IP Header Checksum Journal

34331 Digital Hardware Design

Daniel Brasholt s214675

December 2022

Contents

Part 1 Objective

The objective of this mini project was to write an entity for IP Header Checksum verification in VHDL as well as a test bench which serves to verify this entity. This verification entity should report the result of the test as well as the number of passed and failed tests.

Part 2 IP Headers

The format of an IPv4 header can be seen on the figure 1 below. Since most of these are only 20 octets long, the code written for verifying these assume this length of an IPv4 header. In theory, it would be possible to read the length of the header in the correct field and act accordingly; this is however not implemented, since the length is so often 20. Therefore, the code should work for most headers regardless.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: *IPv4 header as defined in RFC 791.*

The method for verifying the checksum is simply summing all the fields in the header. If this result produces carry bits (e.g. `2fffd`), these carry bits are added to the sum (e.g. `fffd` +2 = `ffff`). Taking the one's complement of this number (flipping all the bits), should yield 0000, if the header checksum matches. If this is not the case, the IP header contains an error.

Because taking one's complement of a number is simply flipping the bits, this operation can be excluded, if the check is performed for the sequence `ffff` instead.

Part 3 Design in VHDL

The code for verifying the checksum is implemented in VHDL in the entity `header_checksum`. The entire code base can be seen in the appendix. The entity has 4 inputs and 4 outputs. The inputs are the clock, a reset signal, a signal showing the start of a packet stream, and a `std_logic_vector` containing an octet of an IP packet.

The outputs are two `std_logic` and two `std_logic_vector`. The former show that a result is ready and whether the checksum test has passed. The latter show how many correct and incorrect checksums have been tested - these simply serve as counters.

The architecture of the entity has some internal signals as well. These are mainly for letting the logic of the code be easier to implement. There are also two internal counter signals, which are mapped to the output counters, since one cannot increment an output signal inside of a process, hence the work-around.

The main logic is written in a process called `LISTENER`. This process contains the clock signal as well as the `data_in`-signal in the sensitivity list. It only runs, however, when the clock signal is on a rising edge. The incoming data is mapped to the signal `previous_data_in` since the data arrives in octets. The checksum, however, expects the data to be 16 bits. Therefore, the previous octet must be saved and used in the calculation. If the `start_of_data` signal is `1`, the entity will start listening for data. It also increments a counter, which stops the entity when it has had 20 total octets of input. This will also initialise the logic for outputting the result of the checksum calculation.

Since the data arrives in octets, another signal is flipped each time an octet arrives; `phase_one`. When this signal is `1`, it means two octets have arrived. These two octets are concatenated and added to the total current checksum.

When all the data has been read and added to the checksum, the final calculation is performed; the carry bits are added to the result. Since the result should be `ffff`, a final check is done to test this. If it matches, `cksum_ok` is pulled high for a single clock cycle, and the counter is incremented by one. Otherwise, `cksum_ok` continues to be `0` and the "incorrect" counter is incremented by one instead.

The `reset` signal is unused in this implementation, since it would have no effect. It could, however, be used to reset the counters and current sum, as well as ready the entity for the start of a new IP packet. This has not been implemented.

Part 4 Test bench

To verify that the entity behaves as expected, a test bench is implemented. This test bench adds the `header_checksum` as a component and stimulates it with a clock signal and data on the input signals. The code for the test bench can also be seen in the appendix.

The test bench reads the data from a file. This file contains a column of hex values corresponding to each input octet, as well as a column of either 1 or 0, which indicates whether or not is the start of a packet. This is then given to the `header_checksum`-entity. Lastly, when the calculation is completed (`cksum_calc` is 1), the result of the test is written to a file along with the number of the test (1 to 52 since there are 52 total tests). Between stimulating the entity, the test bench waits for 10 ns, which corresponds to a single clock cycle.

