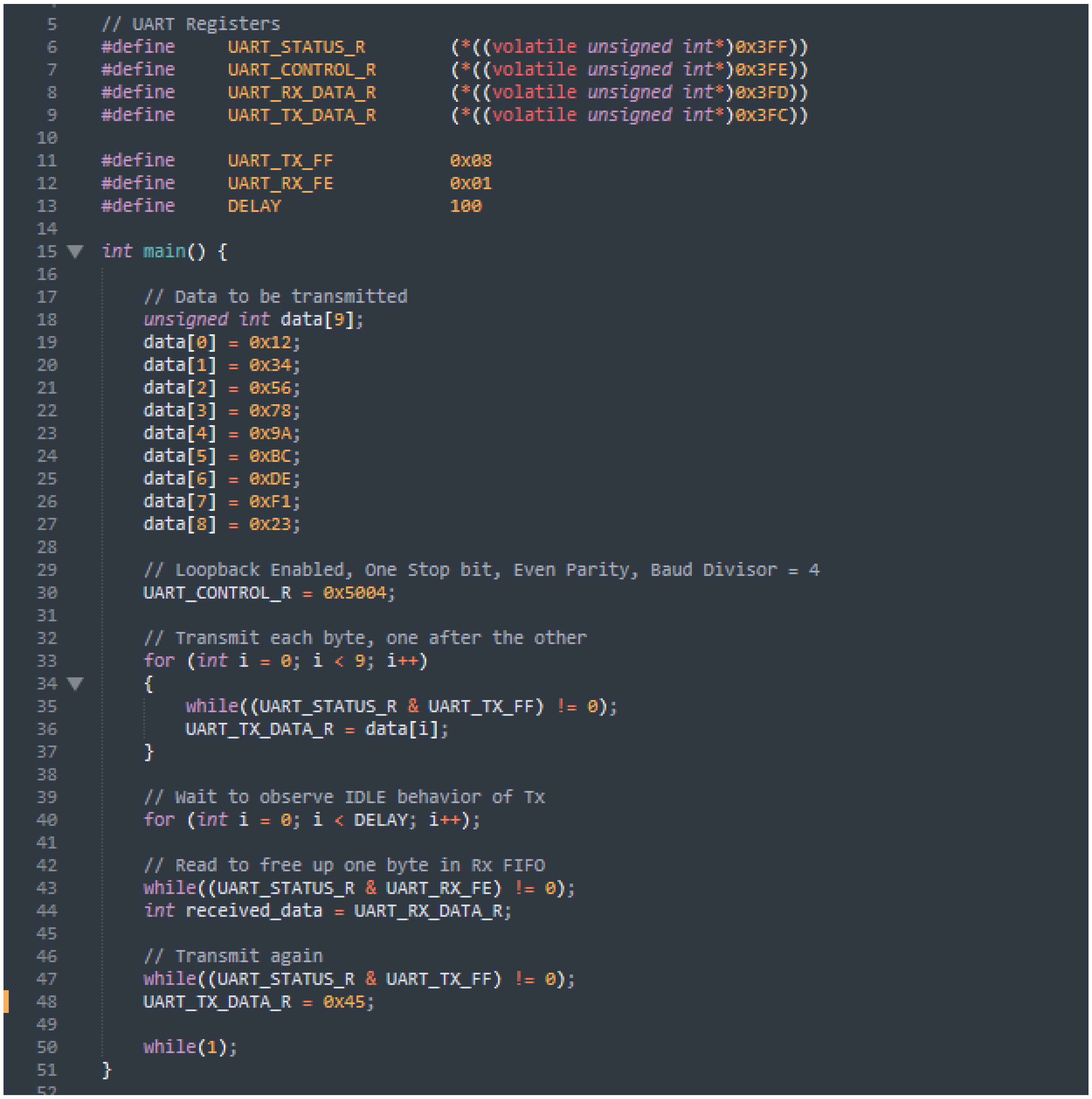
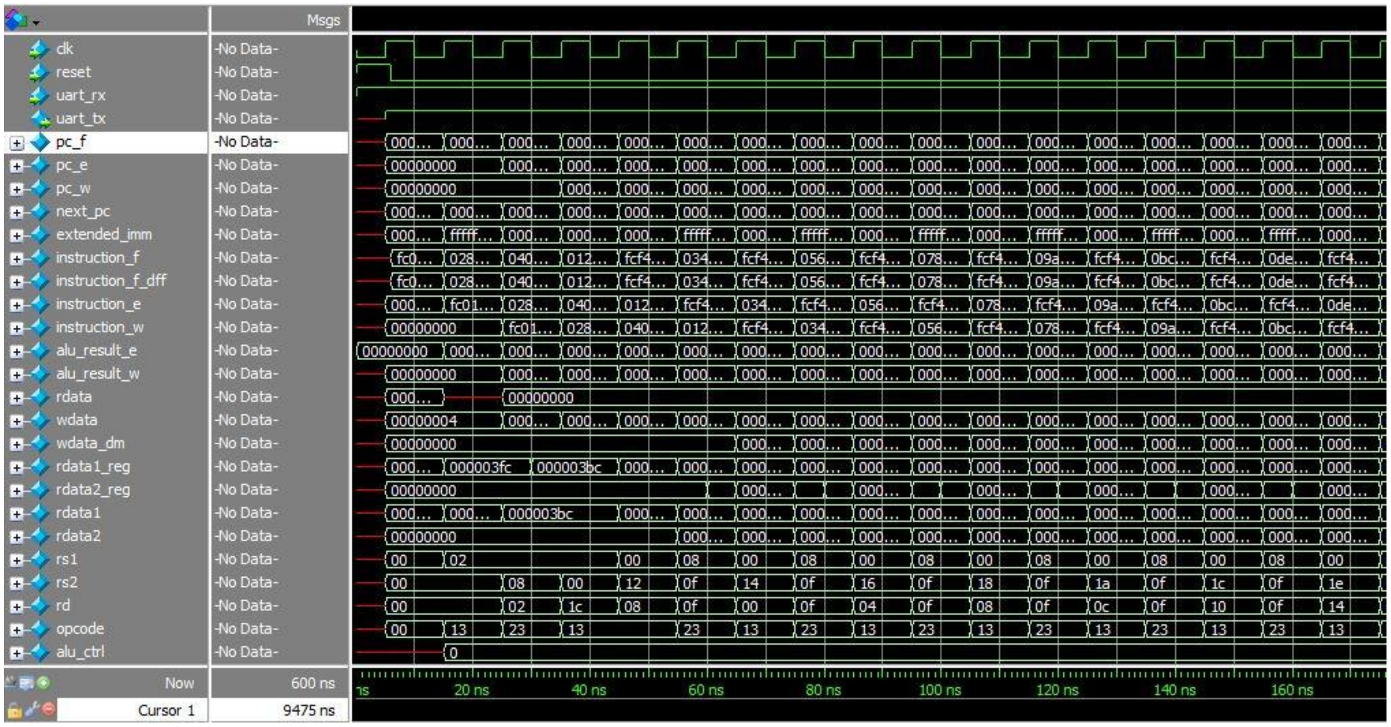
UART INTEGRATED PIPELINE VERIFICATION

