Minimalistic SDHC-SPI Host Reader

Paulino Ruiz-de-Clavijo Vázquez

Departamento de Tecnología Electrónica, Universidad de Sevilla

April, 2017 v1.1

Overview

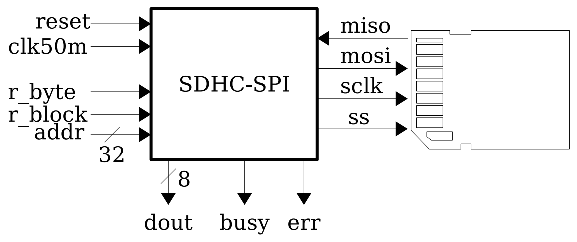

Figure 1: Top view module

MinSDHC-SPI Host is an IP core written in VHDL that implements the SD card protocol using Finite State Machines (FSM). It is has been designed to use a low resources but losing some performance compared with other solutions. The implementation uses SD SPI mode and a minimal set of SD commands to initialize and read SD cards.

Features

The module performs two operations: SD card initialization and blocks read. SD card initialization is auto-triggered when the *reset* signal is de-asserted (set to 0). Read operation is controller by signals *r_block* and *r_byte*. This implementation uses the minimal set of CMD commands, therefore the CMD12 is not used and the read process is aborted waiting for all block bytes. The read operation is performed in the single block mode.

Main features are:

- Only SDHC cards are supported
- Programmable SPI pre-scaler
- Picoblaze interface
- Alternative architecture in VDHL for simulation purpose

The main purpose of this implementation is to achieve low footprint, sacrificing performance, the table 1 show results for some Xilinx's FPGAs.

Functional description and I/O signals

The Figure 1 presents the I/O interface signals, they are described in Table 2.

FPGA	Slices	Slices Reg.	LUTs	Slices (%)
Spartan-3 - XC3S100E	179	111	270	18.6%
Virtex-5 - XC5VLX50T	116	111	211	1.61%
Artix-7 - XC7A35T	63	88	211	0.77%

Table 1: Resources utilization for some Xilinx's FPGAs

Signal	Direc.	Description
reset	input	Reset / SD Card Initialization trigger
clk50m	input	50MHz input clock
addr[31:0]	input	32-bit SD card block address to be read
r_block	input	Reads a new block addressed by addr[31:0]
r_byte	input	Gets next byte of the current block
dout[7:0]	output	Output data bus
busy	output	Operation running indicator
err	output	Error condition indicator

Table 2: Core I/O signals

SD card initialization is auto-triggered when the *reset* signal is de-asserted (set to 0). The Figure 2 contains a timing diagram with the initialization sequence. In the Figure the *busy* signal keeps asserted while the initialization process is not finished. This operation is completed when *busy* signal falls, on error (card is not present or not supported) *err* signal is asserted.

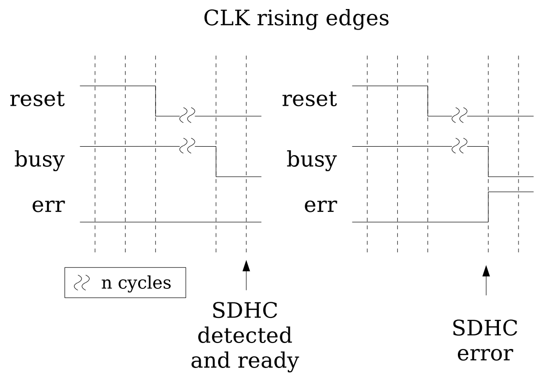

Figure 2: Top view module

Once the initialization process success, the read operation is performed setting the SD block number at *addr* bus and asserting the signal *r_block*. The module behavior is similar to the initialization process, while the block is not ready, the *busy* signal keeps asserted. After *busy* fall, if *err* is asserted an error happened otherwise, the first byte of the 512-byte block is ready in *dout* bus. The next 511 bytes are gotten asserting the *r_byte* signal 511 times as is shown in the timing diagram of the Figure 3.

The block read operation can be interrupted at any time by de-asserting the *r_block* signal.

