



Technical notes on using Analog Devices DSPs, processors and development tools
Visit our Web resources <http://www.analog.com/ee-notes> and <http://www.analog.com/processors> or
e-mail processor.support@analog.com or processor.tools.support@analog.com for technical support.

Interfacing SD Cards with Blackfin® Processors

Contributed by Aseem Vasudev and Jayanti Addepalli

Rev 1 – February 19, 2010

Introduction

Secure Digital™ (SD) memory cards are popular storage devices that see application in a wide range of modern consumer electronic products. This EE-Note details their interface with ADSP-BF52x Blackfin® processors. The approach described in this document also applies to other Blackfin processors. Certain Blackfin processors have on-chip SD support. Refer to the processor selection guide to find out which Blackfin processors support SD natively. The Blackfin processor selection table is available at:

http://www.analog.com/en/embedded-processing-dsp/blackfin/processors/selection-tables/Blackfin_Selection_Table/resources/fca.html

Today, SD cards are very commonly used in products such as mobile phones, PDAs, smart phones, digital cameras, digital recorders, music and video players, pagers, electronic toys, and organizers. These flash-based memory cards are designed to meet the security, capacity, performance, and environment requirements inherent in these newly emerging devices. Their popularity stems from features such as high mobility, high performance at a low price, low power consumption, and high data throughput at the memory card interface. Matsushita, SanDisk, and Toshiba developed the SD card format for portable devices. The card includes a copyright mechanism that complies with the security of the SDMI standard.

SD cards support two modes of operation – a serial mode that is compatible with the Serial Peripheral Interface (SPI), and a parallel mode commonly referred to as SD mode. SPI mode is advantageous in that the cards can be interfaced seamlessly with many controllers or digital signal processors (DSPs) that have on-chip SPI support; but supports a limited set of features. SD mode must have a dedicated controller (or a complex interface); but supports the complete feature set of these cards. This EE-Note deals with various mechanisms employed to implement the SD card interface with a Blackfin processor such as the serial mode over the SPI port, and the parallel mode through different approaches, two of them on the Blackfin Parallel Peripheral Interface (PPI) and one on the Blackfin Asynchronous Memory Interface (AMI).

The following sections of this document detail the fundamentals, command format, and interface. Application code in the .ZIP file supplied with this EE-Note demonstrates how these cards can be interfaced seamlessly with Blackfin processors. The software implements various commands used to configure and operate the cards. Code examples have been supplied for operation in all the modes.

SD Memory Card Fundamentals

SD cards are highly integrated flash memories with serial and random access capability. They are accessible via a dedicated serial/parallel interface optimized for fast and reliable data transmission. This interface allows several cards to be stacked by connecting their peripheral contacts. [Figure 1](#) shows some of these cards in different form factors.



Figure 1. SD memory cards

The host controls any communication between the cards and itself. The SD memory card system defines two alternative communication protocols: SPI and SD. The host system can choose either of the modes. The card detects the mode (SPI or SD) requested by the host when the reset command is received and expects all further communication to be in the same mode.

The following two operation modes are defined for the SD memory card system (host and cards):

- Card identification mode: The host is in card identification mode after reset and while it is looking for new cards on the bus. Cards are in this mode after reset until the `SEND_RCA` command (`CMD3`) is received.
- Data transfer mode: Cards enter data transfer mode after their `RCA` is first published. The host enters data transfer mode after identifying all the cards on the bus.

Features

The SD card has a 9-pin interface with four data lines, and the micro-SD card supports an 8-pin interface with four data lines.

The maximum operating clock frequency can go up to 25 MHz (in the default mode) or up to 50 MHz (in high-speed mode). The data written in any of these modes can be read by a host in either mode. The operating voltage ranges from 2.7 V to 3.6 V, and the maximum capacity ranges from 4 MB to the gigabyte range. The voltage range that is supported for the initialization/identification process is defined in the `CSD` register.

Internal Blocks

All the units in an SD card are clocked by the internal clock generator. The interface driver unit synchronizes the data (`DAT0-DATx`) and `CMD` signals from the external `CLK` to the internally used clock signal.

For the identification of a single SD card in a stack of cards, a card identification register (`CID`) and a relative card address register (`RCA`) are used. The card-specific data register (`CSD`) contains different types of operation parameters.

The card has its own power-on detection unit. It is protected against short circuit during insertion and removal while the SD card system is powered up. The programming voltage is generated on the card.

Interface Signals

The basic SD card concept is based on transferring data via a minimal number of signals. The communication signals for both the modes are given below.

SD Mode

- **CLK:** Host to card clock. With each cycle of this signal, a one-bit transfer on the command and data lines is done. The frequency may vary between zero and the maximum clock frequency.
- **CMD:** Bi-directional command/response signal. This bi-directional signal is used for card initialization and data transfer commands. The **CMD** signal has two operation modes: open-drain for initialization mode, and push-pull for fast command transfer. Commands are sent from the SD bus master to the card, and responses are returned from the cards to the host.
- **DAT0 – DATx (x = 0, 3, or 7):** Bi-directional data channels. The **DAT** signal operates in push-pull mode. Only one card or the host is driving this signal at a time.

SPI Mode

- **CS/:** Host-to-card chip select signal. An active low signal that selects a particular SD card.
- **CLK:** Host-to-card clock signal.
- **MOSI:** (Master Out Slave In) Host-to-card single-bit data signal
- **MISO:** (Master In Slave Out) Card-to-host single-bit data signal

Pin Description

The pin descriptions for a 9-pin interface SD memory card is shown in [Table 1](#).

Pin	Name	Description
1	DAT3, CD, or CS/	Data line 3 (DAT3) – MSB: SD mode, also used as card detect signal; chip select (CS/): SPI mode
2	CMD or MOSI	Command (CMD): SD mode; Master Out Slave In (MOSI): SPI mode
3	VSS	Ground
4	VDD	Power supply
5	CLK	External clock
6	VSS	Ground
7	DAT0 or MISO	Data line 0 (DAT0) – LSB: SD mode; Master In Slave Out (MISO): SPI mode
8	DAT1	Data line 1 (DAT1)
9	DAT2	Data line 2 (DAT2)

Table 1. Pin descriptions for an SD card

The extended data lines (DAT1 – DAT3) are input on power-up. They start operating as DAT lines after an issue of the `SET_BUS_WIDTH` command (ACMD6). Furthermore, CD line is input on power-up with a 50-K Ω pull-up resistor (can be used for card detection or SPI mode selection). The pull-up should be disconnected by the user, during regular data transfer, with the `SET_CLR_CARD_DETECT` command (ACMD42).

Registers

Eight registers are defined for the SD card: CID, RCA, DSR, CSD, SCR, OCR, SSR, and CSR. Some of them are described in Table 2:

Name	Width	Description
CID	128	Card identification number
RCA	16	Relative card address; local system address of a card, dynamically suggested by the card and approved by the host during initialization
CSD	128	Card-specific data; information about the card's operating conditions
OCR	32	Operation conditions register
CSR	32	Card status register

Table 2. SD card registers

Interface with Blackfin Processors: SPI Mode

The SPI implementation uses a subset of the SD memory card protocol and command set. The advantage of this mode is the capability of using an off-the-shelf host. The disadvantage is the loss of performance of the SPI mode versus SD mode (for example, single data line and hardware CS/ signal per card).

Serial Peripheral Interface (SPI) of the Blackfin Processor

The Blackfin processor's SPI is a four-wire interface consisting of two data pins (MOSI and MISO), a device select pin (SPISS/), and a gated clock pin (SCK). SPI is a full-duplex synchronous serial interface, supporting master modes, slave modes, and multi-master environments. The SPI-compatible peripheral implementation also supports programmable baud rate and clock phase/polarities.

The interface is essentially a shift register that serially transmits and receives data bits, one bit at a time at the SCK rate, to and from other SPI devices. SPI data is transmitted and received at the same time through the use of a shift register. The SCK synchronizes the shifting and sampling of the data on the two serial data pins.

The SPI can be configured as an SPI master port (generates SCK and SPISS/ signals) or can be configured as an SPI slave port (receives SCK and slave select signals externally). When the SPI port is configured as a master, it drives data on MOSI pin and receives data on MISO pin. It drives the slave select signals for SPI slave devices and provides serial bit clock SCK. The Blackfin processor's SPI supports four functional modes by using combinations provided by CPOL (clock polarity) and CPHA (clock phase) bits. In a multimaster or multislave SPI system, the data output pins (MOSI and MISO) can be configured to behave as open drain outputs, which prevent contention and possible damage to pin drivers. The WOM bit controls this option. For detailed information on the Blackfin SPI port, refer to the *ADSP-BF52x Blackfin Processor Hardware Reference Manual* ^[7].

Fundamentals – Protocol, Commands, and Responses

The SPI interface is selected during the first reset command after power up (CMD0) and cannot be changed once the part is powered on.

The SPI channel is byte-oriented. Every command or data block is 8 bits long and is byte-aligned to the \overline{CS} signal (that is, the length is a multiple of eight clock cycles). The card starts to count SPI clock cycles at the assertion of the \overline{CS} signal. Every command or data token is aligned to the 8-clock cycle boundary. The SPI messages consist of command, response, and data block tokens.

The host starts every bus transaction by asserting the \overline{CS} signal low. The selected card always responds to the command. When the card encounters a data retrieval problem in a read operation, it will respond with an error response (which replaces the expected data block). Additionally, every data block sent to the card during write operations will be responded with a data response token.

For a standard-capacity memory card, a data block can be as big as one card write block or as small as a single byte. Partial block read/write operations are enabled by card options specified in the CSD register.

Every SD card command transferred on the bus is protected by CRC bits. In SPI mode, the SD memory card offers a CRC ON mode, which enables systems built with reliable data links to exclude the hardware or firmware required for implementing the CRC generation and verification functions. In CRC OFF mode, the CRC bits of the command are defined as “don’t care” for the transmitter and are ignored by the receiver.

The SPI interface is initialized in the CRC OFF mode in default. However, the RESET command (CMD0) used to switch the card to SPI mode is received by the card while in SD mode and, therefore, must have a valid CRC field. Since CMD0 has no arguments, the content of all the fields, including the CRC field, are constants and need not be calculated during run time. A valid reset command is: 0x40, 0x0, 0x0, 0x0, 0x0, and 0x95. After the card is put into SPI mode, a CRC check for all commands including CMD0 will be performed according to CMD59 (CRC_ON_OFF) setting.

Command Format

Commands are six bytes long. The command transmission always starts with the left-most bit (MSB) of the bit string corresponding to the command codeword. The format is as follows:

Bit Position	47	46	[45:40]	[39:8]	[7:1]	0
Width (bits)	1	1	6	32	7	1
Value	‘0’	‘1’	X	X	X	‘1’
Description	Start bit	Transmission bit	Command index	Argument	CRC7	End bit

Table 3. Command format – SPI mode

Response Format

Several types of response tokens (R1, R1b, R2, R3 and R7) are fixed for each command. The response is transmitted as MSB first by the card. The R1 response is discussed here, since it is most commonly used.