We will run the following C code on our processor.



```c
// UART Registers
#define     UART_STATUS_R       (*((volatile unsigned int*)0x3FF))
#define     UART_CONTROL_R      (*((volatile unsigned int*)0x3FE))
#define     UART_RX_DATA_R      (*((volatile unsigned int*)0x3FD))
#define     UART_TX_DATA_R      (*((volatile unsigned int*)0x3FC))

#define     UART_TX_FF          0x08
#define     UART_RX_FE          0x01
#define     DELAY               100

int main() {

    // Data to be transmitted
    unsigned int data[9];
    data[0] = 0x12;
    data[1] = 0x34;
    data[2] = 0x56;
    data[3] = 0x78;
    data[4] = 0x9A;
    data[5] = 0xBC;
    data[6] = 0xDE;
    data[7] = 0xF1;
    data[8] = 0x23;

    // Loopback Enabled, One Stop bit, Even Parity, Baud Divisor = 4
    UART_CONTROL_R = 0x5004;

    // Transmit each byte, one after the other
    for (int i = 0; i < 9; i++)
    {
        while((UART_STATUS_R & UART_TX_FF) != 0);
        UART_TX_DATA_R = data[i];
    }

    // Wait to observe IDLE behavior of Tx
    for (int i = 0; i < DELAY; i++);

    // Read to free up one byte in Rx FIFO
    while((UART_STATUS_R & UART_RX_FE) != 0);
    int received_data = UART_RX_DATA_R;

    // Transmit again
    while((UART_STATUS_R & UART_TX_FF) != 0);
    UART_TX_DATA_R = 0x45;

    while(1);
}
```

