Example application under PRET environment – (Programming a MultiMediaCard)

COMS – E6901, Columbia University

Devesh Dedhia(UNI – ddd2121)

Abstract

PRET philosophy proposes the temporal characteristics to be made predictable. However for various applications the PRET processor will have to interact with a non predictable environment. In this paper an example of one such environment, an MultiMediaCard(MMC) is considered. This paper illustrates a method to make the response of the MMC predictable.

I. Introduction

PRET processor requires static time predictability. However the datasheets of an MMC card specify a time range for the response of the MMC card rather than the precise time. Hence the time required to program an MMC card would be variable depending on exactly when the card responds to the sent commands. Also the data transfer speeds supported and Read/ Write access time ranges of an MMC card are variable and specified by Card Specific Data (CSD) register. In order to make the response predictable we wait for the worst case time defined in the specified time range. The Timing specific data from the CSD is retrieved and communication is aborted if the card supports a maximum frequency less than the 20 MHz

II. Programming an MMC card

The MMC card can be programmed in 2 modes

MMC Mode:

The communication between the host and the MMC card consists of command, response and data – block tokens. Every command or data bit stream is initiated by a start bit and terminated by a stop bit. It has a 10 wire bus consisting of the following lines.

CLK: Each cycle of this signal directs a one bit transfer on the command and on all the data lines. The frequency may vary between zero and the maximum clock frequency.

CMD: This signal is a bidirectional command channel used for card initialization and transfer of commands. Commands are sent from the MultiMediaCard bus master to the card and responses are sent from the card to the host.

DAT0-DAT7: These are bidirectional data channels. By default, after power up or reset, only DAT0 is used for data transfer. A wider data bus can be configured for data transfer, using either DAT0-DAT3 or DAT0-DAT7, by the Multimedia Card controller.

SPI Mode:

This mode is a subset of the MultiMediaCard protocol, designed to communicate with a SPI channel, commonly found in microcontrollers. The Serial Peripheral Interface standard defines the physical link and not the complete data transfer protocol. The MultiMedia-Card SPI implementation uses a subset of the MultiMediaCard protocol and command set. Every command or data block is built of 8–bit bytes and is byte aligned with the Chip select signal. The bidirectional CMD and DAT lines used in the MMC mode are replaced by unidirectional *dataIn* and *dataOut* signals. It has a 4 wire bus consisting of the following lines.

CS: Chip Select signal is driven by the host. The CS signal must be continuously asserted (Active Low) for the duration of the SPI transaction.

CLK: Each cycle of the signal causes a one bit transfer on the DataIn or DataOut line.

DATAIN: The host sends all the commands and Write Data on this line.

DATAOUT: The host receives read data and response on this line.

It is assumed that PRET processor in its later stages of development shall have a hardware SPI engine. Therefore in this project SPI mode has been chosen for communication with the MMC card.

III. Real world model:

Before designing the system model in the PRET environment it is important to study the Real world model of the Host- MMC card communication. The hardware SPI engine inside the microcontroller consists of an 8 bit buffer called Serial Receive/ Transmit Buffer (SSPBUF). There is buffer full flag which is set when the SSPBUF is full. The shift register is used to shift data in and out serially through pins SDI and SDO respectively. The microcontroller (Host) selects the MMC card as a slave by asserting the SS (Slave Select) signal. The clock line of the SPI bus is driven by the clocking unit inside the microcontroller through the SCK pin.

