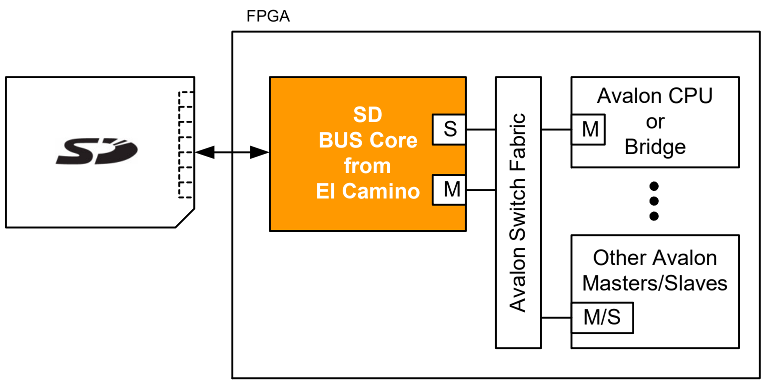
SD BUS Core

with Avalon Interface

General Description

The SD BUS Core with Avalon Interface allows for Platform Designer or Qsys systems to access standard SD, MMC or eMMC flash based memory devices. It comes with low-level SD Card driver routines for Nios II and is integrated into the HAL generic device model classes as a FLASH memory device. Therefore you do not need to write any additional low level code to read or write raw data from or to SD cards.
El Camino offers an optional stand-alone FAT12/16/32 file system that can be used to read or write files on SD cards from a NIOS system. Furhtermore the core is implemented such that it works with the standard sdhci/sdhci-pltfm Linux drivers e.g. on Altera SoC devices.

Figure 1: Block Diagram



Features

- Supports Secure Digital Card (SD, SDHC, SDXC), Multimedia Card (MMC) and embedded Multimedia Card (eMMC)
- 1 bit and 4 bit (wide bus) operation
- Compatible with SD Host Controller Standard Specification V3.01
- Supports High Speed Mode (SDHS) with up to 50 MHz SD Clock rate
- Low-level Nios II drivers included
- Optional stand-alone FAT12/FAT16/FAT32 file system available
- Compatible with Linux sdhci/sdhci-pltfm drivers
- DMA support for high data throughput

Applications

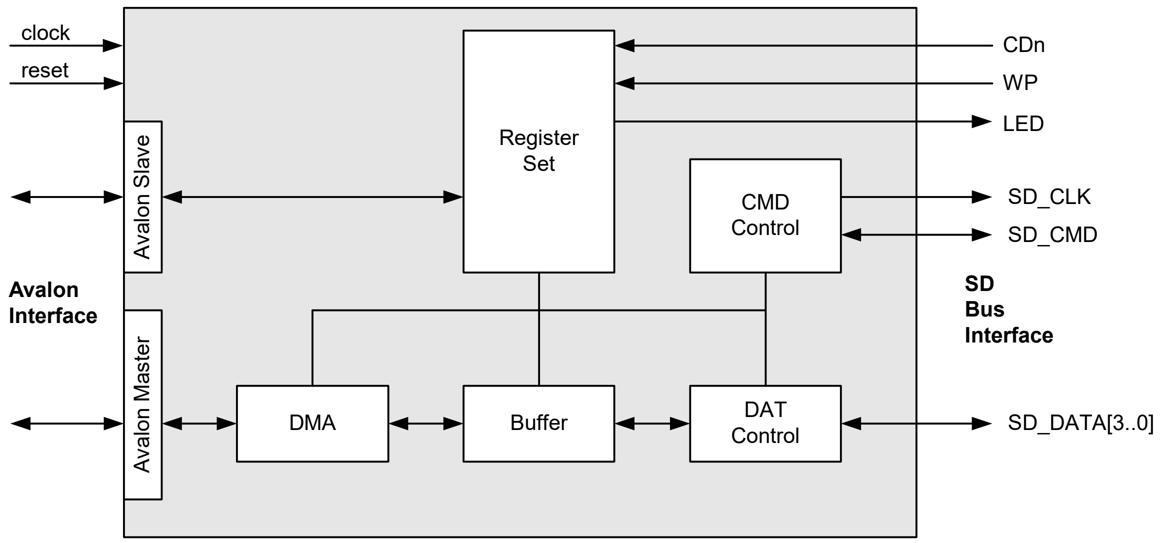
The SD Bus Core is ideal for applications where a mobile, standard and exchangeable storage media is required for NIOS II or SoC applications. Together with our Windows utility it is easy to exchange raw data between a NIOS II application and the PC platform.
When used together with the El Camino SD/MMC loader, FPGA configuration data can be combined with application data or program storage on a removable, common and compact storage media.

