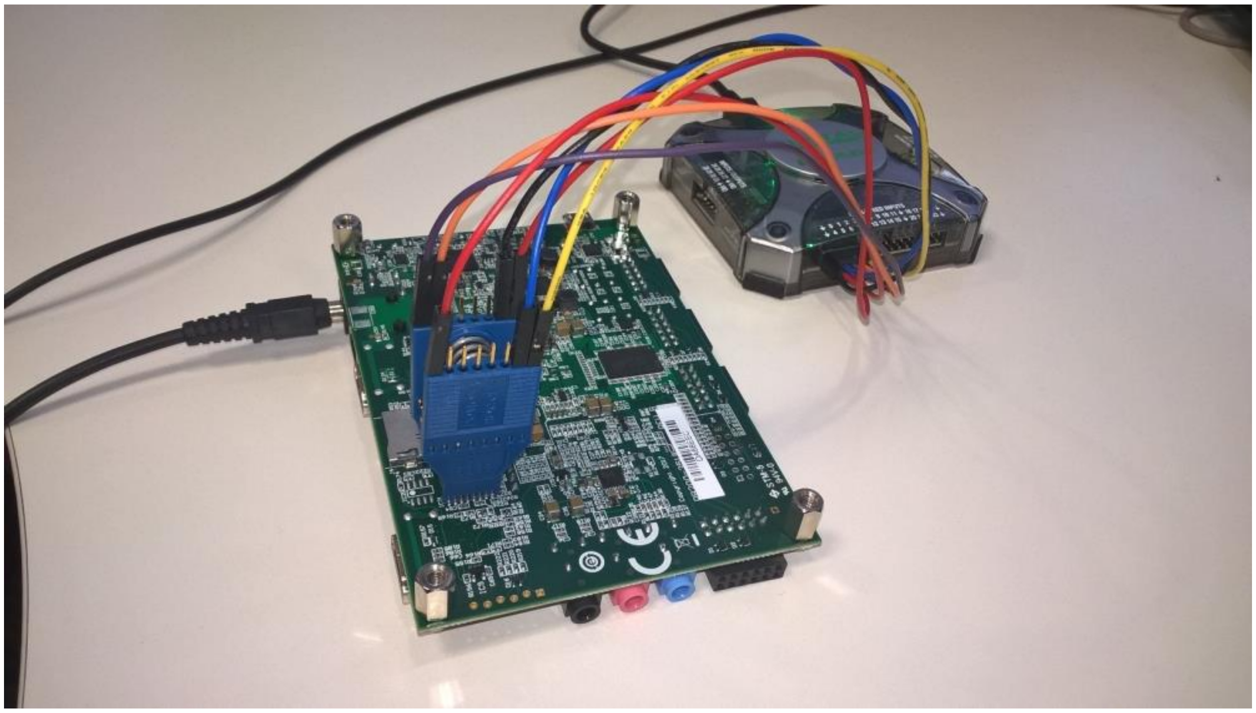
How to debug flash with Digital Discovery

When developing a new host board, it's important to know the hardware specification and observe the timing of signals. Digilent Digital Discovery provides a High Speed Logic Analyzer that allows users to visualize and analyze the signals traversing through development board. For example, in the process of developing a new field programmable gate array (FPGA) board, the speed of the QSPI transactions in the boot sequence isn't specified clearly. To solve this issue, we use the Digital Discovery to visualize the boot sequence to debug the logic.

Step 1: Connecting the Digital Discovery

First of all, we use the SOIC clip connect Digital Discovery with the on-board flash.