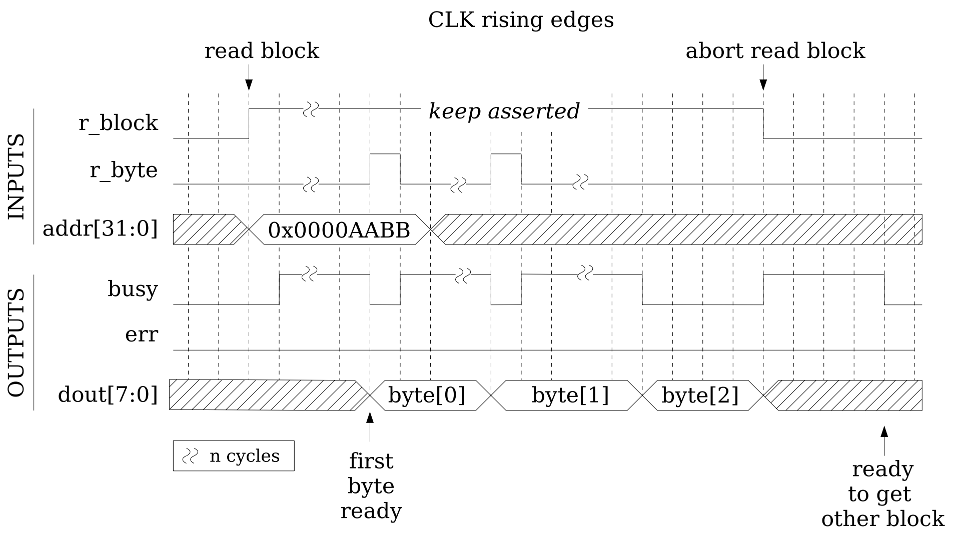

Figure 3: Read block operation

The following examples shows how the module can be controlled from a FSM written in VHDL. The full VHDL code for the examples are is available in the examples directory.

The first example shows the initialization process controlled by another FSM using two states. The following VHDL code starts resetting the module and wait until the SD card is ready or an error happens.

```
    when ST_INIT_SD =>                -- Wait SDHOST INIT
      sdhost_reset <= '1';
      next_st <= ST_WAIT_SD_READY;

    when ST_WAIT_SD_READY =>          -- Wait for SDHOST ready
      sdhost_reset <= '0';
      if sdhost_busy= '1' then
        next_st <= ST_WAIT_SD_READY;
      elsif sdhost_err='1' then
        next_st <= ST_SD_INIT_ERR;
      else
        next_st <= ST_SD_INIT_DONE;
      end if;
```

The next example is a one block read operation and it is achieved using three states. A counter is added to detect when the 512 bytes of the block are read.

```
     when ST_READ_BLOCK =>
       sdhost_r_block <= '1';       -- Read block
       next_st        <= ST_WAIT_BYTE;

     when ST_WAIT_BYTE =>           -- Wait for a byte ready
       sdhost_r_block <= '1';       -- Mandatory keep signal asserted
       if sdhost_busy= '1' then
         next_st <= ST_WAIT_BYTE;
       elsif sdhost_err='1' then
         next_st <= ST_ERR;
       else
         ...                        -- Byte ready, do something ...
         next_st <= ST_LOOP;
       end if;

      when ST_LOOP =>              -- Loop to read 512 bytes from SD card.
        sdhost_r_block  <= '1';    -- Keep asserted to get
                                   -- other byte on same block
        if (byte_counter = b"111111111") then -- 512 bytes read
          next_st <= ST_END_BLOCK;
        else
          byte_counter_up <= '1'; -- byte_counter = byte_counter + 1;
          sdhost_r_byte   <= '1'; -- r_byte pulse to get other byte
          next_st         <= ST_WAIT_BYTE;
        end if;
```

Picoblaze-3 interface

An example with the module connected as a peripheral to Picoblaze-3 (KCPSM3 [1]) microcontroller is supplied. It allow read SD cards with smalls pieces of Picoblaze assembler code. Although an SD card can be controlled from Picoblaze adding an SPI peripheral, the code required spends a lot of program code due Picoblaze-3 has only 1024-instructions for program code available.

The file *rtl/if_picoblaze.vhdl* contains an example with the glue logic for Picoblaze-3. The proposed logic uses one output port of Picoblaze and two input ports. Writing a byte in the out port, an operation of the table 3 is triggered. The operation 0x01 need a 4-byte argument, the SD card block (32-bit number). This argument must be sent after 0x01 with 4 writings in this port. The two input ports are selected by *addr*.

This example has some issues like the absence of reset operation, but it can be done by adding the module reset signal to some port. It will improve in future version using the KCPSM6.

Value	**Description**
0x01	Send the 32-bit address, after this byte 4 bytes more are expected with the SD card block address
0x02	Starts a read for the block address received with the command 0x01
0x04	Reads the next byte of the current block

Table 3: Write port operation

addr	**Description**
0	Read status register
1	Read SDHC-SPI byte out

Table 4: Read port operations

bit 7	**bit 6**	**bit 5**	**bit 4**	**bit 3**	**bit 2**	**bit 1**	**bit 0**
-	-	-	-	-	sdhost_busy	sdhost_err	byte_ready

Table 5: Status register

References

[1] Xilinx Inc., PicoBlaze 8-bit Embedded Microcontroller User Guide for Extended Spartan-3 and Virtex-5 FPGAs. Introducing PicoBlaze for Spartan-6,Virtex-6, and 7 Series FPGAs (2011).