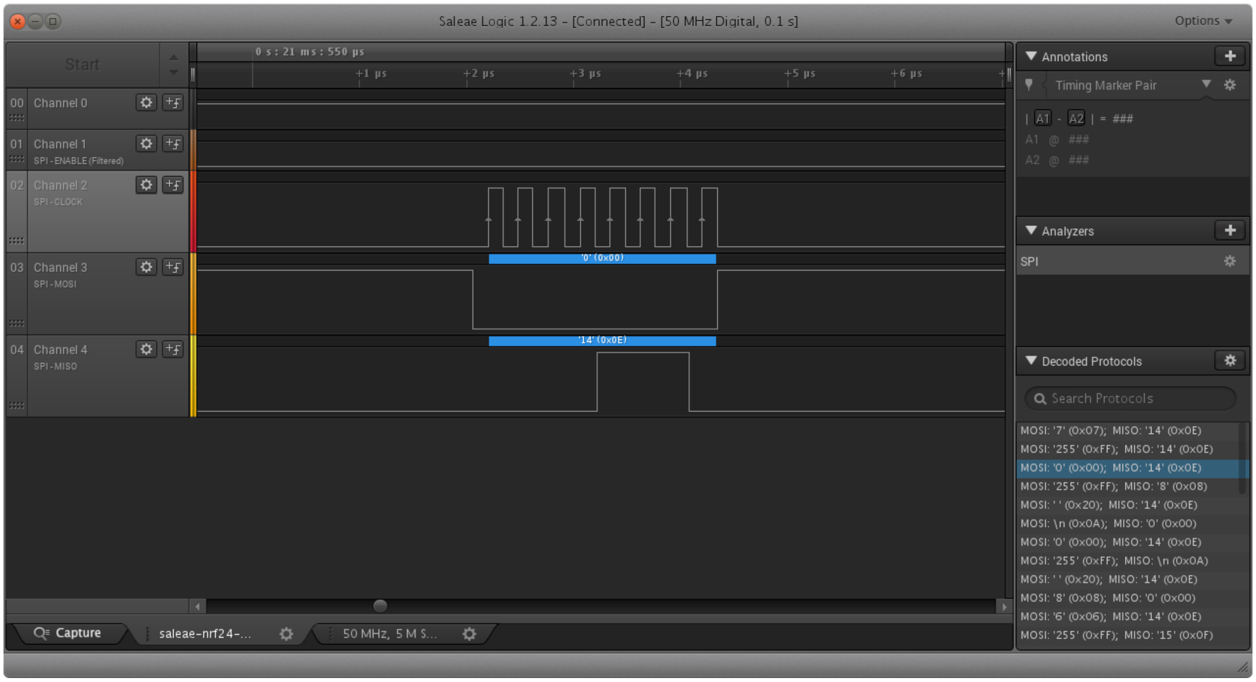

Fig. 9: Sending the command to explicitly read the config register, *0x00 | 0x00*, on MOSI, and the default status register on MISO, *0x0E*.

