SD/MMC Bootloader Specification

Author: Arnim Läuger
arniml@opencores.org

Rev. 2.0
March 9, 2005

This page has been intentionally left blank.

Revision History

Rev.	Date	Author	Description
1.0	27-Feb-2005	Arnim Läuger	First Version
2.0	03/09/05	Arnim Läuger	Update for set concept

Contents

1 Introduction

The SD/MMC Bootloader is a CPLD design that manages configuration and bootstrapping of FPGAs. It is able to retrieve the required data from SecureDigital (SD) cards or MultiMediaCards (MMC) and manages the FPGA configuration process. SD cards as well as MMCs are operated in SPI mode which is part of both standards thus eliminating the need for dedicated implementations. The SD/MMC Bootloader fits both. Beyond configuration, this core supports a bootstrapping strategy where multiple images are stored on one single memory card.

For example consider a system completely based on SRAM. The bootloader provides an initial configuration data from the first image to the FPGA. This image contains a design which pulls the next image from the memory card and transfers this data to SRAM. In the third step the final FPGA design is loaded from the third image.

Features

- Configuration mode: configures SRAM based FPGAs via slave serial mode (Xilinx and Altera)
- Data mode: provides stored data over a simple synchronous serial interface
- Broad compatability using SPI mode
    - SecureDigital cards using dedicated initialization command
    - MultiMediaCards (see below)
- Operation triggered by power-up or card insertion

The SD/MMC Bootloader project is maintained and released on the OpenCores web server at

http://www.opencores.org/projects.cgi/web/spi_boot/overview/

Updates of this core can be obtained via the project pages.

2

Architecture

The architecture of the SD/MMC Bootloader is depicted in Figure 1. It consists of the controller and command FSMs, an SPI port, two config/data ports and three counters.

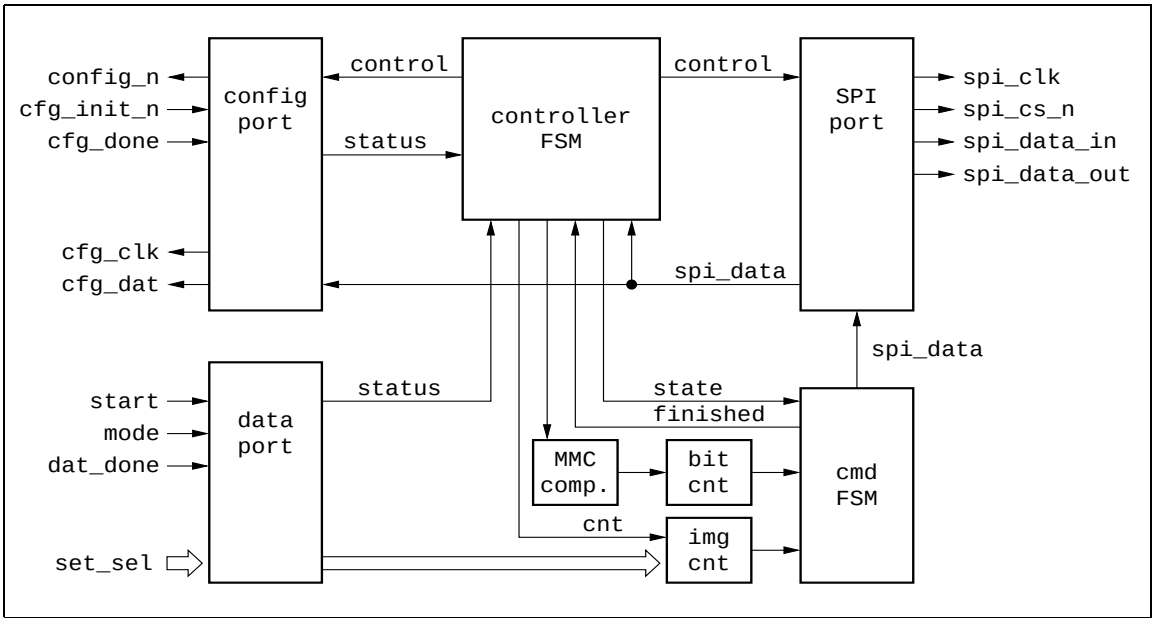

Figure 1: SD/MMC Bootloader block diagram

Controller FSM

The controller FSM manages the overall functionality of the core. On one hand, this includes the complete SD/MMC SPI mode protocol with initialization, data retrieval and abort. On the other hand, configuration and data requests are handled.

