ENGR-E 399/599: Embedded systems reverse engineering
Indiana University — Spring 2022
Prepared by: Austin Roach — Assigned: March 10, 2022

Assignment 4: SPI traffic analysis

Due date: April 15, 2022

For this assignment, you will analyze traffic captured from an SPI bus joining a processor and an SPI flash memory. You have been provided with the CSV output of an SPI protocol analyzer. Your goal is to reconstruct the memory contents from the data that traversed the bus. You should submit the source code used for analysis as well as a file containing the reconstructed memory contents.

0.1 Memory description

The SPI traffic was captured as the processor read data from a Macronix MX25L1605D SPI flash memory. The datasheet for this component is provided in the `resources` directory. The capacity of the memory is 16,777,216 bits (2,097,152 bytes).

The valid SPI commands for this component are summarized in Table 4 of the datasheet. The timing diagram for the FAST READ (0x0b) command is shown in Figure 18. Note that the command byte and address bytes are sent to the memory on MOSI and the data is returned on MISO. Note also that the amount of data returned from the device is variable–the memory will continue to send data until the master raises `CS#` high.

0.2 SPI analyzer output file

In class, we captured the memory traffic using a logic analyzer and applied an SPI protocol analyzer to deserialize the captured signals and to separate the traffic into individual transactions. The exported analyzer output is in an xz-compressed CSV file in the resources directory. You can inflate the contents using the `unxz` command.

Each transaction on the bus is labeled by an ID. A new transaction begins when `CS#` goes low. All of the traffic captured while `CS#` remains low is listed with the same ID. At the end of a transaction, `CS#` goes high. When `CS#` goes low again, any subsequent traffic will be labeled with an incremented transaction ID.

The protocol analyzer has already deserialized the data on MISO and MOSI for you. Each group of 8 bits is listed as a single (hexadecimal) byte value in the CSV.

Note that the values on both the MOSI and MISO signal lines are shown at each instant in the CSV, although as you may have noted by inspection of the timing diagrams we typically only care about the value on one of these signal lines depending on where we are in the command.

Your script should parse the CSV to separate out each of these transactions based on ID number. You will then be able to parse out the command code byte and any other values represented in each individual transaction (address, data, etc.)

1 Initial analysis: 2 points

What command codes do you see represented in the capture? In your assignment report, give both the command code byte values that you see represented, as well as the corresponding human-readable command definitions from Table 4 of the data sheet. You should see three command codes represented. Does it look like we will encounter any problems with memory contents being modified during the course of our capture?

2 Memory contents reconstruction: 8 points

Use the information from each bus transaction to reconstruct the memory contents that would have produced the observed bus traffic. For each read command that you have identified, you should be able to recover an address and some number of data bytes read from that address and subsequent addresses. Use this information to populate your model of the memory state.

Not every memory address was read during the capture period. You should represent all bytes that are unknown with the value `0xff`.

The final binary size should be 2,097,152 bytes (the capacity of the memory). Listed below are hexdumps of a few samples of the expected memory contents that you can use to validate your output. Provide both the reconstructed binary file and the script that produced it with your assignment submission.

```
00000000: 00b8 123c 5053 0724 0000 518e ffff 0a34  ...<PS.$..Q....4
00000010: 2450 2a02 0f00 4715 0000 0000 f000 0a3c  $P*...G........<
00000020: 2450 2a02 1000 073c 4000 4711 0000 0000  $P*....<@.G.....
...
00000400: 464c 5348 c802 0000 af01 1904 0302 1d70  FLSH...........p
00000410: 7000 0000 626f 6172 6474 7970 653d 3078  p...boardtype=0x
00000420: 3035 3035 0062 6f61 7264 6e75 6d3d 3230  0505.boardnum=20
...
00100000: 19ba 669f ea0c c5e6 55cb 5009 099a 670a  ..f.....U.P...g.
00100010: d9d8 ca8d 85ab 8b35 4080 845b 2bb6 426d  .......5@..[+.Bm
00100020: 1475 0667 7235 9097 9a61 ef2d fbf8 e33d  .u.gr5...a.-...=
...
00180000: b139 110f 0a08 0000 ffff ffff ffff ffff  .9..............
00180010: ffff ffff ffff ffff ffff ffff ffff ffff  ................
```

```
00180020: ffff ffff ffff ffff ffff ffff ffff ffff  ................
...
```