The R1 response token is sent by the card after every command, with the exception of SEND_STATUS command. It is one byte long, and the MSB is always set to zero. The other bits are error indications; an error being signaled by a 1. The structure of the R1 format is shown here. The meaning of the flags is defined as follows:

- **In idle state:** The card is in idle state and running the initializing process.
- **Erase reset:** An erase sequence was cleared before executing, because an out-of-erase sequence command was received.

- **Illegal command:** An illegal command code was detected.
- **Communication CRC error:** The CRC check of the last command failed.
- **Erase sequence error:** An error in the sequence of erase commands occurred.
- **Address error:** A misaligned address that did not match the block length was used in the command.
- **Parameter error:** The command's argument (for example, address or block length) was outside the allowed range for this card.

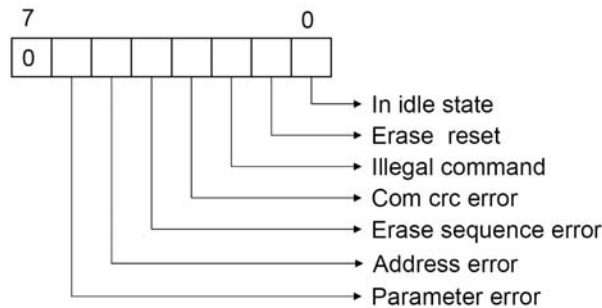


Figure 2. R1 response format – SPI mode

Read and Write

Single block read and multiple block read operations (CMD17 or CMD18) are supported. Upon reception of a valid read command, the card will respond with a response token followed by a data token. For standard-capacity cards, the size of the data token is determined by the block length set by SET_BLOCKLEN (CMD16). For high-capacity cards, the data size in the data token for is fixed to 512 bytes, regardless of the block length set by CMD16.

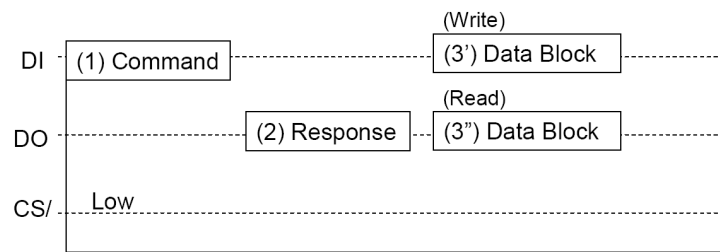


Figure 3. Data read and write – SPI mode

Single block and multiple block write commands are supported. Upon reception of a valid write command (CMD24 or CMD25), the card will respond with a response token and will wait for a data block to be sent from the host.

All data bytes are transferred MSB first. The data transfers associated with read and write commands (data tokens) are 4 to 515 bytes long, and have the following format (in the case of single block transfers):

- First byte (Start Block): 1111 1110
- Bytes 2 to 513 (depends on the data block length): User Data
- Last two bytes: 16 bit CRC

Every data block written to the card is acknowledged by a data response token, as shown below:

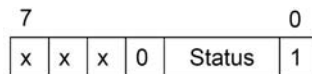


Figure 4. Data response token – SPI mode

It is one byte long and contains three bits to indicate the status of the write. The meaning of the bits is as follows:

010 – Data accepted

101 – Data rejected due to a CRC error

110 – Data rejected due to a write error

If a read operation fails and the card cannot provide the required data, it will send a data error token instead. This token is one byte long and is formatted as follows:

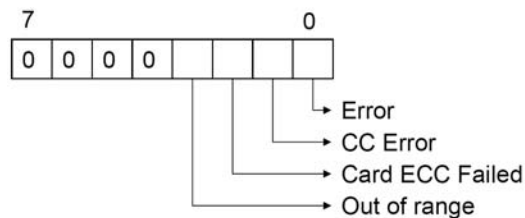


Figure 5. Data error token – SPI mode

Hardware Interface

Block Diagram

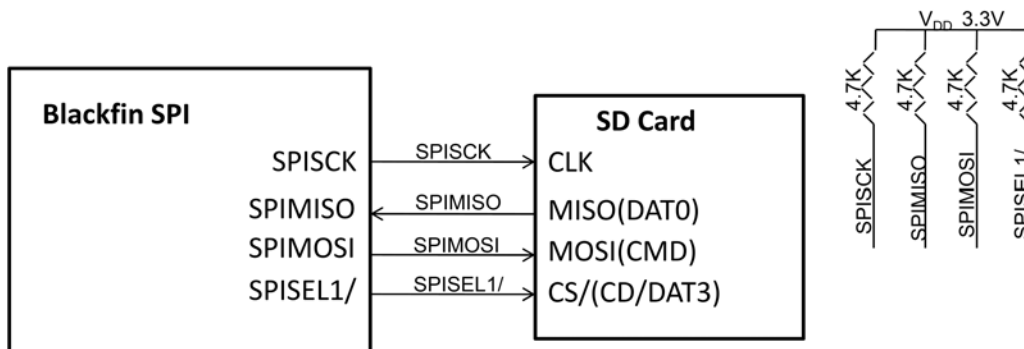


Figure 6. Blackfin Processor - SD card interface: SPI mode

Description

In SPI mode, four signals (SPISCK, SPIMOSI, SPIMISO, and SPISEL1/) are used for the interface. The clock is used to drive data out on the SPIMOSI pin and receive data on the SPIMISO pin. The host drives commands and data to the SD card over the SPIMOSI pin. The host receives response and data from the card on the SPIMISO pin. The chip select signal (SPISEL1/) enables the card during data and command transfer.

The chip select signal is also used initially to drive the card in SPI mode. If the $CS/\overline{}$ line is low when transmitting $CMD0$, the card is configured for SPI mode. In SPI mode, data is transferred in units of eight clock cycles.

Algorithm for the Application Code

1. After power up, drive $SPISSEL1/\overline{}$ inactive (logic high). This disables the card.
2. Issue at least 80 dummy clock cycles for the SD card initialization.
3. Drive $SPISSEL1/\overline{}$ low and transmit a $CMD0$ command. At this point, the card mode changes from default SD mode to SPI mode.

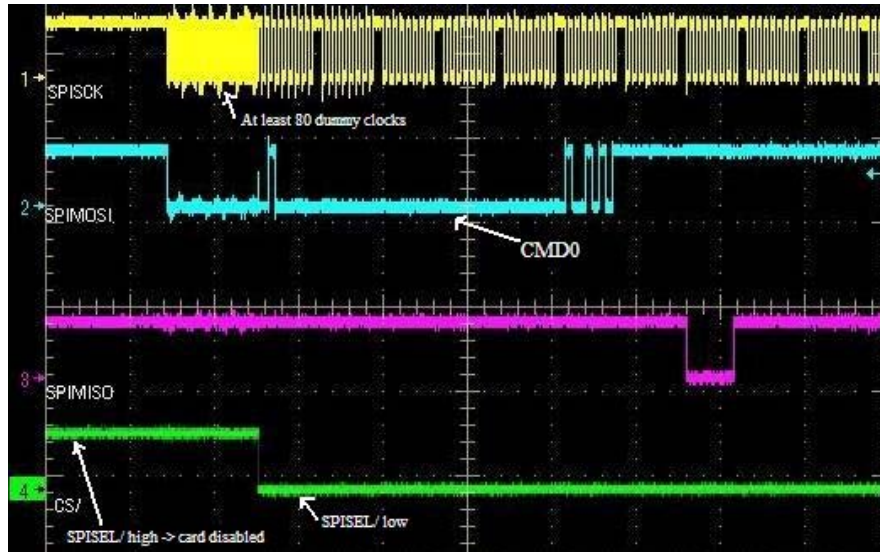


Figure 7. Initialization followed by $CMD0$ command

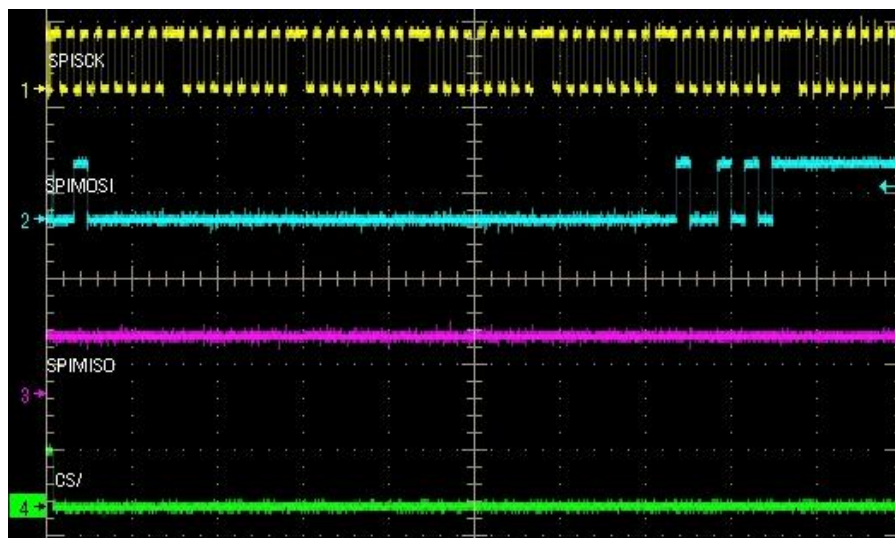


Figure 8. $CMD0$ command

4. Wait for an $R1$ response from the card. The $R1$ response should be $0x01$. Any other value indicates an error condition. Re-execution starting from powering on may be required.

5. Issue a `CMD1` command and wait for an `R1` response. The `R1` response should be `0x00`. Any other response value indicates of error condition.
6. After a correct `R1` response for `CMD1`, data transfer can occur. Note that until the `CMD1` command is successful, the SPI baud rate (clock frequency) should be less than 400 kHz. Before data transfer can begin, the SPI baud rate can be increased.
7. Commands such as set block length can now be issued.
8. Issue commands to read/write the SD card.
9. When powering off, check that the card is in the ready state and drive `SPISSEL1/` high (inactive) before turning off the power supplies.