Command FSM

The command FSM sequences each single SPI mode command and generates the corresponding bit stream. Each sequence consists of the command itself, the card's response and optional data. Whenever this sequence has finished, the controller FSM is flagged, triggering it to step to the next command.

Bit Counter

The bit counter provides a generic counting service to the command FSM. I.e. it times each part of a command sequence and generates an overflow indicator to the command FSM.

Image Counter

The image counter tracks the number of the current image. It increments according to the instructions of the controller FSM.

MMC Compatability Counter

For full compatability with the MMC standard it is required to initialize the card with a maximum clock frequency of 400 kHz. The clock division is done with the MMC compatibility counter which signals its overflow to the bit counter. As soon as the initialization phase has finished, the MMC compat counter is disabled by the controller FSM.

SPI Port

The SPI port connects to the pins of the SD or MM card according to Table 1.

Signal	Connector	Description
spi_clk	Pin 5, CLK	Clock
spi_cs_n	Pin 1, CS	Chip Select (Active low)
spi_data_in	Pin 7, DataOut	Card to Host Data and Status
spi_data_out	Pin 2, DataIn	Host to Card Commands and Data

Table 1: SD/MMC connections of SPI port

Furthermore, this port contains an output enable signal to put all outputs to tri-state.

Configuration Port

This port interfaces to the configuration facilities of the FPGA. It matches both Altera and Xilinx products. The mapping is given in Table 2.

Signal	Altera	Xilinx	Description
config_n	nCONFIG	PROGRAM#	Initiates FPGA configuration sequence
cgf_init_n	nSTATUS	INIT#	Rising edge indicates end of initialization
cfg_done	CONF_DONE	DONE	Loading the configuration completed
cfg_clk	DCLK	CCLK	Configuration clock
cfg_dat	DATA0	DIN	Configuration data

Table 2: Configuration port mapping

Data Port

The data port accepts control signals that control the sequence when reading multiple images and sets from the card. Table 3 describes their meaning.

Signal	Description
start	Initiates configuration sequence when asserted low
mode	Mode selector: 0 → configuration mode, 1 → data mode
dat_done	Loading in data mode completed
set_sel	Set selection

Table 3: Data port signals

Memory Organization

Data on the card is paritioned into sets, with each set consisting of one or more images. Sets are a static containers while images form the dynamic part of this two-fold scheme. The core automatically increments its image counter whenever an image is requested via the `start` trigger. Hence the dynamic attribute of images. The address offset for the images involved is calculated by the set selection inputs. They are applied externally and are static during a bootstrapping sequence.

Figure 2 shows the generic memory organization.