Part 5 The tests

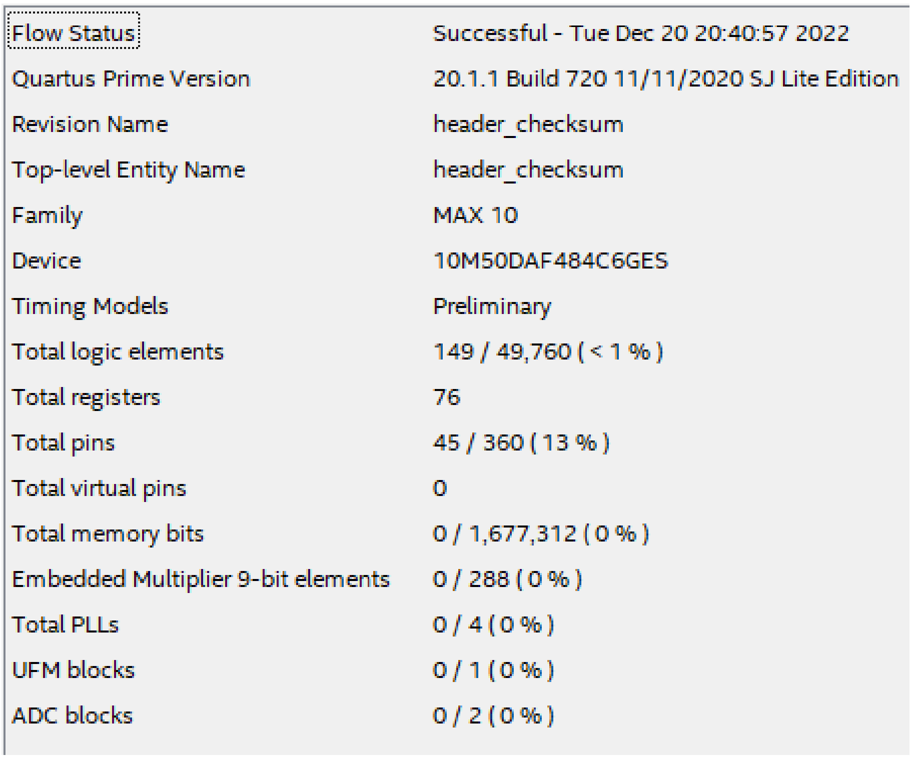
The IP headers used in the tests have been extracted from a Wireshark-trace. Since most of these arrive with the correct checksum in the header, errors have been manually introduced to these by changing some of the hex. The `input_file.txt` contains 52 total headers seperated by a few octets of dummy data. This dummy data is the same between all packets, however it can be seen on the wave-form in the simulation that these are not counted or used for anything. To make it easy to verify that the `header_checksum` entity works as expected, the 52 tests are put in the following order: 2 which should pass, 2 which should fail, 2 which should pass, and so on.

The results of the tests are written to the `output.txt`-file. The checksums are verified correctly if the pattern of the results match what is explained above.

Both the input- and output files can be found in the appendices. The input file, however, is extremely long.

Part 6 Compilation

Unfortunately, I could not get the hardware test bench to work. Running one of the old exercise solutions on the board worked fine; however when declaring everything myself and trying to control the LED's and segment display, random errors kept popping up. Therefore, no hardware test bench has been written and no tests have been performed on the board itself. There is, however, a compilation result from Quartus. This can be seen below:



Flow Status	Successful - Tue Dec 20 20:40:57 2022
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	header_checksum
Top-level Entity Name	header_checksum
Family	MAX 10
Device	10M50DAF484C6GES
Timing Models	Preliminary
Total logic elements	149 / 49,760 (< 1 %)
Total registers	76
Total pins	45 / 360 (13 %)
Total virtual pins	0
Total memory bits	0 / 1,677,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

Figure 2: *Screenshot of compilation summary*

Also, two Fmax values were found; one for ”Slow 1200mV 85c model” and another for ”Slow 1200mV 0C model”. These were 143.16MHz and 157.68MHz respectively.