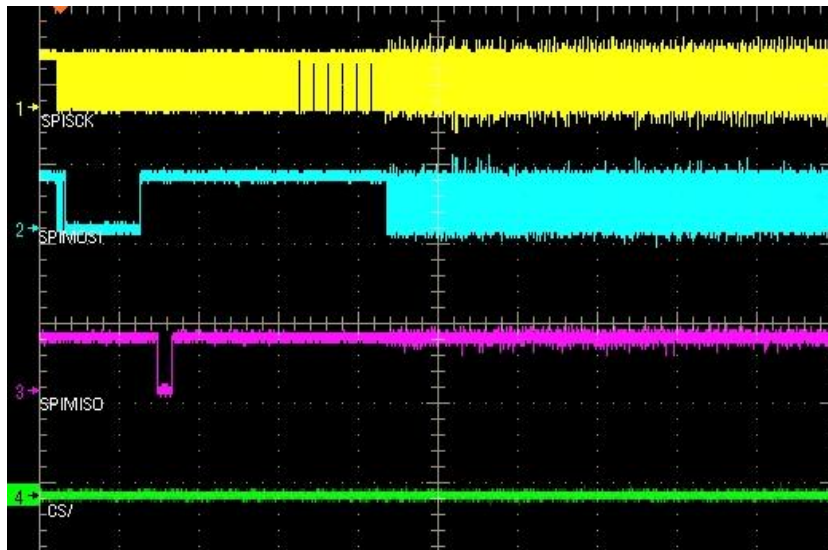


Figure 9. Data write

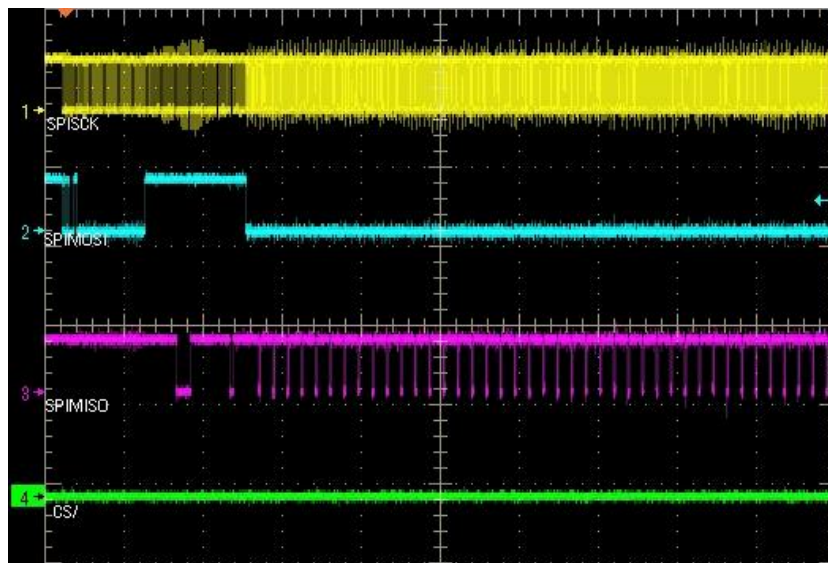


Figure 10. Data read

Interface with Blackfin Processors: SD/MMC Mode

This section describes three methods for the SD mode interface. The first two approaches use the Parallel Peripheral Interface (PPI), and the third approach uses the Asynchronous Memory Bank of the External Bus Interface Unit (EBIU). Interface I uses no glue logic, but uses time-critical interrupts. Interface II is OS (Operating System) friendly, and requires a little glue logic. This approach does not use any deterministic interrupts. The complete control is done in software, thus giving a seamless hardware interface.

Fundamentals – Protocol, Commands, and Responses

Command Format

Each command token is preceded by a start bit (0) and succeeded by an end bit (1). The total length is 48 bits. Each token is protected by CRC bits so that transmission errors can be detected and the operation may be repeated. The format is shown below:

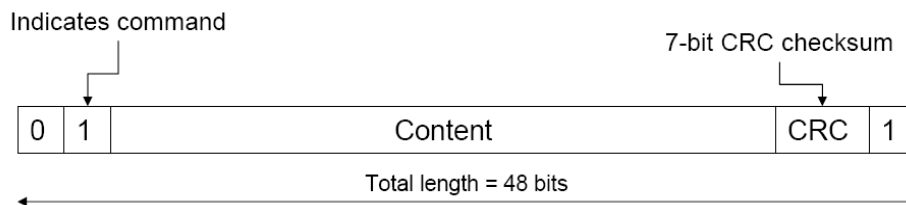


Figure 11. Command format - SD mode

Response Format

Response tokens have one of four coding schemes, depending on their content. The token length is either 48 or 136 bits.

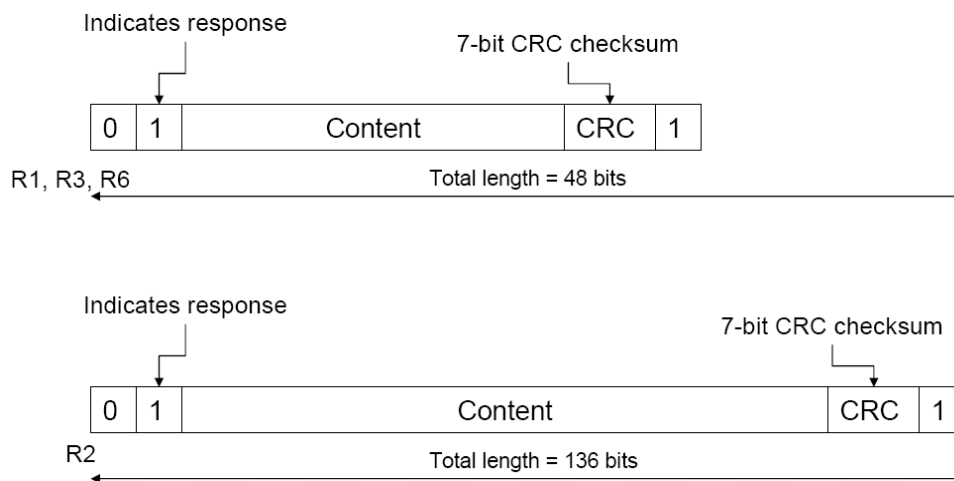


Figure 12. Response format - SD mode

Data Format

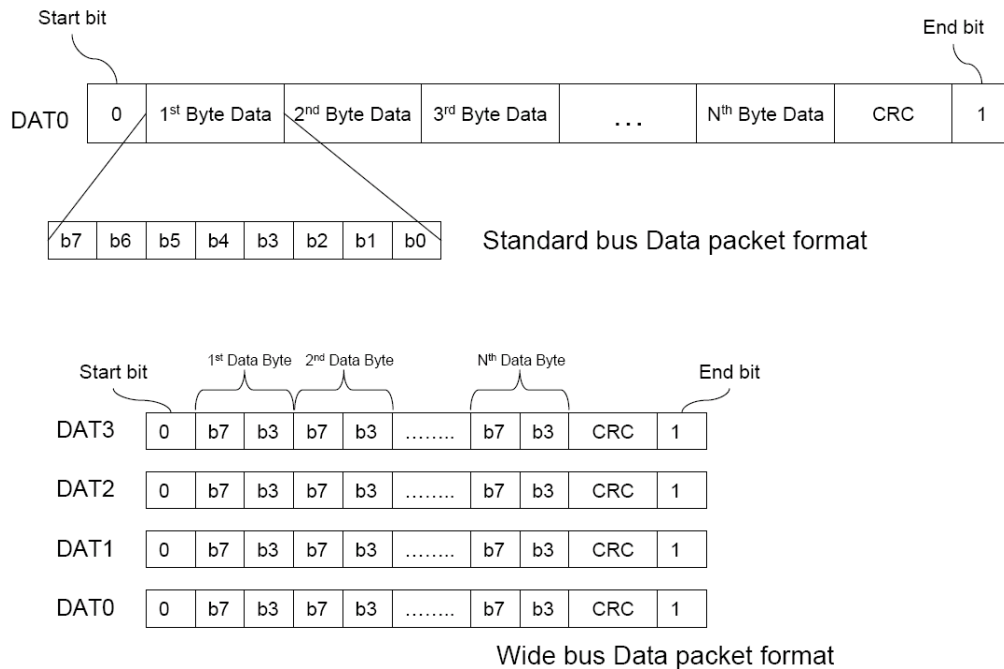


Figure 13. Data format

Cyclic Redundancy Check (CRC)

The CRC is intended to protect SD memory card commands, responses, and data transfer against transmission errors on the SD Memory Card bus. A CRC code is generated for every command and checked for every response on the CMD line. For data blocks, one CRC per transferred block is generated. The CRC is generated and checked as described below.

CRC7

The CRC7 check is used for all commands, for all responses except type R3, and for the CSD and CID registers. The CRC7 is a 7-bit value that is computed as follows:

Generator Polynomial: $G(x) = x^7 + x^3 + 1$
 $M(x) = (\text{first bit}) * x^n + (\text{second bit}) * x^{n-1} + \dots + (\text{last bit}) * x^0$
 $CRC[6:0] = \text{Remainder } [M(x) * x^7 / G(x)]$

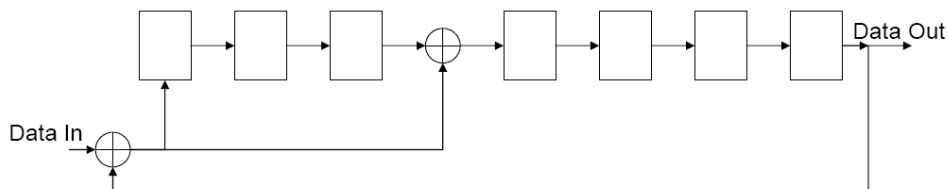


Figure 14. CRC7

CRC16

In the case of one DAT line usage, the CRC16 is used for payload protection in block-transfer mode. The CRC check sum is a 16-bit value and is computed as follows:

Generator Polynomial: $G(x) = x^{16} + x^{12} + x^5 + 1$
 $M(x) = (\text{first bit}) * x^n + (\text{second bit}) * x^{n-1} + \dots + (\text{last bit}) * x^0$
 $\text{CRC}[15:0] = \text{Remainder} [M(x) * x^{16} / G(x)]$

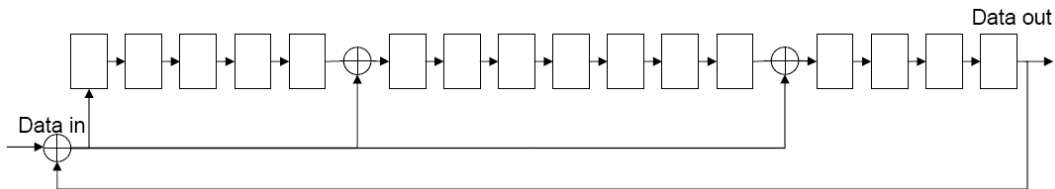


Figure 15. CRC16

Application Software

This section describes the details of software for all the three SD mode interfaces. An overall flowchart is given in Figure 16. The algorithms for each step are given with the detailed description of the interfaces.