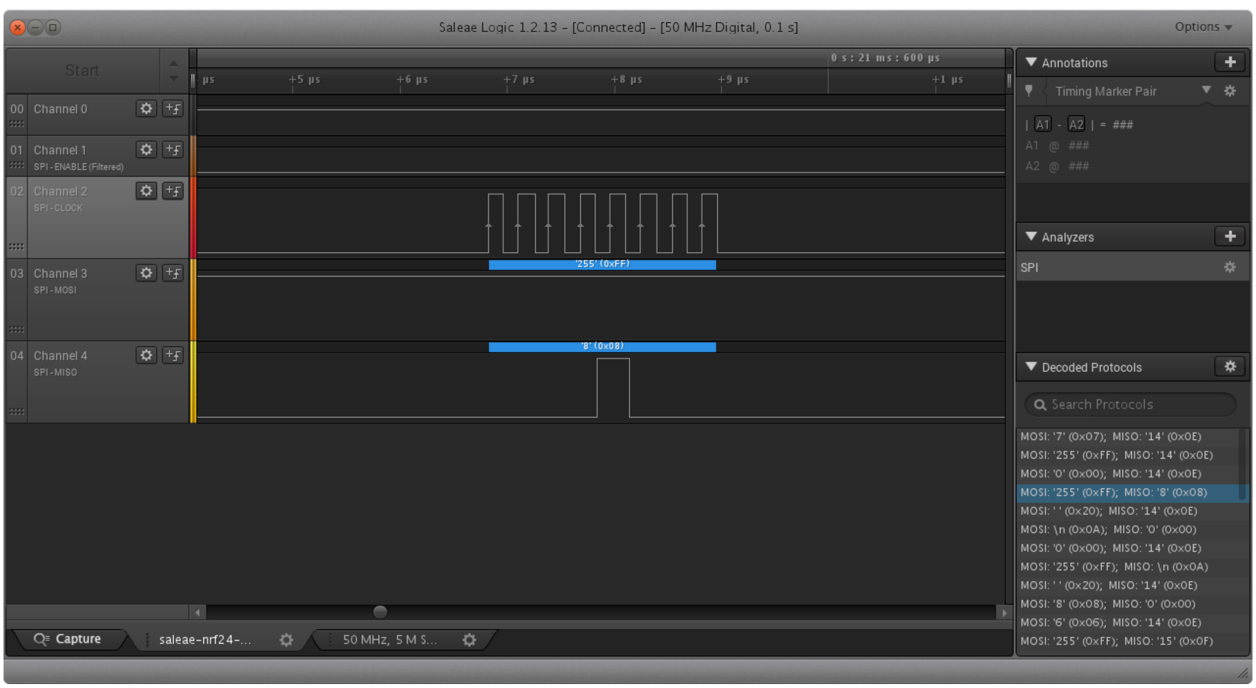

Fig. 10: Sending the second of the two byte sequence to read the config register, a no-op command, *0xFF*, to receive the power up state of the config register, *0x08*.

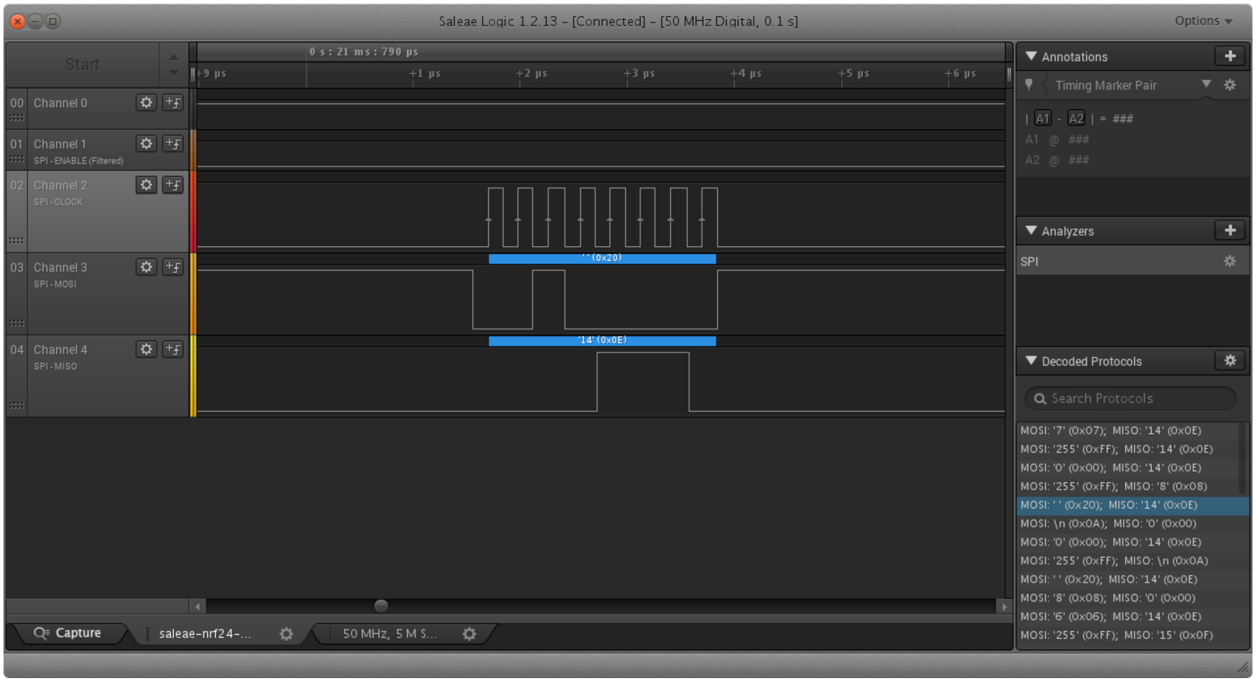

Fig. 11: Sending the first of the two byte command sequence to write the config register, *0x20* | *0x00*, on MOSI, and the default status register on MISO.