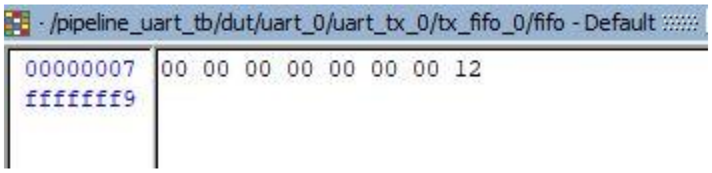
First of all, the program loads the data into the data memory.



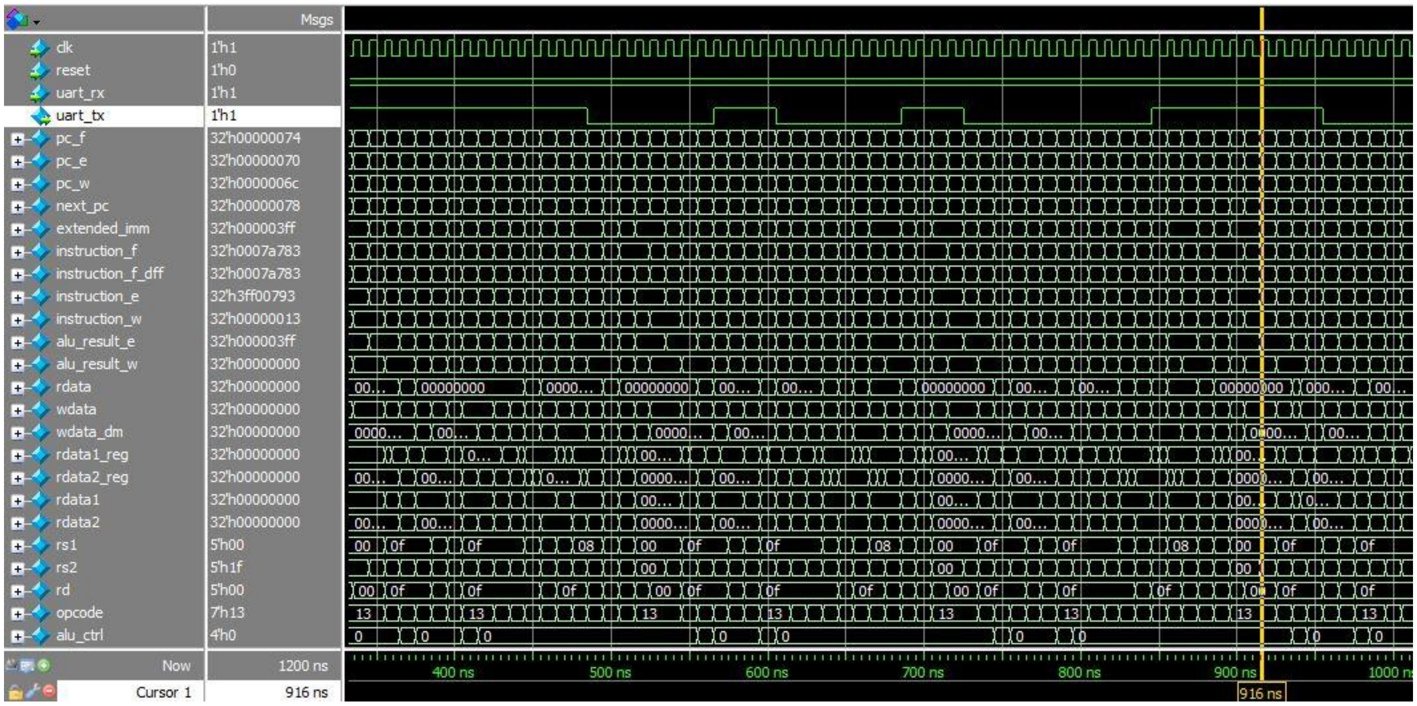
```
                 Msgs
clk              -No Data-
reset            -No Data-
uart_rx          -No Data-
uart_tx          -No Data-
pc_f             -No Data-
pc_e             -No Data-
pc_w             -No Data-
next_pc          -No Data-
extended_imm     -No Data-
instruction_f    -No Data-
instruction_f_dff -No Data-
instruction_e    -No Data-
instruction_w    -No Data-
alu_result_e     -No Data-
alu_result_w     -No Data-
rdata            -No Data-
wdata            -No Data-
wdata_dm         -No Data-
rdata1_reg       -No Data-
rdata2_reg       -No Data-
rdata1           -No Data-
rdata2           -No Data-
rs1              -No Data-
rs2              -No Data-
rd               -No Data-
opcode           -No Data-
alu_ctrl         -No Data-
Now              600 ns
Cursor 1         9475 ns
```