Flowchart

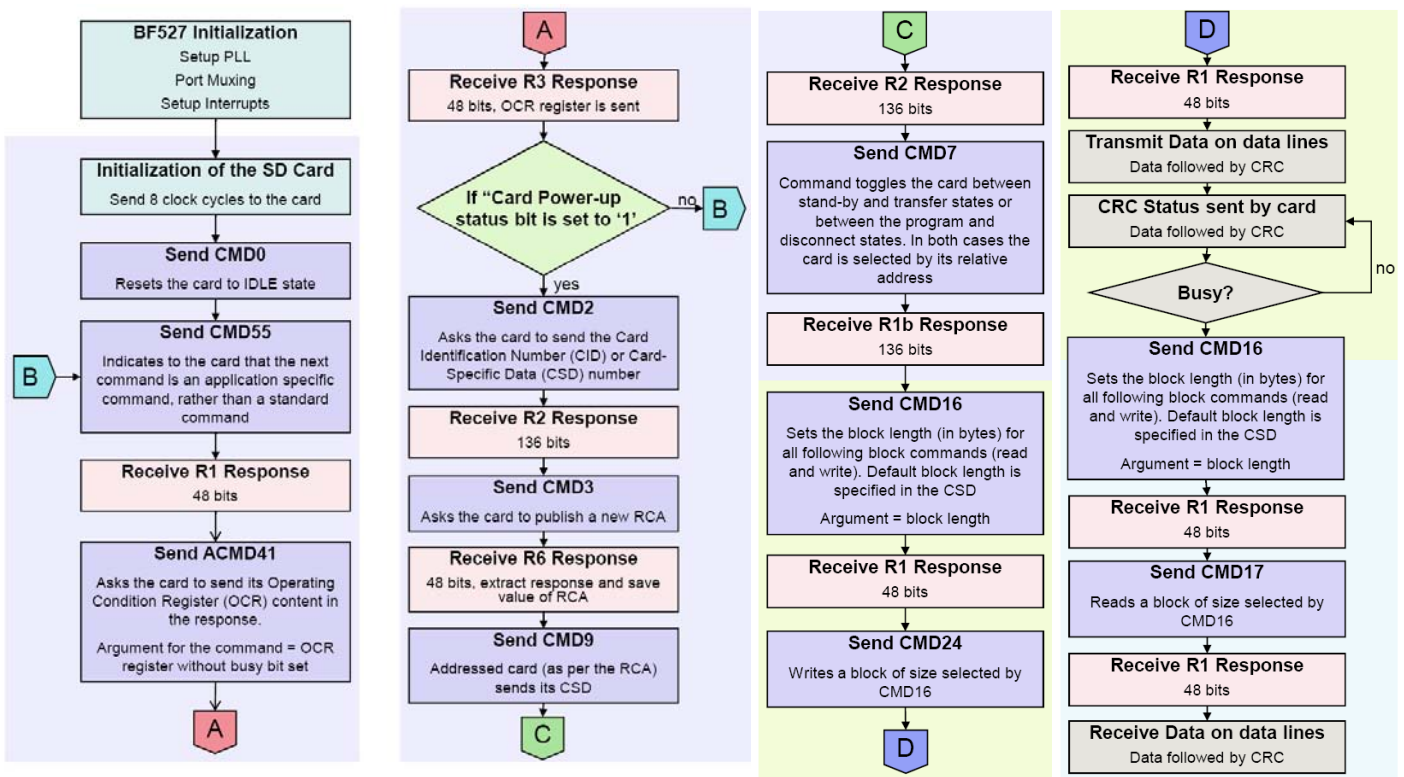


Figure 16. Application flowchart

Interface I and II – PPI

The SD card is interfaced to the Blackfin processor through the Parallel Peripheral Interface (PPI). The parallel port is to implement the bi-directional data/command protocol of the SD card. It supports the 4-bit SD mode. One of the PPI data lines (PPID4) is used for the command/response transfer.

Parallel Peripheral Interface (PPI) of the Blackfin Processor

The PPI is a half-duplex, bidirectional port accommodating up to 16 bits of data. It has a dedicated clock pin and three multiplexed frame sync pins. The PPI_CLK pin accepts an external clock input. It cannot source a clock internally. The general-purpose (GP) PPI modes are intended to suit a wide variety of data capture and transmission applications.

The PPI Control register (PPI_CONTROL) configures the PPI for operating mode, control signal polarities, and data width of the port. For a PPI receive mode with one frame sync, after the PPI receives the hardware frame sync pulse (PPI_FS1), it delays for the duration of the PPI_CLK cycles programmed into PPI_DELAY. The DMA controller then transfers in the number of samples specified by PPI_COUNT. Every sample that arrives after this, but before the next PPI_FS1 frame sync arrives, is ignored and not transferred onto the DMA bus.

For PPI transmit mode with one frame sync, after PPI_FS1 is asserted, there is a latency of one PPI_CLK cycle, and then there is a delay for the number of PPI_CLK cycles programmed into PPI_DELAY. Next, the DMA controller transfers out the number of samples specified by PPI_COUNT. No further DMA takes place until the next PPI_FS1 sync and programmed delay occur.

Approach I

Block Diagram for PPI Approach I interface between Blackfin and an SD card

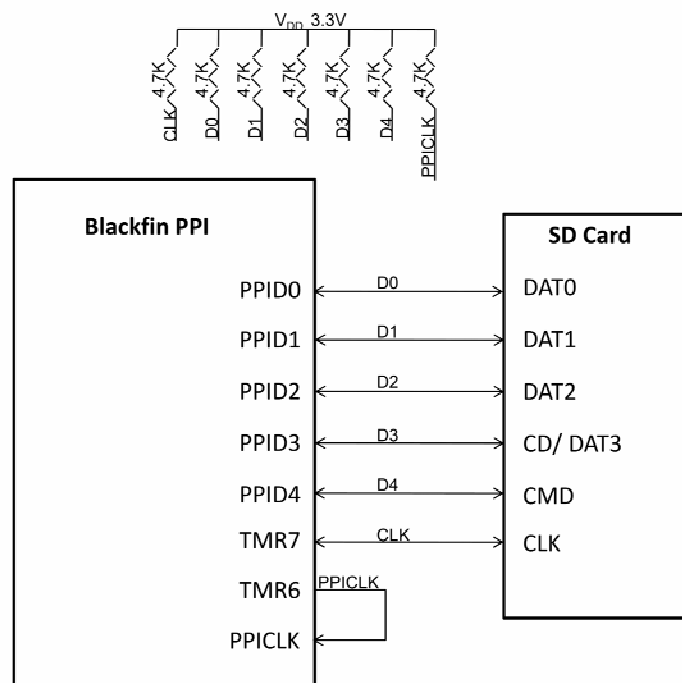


Figure 17. Approach 1: Interface of an SD card with a Blackfin Processor: SD mode

Description

Figure 17 shows a block diagram for Interface I. The PPI data lines are used for both command/response and data transfer. The PPID0-PPID3 signals are connected to the data lines of the SD card (DAT0-DAT3), and PPID4 is connected to CMD of the card. When a command has to be sent to the card, it is embedded into an 8-bit PPI data word. This is done in a manner that in effect sends the command word over the CMD line serially. Similarly, software manipulates data for receiving command words and sending/receiving data words. $4.7\text{-K}\Omega$ pull-up resistors are required on each of the data lines; because the first bit of any transfer word is expected to be '0'.

The CMD line is connected to a GPIO pin, and is configured as input to interrupt on a falling edge. A $10\text{-K}\Omega$ pull-up resistor of is used on this line. After receiving a command, the card sends back a response word after a few (variable number) clock cycles. The start bit of the response is always 0, which implies that a falling edge on the line indicates the start of transmission by the card. The receive operation by Blackfin is started after this.

General purpose timers (TMR6 and TMR7) are used for clock generation. TMR6 supplies a clock to the Blackfin PPI, and TMR7 is used to clock the card. When a word is to be transferred from the Blackfin processor to the card, or vice-versa, the two timers are started in sync. Two timers are used because of certain limitations of Blackfin PPI and SD card.

PPI is a peripheral that is predominantly used for video data transfer applications. Because of this, clock and frame sync signals are not designed for control applications, such as this. When PPI is enabled for transmit, the transmission starts only a few clock cycles after PPI is enabled. So, for this period, clock has to be given only to PPI, and not to the SD card. Also, after the Blackfin processor sends a command, the card is expected to respond after a number of clocks. Until the card sends the first zero, the clock has to be continuous for the card. But the Blackfin processor should not receive data during this time. These reasons prompted the need for separate clock domains.

$10\text{-K}\Omega$ pull-up resistors of are used on the clock lines. This is needed because the card latches data on the rising edge and transmits on the falling edge.

Algorithms for the Software

The flowchart in Figure 16 shows the overall program flow. Each of the steps for Approach I are described in the algorithms below.

Set Up N Clock Cycles to the SD Card

1. Disable TMR7 (clock to the SD card).
2. Configure the core timer to count for a time equal to N SD clock cycles and interrupt.
3. Configure TMR7 in PWM out mode for the desired width and period.
4. Enable TMR7 .
5. Enable the core timer.
6. Disable the core timer in the core timer ISR.

Set Up a Continuous SD Card Clock

1. Disable TMR7 (clock to the SD card).
2. Configure TMR7 in PWM out mode for the desired width and period.
3. Enable TMR7.

Set Up a Continuous Blackfin Clock

1. Disable TMR6 (clock to Blackfin PPI).
2. Configure TMR6 in PWM out mode for the desired width and period.
3. Enable TMR6.

Set Up N Clock Cycles to Blackfin PPI for 1FS Receive

1. Disable TMR6 (clock to Blackfin PPI).
2. Configure the core timer to count for a time equal to N PPI clock cycles and interrupt.
3. Configure TMR6 in PWM out mode for the desired width and period.
4. Configure TMR0 in PWM out mode for the desired width to derive clock source from the PPI clock.
5. Enable TMR6 and TMR0.
6. Enable the core timer.
7. Disable TMR0.
8. Disable the core timer and TMR6 in the core timer ISR.

Set Up N Clocks to Blackfin PPI and the SD Card

1. Disable TMR7 (clock to the SD card).
2. Configure the core timer to count for a time equal to N clock cycles and interrupt.
3. Configure TMR6 in PWM out mode for the desired width and period.
4. Configure TMR7 in PWM out mode for the desired width and period.
5. Enable TMR6 and TMR7.
6. Enable the core timer.
7. Disable the core timer and TMR6 in the core timer ISR.

Set Up Command Array

CommandBuffer[6][8] is filled up so that the bits of the command word occupy bit 4 of each word. For example, the first byte of CMD0 is 0x40. This translates to CommandBuffer [0][7] = 0; CommandBuffer [0][6] = 0x10; CommandBuffer [0][5] = 0; CommandBuffer [0][4] = 0; CommandBuffer [0][3] = 0; CommandBuffer [0][2] = 0; CommandBuffer [0][1] = 0; and CommandBuffer [0][0] = 0. CRC7 is calculated for the last byte of each command word.