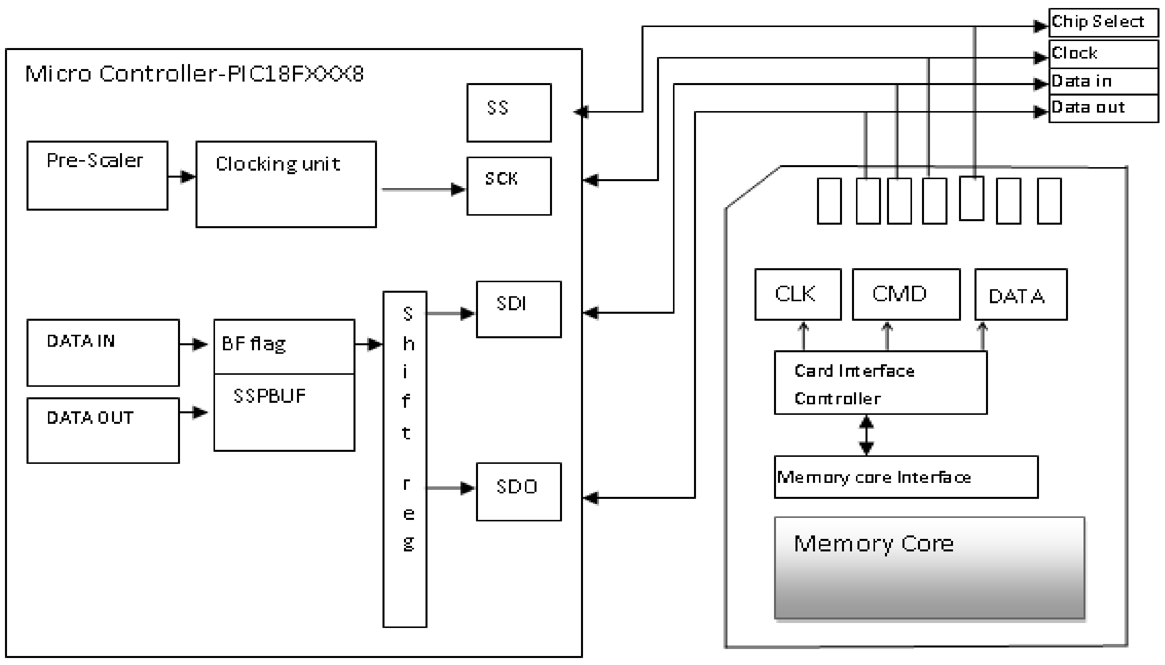


Figure 1: Block diagram of communication between MMC card and a microcontroller using SPI

IV. PRET Architecture:

The PRET PROCESSOR component implements a six-stage thread-interleaved pipeline in which each stage executes a separate hardware thread. Each hardware thread has its own register file, local on-chip Memory, and assigned region of off-chip memory.

Memory map: Each piece of memory in the system has a unique global address (main memory and SPMs), but each thread only has access to part of the overall memory map. Addresses 0x3F800000 – 0x405FFFFF (14 MB) are main memory, visible to every thread. This part of the memory can be used for communication between threads. Peripherals start at 0x80000000; in the Memory-mapped I/O space.