Part 7 Appendix

Part 7.1. Main entity VHDL

Note: because of formatting, some lines are too long. Therefore they have been wrapped. A wrapped line is indicated with a ↪ .

```
--========================================================
-- Required libraries
--========================================================

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

--========================================================
-- Entity decleration
--========================================================

entity header_checksum is
  port ( clk     : in std_logic;
               reset     : in std_logic;
               start_of_data    : in std_logic;
               data_in     : in std_logic_vector(7 downto 0);
               cksum_calc    : out std_logic;
               cksum_ok    : out std_logic;
               cksum_ok_cnt    : out std_logic_vector(15 downto 0);
               cksum_ko_cnt    : out std_logic_vector(15 downto 0)
               );
end entity header_checksum;

--========================================================
-- Architecture declaration
--========================================================

architecture header_checksum_arch of header_checksum is

-- Signals needed to perform operations
signal currently_recording : std_logic := '0';
signal phase_one : std_logic := '0';
signal previous_data_in : std_logic_vector(7 downto 0);
signal recorder_counter : std_logic_vector(7 downto 0) := x"00";
signal cksum_ok_cnt_internal : std_logic_vector(15 downto 0) := x"0000";
signal cksum_ko_cnt_internal : std_logic_vector(15 downto 0) := x"0000";

begin

  -- main (and only) process running on clock and data
  LISTENER : process (clk, data_in)
    -- variables used to perform calculations
    variable output_sum : std_logic_vector(15 downto 0) := x"0000";
    variable total_internal_sum : std_logic_vector(23 downto 0) := x"000000";
    begin
      if (rising_edge(clk)) then
        -- store data, force outputs
        previous_data_in <= data_in;
        cksum_ok <= '0';
        cksum_calc <= '0';
        -- listen for incoming data
        if (start_of_data = '1') then
```

```
          currently_recording <= '1';
          total_internal_sum := x"000000";
          phase_one <= '1';
        -- if already in the middle of listening:
        elsif (currently_recording = '1') then
          recorder_counter <= recorder_counter + 1;
          phase_one <= not phase_one;
          -- concatenate two octets and add to sum
          if (phase_one = '1') then
            total_internal_sum := total_internal_sum + (previous_data_in & data_in
                ↪ );
          end if;
          -- reset and specify outputs once header is completely transmitted
          if (conv_integer(recorder_counter) = 18) then
            recorder_counter <= x"00";
            currently_recording <= '0';
            cksum_calc <= '1';
            output_sum := x"0000" + total_internal_sum(15 downto 0) +
                ↪ total_internal_sum(23 downto 16);
            -- Check if checksum is correct
            if (output_sum = x"FFFF") then
              cksum_ok <= '1';
              cksum_ok_cnt_internal <= cksum_ok_cnt_internal + 1;
            else
              cksum_ok <= '0';
              cksum_ko_cnt_internal <= cksum_ko_cnt_internal + 1;
            end if;
          end if;

        end if;
      end if;
  end process;

  cksum_ok_cnt <= cksum_ok_cnt_internal;
  cksum_ko_cnt <= cksum_ko_cnt_internal;

end architecture;
```

Part 7.2. Test bench VHDL

As with the main entity, a wrapped line is indicated with ↪ .

```
--=========================================================
-- Required libraries
--=========================================================

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.numeric_std_unsigned.all;
use ieee.std_logic_textio.all;

library std;
use std.textio.all;

--=========================================================
-- Entity decleration (empty)
--=========================================================

entity header_checksum_TB is
end entity;
```

```
--=========================================================
-- Architecture declaration
--=========================================================

architecture header_checksum_TB_arch of header_checksum_TB is

  -- Use header_checksum as component
  component header_checksum
    port (
               clk      : in std_logic;
               reset     : in std_logic;
               start_of_data    : in std_logic;
               data_in     : in std_logic_vector(7 downto 0);
               cksum_calc    : out std_logic;
               cksum_ok    : out std_logic;
               cksum_ok_cnt    : out std_logic_vector(15 downto 0);
               cksum_ko_cnt    : out std_logic_vector(15 downto 0)
               );
  end component;

  -- signals to bind to header_checksum
  signal clk_TB, reset_TB, start_of_data_TB : std_logic;
  signal data_in_TB : std_logic_vector(7 downto 0);
  signal cksum_calc_TB, cksum_ok_TB : std_logic;
  signal cksum_ok_cnt_TB, cksum_ko_cnt_TB : std_logic_vector(15 downto 0);

  begin

  DUT : header_checksum port map (clk_TB, reset_TB, start_of_data_TB,
                              data_in_TB,
                              cksum_calc_TB, cksum_ok_TB,
                              cksum_ok_cnt_TB, cksum_ko_cnt_TB);

  -- clock process which is constantly running
  CLOCK_PROC : process
    begin
      clk_TB <= '0'; wait for 5 ns;
      clk_TB <= '1'; wait for 5 ns;
  end process;

  -- process which applies data stimulus
  STIMULUS : process

    -- read/write input/output from/to files
    file Fin : TEXT open READ_MODE is "input_file.txt";
    file Fout : TEXT open WRITE_MODE is "output_file.txt";

    variable current_line : line;
    variable current_read_line : line;
    variable current_read_field1 : std_logic_vector(7 downto 0);
    variable current_read_field2 : std_logic;
    variable current_write_line : line;
    variable test_number : integer := 0;

    begin
      -- wait for 50ns, then begin stimulus
      wait for 50 ns;
      while (not endfile(Fin)) loop
        -- read line, then read hex and bits
        readline(Fin, current_read_line);
        hread(current_read_line, current_read_field1);
        read(current_read_line, current_read_field2);

        -- apply stimulus
        start_of_data_TB <= current_read_field2;
        data_in_TB <= current_read_field1;

        -- write output to file when done
        if (cksum_calc_TB = '1') then
          test_number := test_number + 1;
          write(current_line, string'("Test␣number␣"));
          write(current_line, test_number);
          write(current_line, string'("␣result:␣"));
          write(current_line, cksum_ok_TB);
          writeline(Fout, current_line);
        end if;

        wait for 10 ns;
      end loop;

  end process;

end architecture;
```