Deliverables

- Platform Designer / Qsys Compliant IP core in Verilog
- Low-level Nios II software drivers for initialization, read and write access
- Windows Utility for reading and writing raw data (on request)

Architecture Specification

Figure 2: SD/MMC Bus Core Block Diagram



The SD Bus core has the following interfaces:

- Avalon Interface
  - Control Signals
    clock and reset signals driven from the Avalon switch fabric and interrupt signal driven to the Avalon switch fabric

  - Avalon Slave Interface
    read and write access to the core registers and the data buffer for non-DMA data transfers
  - Avalon Master Interface
    connection to a DMA controler inside the core for SDMA (Single operation DMA) support as specified in the SD Host Controller Standard Specification

- SD Bus Interface
  - CDn
  - WP
  - LED
  - SD_CLK
  - SD_CMD
    (optional unidirectional signals: _i/_o/_en)
  - SD_DAT3...SD_DAT0
    (optional unidirectional signals: _i/_o/_en)

The registers provide an interface to the SD Bus core and are visible via the Avalon slave port. The SD_CLK, SD_CMD, SD_DAT3-SD_DAT0, CDn and WP ports provide the hardware interface to the SD card.
The core logic is synchronous to the clock input provided by the Avalon interface. The Avalon clock is divided to generate the SD_CLK output.