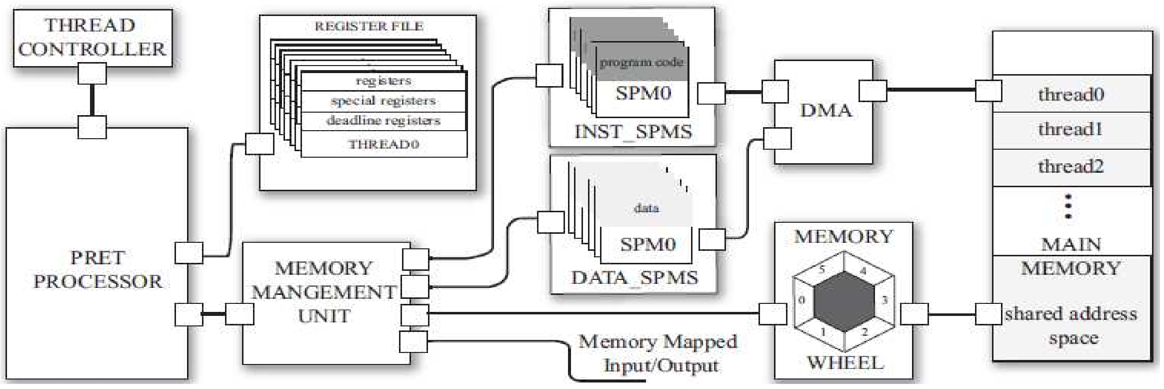
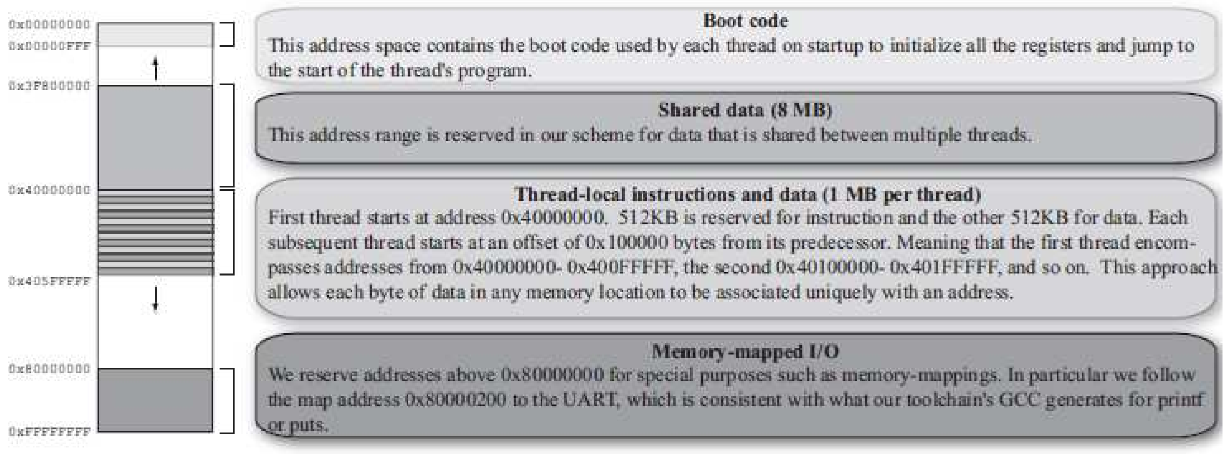


Figure 2: Block diagram of the PRET Architecture



Boot code
This address space contains the boot code used by each thread on startup to initialize all the registers and jump to the start of the thread's program.

Shared data (8 MB)
This address range is reserved in our scheme for data that is shared between multiple threads.

Thread-local instructions and data (1 MB per thread)
First thread starts at address 0x40000000. 512KB is reserved for instruction and the other 512KB for data. Each subsequent thread starts at an offset of 0x100000 bytes from its predecessor. Meaning that the first thread encompasses addresses from 0x40000000- 0x400FFFFF, the second 0x40100000- 0x401FFFFF, and so on. This approach allows each byte of data in any memory location to be associated uniquely with an address.

Memory-mapped I/O
We reserve addresses above 0x80000000 for special purposes such as memory-mappings. In particular we follow the map address 0x80000200 to the UART, which is consistent with what our toolchain's GCC generates for printf or puts.

Figure 3: Memory Map

V. System Model in the PRET environment:

The entire system is modeled with two threads. Thread 0 emulates a processor sending commands while thread 1 emulates a memory card sending the desired response. As focus of the project is to study and simulate the time response of the MMC card all the timing requirements have to be considered. An application to read a block on 512 bytes from the MMC card has been created.

Flags: Flags have been used to prevent overwriting of data by either of the threads. They are acting like semaphores to protect the Read/Write buffers.

Writeflag: It is a byte in the shared memory address space at memory location 0x3F800001. Thread 0 after writing a value to the writebuffer makes the Writeflag =1, while thread 1 reads the value and makes the Writeflag =0.

Readflag: It is a byte in the shared memory address space at memory location 0x3F800002. Thread 1 after writing a value to the Readbuffer makes the Readflag =1, while thread 0 reads the value and makes the Readflag=0.