Part 7.3. Output file

```
Test number 1 result: 1
Test number 2 result: 1
Test number 3 result: 0
Test number 4 result: 0
Test number 5 result: 1
Test number 6 result: 1
Test number 7 result: 0
Test number 8 result: 0
Test number 9 result: 1
Test number 10 result: 1
Test number 11 result: 0
Test number 12 result: 0
Test number 13 result: 1
Test number 14 result: 1
Test number 15 result: 0
Test number 16 result: 0
Test number 17 result: 1
Test number 18 result: 1
Test number 19 result: 0
Test number 20 result: 0
Test number 21 result: 1
Test number 22 result: 1
Test number 23 result: 0
Test number 24 result: 0
Test number 25 result: 1
Test number 26 result: 1
Test number 27 result: 0
Test number 28 result: 0
Test number 29 result: 1
Test number 30 result: 1
Test number 31 result: 0
Test number 32 result: 0
Test number 33 result: 1
Test number 34 result: 1
Test number 35 result: 0
Test number 36 result: 0
Test number 37 result: 1
Test number 38 result: 1
Test number 39 result: 0
Test number 40 result: 0
Test number 41 result: 1
Test number 42 result: 1
Test number 43 result: 0
Test number 44 result: 0
Test number 45 result: 1
Test number 46 result: 1
Test number 47 result: 0
Test number 48 result: 0
Test number 49 result: 1
Test number 50 result: 1
Test number 51 result: 0
Test number 52 result: 0
```