Set Up Transmit Data

TransmitData[516][8] is filled up so that the bits of the data occupy bit 0 of each word. For example, for a data byte = 0x33, TransmitData[0][7] = 0; TransmitData[0][6] = 0; TransmitData[0][5] = 1; TransmitData[0][4] = 1; TransmitData[0][3] = 0; TransmitData[0][2] = 0; TransmitData[0][1] = 1; and TransmitData[0][0] = 1. CRC16 is calculated for the entire block.

Receive Response

1. Routine to sense the first zero sent by the card
 - a) Set up PPID4 as input to interrupt on the falling edge.
 - b) Set up a continuous SD clock.
 - c) Disable continuous clock in the ISR.
2. Dummy DMA receive transaction to account for the delay
3. Actual DMA receive
4. Extract response: Bit 4 of every byte of the response, ResponseBuffer[17][8] is extracted into FinalResponseBuffer[17].

Send Command

1. Send 12 SD clocks – a minimum of 8 are needed as per the SD specification.
2. Set up command array.
3. Set up and enable DMA transmit.
4. Set up and enable PPI transmit.
5. Set up 4 clocks to the Blackfin – for the delay.
6. Set up 48 clocks to the Blackfin and the card.
7. Check for DMA_RUN bit to be cleared and then disable PPI in the ISR.

Receive Data

1. Routine to sense the first zero sent by the card
 - a) Set up PPID0 as input to interrupt on the falling edge.
 - b) Set up a continuous SD clock.
 - c) Disable continuous clock in the ISR.
2. Dummy DMA receive transaction to account for the delay
3. Actual DMA receive
4. Extract response: Bit 0 of every byte of the response, ReceiveData [517][8] is extracted into FinalReceiveData [514].

Send Data

1. Set up data in the data buffer.
2. Set up and enable DMA transmit.
3. Set up and enable PPI transmit.
4. Set up 4 clocks to the Blackfin – for the delay.
5. Set up 4121 clocks to the Blackfin and the card.
6. Dummy DMA receive for 8 words – workaround for anomaly# 05000401 - PPI data signals D0 and D8 do not three-state after disabling PPI.
7. Routine to sense the first zero sent by the card
 - a) Set up PPID0 as input to interrupt on the falling edge.
 - b) Set up a continuous SD clock.
 - c) Disable continuous clock in the ISR.
8. Actual DMA receive operation

The commands and responses detailed in the flowchart for Interface I are given in Figure 23 to Figure 32.

Approach II

Block Diagram

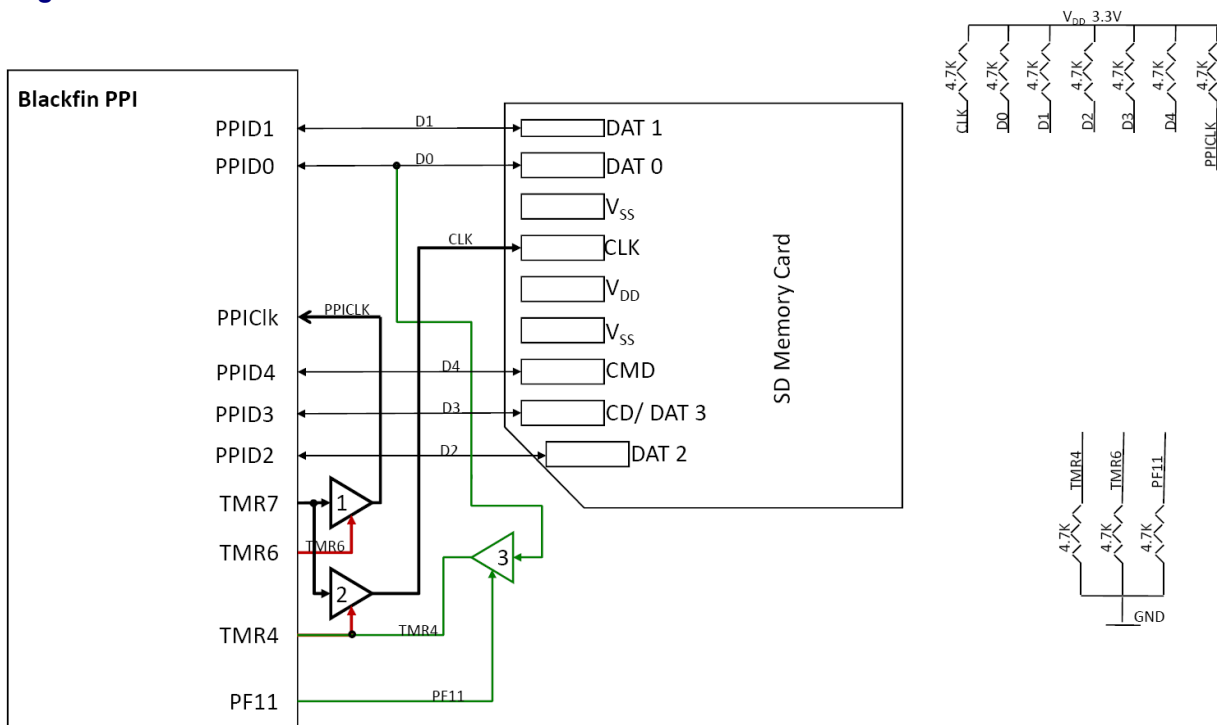


Figure 18. Approach II: Interface of an SD card with a Blackfin Processor: PPI mode

Description

Figure 18 shows a block diagram for Interface II. Here, the clocks to the SD card and PPI are gated through non-inverting three-state buffers. Two other timers (TMR3 and TMR4) are used as output-enable signals. In the first approach, the number of clocks that reach the SD card and PPI are limited by stopping the two timers in the interrupt service routine (ISR) of the core timer (which counts the number of clocks). So, there was a requirement to service the ISRs immediately, without any delay. In this approach, the clocks are controlled by pulses of the required duration generated by the TMR3 and TMR4 timers. This makes it independent of interrupt service time.

When a command is sent by the card and the Blackfin processor is waiting for a response, the response is sent after a variable number of clock cycles. In this case, the number of clock cycles needed is not fixed. Thus, single pulses are sent on the TMR6 and TMR7 lines, and after each pulse the values on the data lines are read. In order to do this, the multiplexing on these lines is changed from PPI data lines to GPIO pins.

4.7-K Ω pull-up resistors are needed on the two output enable lines of the buffers (TMR3 and TMR4). This ensures that the buffers are not enabled when these lines are not driven.

Algorithms for the software

The flowchart given in Figure 16 shows the overall program flow. Each of the steps for Approach II is described in the algorithms below.

Set Up N Clock Cycles to the SD Card

1. Configure TMR7 in PWM out mode to generate a continuous clock.
2. Configure TMR6 to generate a single pulse of a width equal to the number of clock cycles that need to be generated.
3. Enable both timers.

Set Up a Continuous SD Card Clock

1. Configure TMR6 as a GPIO output.
2. Drive HI on this pin.
3. Configure TMR7 in PWM out mode to generate a continuous clock.
4. Enable TMR7.
5. Drive LO on GPIO to stop the clock.

Set Up a Continuous Blackfin Clock

1. Configure TMR4 as a GPIO output.
2. Drive HI on this pin.
3. Configure TMR7 in PWM out mode to generate a continuous clock.
4. Enable TMR7.
5. Drive LO on GPIO to stop the clock.

Set Up N Clock Cycles to Blackfin PPI

1. Configure TMR7 in PWM out mode to generate a continuous clock.
2. Configure TMR6 to generate a single pulse whose width equals the number of clock cycles that need to be generated.
3. Enable both timers.

Set Up N Clocks to Blackfin PPI and the SD Card

1. Configure TMR7 in PWM out mode to generate a continuous clock.
2. Configure TMR4 and TMR6 to generate a single pulse whose width equals the number of clock cycles that need to be generated.
3. Enable TMR4, TMR6, and TMR7.

Frame Sync Generation for PPI Transmit

1. Configure TMR0 to derive clock from PPI clock and generate a single pulse.
2. Enable TMR0.

Receive Response

1. Set up and enable DMA to receive a “number” of words equal to the response length plus 24.

When PPI is used in receive mode with 0 frame syncs, a delay of at least two clock cycles occurs before actual data is received. In this approach, eight extra words are received to account for this delay. The card transmits the first ‘0’ after a variable number of clocks after receiving the stop bit of the command. This amounts to a total of 24 extra clocks.

2. Set up and enable PPI to receive in 0 frame sync GP mode with internal trigger.
3. Send eight clock cycles only to the Blackfin processor.

When PPI is used in receive mode with 0 frame syncs, a delay of at least two clock cycles occurs before actual data is received. In this approach, eight extra words are received to account for this delay. The card transmits the first ‘0’ after a variable number of clocks after receiving the stop bit of the command. This amounts to a total of 24 extra clocks.

4. Send “response length + 24” clock cycles to both the PPI and the SD card.
5. Disable PPI in the DMA ISR.
6. Extract response
 - a) Detect the first 0 in the buffer. This is the start bit.
 - b) Extract bit 0 from every word out of “response length” words to form the complete response.

Send Command

1. Send 12 SD clocks – a minimum of eight are needed as per the SD specification.
2. Set up command array.
 - a) Fill up the command buffer.
 - b) Compute CRC7.
 - c) Rearrange the command word to occupy only bit 4 of every PPI command word.
3. Set up and enable DMA to transmit 48 words (equivalent to 48 bits on the CMD line).
4. Set up and enable PPI in 1 frame sync (FS) GP mode to transmit 48 words.
5. Generate FS and send four clock cycles only to the Blackfin processor (to account for PPI delays).
6. Send 48 clock cycles to both the PPI and the SD card.
7. Check for the `DMA_RUN` bit to be cleared and then disable PPI.

Interface III – Asynchronous Memory Interface

Block Diagram

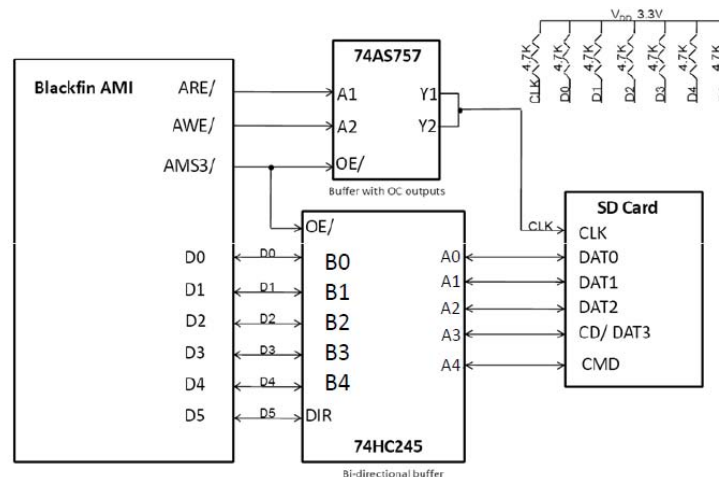


Figure 19. Approach III: Interface of an SD card with Blackfin Processors: Asynchronous Memory Interface

Description

The SD card is interfaced as a memory-mapped device on the Asynchronous Memory Interface (AMI), bank 3.