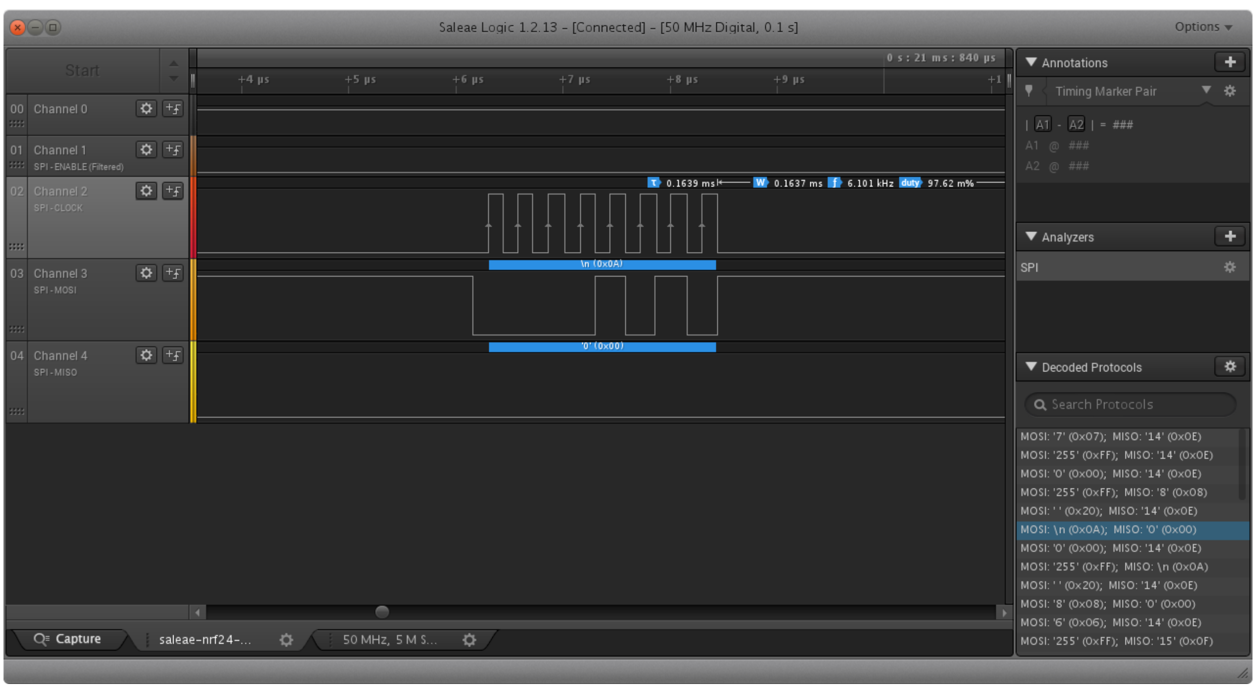

Fig. 12: Sending the second of the two byte sequence to write the config register, the value to be written, *0x0A*, and the no-op return on MISO, *0x00*.

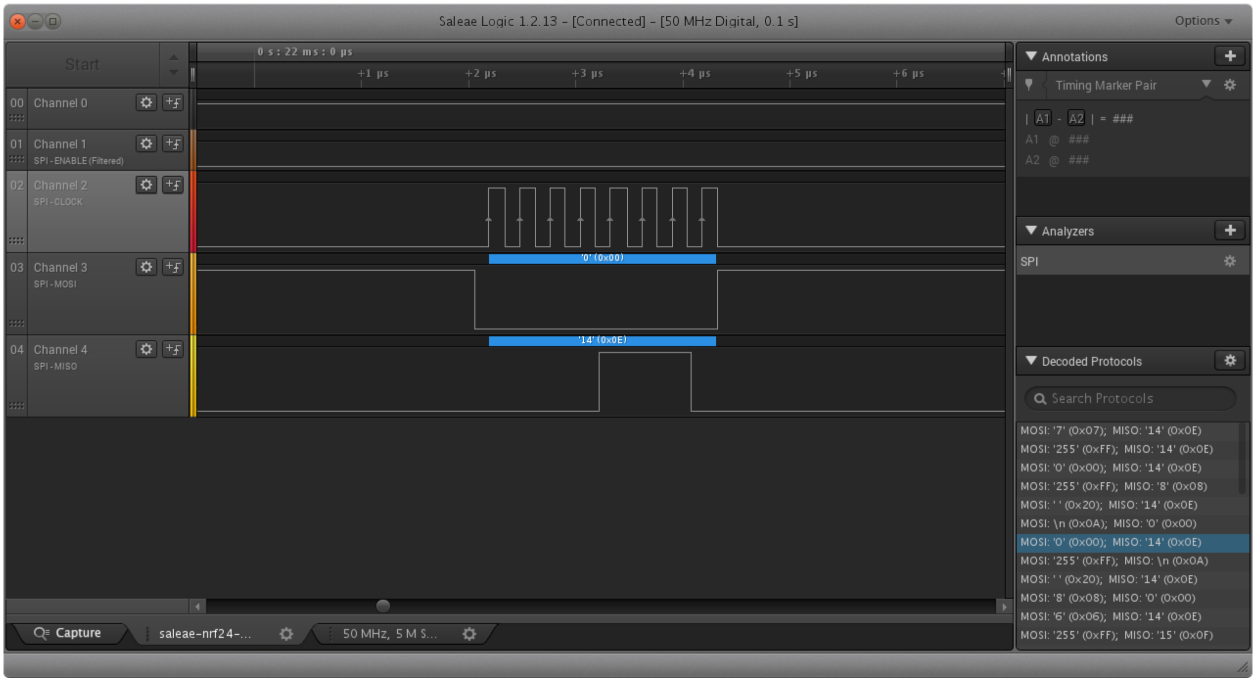

Fig. 13: Resending the read register command for the config register, *0x00* | *0x00*, and the default status return, *0x0E*.