We have selected the Baud Divisor to be 4, Even Parity, One Stop bit and Loopback enabled.

Then the program moves onto transmitting the data, as soon as data is written to **UART_TX_DR.**

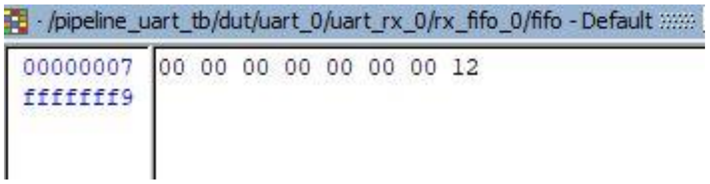


```
- /pipeline_uart_tb/dut/uart_0/uart_tx_0/tx_fifo_0/fifo - Default
00000007  00 00 00 00 00 00 00 12
fffffff9
```



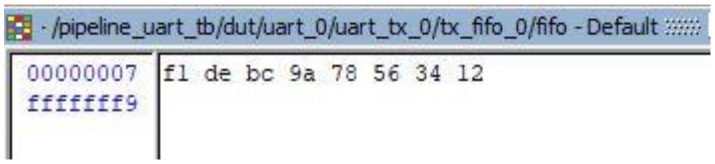
```
                    Msgs
clk                 1'h1
reset               1'h0
uart_rx             1'h1
uart_tx             1'h1
pc_f                32'h00000074
pc_e                32'h00000070
pc_w                32'h0000006c
next_pc             32'h00000078
extended_imm        32'h000003ff
instruction_f       32'h0007a783
instruction_f_dff   32'h0007a783
instruction_e       32'h3ff00793
instruction_w       32'h00000013
alu_result_e        32'h000003ff
alu_result_w        32'h00000000
rdata               32'h00000000
wdata               32'h00000000
wdata_dm            32'h00000000
rdata1_reg          32'h00000000
rdata2_reg          32'h00000000
rdata1              32'h00000000
rdata2              32'h00000000
rs1                 5'h00
rs2                 5'h1f
rd                  5'h00
opcode              7'h13
alu_ctrl            4'h0
Now                 1200 ns
Cursor 1            916 ns
```

As loopback mode is enabled, we can see that as the transmission completes the same data ends up in RX_FIFO.