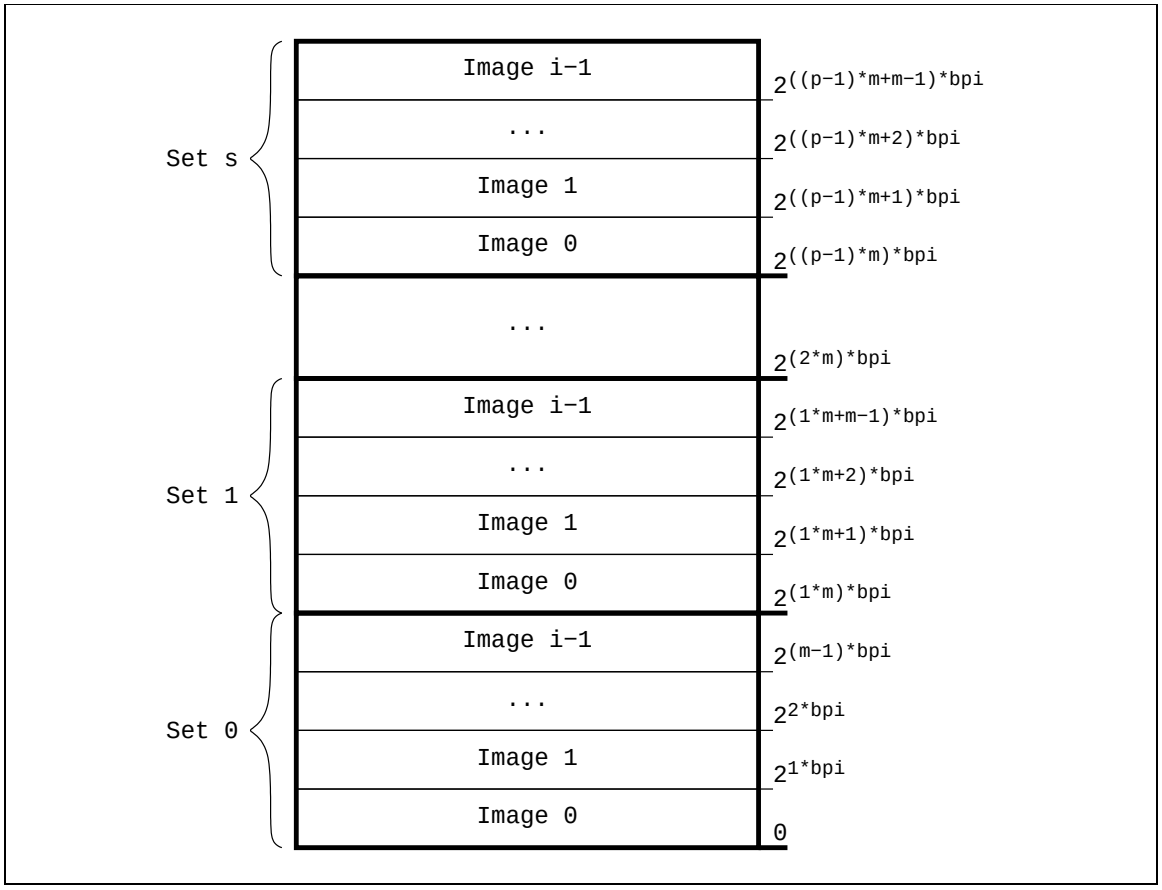

Figure 2: Memory organization

Parameters in Figure 2 are defined as follows (see also Table 5)

- bpi, bits per image: Number of bits required for the image address range.
  bpi = 18 → image contains 2^{18} = 256 KB
- i: Number of images within a set.
- m: Width of set counter.
  m = 2 → 2^2 = 4 images
- s: Number of sets on card.
- P: Width of set selection vector
  p = 4 → 2^4 = 16 sets

3

Operation

The SD/MMC Bootloader has three operation states coupled to the interaction with the memory card. After reset, the core is in initialization state and automatically configures the SD or MM card. Next is the idle state where the core deactivates its outputs on the SPI interface. Upon an external request, the core switches to the transfer state and requests data from the memory card. The transfer state itself has two modes for either configuring an FPGA or simply passing through the card's data.

Initialization State

The flow diagram of the initialization state is shown in Figure 3.