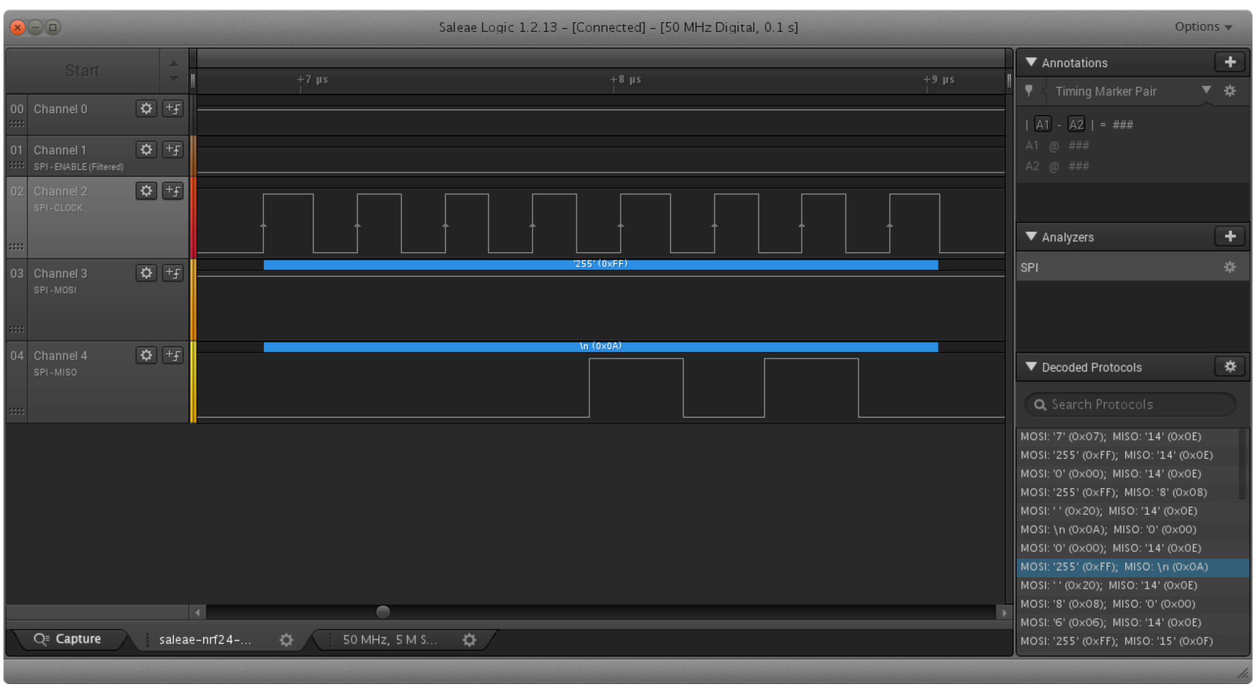

Fig. 14: Sending the second of the two byte sequence to read the config register, a no-op command, *0xFF*, to receive the modified config register, *0x0A*.

4.3 TX ADDR register

Reading and writing to the TX_ADDR register is a multi-byte sequence consisting of one command byte and five data bytes. Fig. 15 shows an over view of the read process. First a single byte read command is sent, Fig. 16, then the five byte address is returned, as shown in Fig. 17 for the first byte. The write process is shown in Fig. 18. A single byte write command is sent, Fig. 19 , followed by the five address bytes, Fig. 20. In this example, the same five bytes are repeated for the send address. To verify the write, we read the register again. The Command is the same shown in Fig. 16. This time the return data is the values we just wrote, Fig. 21.