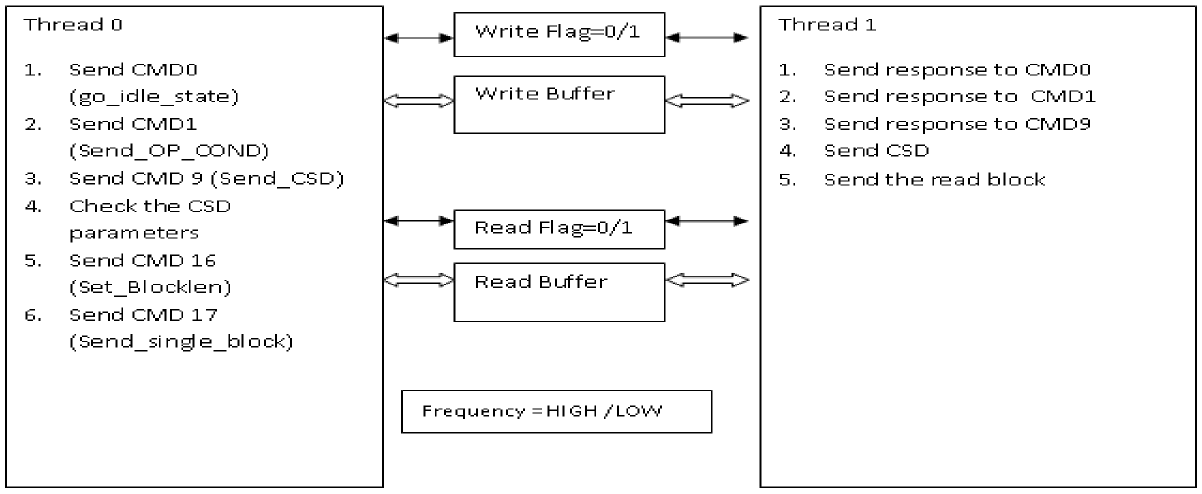


Figure 4: Block diagram of the system model in the PRET environment.

Buffers: Buffers are used to exchange data between the threads.

Writebuffer: It emulates the DataOut line on the SPI bus. It is a byte in the shared address space at memory location 0x3F800003.

Readbuffer: It emulates the DataIn line on the SPI bus. It is a byte in the shared address space at memory location 0x3F800004.

Frequency variable: The MMC card is required to be programmed at a frequency less than 400 KHz before power up. After power up the frequency can be increased to 20MHz. Hence to use these two frequencies, the frequency variable is set to LOW (400 KHz) or HIGH (20MHz).

Implementation of frequency: As Clock is not used; delays have been used to realize the required read/write frequencies. In a Real time SPI system the data written in and read from the buffer (SSPBUF) in terms of bytes, the time required to write a single byte is considered. An SPI system working at a frequency of 400 KHz will write a byte at a frequency of 50 KHz. Therefore a read write frequency of 50 KHz is achieved as follows.

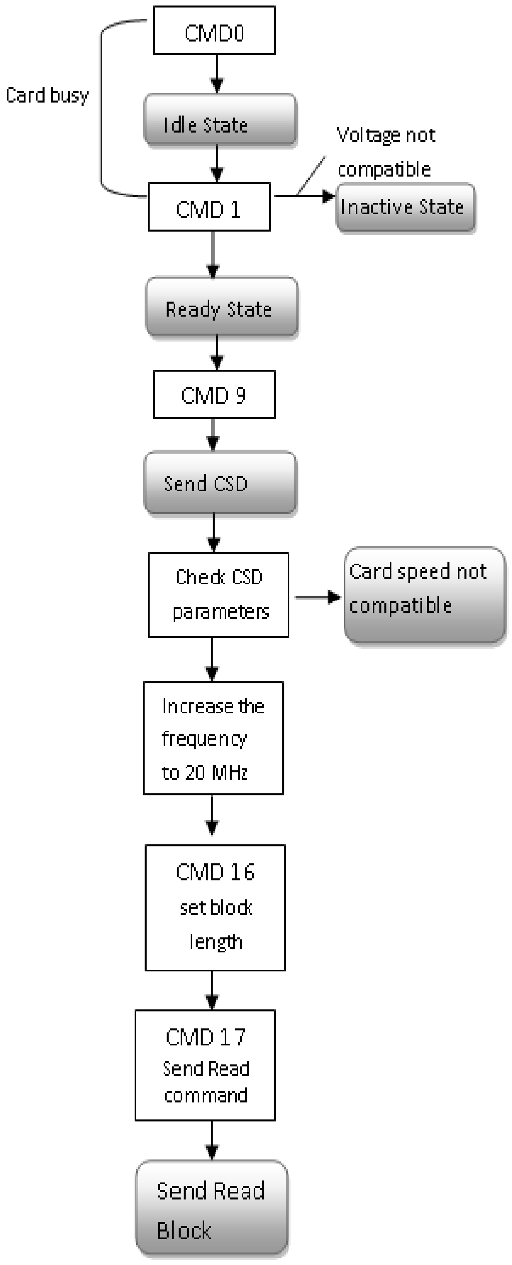
```
if (Frequency_MMC=LOW){
 DEAD (864);
 DEAD (0);
}
```