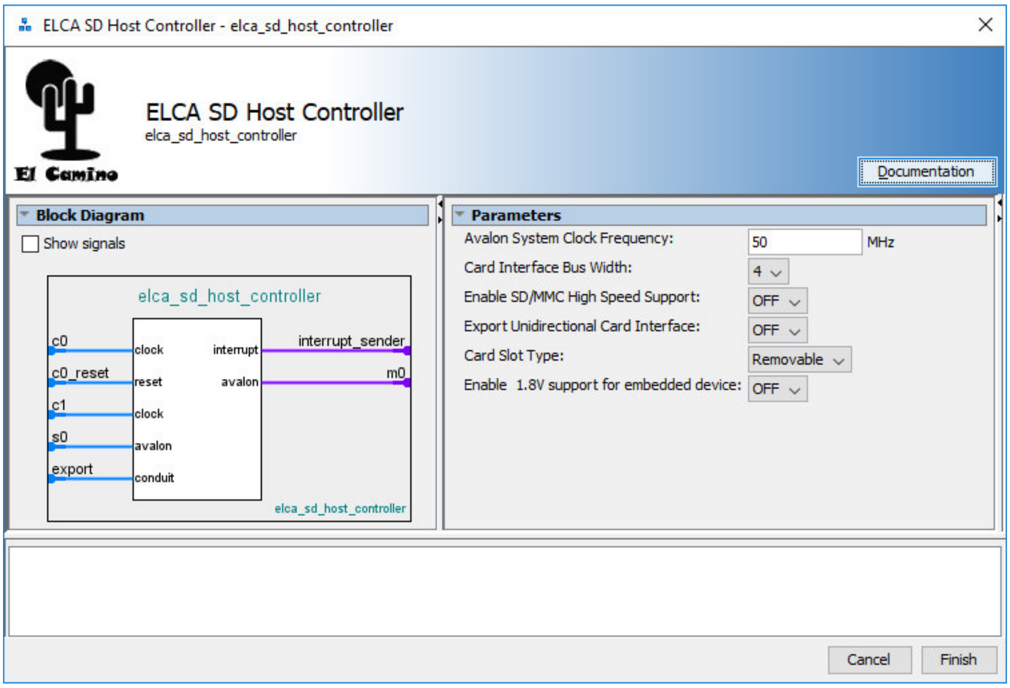
Table 1: Port Description

Port	Direction	Function	Connect to
SD_CLK	output	Clock Signal to the SD/MMC card. This clock is derived from the system clock by a parameterizable clock divider. The frequency determines the data rate and is set automatically by the sofware driver	SD card pin 5
SD_CMD	input/output	Bidirectional Command/Response Signal Optional Unidirectional Signals for implementing external tri-state drivers SD_CMD_o, SD_CMD_i, SD_CMD_en (enable when high)	SD card pin 2
SD_DAT[3..0]	input/output	Bidirectional Data signals Optional unidirectional signals for implementing external tri-state drivers SD_DAT_o[3..0], SD_DAT_i[3..0], SD_DAT_en[3..0] (enable when high)	SD card SD_DAT3 - pin 1 SD_DAT2 - pin 9 SD_DAT1 - pin 8 SD_DAT0 - pin 7
CDn	input	Card Detect signal form the SD card socket logic 0 -> card is present logic 1 -> socket empty	SD card connector card detect switch
WP	input	Write Protect signal from the SD card socekt logic 0 -> card is not write protected logic 1 -> card is write protected	SD card connector write protect switch
Common Avalon Control Signals			
csi_c0_clk	input	Avalon clock signal	automatically connected by Platform Designer (Qsys)
csi_c1_clk_sd_clk	input	Clock signal driving only the flipflop that generates the SD_CLK output. This clock can be the same as csi_c0_clk or shifted forward a little (switches earlier). When this clock is shifted forward it is easier to meet SD card timing with newer device families like Cyclone 10 GX, Arria 10 or Stratix 10.	
csi_c0_reset_n	input	Avalon reset signal - low adctive	
ins_i0_irq	output	Avalon interrupt signal	
Avalon MM slave interface			
avs_s1_writedata[31..0]	input	Avalon write data bus	automatically connected by Platform Designer (Qsys)
avs_s1_readdata[31..0]	output	Avalon read data bus	
avs_s1_address[5..0]	input	Avalon address bus	
avs_s1_byteenable_n[3..0]	input	Avalon byteenable signals - active low	
avs_s1_chipselect	input	Avalon chipselect signal	
avs_s1_read_n	input	Avalon read signal - active low	
avs_s1_write_n	input	Avalon write signal - active low	
Avalon MM master interface			
avm_m1_writedata[31..0]	output	Avalon write data bus	automatically connected by Platform Designer (Qsys)
avm_m1_readdata[31..0]	input	Avalon read data bus	
avm_m1_address[31..0]	output	Avalon address bus	
avm_m1_byteenable_n[3..0]	output	Avalon byteenable signals - active low	
avm_m1_read_n	output	Avalon read signal - active low	
avm_m1_write_n	output	Avalon write signal - active low	
avm_m1_waitrequest_n	input	Avalon wait request signal - active low	