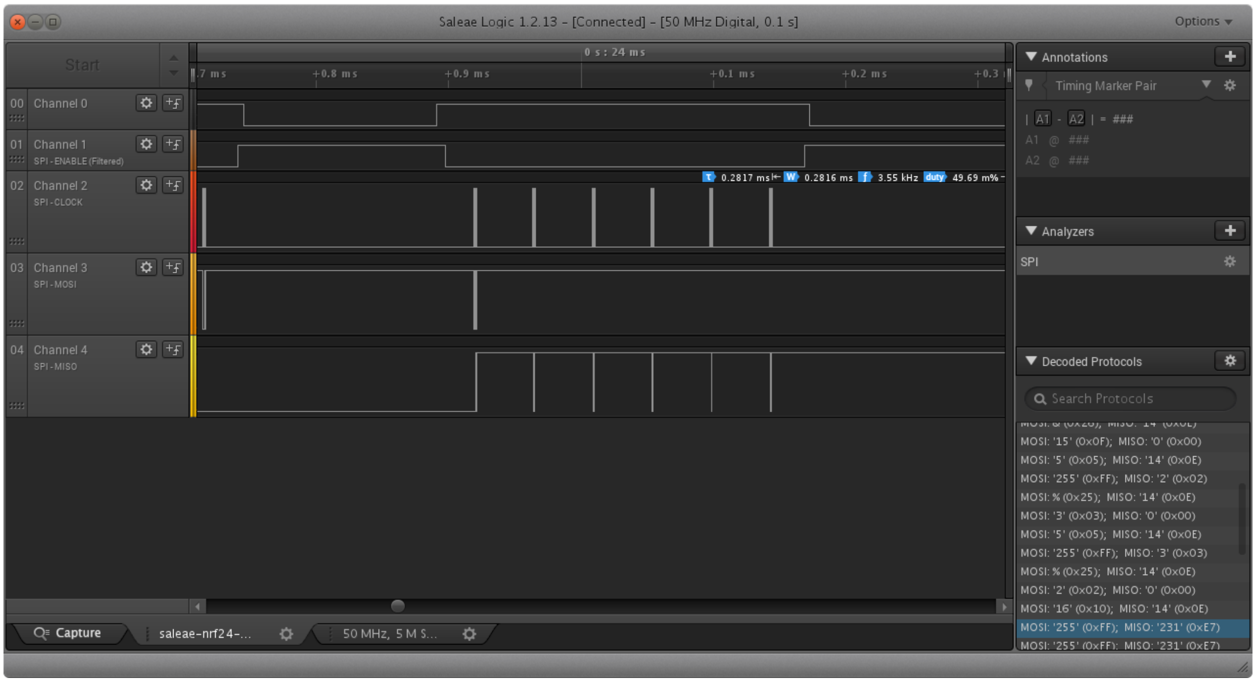

Fig. 15: Over view of the six byte read process for the TX_ADDR register.

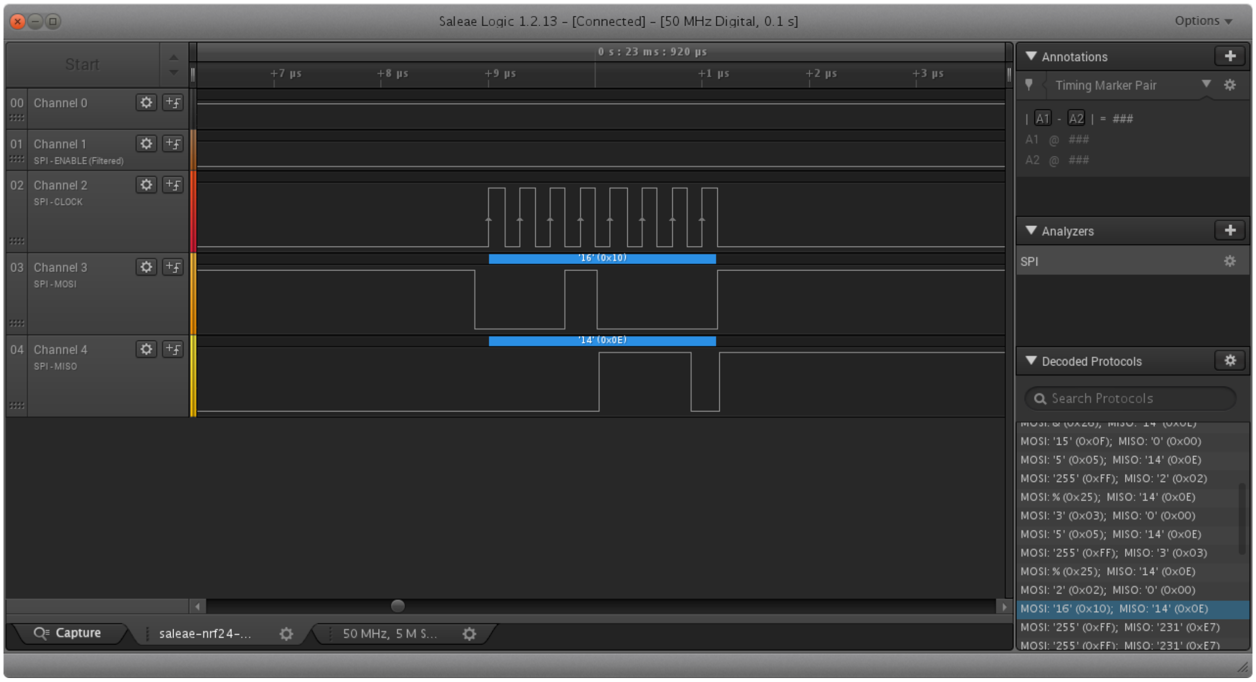

Fig. 16: First a read command is sent for TX_ADDR, *0x00* | *0x10*, on MOSI. The slave sends its status, *0x0E*, for the first byte on MISO.