Step 2: QSPI script

Then, a custom interpreter is used to translate the QSPI signals into data. This is activated by adding a “Custom” channel from the Logic instrument in Digilent WaveForms. (see below javascript code)

```
// rgData: input, raw digital sample array
// rgValue: output, decoded data array
// rgFlag: output, decoded flag array

var c = rgData.length // c = number of raw samples
var pClock = false; // previous cock signal level
var iStart = 0;     // used to keep track on word start index
var cByte = 0;      // byte count per transmission
var cBits = 0;      // bit counter
var bValue = 0;     // value variable
var fCmd = true;

for(var i = 0; i < c; i++){ // for each sample
    var s = rgData[i]; // current sample
    var fSelect = 1&(s>>0); // pin0 is the select signal
    var fClock = 1&(s>>1); // pin1 is the clock signal
    var fData = 1&(s>>2); // pin2 is the data signal
    var fData4 = 0xF&(s>>2); // DIN 2-5 DQ 0-3

    if(fSelect != 0){ // select active low
        // while select inactive reset our counters/variables
        iStart = i+1; // select might become active with next sample
        cByte = 0;
        cBits = 0;
        bValue = 0;
        pClock = false;
        fCmd = true;
        continue;
    }
    if(pClock == 0 && fClock != 0){ // sample on clock rising edge

            bValue <<= 4; // serial data bit, MSBit first
            bValue |= fData4;

            cBits++;
            if(cBits==2){ // when got the 8th bit of the word store it
                cByte++;
                // store rgValue/Flag from word start index to current sample
position
                for(var j = iStart; j < i; j++){
                    // Flag change will be visible on plot even when data remains
constant.
                    // This is useful in case we get more consecutive equal
values.
                    rgFlag[j] = cByte;
                    rgValue[j] = bValue;
                }
                iStart = i+1; // next word might start after this sample
                cBits = 0;  // reset bit count for the next byte
                bValue = 0; // reset value variable
            }

    }
    pClock = fClock; // previous clock level
}
```

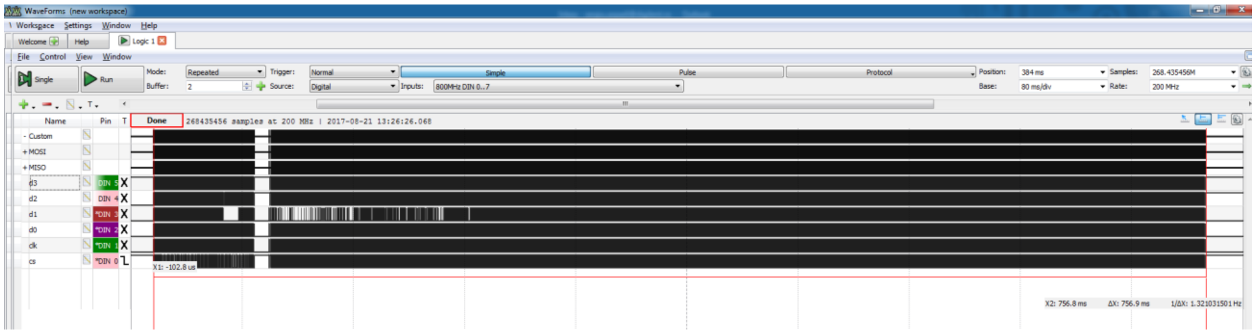
Step 3: Trigger and acquisition

Although the maximum QSPI clock frequency is about 100 MHz, when booting, a maximum frequency of 25 MHz is used. Also, the entire boot transfer takes about 700 ms. Because of this, both a large number of samples and a decent sample rate are required, and this is where the Digital Discovery comes in handy. 268 million samples at 200 MHz would translate into a ~1.3 second frame.

The acquisition itself is quite demanding, using a lot of the PC's memory (16 GB) and it also takes a long time to process the data.

The trigger is set on the falling edge of the CS signal.

Below is the entire QSPI transaction captured by Digilent Waveforms.