Part 7.4. Input file

```
45 1
00 0
00 0
40 0
19 0
1e 0
40 0
00 0
40 0
11 0
9d 0
55 0
c0 0
a8 0
01 0
7e 0
c0 0
a8 0
01 0
6b 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
40 0
19 0
1f 0
40 0
00 0
40 0
11 0
9d 0
54 0
c0 0
a8 0
01 0
7e 0
```

```
c0 0
a8 0
01 0
6b 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
c4 0
c6 0
c9 0
40 0
00 0
3f 0
11 0
f0 0
75 0
c0 0
a8 0
01 0
6b 0
c0 0
a8 0
01 0
7e 0
ba 0
ab 0
ff 0
45 1
00 0
00 0
49 0
f9 0
e7 0
40 0
00 0
ff 0
21 0
de 0
90 0
c0 0
a8 0
01 0
87 0
e0 0
00 0
00 0
fb 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
e8 0
1d 0
b0 0
40 0
00 0
```

```
40 0
06 0
6f 0
ab 0
c0 0
a8 0
01 0
7e 0
a2 0
7d 0
48 0
11 0
ca 0
ab 0
ff 0
45 1
00 0
1c 0
18 0
1d 0
b1 0
40 0
00 0
40 0
06 0
54 0
7a 0
c0 0
a8 0
01 0
7e 0
a2 0
7d 0
48 0
11 0
ca 0
ab 0
ff 0
45 1
00 0
16 0
84 0
1d 0
b6 0
40 0
00 0
40 0
06 0
5a 0
09 0
c0 0
a8 0
01 0
7e 0
a3 0
7d 0
48 0
11 0
ca 0
ab 0
ff 0
```

```
45 1
00 0
01 0
34 0
78 0
b1 0
40 0
00 0
35 0
06 0
20 0
5e 0
a2 0
7d 0
48 0
11 0
c0 0
a8 0
01 0
7e 0
ca 0
ab 0
ff 0
45 1
00 0
05 0
f0 0
1d 0
ba 0
40 0
00 0
40 0
06 0
6a 0
99 0
c0 0
a8 0
01 0
7e 0
a2 0
7d 0
48 0
11 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
34 0
78 0
b2 0
40 0
00 0
35 0
06 0
20 0
5d 0
a2 0
7d 0
48 0
```

```
11 0
c0 0
a8 0
01 0
7e 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
34 0
78 0
b3 0
50 0
00 0
35 0
06 0
20 0
5c 0
a2 0
7d 0
48 0
11 0
c0 0
a8 0
01 0
7e 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
34 0
78 0
b4 0
40 0
00 0
35 0
06 0
20 0
5b 0
a2 0
7d 0
c8 0
11 0
c0 0
a8 0
01 0
7e 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
34 0
78 0
b5 0
40 0
```

```
00 0
35 0
06 0
20 0
5a 0
a2 0
7d 0
48 0
11 0
c0 0
a8 0
01 0
7e 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
34 0
78 0
b6 0
40 0
00 0
35 0
06 0
20 0
59 0
a2 0
7d 0
48 0
11 0
c0 0
a8 0
01 0
7e 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
34 0
78 0
b7 0
40 0
00 0
35 0
66 0
20 0
58 0
a2 0
7d 0
48 0
11 0
c0 0
a8 0
01 0
7e 0
ca 0
ab 0
```

```
ff 0
45 1
00 0
00 0
34 0
78 0
b8 0
40 0
00 0
35 0
06 0
20 0
a7 0
a2 0
7d 0
48 0
11 0
c0 0
a8 0
01 0
7e 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
34 0
78 0
b8 0
40 0
00 0
35 0
06 0
20 0
57 0
a2 0
7d 0
48 0
11 0
c0 0
a8 0
01 0
7e 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
34 0
78 0
ba 0
40 0
00 0
35 0
06 0
20 0
55 0
a2 0
7d 0
```

```
48 0
11 0
c0 0
a8 0
01 0
7e 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
34 0
78 0
cb 0
40 0
00 0
35 0
06 0
20 0
54 0
a2 0
7d 0
48 0
11 0
c0 0
a8 0
01 0
7e 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
e1 0
78 0
bc 0
40 0
00 0
35 0
06 0
2f 0
a6 0
a2 0
7d 0
48 0
11 0
c0 0
a8 0
01 0
7e 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
34 0
1d 0
bc 0
```

```
40 0
00 0
40 0
06 0
70 0
53 0
c0 0
a8 0
01 0
7e 0
a2 0
7d 0
48 0
11 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
44 0
ba 0
ab 0
40 0
00 0
40 0
11 0
fb 0
1f 0
c0 0
a8 0
01 0
8e 0
c0 0
a8 0
01 0
ff 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
40 0
da 0
7e 0
40 0
00 0
40 0
11 0
db 0
f4 0
c0 0
c8 0
01 0
7e 0
c0 0
a8 0
01 0
6b 0
ca 0
```

```
ab 0
ff 0
45 1
00 0
00 0
40 0
da 0
7f 0
40 0
00 0
40 0
12 0
db 0
f3 0
c0 0
a8 0
01 0
7e 0
c0 0
a8 0
01 0
6b 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
74 0
d2 0
1b 0
40 0
00 0
3f 0
11 0
e5 0
23 0
c0 0
a8 0
01 0
6b 0
c0 0
a8 0
01 0
7e 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
80 0
d2 0
1c 0
40 0
00 0
3f 0
11 0
e5 0
16 0
c0 0
```

```
a8 0
01 0
6b 0
c0 0