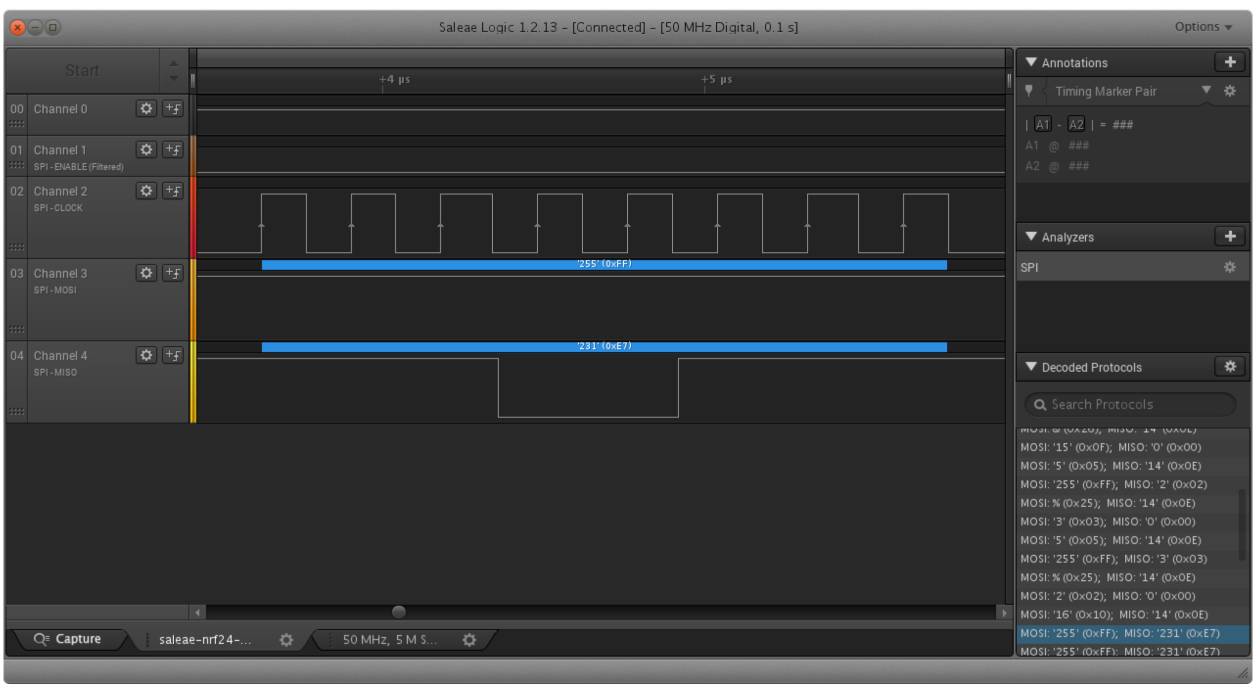

Fig. 17: The read register command is followed by five no-op commands, *0xFF*, on MOSI. The nordic responds providing the first byte of data on MISO. This frame is repeated five times. The default address is *0xE7E7E7E7E7*.