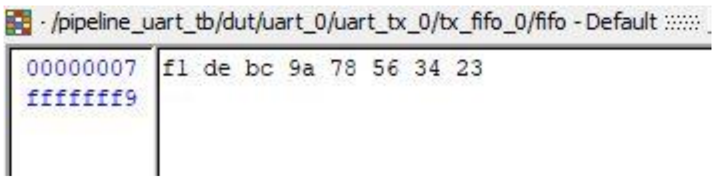
```
· /pipeline_uart_tb/dut/uart_0/uart_rx_0/rx_fifo_0/fifo - Default
00000007  00 00 00 00 00 00 00 12
fffffff9
```

The Program keeps on writing to UART_TX_DR, which goes to the TX_FIFO.

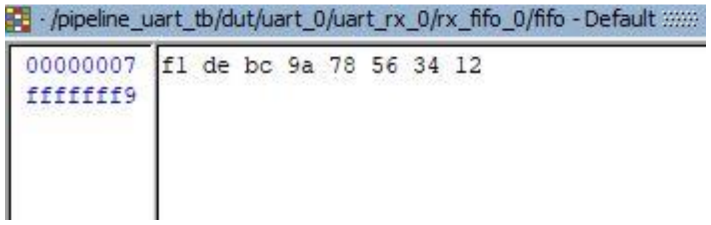


```
· /pipeline_uart_tb/dut/uart_0/uart_tx_0/tx_fifo_0/fifo - Default
00000007  f1 de bc 9a 78 56 34 12
fffffff9
```

As the data is being transmitted when we are writing to the FIFO, so it also freeing up and we can write more data to it. We can see that **0x12** has been replaced by **0x23**.

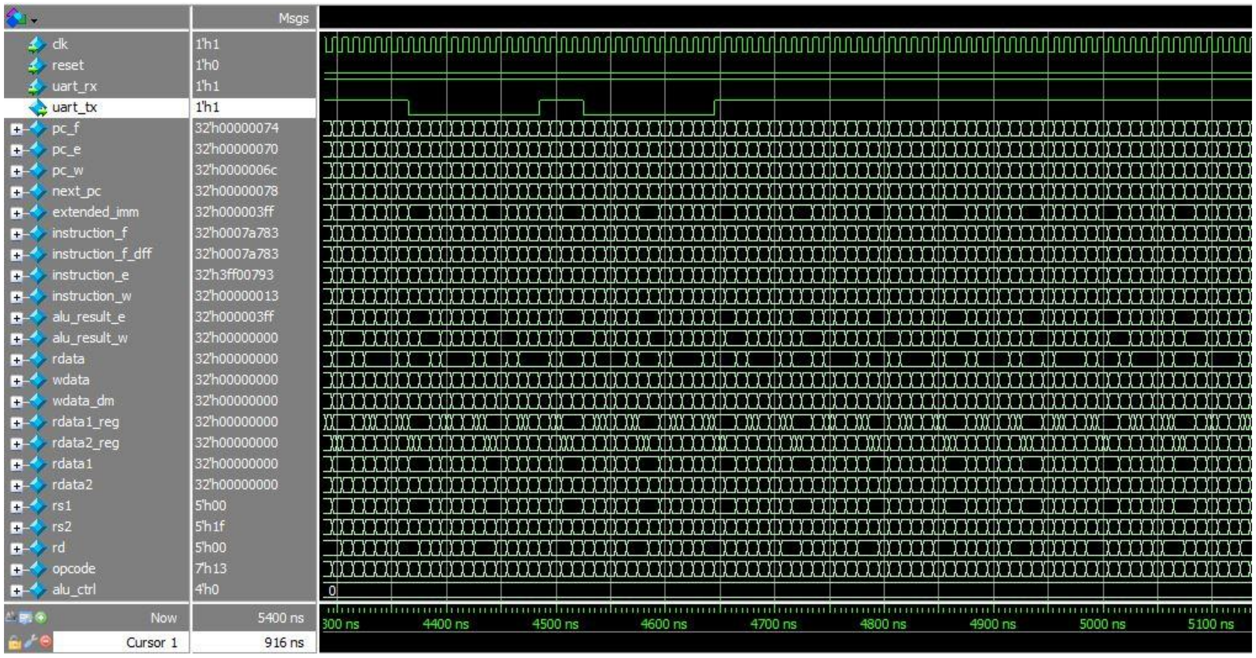

```
/pipeline_uart_tb/dut/uart_0/uart_tx_0/tx_fifo_0/fifo - Default
00000007  f1 de bc 9a 78 56 34 23
fffffff9
```