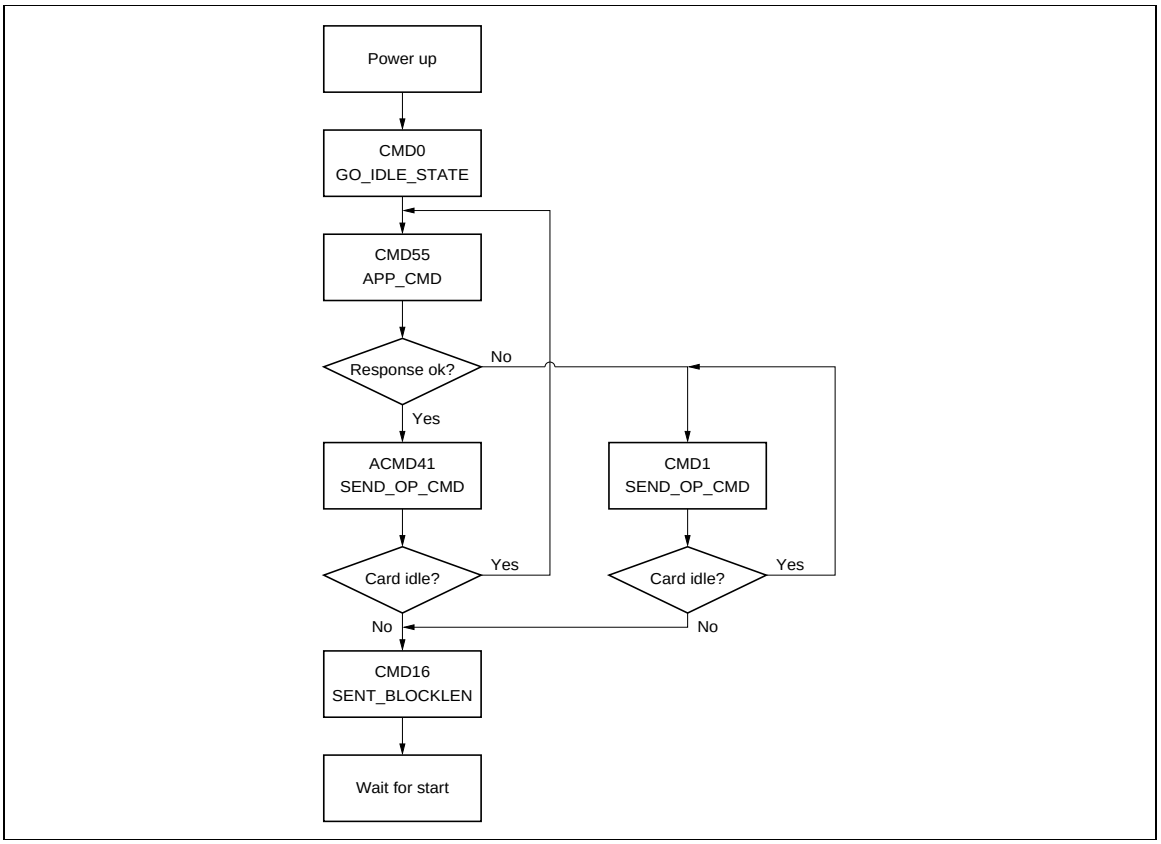

Figure 3: Initialization flow diagram

After reset, the core remains in the power up state spending a total of 144 clock cycles before the SPI interface is activated. This is more than twice the time the card needs to initialize its internal states (specified to 64 clock cycles). The extra time is to eliminate uncertainties both in power ramp up and card start up. If the application is time critical and there is an external power supply monitor it is safe to reduce the power up time to something around 74 clocks. This modification has to be applied to the VHDL source code of the core.

The first command issued by the core is GO_IDLE_STATE (CMD0) with parallel assertion of CS. This resets the card and puts it in SPI mode. Them the core sends the command APP_CMD (CMD55) to escape the next extended command. MultiMediaCards will respond to this with an illegal command error. The core detects this and uses CMD1 in the further process. In both ways (CMD55 + ACMD41 and CMD1) the idle status of the card is polled repeatedly. Once it left idle state, the core sets the desired block length with SET_BLOCKLEN (CMD16). The block length is derived from the generic parameter width_bit_cnt (refer to Table 5).

The core is now idle and SPI output signals are tri-stated.

Transfer State

Whenever a start trigger is detected, the core turns to transfer state. The trigger consists of a low-to-high transition of the start_i input. To allow automatic operation, the core also treats a constant high level at start_i as a trigger after reset. I.e. when the core reaches the idle state for the first time, it continues immediately when start_i is high. For subsequent loops through transfer and idle state start_i has to go low and high again.

When going from idle to transfer mode, the core samples the mode_i input which determines whether the transfer should be done in configuration or data mode. This operation is shown in Figure 4.