Table 2: Qsys Component Settings

Parameter	Legal Values		Radix	Description
Avalon System Clock Frequency (SYSTEM_CLOCK_FREQUENCY)	1-512		Integer MHz	This parameter is used to set the frequency of the Avalon clock (csi_c0_clk) driven from the Avalon System Interconnect Fabric (SIF) into the core. The parameter needs to match the actual frequency of the clock connected to the core in Qsys. The setting is used to pass the system clock frequency of the core to the software and calculate the neccessary clock divider.
Card Interface Bus Width (SD_BUS_WIDTH)	1 4	1 bit mode only 4 bit mode	Integer	This parameter is used to set the maximum SD Bus width used by the core and the software driver. Legal values for this parameter are 1 or 4. If set to 1, only SD_DAT[0] will be used for communication with the SD card. SD_DAT[3..1] will still be present and can be left unconnected. A setting of 1 can be used for example with the Altera Nios Embedded Evaluation Kit (NEEK) which has only SD_DAT[0] connected.
Enable SD/MMC High Speed Support (SD_HS_SUPPORT)	OFF (0) ON (1)	high speed off high speed on	Integer	This parameter is used to turn on or off high speed mode. High speed mode uses up to 50 MHz clock rates and requires careful routing of the SD signals. Not every hardware implementation will support high speed mode, which is why automatic switching to high speed mode can be turned off here.
Export Unidirectional Card Interface (UNIEN)	OFF (0) ON (1)	bidirectional unidirectional	Integer	This parameter allows to export separate input, output and enable signals for CMD and DAT. With these signals external tri-state drivers can be implemented.
Card Sot Type:	Removable Embedded	0 1	Integer	This affects the „Slot Type“ bits in the capabilities register that can be read by the software driver.
Enable 1.8V support for embedded device	OFF (0) ON (1)			This affects the „Voltage Support 1.8V“ bit in the capabilities register that can be read by the software driver.

Figure 3: Platform Designer (Qsys) Component GUI

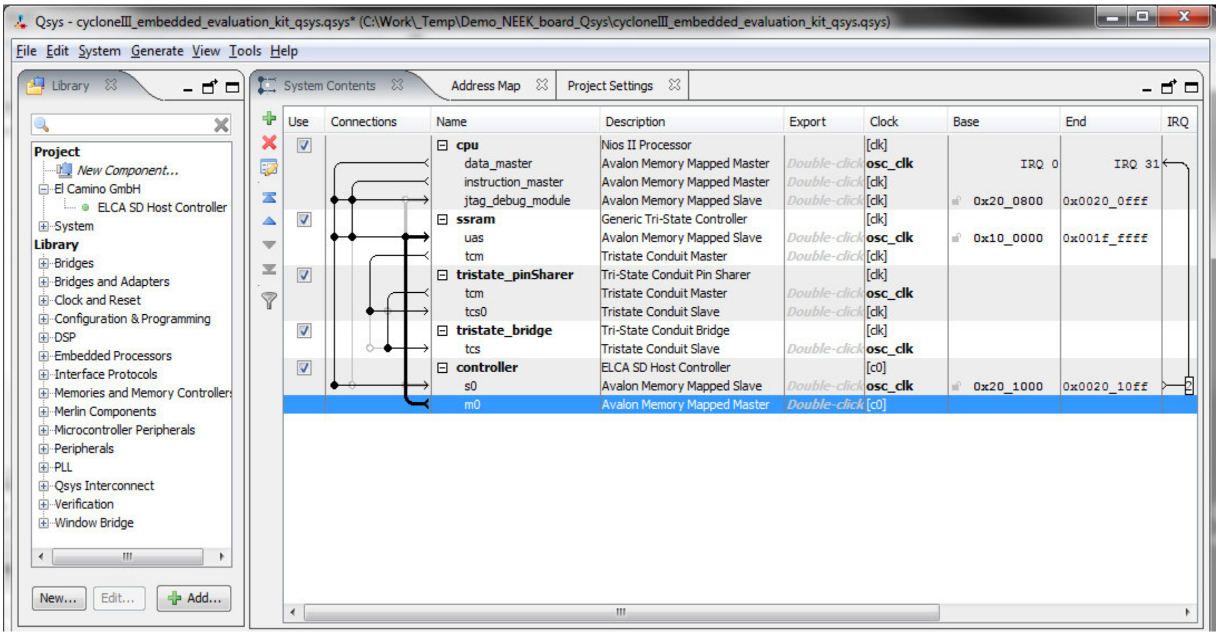


Integration into Qsys System

The following picture shows a typical Platform Designer (Qsys) system with a NIOS processor and an SD Bus core. The master (DMA) interface connections of the SD Bus core is highlighted.