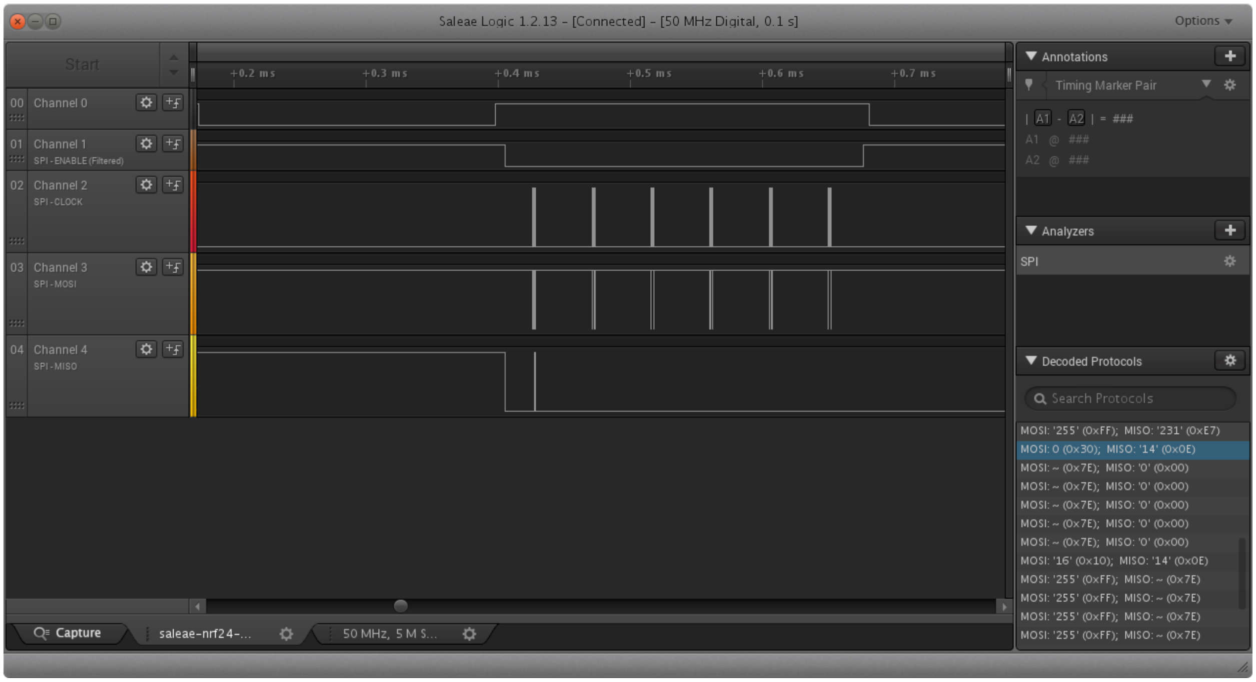

Fig. 18: Over view of the six byte write process for the TX_ADDR register.

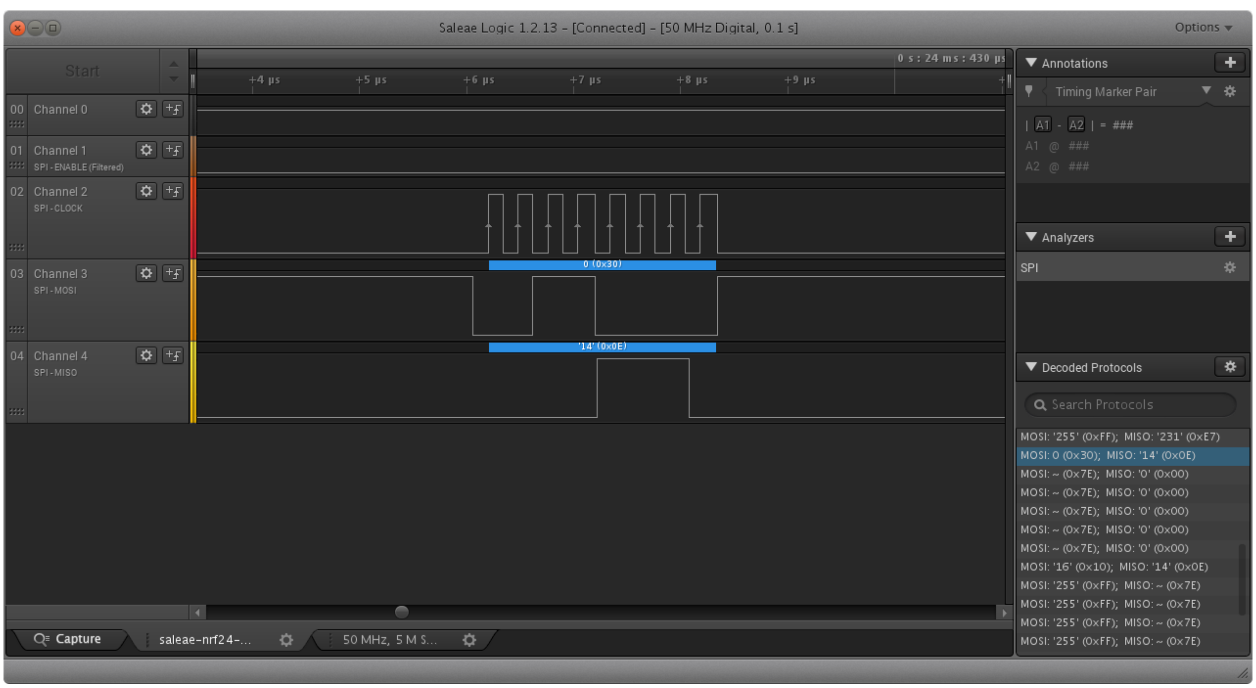

Fig. 19: The write request for the TX_ADDR register, *0x20 | 0x10*.