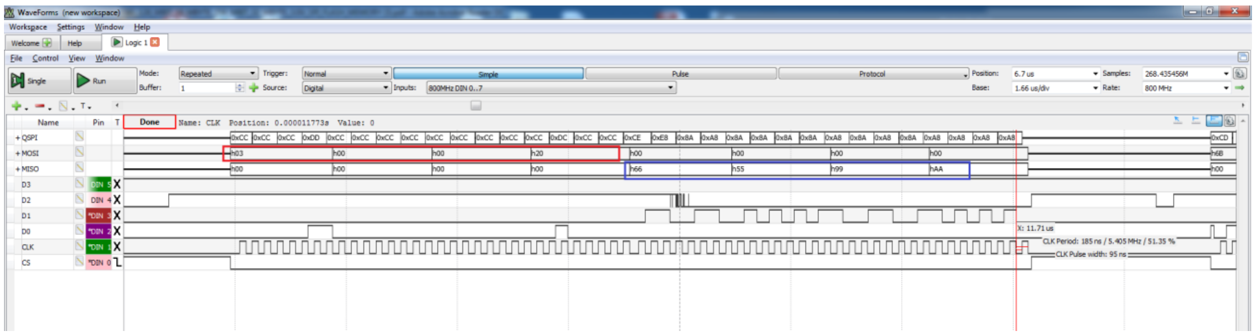


```
Mode: Repeated   Trigger: Normal   Simple   Pulse   Protocol   Position: 384 ms   Samples: 268.435456M
Buffer: 2        Source: Digital   Inputs: 800MHz DIN 0..7     Base: 80 ms/div    Rate: 200 MHz
Done  268435456 samples at 200 MHz | 2017-08-21 13:26:26.068
X1: -102.8 us
X2: 756.8 ms   ΔX: 756.9 ms   1/ΔX: 1.321031501 Hz
```

Step 4: Boot transfers

There are two documents that need to be read in order to understand what the data transfers represent. One is the Zynq TRM and the other one is the flash memory's datasheet.

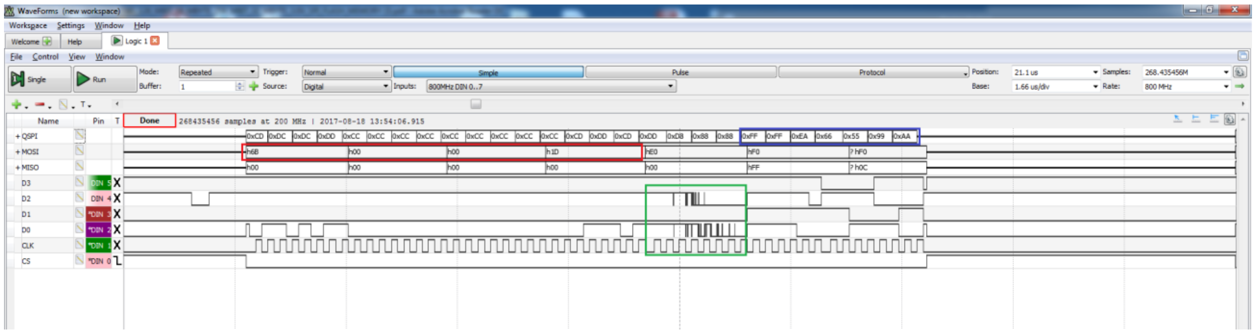
The instructions sent from the Zynq to the flash memory are always sent via SPI using D0. The first instruction sent is 0x03 0x00 0x00 0x20 which means SPI READ from address 0x20 and the reply is also received via SPI using D1, 0x66 0x55 0x99 0xaa. The flash read instruction is explained on page 85 of the datasheet.



```
Mode: Repeated   Trigger: Normal   Simple   Pulse   Protocol   Position: 6.7 us     Samples: 268.435456M
Buffer: 1        Source: Digital   Inputs: 800MHz DIN 0..7     Base: 1.66 us/div  Rate: 800 MHz
Done  Name: CLK  Position: 0.000011773s  Value: 0
QSPI: 0xCC 0xCC 0xCC 0xDD 0xCC 0xCC 0xCC 0xCC 0xCC 0xCC 0xCC 0xCC 0xCC 0xDC 0xCC 0xCC 0xCE 0xE8 0x8A 0xA8 0x8A 0x8A 0x8A 0x8A 0xA8 0x8A 0xA8 0x8A 0xA8 0xA8 0xA8 0xA8 ... 0xCD
MOSI: h03 h00 h00 h20 h00 h00 h00 h00 ... h6B
MISO: h00 h00 h00 h00 h66 h55 h99 hAA ... h00
X: 11.71 us
CLK Period: 185 ns / 5.405 MHz / 51.35 %
CLK Pulse width: 95 ns
```

In the Zynq TRM pages 170 and 179 explain what that reply means. In short, that set of bytes tell the Zynq that the memory is QSPI capable. It is also important to observe that, at this point, the SPI clock frequency is 5.405 MHz, which is a relatively low speed.