The Avalon MM Slave port should be connected to the CPU, so that it can access the registers in the core.
The Avalon MM Master port needs to be connected to the memory, that holds the read and write buffers, passed to the HAL flash API.

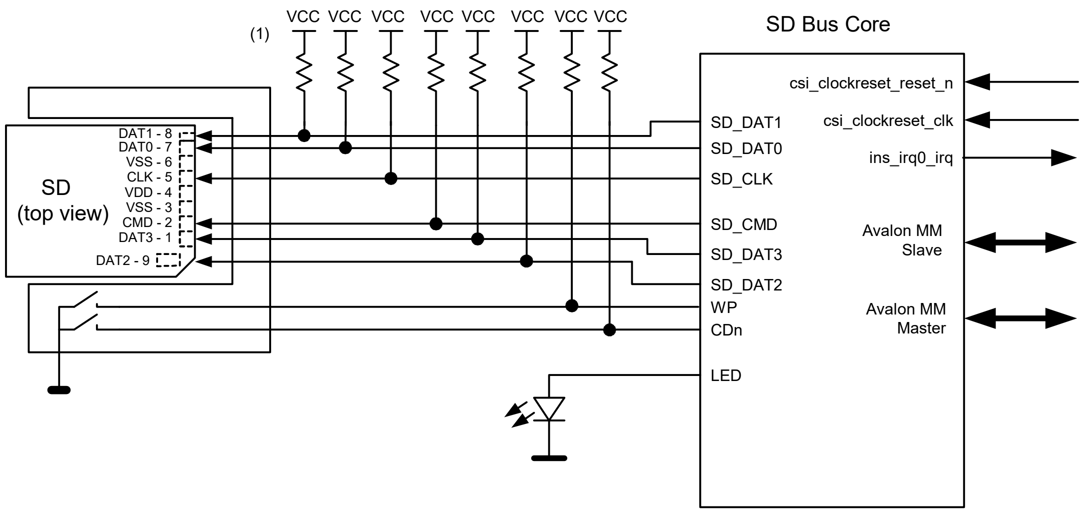
Figure 4: Typical Qsys system featuring a NIOS II CPU and the SD Bus Core



Connecting the Core to an SD Card

The following picture shows how to connect the SD Bus Core to an SD card. If the SD_BUS_WIDTH parameter is set to 4 the software driver will use the 4-bit mode and all four data signals (SD_DAT[3..0]) need to be connected.

Figure 5: Connecting the SD Bus Core



(1) Recommended Pull-up resistors are 10kOhm

Register Model

An Avalon master peripheral controls and communicates with the SD Bus core via registers, devided into 9 parts as listed below:

Figure 6: Coarse Register Map

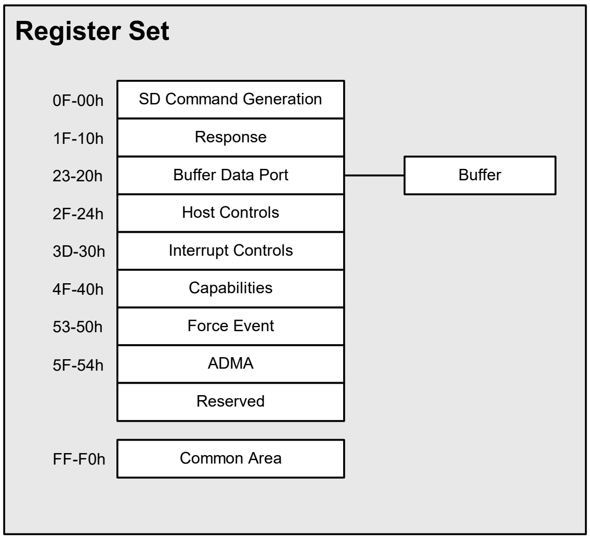


Table 3: Fine Register Map for SD Bus Core

Offset	15..8	7..0	Offset	15..8	7..0
002h	SDMA System Address (High)		000h	SDMA System Address (Low)	
006h	Block Count		004h	Block Size	
00Ah	Argument1		008h	Argument0	
00Eh	Command		00Ch	Transfer Mode	
012h	Response1		010h	Response0	
016h	Response3		014h	Response2	
01Ah	Response5		018h	Response4	
01Eh	Response7		01Ch	Response6	
022h	Buffer Data Port1		020h	Buffer Data Port0	
026h	Present State		024h	Present State	
02Ah	Wakeup Control	Block Gap Control	028h	Power Control	Host Control
02Eh	Software Reset	Timeout Control	02Ch	Clock Control	
032h	Error Interrupt Status		030h	Normal Interrupt Status	
036h	Error Interrupt Status Enable		034h	Normal Interrupt Status Enable	
03Ah	Error Interrupt Signal Enable		038h	Normal Interrupt Signal Enable	
03Eh	---		03Ch	Auto CMD12 Error Status	
042h	Capabilities		040h	Capabilities	
046h	Reserved		044h	Reserved	
04Ah	Maximum Current Capabilities		048h	Maximum Current Capabilities	
04Eh	Reserved		04Ch	Reserved	
052h	Force Event for Error Interrupt Status		050h	Force Event for Auto CMD12 Error Status	
	---		054h	---	ADMA Error Status (1)
05Ah	ADMA System Address [31..16] (1)		058h	ADMA System Address [15..0] (1)	
05Eh	ADMA System Address [63..48] (1)		05Ch	ADMA System Address [47..32] (1)	
	---			---	
0F2h	---		0F0h	---	
	---			---	
0FEh	Host Controller Version		0FCh	Slot Interrupt Status	

white areas not implemented in current version of the core

Register Description

Please see the SD Host Controller Standard Specification, Version 3.00 for a detailed register description

Resource Utilization and Performance

The following results are based on synthesis and place & route in Quartus II Version 13.1. The maximum frequency of the data transfer is limited by both the maximum system frequency of the SOPC builder block and the maximum SD_CLK frequency of the SD card.

Table 4: Resources

Device Family	Ressource		Maximum System Clock
	Logic	Memory in Bits	
Cyclone V	853 ALMs	8192	170 MHz
Arria V	848 ALMs	8192	227 MHz
Stratix V	857 ALMs	8192	420 MHz
MAX 10	2063 LEs	8192	168 MHz

Read/Write performance as shown in the following table will largely depend on the type of SD card used.

Table 5: Performance

Platform	NIOS CPU	Avalon System Clock	Low Level Read	Low Level Write	FAT 32 Read	FAT 32 Write
BeMicro CV Cyclone V FPGA Development Kit Pretec 1GB industrial SD card	fast	100 MHz	21.98 MB/s	11.22 MB/s	17.72 MB/s	9.33 MB/s

Timing Requirements

In order to support high speed mode (up to 50 MHz SD clock rate) the following I/O timing requirements have to be met on SD interface signals.

1. tco of SD_CMD, SD_DAT[3..0] has to be at least 2 ns greater than tco of SD_CLK
2. tco of SD_CLK +
   board delay of SD_CLK +
   maximum board delay of either SD_CMD or SD_DAT[3..0] +
   maximum tsu of either SD_CMD or SD_DAT[3..0]
   has to be smaller than 6 ns

It is up to the user, to either turn of High Speed support or implement the necessary timing constraints and ensure they are met.

Nios II Software Support

Basic Concept

The SD Bus peripheral is integrated into the HAL generic device model classes as a FLASH memory device.