Eventually, same data is being received from UART receiver and goes into RX_FIFO.


```
/pipeline_uart_tb/dut/uart_0/uart_rx_0/rx_fifo_0/fifo - Default
00000007  f1 de bc 9a 78 56 34 12
fffffff9
```

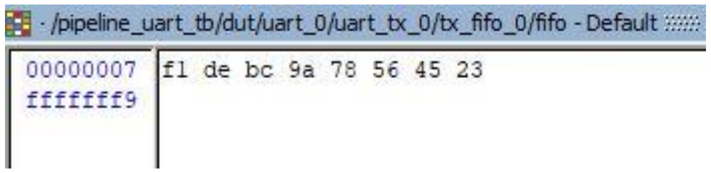
Here, we can observe that the RX_FIFO does not have **0x23**, as the receiver FIFO if full therefore it can’t accept more than 8 bytes.

After transmitting the 9 bytes, the program enters into a state of delay, which can be observed below.


```
                    Msgs
clk                 1'h1
reset               1'h0
uart_rx             1'h1
uart_tx             1'h1
pc_f                32'h00000074
pc_e                32'h00000070
pc_w                32'h0000006c
next_pc             32'h00000078
extended_imm        32'h000003ff
instruction_f       32'h0007a783
instruction_f_dff   32'h0007a783
instruction_e       32'h3ff00793
instruction_w       32'h00000013
alu_result_e        32'h000003ff
alu_result_w        32'h00000000
rdata               32'h00000000
wdata               32'h00000000
wdata_dm            32'h00000000
rdata1_reg          32'h00000000
rdata2_reg          32'h00000000
rdata1              32'h00000000
rdata2              32'h00000000
rs1                 5'h00
rs2                 5'h1f
rd                  5'h00
opcode              7'h13
alu_ctrl            4'h0
Now                 5400 ns
Cursor 1            916 ns
```

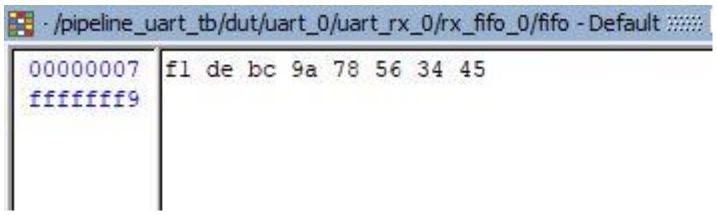
After the delay finishes, we read from the RX_FIFO, which frees up one byte from the RX_FIFO. Therefore, new incoming data can now be stored in RX_FIFO.