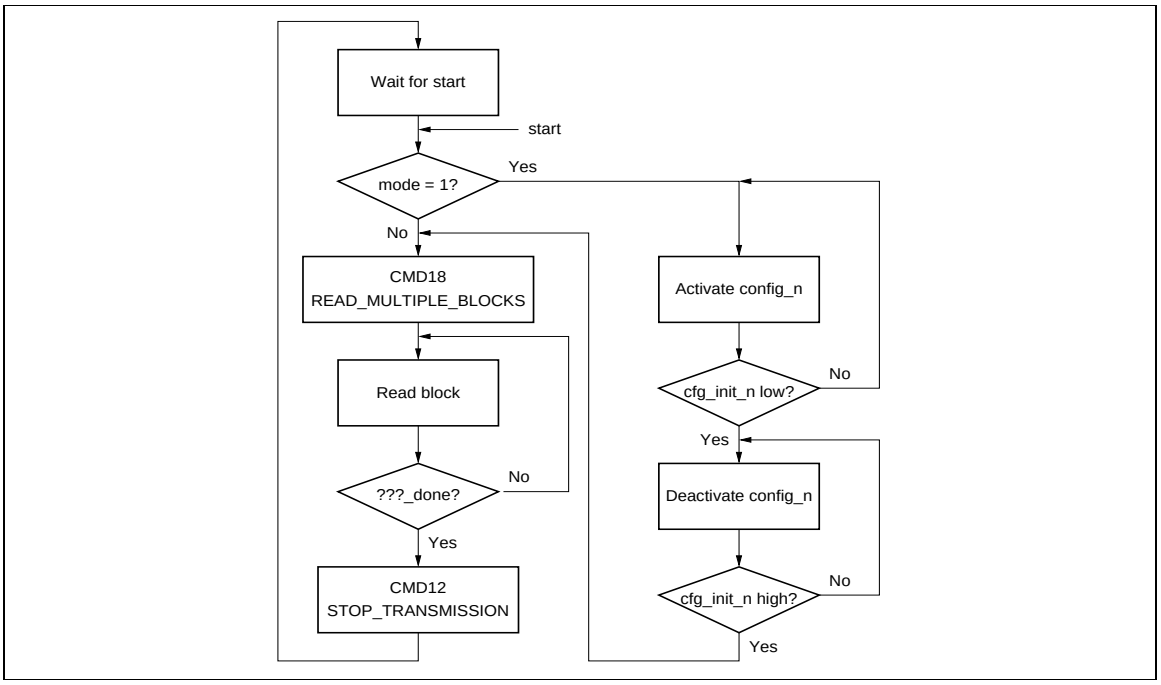

Figure 4: Transfer flow diagram

Configuration and data mode are the same except that in configuration mode a configuration cycle for the FPGA is generated. This cycle is similar for Altera and Xilinx devices (also refer to Table 2):

1. Activation of config_n → configuration memory is cleared
2. Wait for low level on cfg_init_n → FPGA acknowledges assertion of config_n
3. Wait for high level on cfg_init_n → memory cleared, ready for configuration data

This sequence is skipped for data mode. The core continues with the data transfer itself in both modes.

First step is to request a block of data by sending READ_MULTIPLE_BLOCKS command (CMD18). The address of the first block depends on the selected set and the current value of the image counter. Starting from 0, it counts the number of images transferred and the start address of an image is derived from the following formula

$$start\_address = 2^{num\_bits\_per\_image} * (cnt_{img} + set * 2^{width\_img\_cnt})$$

As soon as the card has retrieved the specified block, the bit stream is presented at the cfg_clk and cfg_dat outputs. Data at cfg_dat can be sampled with the rising edge of cfg_clk. The card now sends one block after another without intervention of the core. Configuration clock and data outputs are only operated when there is a valid bit stream from the card. I.e. during gaps between two blocks cfg_clk remains on high level. It changes from high to low for the next valid bit on cfg_dat.

This sequence is terminated when either cfg_done or cfg_dat is activated. The configuration clock is stopped immediately and the core sends STOP_TRANSMISSION (CMD12) to the card. It may take some time before the core is finally in idle state again depending on the block size and the time the done signal has been activated.

4

Integration

This chapter provides informations on the integration of the SD/MMC Bootloader in a FPGA system.

Configuration Timing

As written above, the configuration clock is stopped immediately when the core samples a high level on either cfg_done or dat_done. The intention is to prevent any unwanted data bits to be transferred to the FPGA. This scheme requires on the other hand that the FPGA has terminated its configuration process at this time. For Spartan IIe devices (and probably others) it is therefore necessary to program the DONE pin to the last cycle of the startup sequence (one clock cycle earlier is probably also ok). With the default settings of Xilinx WebPack the FPGA will not be able to finish its startup sequence because DONE is asserted too early.