The data lines are used for both command/response and data transfer. `D0-D3` signals are connected to the data lines of the SD card (`DAT0 – DAT3`), while `D4` is connected to `CMD` of the card. When a command has to be sent to the card, it is framed into a series of 8-bit data words; command bits being driven out on `D4`. Effectively, the command word is serially sent over the `CMD` line. Similarly, software does data rearrangement of the received responses, and while sending and receiving content data. 4.7-K Ω pull-up resistors are needed on each of the data lines; because the first bit of any transfer word is expected to be a '0'.

The SD card clock signal is derived from the $\overline{ARE}/$ and $\overline{AWE}/$ signals. $\overline{ARE}/$ and $\overline{AWE}/$ are connected to the inputs of a buffer with open collector outputs. $\overline{AMS3}/$ drives the output enable of this buffer. The outputs of this buffer are shorted and drive the SD card clock. The SD card clock is therefore aware of the $\overline{ARE}/$ and $\overline{AWE}/$ signals.

The data lines are also buffered through a bi-directional buffer, which isolates the SD card from any other devices sitting on the same AMI interface.

The \overline{DIR} pin of the bi-directional buffer is connected to the data line (D5) of the Blackfin AMI. A ‘1’ on this line configures the direction to be “card to Blackfin”. The 4.7-K Ω pull-up resistor ensures that this line is HI when data is being read from the card. When the Blackfin processor writes data into the card, software ensures that D5 is driven low. This sets the direction to “Blackfin to card”.



Use 4.7-K Ω pull-up resistors on all unused inputs of CMOS ICs.

Card Detection

The CD pin on an SD card has a 50-K Ω pull-up at power-up. For card detection, the host detects that the line is pulled high. Disconnect this pull-up resistor during regular data transfer with the `SET_CLR_CARD_DETECT` (ACMD42) command. Several mechanisms can be implemented for card detection. In addition to the circuits described below, a system designer can also rely on the hardware switches (to detect card insertion/removal and write protect) implemented within the card slots; most card slots support them. Using the hardware switches in the card slots is perhaps the simplest mechanism to detect card insertion and removal.

Method 1

Circuit Diagram

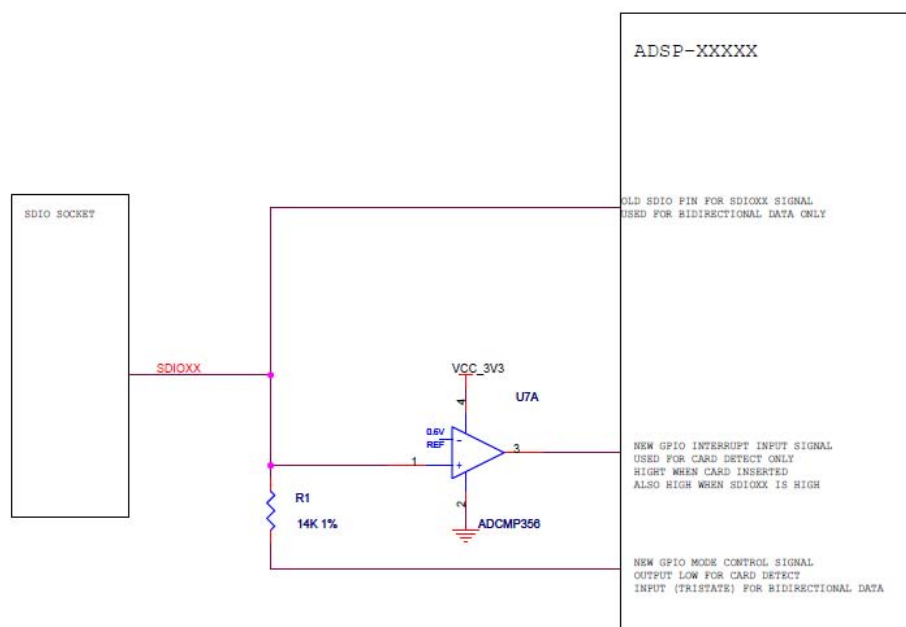


Figure 20. Method 1: Card detection

Description

In this method the GPIO connected to one end of R1 is driven low for card detection. After card detection, the GPIO is configured as an input to disable the pull-down resistor. After the mechanical approach that fully relies on the hardware card switches, this is the strongly recommended card detection circuitry as it takes in to account the positive and negative three-state leakage current [I_{OZH} and I_{OZL}] specification of $10\ \mu\text{A}$ on I/O signals. The circuits shown in Method 2 and Method 3 are also functional because the I/O leakage currents are typically not as high as $10\ \mu\text{A}$.

Method 2

Circuit Diagram

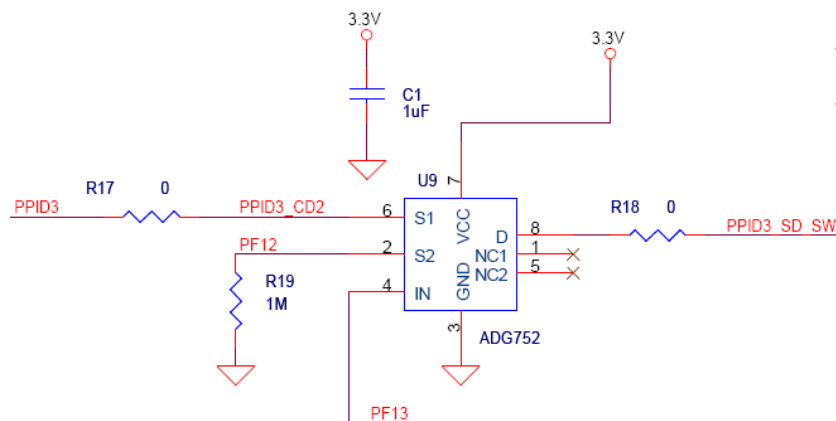


Figure 21. Method 2: Card detection

ADG752 IN	Switch S1	Switch S2
0	ON	OFF
1	OFF	ON

Table 4. Truth table for ADG752

Description

The ADG752 is a low-voltage, single pole, double throw (SPDT) switch.

In the block diagram, PPID3 is connected to Blackfin PPID3. PPID3_SD_SW is connected to CD of the card. A programmable flag (PF12) is connected to S2. PF13 is connected to the IN input. Initially, PF13 is driven HI. This connects S2 to D (that is, it connects DAT3 of the card to a 1-M Ω pull-down resistor. PF12 is configured as an input to interrupt on a rising edge.

When the card is inserted, a rising edge on PF12 causes an interrupt. In the ISR, a low is driven on PF13 and the 50-K Ω pull-up resistor on CD is disabled.

Method 3

Block Diagram

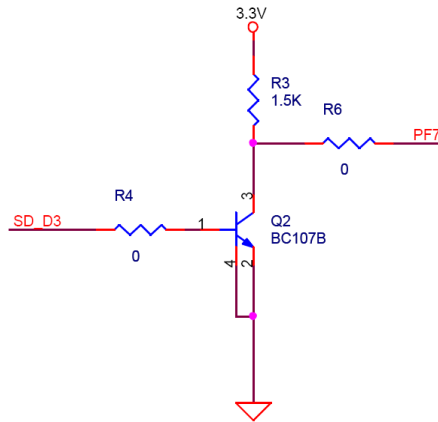


Figure 22. Method 3: Card detection

Description

SD_D3 is connected to the DAT3 line of the card. Initially, when the card is not inserted, this pin is floating. The transistor is OFF, and PF7 is high. When the card is inserted, the transistor turns ON. This causes a low on PF7.

The disadvantage of this method is that the internal 50-K Ω pull-up resistor must be used.