Let us transmit another byte **0x45.**



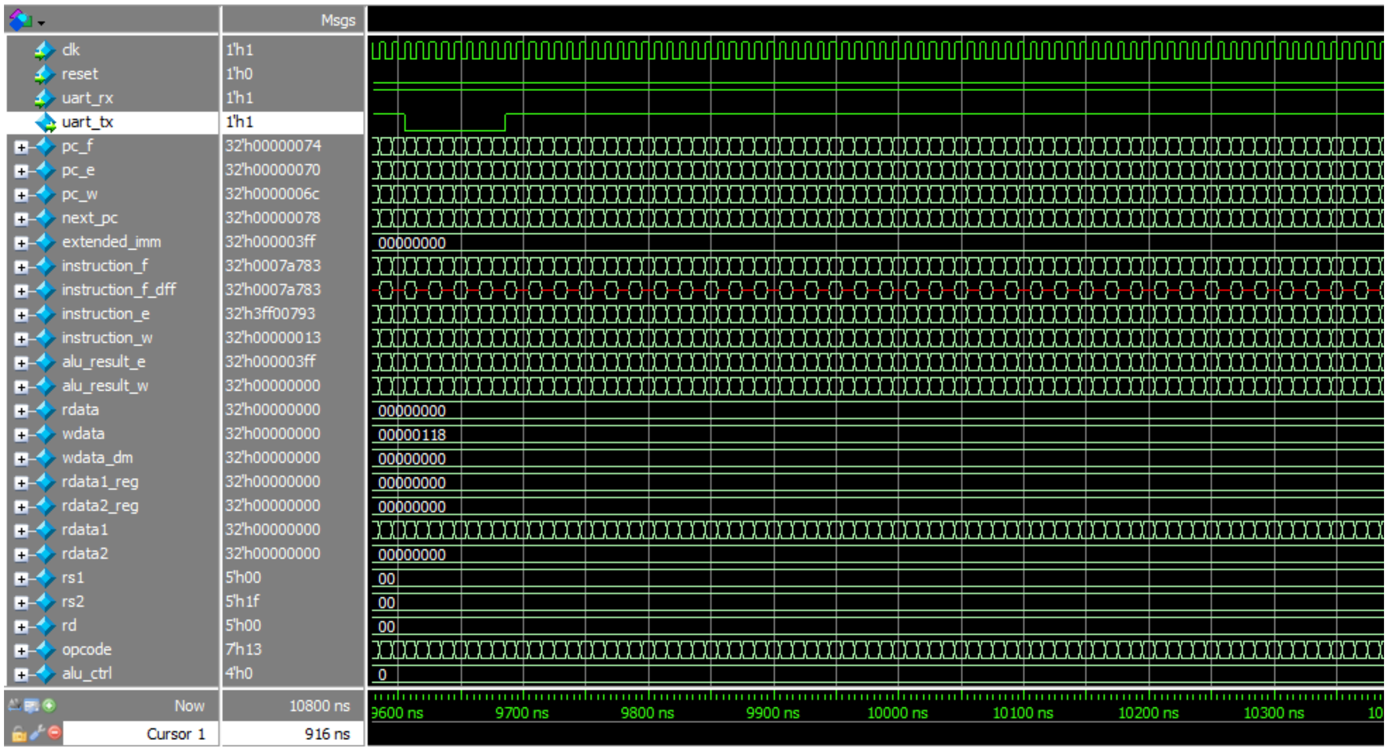
```
- /pipeline_uart_tb/dut/uart_0/uart_tx_0/tx_fifo_0/fifo - Default
00000007  f1 de bc 9a 78 56 45 23
fffffff9
```

Which is successfully read stored in RX_FIFO.



```
- /pipeline_uart_tb/dut/uart_0/uart_rx_0/rx_fifo_0/fifo - Default
00000007  f1 de bc 9a 78 56 34 45
fffffff9
```

After that the program finishes, and it remains in a state of infinite loop.



```
                 Msgs
clk              1'h1
reset            1'h0
uart_rx          1'h1
uart_tx          1'h1
pc_f             32'h00000074
pc_e             32'h00000070
pc_w             32'h0000006c
next_pc          32'h00000078
extended_imm     32'h000003ff       00000000
instruction_f    32'h0007a783
instruction_f_dff 32'h0007a783
instruction_e    32'h3ff00793
instruction_w    32'h00000013
alu_result_e     32'h000003ff
alu_result_w     32'h00000000
rdata            32'h00000000       00000000
wdata            32'h00000000       00000118
wdata_dm         32'h00000000       00000000
rdata1_reg       32'h00000000       00000000
rdata2_reg       32'h00000000       00000000
rdata1           32'h00000000
rdata2           32'h00000000       00000000
rs1              5'h00              00
rs2              5'h1f              00
rd               5'h00              00
opcode           7'h13
alu_ctrl         4'h0               0
Now              10800 ns
Cursor 1         916 ns
9600 ns  9700 ns  9800 ns  9900 ns  10000 ns  10100 ns  10200 ns  10300 ns
```