The HAL provides a generic device model for nonvolatile flash memory devices such as SD cards. The HAL API provides functions to write data to flash. For example, you can use these functions to implement a SD based filing subsystem.
Although it is not necessary for general, parallel FLASH devices, the HAL API also provides functions to read flash. For most flash devices, programs can treat the flash memory space as simple memory when reading, and do not need to call special HAL API functions. If the flash device has a special protocol for reading data, just like SD cards, you must use the HAL API to both read and write data.

The following two NIOS II HAL APIs provide a different level of access to the flash:

- Simple flash access
  a simple API for writing buffers into flash and reading them back, which in general does not preserve the prior contents of other flash erase blocks.

- Fine-grained flash access
  finer-grained functions for programs that need control over writing or erasing individual blocks. This functionality is generally required for managing a file subsystem.

With SD cards you can write single bytes without the need to erase whole blocks or sectors. So even when using simple flash access all data within a sector or block is preserved even when not writing the complete sector or block. Only for compatibility reasons with other flash devices it might make sense to use fine-grained flash access even with SD cards.

The API functions for flash devices are defined in sys/alt_flash.h.

☞ Since the SD Bus core uses DMA to transfer data to and from SD cards it is important to consider cache/memory coherence when using a NIOS CPU that features a data cache. It is therefor strongly recommended to allocated the data buffers for read and write functions in uncached memory or to bypass the cache (by setting address bit 31 to ’1’) whenever accessing the read or write buffers from the CPU.
A flush of the data cache before calling the flash HAL functions is not sufficient as a single cache line may overlap with variables used inside the driver and the read- or write buffers!

Simple Flash Access

This interface comprises:
alt_flash_open_dev(), alt_write_flash(), alt_read_flash(), and alt_flash_close_dev().
Writing and reading can start at any address and can be of any length as long as one stays within the boundarys of the SD card. For maximum compatibility with older SD cards however it is recommneded to start writing only on block boundaris. The block size can be determined with the „alt_get_flash_info()“ function. The typical block size for most SD cards is 512 bytes.
The code “Example: Using the Simple Flash API Functions” in the Nios II Software Developer's Handbook shows the usage of all of these functions in one code example.

Fine Grained Flash Access

There are three additional functions that provide complete control over writing flash contents at the highest granularity:
alt_get_flash_info(), alt_erase_flash_block(), and alt_write_flash_block().
These functions are implemented for compatibility reasons with other flash devices however are not necessary when accessing SD cards.

alt_get_flash_info() reports the region_size and the block_size in kBytes
The size of the SD card is equivalent to the region_size * 1024 or the block_size * 1024 * number_of_blocks.

Linux Software Support

This core is compatible with the Linux sdhci/sdhci-pltfm drivers. It has been tested on an Altera Cyclone V SoC development kit.

The Altera SoC device tree flow, will automatically add the necessary parameters to the device tree.

For the sdhci-pltfm driver to work with the device tree flow, the following patch from Alistair Pobble is required:
[PATCH 1/5] SDHCI: Add a generic registration to the SDHCI platform driver

Command Line Tool

The utility „sd_rd_wr“ allows to read and write raw data from and to SD cards. The tool provides its own documentation in the form of a help page, accessible from the command line. To view the help, open a DOS Command Shell, and type the following command:
sd_rd_wr -h