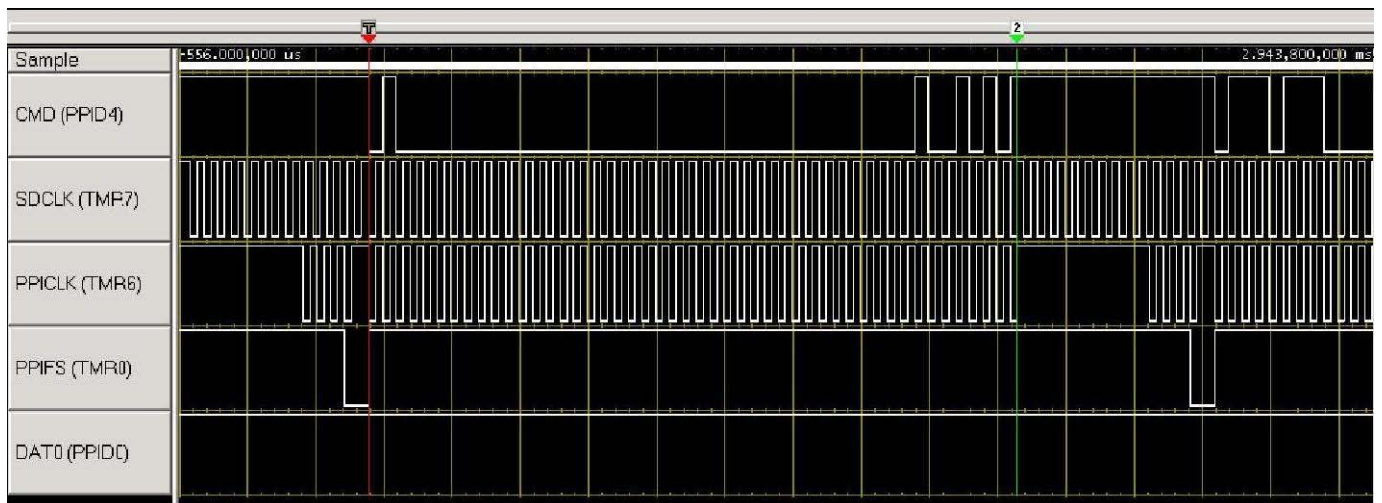


Figure 23. CMD0



Figure 24. CMD 55 followed by response

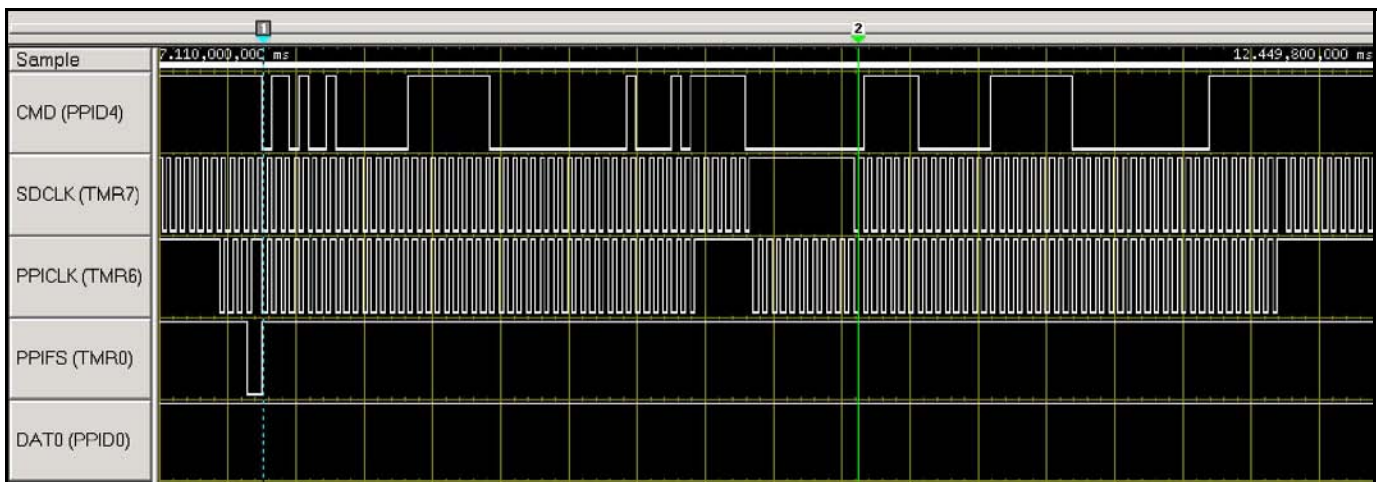


Figure 25. ACMD41 followed by wrong response

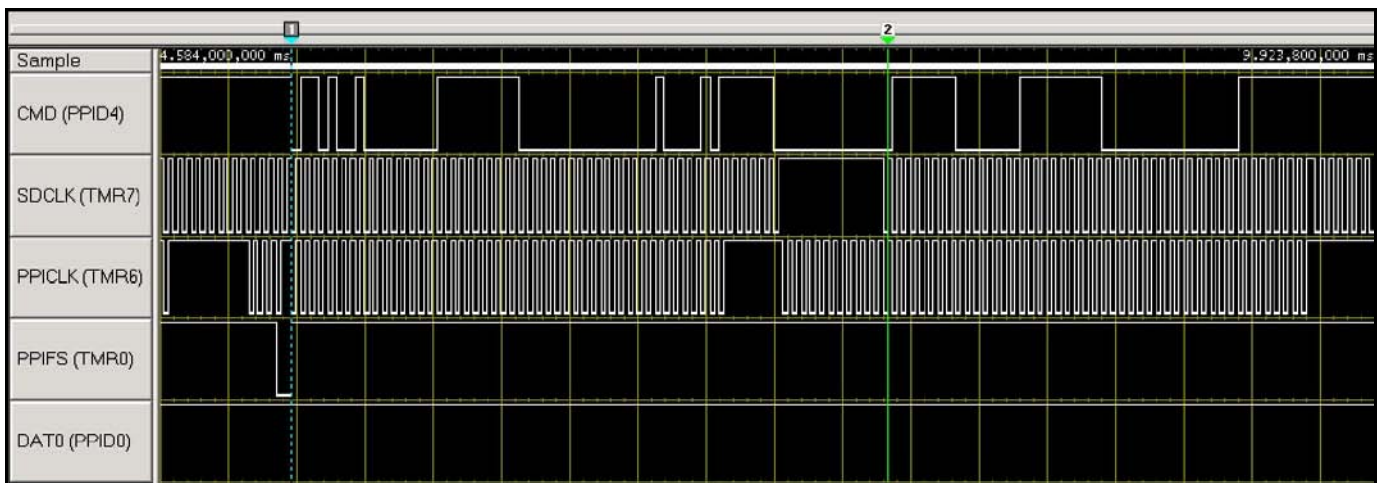


Figure 26. ACMD41 followed by correct response

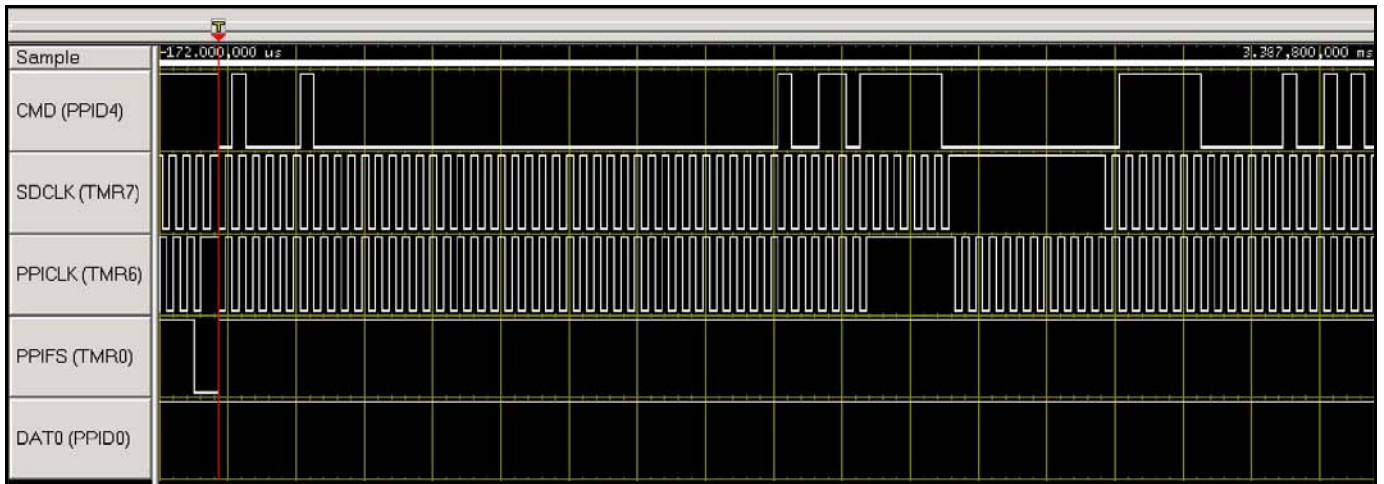


Figure 27. CMD2

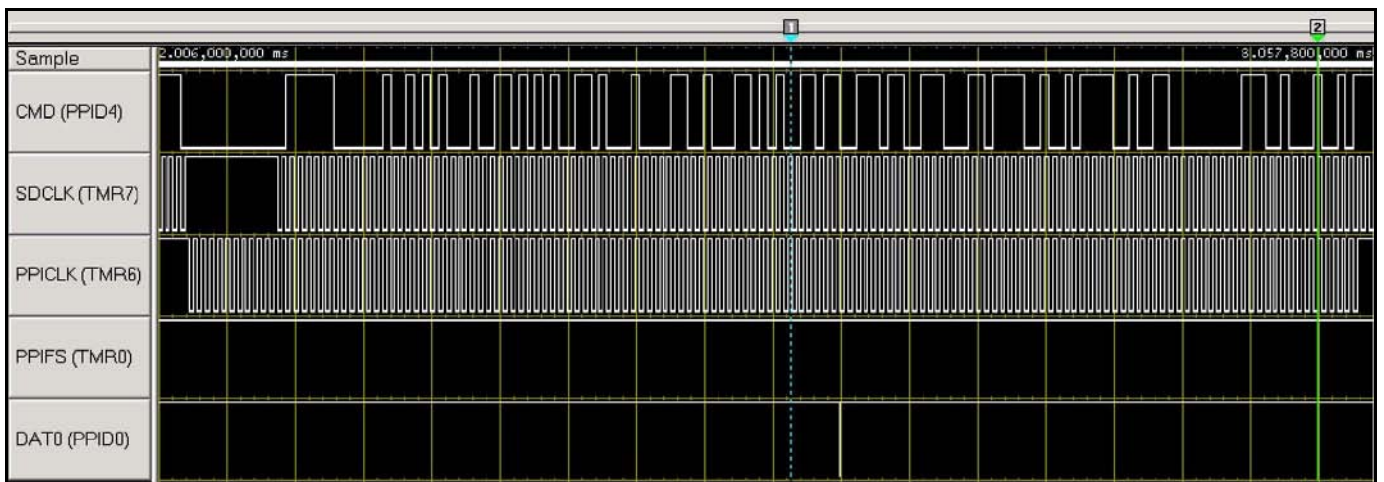


Figure 28. Response for CMD2

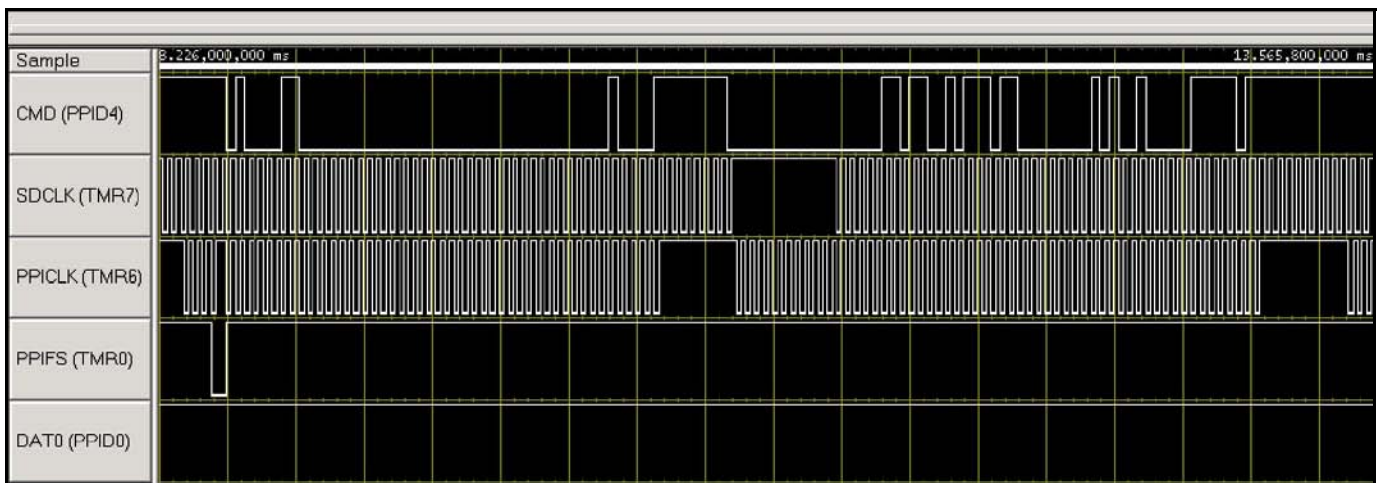


Figure 29. CMD3 followed by response

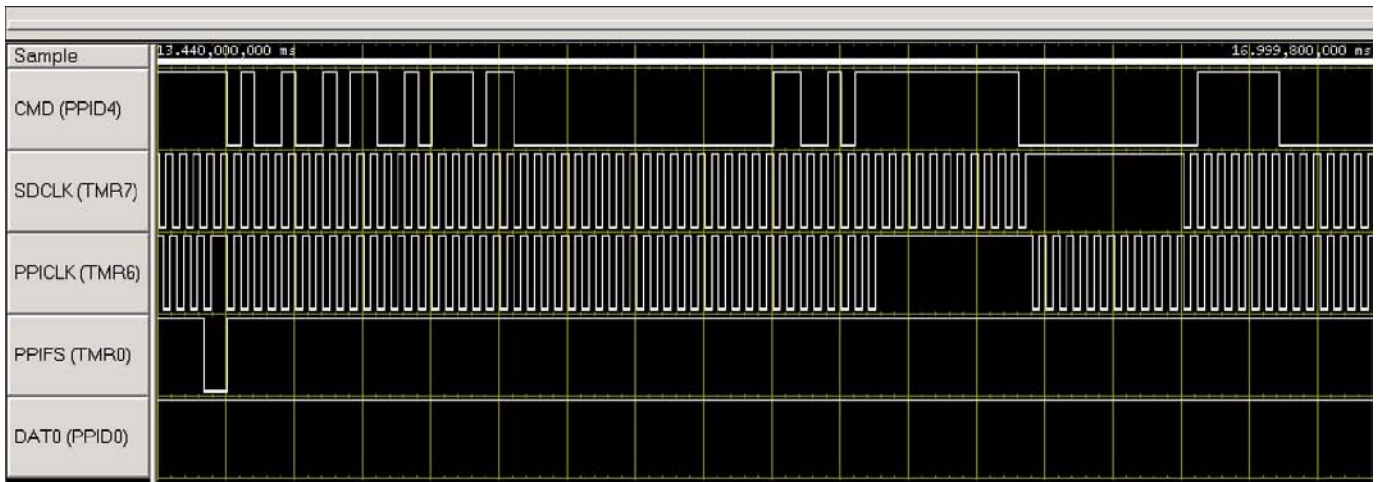


Figure 30. CMD9

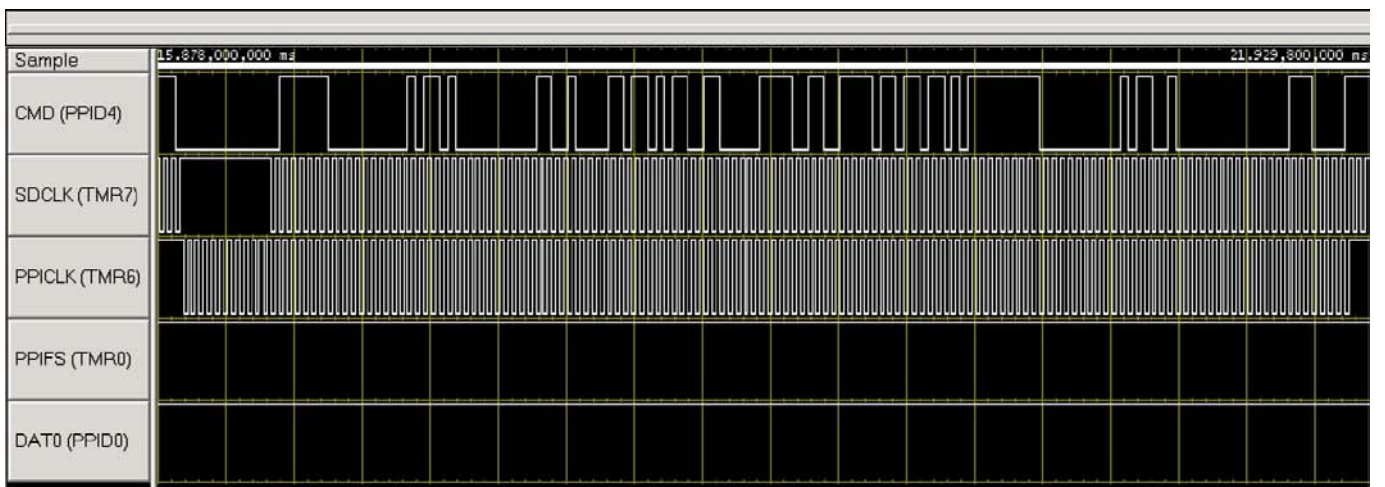


Figure 31. Response for CMD9

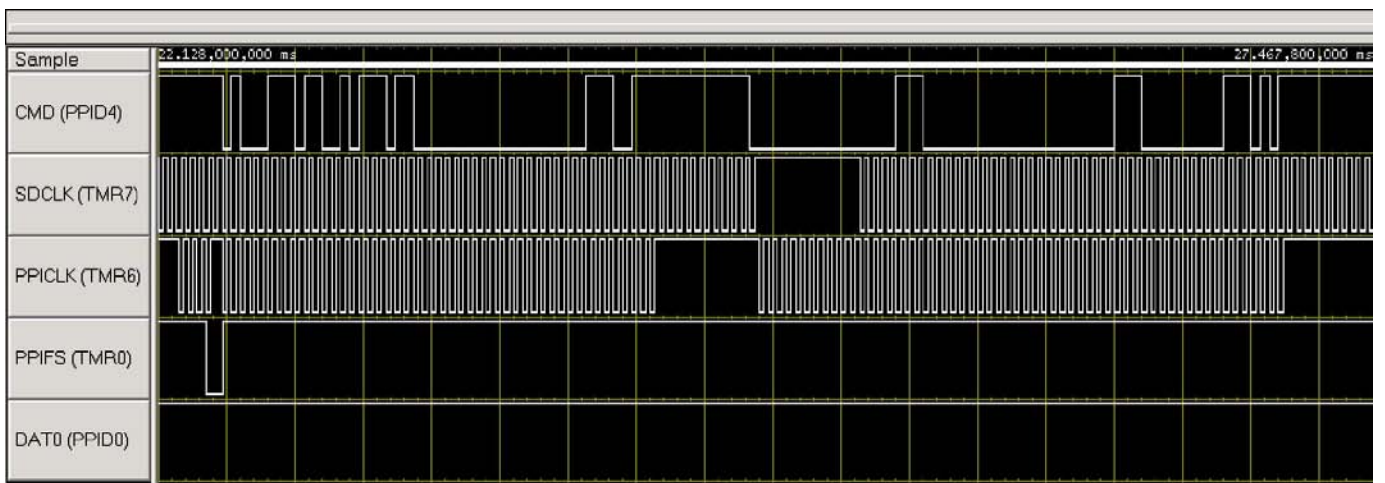


Figure 32. CMD7 followed by response

SD Card Interface Evaluation Platform

Rubico (<http://www.rubico.se>) built an evaluation board and developed uClinux drivers. To evaluate this interface on a hardware platform, contact Rubico for evaluation boards and driver code. The reference code provided in the associated .ZIP file was developed using VisualDSP++® Development Tools and validated on an in-house designed test board. This reference code has been written keeping in mind, as much as possible, the operating system environment.

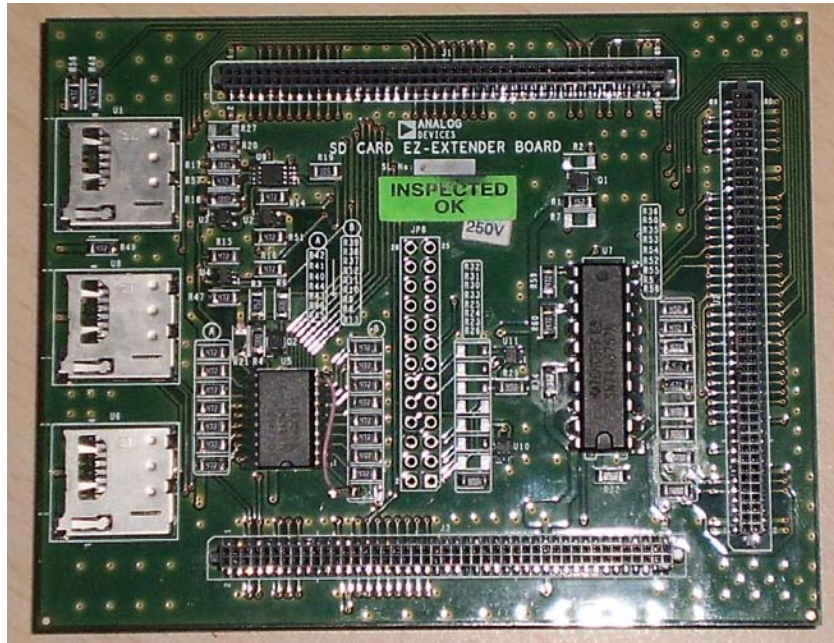


Figure 31 SD-interface with a Blackfin Processor (In-house test platform)

References

- [1] *The MultiMediaCard System Summary*. Based on System Specification Version 3.3. MMCA Technical Committee