From this point on, since it has been determined that the memory supports QSPI, all transactions will be done on all 4 data lines. For instance, the next instruction will be 0x6b followed by a 3 byte address. 0x6b represents a quad read instruction and the response will be seen on the QSPI interpreter after 8 clock periods, which are "dummy" bytes.

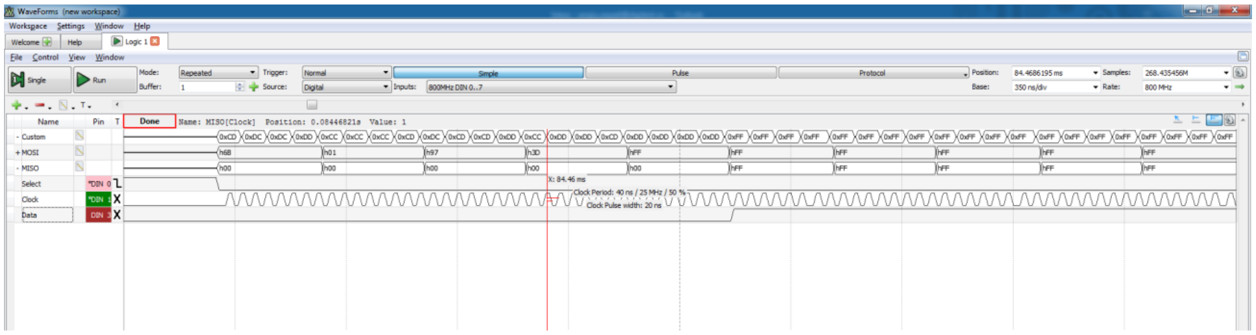


Mode: Repeated Trigger: Normal Position: 21.1 us Samples: 268.435456M
Buffer: 1 Source: Digital Inputs: 800MHz DIN 0..7 Base: 1.66 us/div Rate: 800 MHz

```
268435456 samples at 200 MHz | 2017-08-18 13:54:06.915
QSPI  0xCD 0xDC 0xDC 0xDD 0xCC 0xCC 0xCC 0xCC 0xCC 0xCC 0xCC 0xCC 0xCC 0xCD 0xDD 0xCD 0xDD 0xD8 0x88 0x88 0xFF 0xFF 0xEA 0x66 0x55 0x99 0xAA
MOSI  h6B h00 h00 h1D hE0 hF0 ? hF0
MISO  h00 h00 h00 h00 h00 hFF ? h0C
```

In this case, the address is 0x1d and 7 bytes are read. These bytes are from addresses 0x1d, 0x1e, 0x1f which are part of an interrupt table and then it reads 4 bytes from address 0x20 which are the same bytes read at the first SPI read.

The Zynq will proceed to read bytes, incrementing the address until it reaches 0x45, which is the end of the bootROM header.



Mode: Repeated Trigger: Normal Position: 84.4686195 ms Samples: 268.435456M
Buffer: 1 Source: Digital Inputs: 800MHz DIN 0..7 Base: 350 ns/div Rate: 800 MHz

```
Name: MISO[Clock]  Position: 0.08446821s  Value: 1
Custom  0xCD 0xDC 0xDC 0xDD 0xCC 0xCC 0xCC 0xCD 0xDC 0xCD 0xCD 0xDD 0xCC 0xDD 0xDD 0xCD 0xDD 0xDD 0xDD 0xDD 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
MOSI    h6B h01 h97 h3D hFF hFF hFF hFF hFF hFF
MISO    h00 h00 h00 h00 h00 hFF hFF hFF hFF hFF
```

Unfortunately, because we do not have access to the BootROM code, the rest of the boot sequence is not so transparent. At some point, the FSBL (first stage boot loader) will begin to run, most likely where the SPI clock frequency changes to 25 MHz as seen below, 84 ms after the boot process started.

The FSBL will then read the boot image and analyze the different partitions that it contains, including the .bit file, which will configure the Zynq's PL, and the .elf which will run in the ARM.