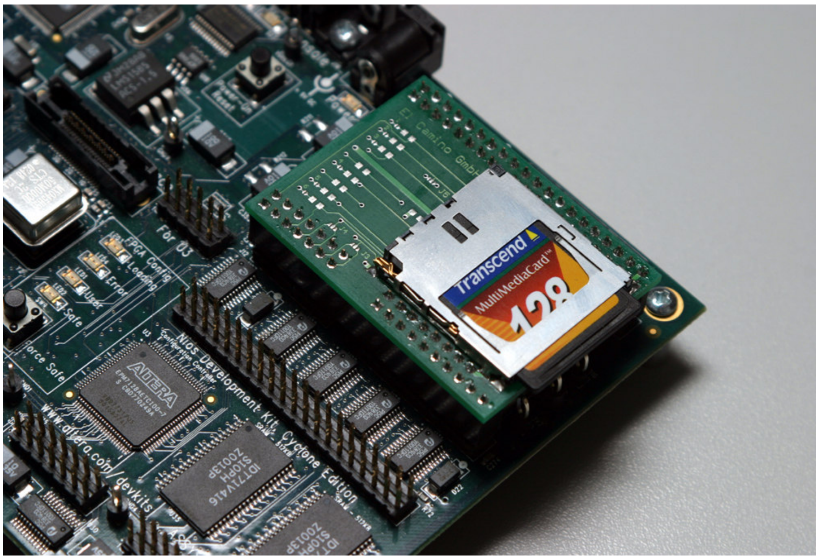
Usage : sd_rd_wr x: file.dat [-w] [-o:offset] [-b:bytes] [-h]

Table 6: sd_rd_wr utility command line options

Paramter	Default when ommitted		Description
	reading	writing	
x:	always required	always required	drive letter of SD card
file.dat	always required	always required	source or desitination file name
-w	default		write data from file.dat to SD card when omitted: read data from SD card and store in file.dat
-o:offset	0	0	decimal byte offset from address 0 of SD card
-b:bytes	512	size of file.dat	number of bytes to read or write
-h			display on-line help

Prototyping Hardware

Figure 7: Prototyping Hardware



The IP core can be ordered with a small prototyping board that can be used together with El Camino or Altera NIOS boards that feature a Santa Cruz prototyping header. The schematics of the prototyping hardware

can be found at the end of this document. The SPI mode chip select, as well as the card detect and write protect switch signals can each be routed to one of two connector pins. This allows to cascade up to two SD prototyping boards. In such a cascaded configuration some signals need to be routed individually. Soldering bridges allow to put individual prototyping boards into an A or B configuration and de-activate the pull-up resistors on one of them. On a request basis, the prototyping boards can be outfitted with connectors that support cascading.
The following table lists the signal mapping.

Table 7: J2B Pin Descriptions

J2B Connector Pin	Schematic Signal	Connect to IP Core Signal	SD/MMC Pin	Function - SPI Bus Mode	Function - SD Bus Mode
3	SD_SD1	SD_DAT[2]	9	Reserved	Data Line [Bit 2]
4	SD_CMD	SD_CMD	2	Host to Card Commands and Data	Command/Response
5	SD_D1	SD_DAT[1]	8	Reserved	Data Line [Bit 1]
6	SD_CLK	SD_CLK	5	Clock	Clock
7	SD_SWWP_A	WP (default)			
8	SD_DAT	SD_DAT[0]	7	Card to Host Data and Status	Data Line [Bit 0]
9	SD_SWWP_B				
11	SD_SWCI_A	CDn (default)			
12	SD_D3CS_A	SD_DAT[3] (default)	1	Chip Select (Active Low)	Card Detect/Data Line [Bit 3]
13	SD_SWCI_B				
14	SD_D3CS_B		1	Chip Select (Active Low)	Card Detect/Data Line [Bit 3]

The SD card connector on older Altera Nios II Embedded Evaluation Kit (NEEK) has only one data line connected (SD_DAT[0]). These NEEKs therefor only supports the 1-bit mode of this core and is limited to about one fourth of the maximum data throughput performance. Because of the buffers in the SD signal paths the timing for high speed mode cannot be met on the NEEK.

Custom Solutions

Please contact El Camino if you require any custom solutions based on this IP.

Notes:

El Camino GmbH
Landshuter Str. 1
84048 Mainburg
Germany
Telefone +49-8751-8787-0
Telefax +49-8751-8787-10
E-mail: info@elca.de
http://www.elcamino.de