a8 0
01 0
7e 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
e7 0
1d 0
bd 0
40 0
00 0
a0 0
06 0
6f 0
9f 0
c0 0
a8 0
01 0
7e 0
a2 0
7d 0
48 0
11 0
ca 0
ab 0
ff 0
45 1
00 0
10 0
f0 0
1d 0
be 0
40 0
00 0
40 0
06 0
5c 0
95 0
c0 0
a8 0
01 0
7e 0
a2 0
7d 0
48 0
11 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
34 0
78 0
```

```
bd 0
40 0
00 0
35 0
06 0
20 0
52 0
a2 0
7d 0
48 0
11 0
c0 0
a8 0
01 0
7e 0
ca 0
ab 0
ff 0
45 1
00 0
01 0
78 0
1d 0
c1 0
40 0
00 0
40 0
06 0
6f 0
0a 0
c0 0
a8 0
01 0
7e 0
a2 0
7d 0
48 0
11 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
34 0
78 0
be 0
41 0
00 0
35 0
06 0
20 0
51 0
a2 0
7d 0
48 0
11 0
c0 0
a8 0
01 0
7e 0
```

```
ca 0
ab 0
ff 0
45 1
00 0
01 0
9e 0
32 0
35 0
40 0
00 0
fa 0
11 0
a5 0
50 0
c0 0
a8 0
01 0
25 0
e0 0
00 0
00 0
fb 0
ca 0
ab 0
ff 0
45 1
00 0
01 0
9e 0
32 0
67 0
40 0
00 0
ff 0
11 0
a5 0
1e 0
c0 0
a8 0
01 0
25 0
e0 0
00 0
00 0
fb 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
c6 0
21 0
c9 0
40 0
00 0
40 0
06 0
a0 0
43 0
```

```
c0 0
a8 0
01 0
7e 0
a2 0
7d 0
13 0
82 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
65 0
85 0
9a 0
40 0
00 0
ff 0
11 0
53 0
34 0
c0 0
a8 0
01 0
1e 0
e0 0
00 0
00 0
fb 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
e8 0
23 0
27 0
40 0
00 0
40 0
06 0
6a 0
34 0
c0 0
a9 0
01 0
7e 0
a2 0
7d 0
48 0
11 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
c4 0
```

```
8a 0
49 0
40 0
00 0
40 0
11 0
ed 0
b9 0
c0 0
a8 0
01 0
7e 0
ff 0
ff 0
ff 0
ff 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
df 0
7b 0
fa 0
40 0
00 0
35 0
06 0
1c 0
6a 0
a2 0
7d 0
48 0
11 0
c0 0
a8 0
01 0
7e 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
76 0
0e 0
94 0
40 0
00 0
40 0
a1 0
6a 0
1d 0
c0 0
a8 0
01 0
1e 0
ff 0
ff 0
ff 0
ff 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
40 0
64 0
71 0
50 0
00 0
40 0
11 0
52 0
02 0
c0 0
a8 0
01 0
7e 0
c0 0
a8 0
01 0
6b 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
e8 0
23 0
31 0
40 0
00 0
40 0
06 0
6a 0
2a 0
c0 0
a8 0
01 0
7e 0
a2 0
7d 0
48 0
11 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
c5 0
8f 0
26 0
40 0
00 0
40 0
06 0
32 0
```

```
e7 0
c0 0
a8 0
01 0
7e 0
a2 0
7d 0
13 0
82 0
ca 0
ab 0
ff 0
45 1
00 0
02 0
1f 0
07 0
4a 0
40 0
00 0
36 0
06 0
c3 0
66 0
a2 0
7d 0
13 0
82 0
c0 0
a8 0
01 0
7e 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
c4 0
91 0
c4 0
40 0
00 0
40 0
11 0
e6 0
3f 0
c0 0
a8 0
01 0
7e 0
ff 0
ff 0
ff 0
ff 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
```

```
e7 0
23 0
3c 0
40 0
00 0
40 0
06 0
6a 0
20 0
c0 0
a8 0
01 0
7e 0
a2 0
7d 0
48 0
11 0
ca 0
ab 0
ff 0
45 1
00 0
01 0
80 0
19 0
a9 0
40 0
00 0
ff 0
11 0
bd 0
90 0
c0 0
a8 0
01 0
8f 0
e0 0
00 0
00 0
fb 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
b5 0
93 0
e8 0
49 0
00 0
3f 0
11 0
23 0
16 0
c0 0
a8 0
01 0
6b 0
c0 0
a8 0
```

```
01 0
7e 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
3c 0
00 0
00 0
40 0
10 0
3b 0
06 0
7d 0
d8 0
22 0
6b 0
dd 0
52 0
c0 0
a8 0
01 0
7e 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
54 0
d4 0
40 0
40 0
00 0
40 0
01 0
94 0
32 0
c0 0
a8 0
01 0
7e 0
08 0
08 0
08 0
08 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
54 0
00 0
00 0
00 0
00 0
3b 0
01 0
```

```
ad 0
73 0
08 0
08 0
08 0
08 0
c0 0
a8 0
01 0
7e 0
ca 0
ab 0
ff 0
45 1
00 0
00 0
b2 0
e6 0
f1 0
50 0
00 0
40 0
06 0
b2 0
7e 0
c0 0
a8 0
01 0
7e 0
5f 0
a6 0
7f 0
09 0
ca 0
ab 0
ff 0
45 1
00 0
02 0
98 0
d0 0
eb 0
40 0
00 0
3c 0
06 0
ca 0
9e 0
5a 0
a6 0
7f 0
09 0
c0 0
a8 0
01 0
7e 0
```