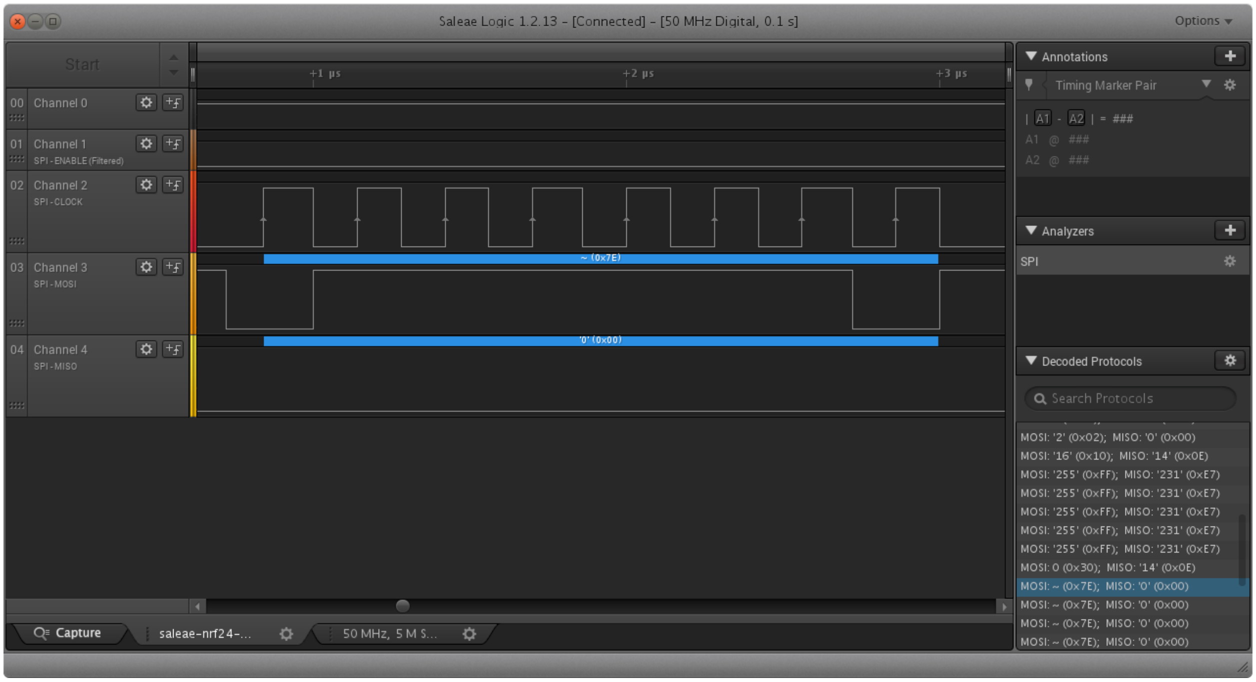

Fig. 20: The write data for the TX_ADDR register, *0x7E*. This packet is sent five times, *0x7E7E7E7E7E*.

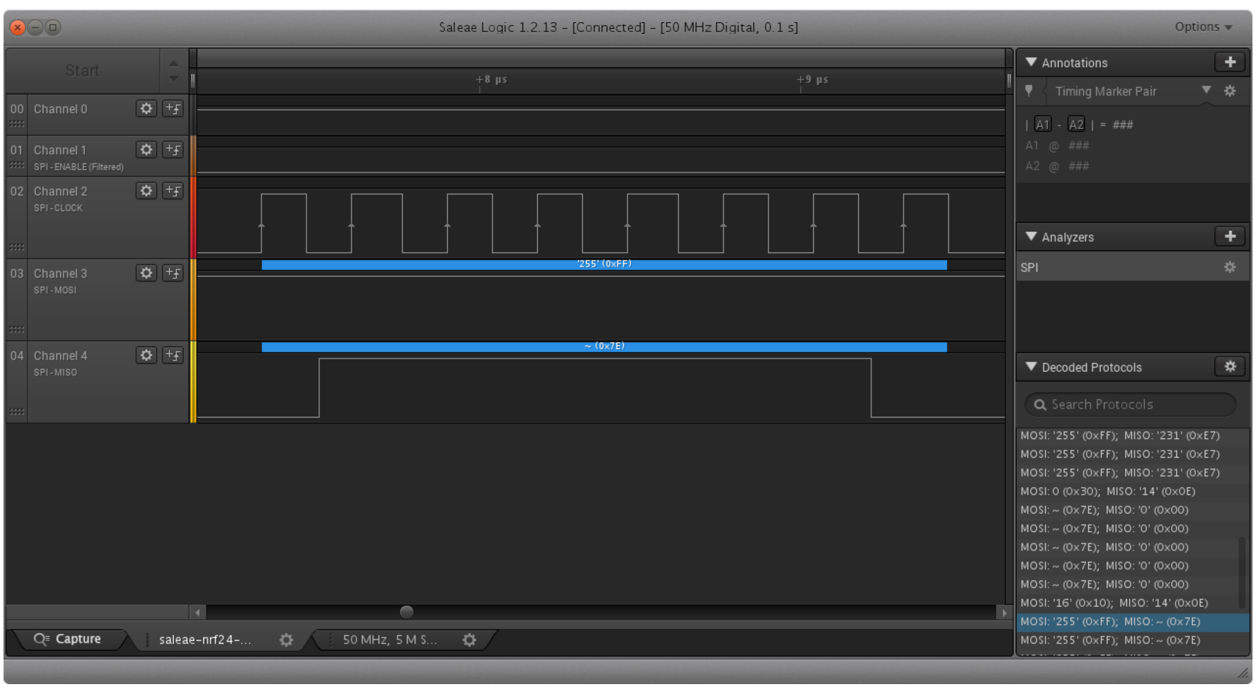

Fig. 21: The read process is repeated for the TX_ADDR register. But this time, the MISO packes are the previously written pattern, *0x7E*.