There is no experience so far with Altera devices.

Writing Data to the Card

Downloading the configuration data to the card is a straight forward process. The images have to be written starting at dedicated locations. For the provided toplevel designs, these locations are multiples of 256 K. I.e. 0, 0x40000, 0x80000 and so forth.

dd (part of the GNU coreutils) serves this purpose:

```
$ dd if=ram_loader.bin of=/dev/sdX bs=512
$ dd if=pongrom_6.bin of=/dev/sdX bs=512 seek=512
$ dd if=pacman.bin of=/dev/sdX bs=512 seek=1024
```

The name of the device node depends on how the card reader is attached to the kernel. For Linux systems this is most often something like /dev/sdX with X ranging from a-z. Please note that it is essential to use the device without any trailing numbers as they refer to partitions leading to wrong offsets for data written to the card.

All this works perfectly for my Spartan IIe device as this FPGA expects the configuration data as it is delivered from the card: Consecutive bytes each with its most significant bit first. Altera devices like the FLEX family are different here. They expect the bytes with least significant bit first. Therefore, the configuration data has to be swapped bitwise before it is written to the card.

Schematic

A sample schematic for embedding SD/MMC Bootloader in an FPGA system is provided in spi_boot_schematic.pdf. I use it to configure/boot the Xilinx Spartan IIe on BurchED's B5-X300 board. SV2 fits the "SERIAL MODE" connector on this board but you will have to add a separate wire from R6 to attach INIT. Please check the proper use of the pull-up resistors for your specific board.

Only the configuration port has to be connected to the FPGA even in case the data mode is not required and only one single configuration sequence should be applied. The core will automatically start configuration as start_i and mode_i inputs are pulled high. When configuration has finished (FPGA sets cfg_done), the core will remain in the idle state because there is no further low-to-high transition at start_i.

Compatability

These cards have been tested with the SD/MMC Bootloader:

- Hama 64 MB SD
- SanDisk 128 MB SD
- SanDisk 64 MB MMC
- Panasonic 32 MB SD

Some MMCs might fail with this core as not all cards support CMD18 (READ_MULTIPLE_BLOCK). Please consult the data sheet of your specific model. In case your MMC does not implement CMD18 you might want to have a look at the FPGA MMC-Card Config project at

http://www.opencores.org/projects.cgi/web/mmcfpgaconfig/overview/

5

IO Ports

The following Table 4 shows the primary IO ports of spi_boot.

Port	Width	Dir	Description
clk_i	1	In	Clock input
reset_i	1	In	Reset input – active level selected via generic
set_sel_i	1 – s	In	Set selection input
spi_clk_o	1	Out	SPI clock output
spi_cs_n_o	1	Out	SPI chip select – low active
spi_data_in_i	1	In	SPI data from card
spi_data_out_o	1	Out	SPI data to card
spi_en_outs_o	1	Out	Tristate driver enable for SPI outputs
start_i	1	In	Start trigger
mode_i	1	In	Mode selection
config_n_o	1	Out	Begin configuration
cfg_init_n_i	1	In	Configuration init handshake – low active
cfg_done_i	1	In	Configuration done
dat_done_i	1	In	Data transfer done
cfg_clk_o	1	Out	Configuration clock output
cfg_dat_o	1	Out	Configuration data

Table 4: List of IO ports

Table 5 lists the generic parameters of the core.

Generic	Value	Description
width_set_sel	1 – p	Width of set selection, 2^p = s sets available
width_bit_cnt	6 – 12	Width of bit counter
width_img_cnt	0 – m	Width of image counter, 2^m = i images available
num_bits_per_img	0 – 31	Number of bits required to address one image
sd_init	0, 1	SD specific initialization command 1 : use ACMD41 0 : do not use ACMD41
mmc_compat_clk_div	0 – n	Maximum count for MMC compatibility counter 0 : do not implement MMC compatibility counter
width_mmc_clk_div	0 – o	Width of MMC compatibility counter
reset_level	0, 1	Active level of reset_i

Table 5: List of generic parameters