- [2] *SD Memory Card Specifications*. Part 1. Physical Layer Simplified Specification. Version 2.00. SD Group
- [3] *SD Memory Card Specifications*. Part 1. Physical Layer Specification. Version 1.0. SD Group
- [4] *ADSP-BF522C/ADSP-BF523C/ADSP-BF524C/ADSP-BF525C/ADSP-BF526C/ADSP-BF527C Blackfin Embedded Processor Data Sheet*. Rev 0, May 2009. Analog Devices, Inc.
- [5] *ADSP-BF522/ADSP-BF523/ADSP-BF524/ADSP-BF525/ADSP-BF526/ADSP-BF527: Blackfin Embedded Processor Data Sheet*. Rev A, November 2009. Analog Devices, Inc.
- [6] *ADSP-BF52x Blackfin Processor Hardware Reference (Volume 1 of 2)*. Rev 0.31 (Preliminary), May 2008. Analog Devices, Inc.
- [7] *ADSP-BF52x Blackfin Processor Hardware Reference (Volume 2 of 2)*. Rev 0.3 (Preliminary) September 2007. Analog Devices, Inc.
- [8] *ADSP-BF527 EZ-KIT Lite Evaluation System Manual*. Rev. 1.1, November 2007. Analog Devices, Inc.
- [9] *ADSP-BF533 Blackfin Processor Hardware Reference*. Rev 3.1, May 2005. Analog Devices, Inc.
- [10] *Interfacing MultiMediaCard™ with ADSP-2126x SHARC Processors (EE-264)*. Rev 1, March 2005. Analog Devices, Inc.
- [11] *MultiMediaCard* (http://en.wikipedia.org/wiki/Multimedia_Card).
- [12] *SD Card Association* (<http://www.sdcard.org>).
- [13] *MultiMediaCard Association* (<http://www.mmca.org>).

Document History

Revision	Description
<i>Rev 1 – February 19, 2010 by Jayanti Addepalli and Aseem Vasudev</i>	Initial release.