The value of the DEAD instruction is calculated as follows. The PRET processor working at a frequency of 250MHz allocates a frequency of 41.66 MHz to each thread. The delay required is 41.66 MHz/50 KHz ≈864.

Similarly to achieve a frequency of 20MHz (i.e. byte frequency of 2.25Mz) the delay required is 41.66MHZ/2.25MHz≈17.

```
if (Frequency_MMC=HIGH){
 DEAD (17);
 DEAD (0);
}
```

VI. Sequence of Instructions:



Commands:
CMD0: Go to Idle state command
CMD1: Send O/P conditions
CMD9: Send Card Specific Data
CMD16: Set Block Length
CMD 17: Read Block

a) Initialization Sequence:

When powered up the MMC wakes up in MMC mode. Every time CMD 0 is sent the card samples the CS signal. It enters SPI mode when the CS signal is asserted during the reception of the reset command CMD0 and sends a response.

On receiving the response for CMD 0, CMD 1 is sent. CMD1 is a special synchronization command used to negotiate the operation voltage range and to poll the card until it is out of its power-up sequence. The card responds with the busy bit in its response cleared once it is out of its power up sequence.

b) Reading the Card Specific Data (CSD) register:

Every MMC card has a CSD register which is hardcoded with values by the manufacturer.

It is 16 byte register which provides information about the transfer speed, card size, read block size, write block size and various other parameters. We read the CSD register to retrieve timing related information.

Register	Width	CSD Slice
TAAC	8	[119:112]
NSAC	8	[111:104]
TRANS_SPEED	8	[103:96]
R2W factor	3	[28:26]

TAAC: Defines the asynchronous part of the data access time.

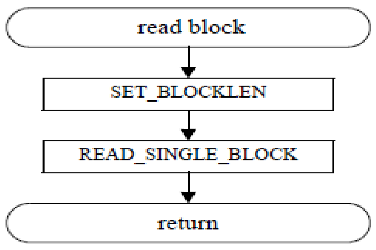
NSAC: Defines the typical case for the clock dependent factor of the data access time.

Worst case Read Access time= 10 * (TAAC *FOP + 100 * NSAC)

TRANS_SPEED: Defines the clock frequency supported by the card. For cards supporting version 4.0, and higher, of the specification, the value shall be 20MHz. Therefore we do not communicate with cards that support a maximum frequency less than 20MHz.

R2W_FACTOR: Defines the typical Write block time in multiples of the Read access time.

c) Reading a block



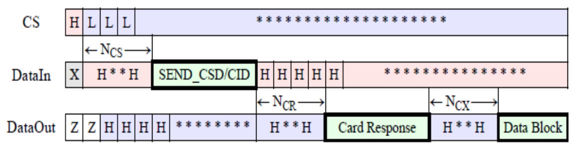
The procedure starts by setting the required block length with the SET_BLOCKLEN (CMD16) command. If the card accepts this setting, the data block is transferred via command READ_SINGLE_BLOCK (CMD17), starting at the given address

VII. Timing specification:

- CMD 0 and CMD 1

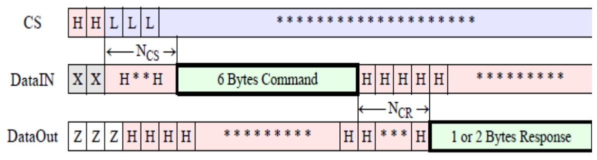
After power is applied thread 0 needs to wait for the supply ramp up time before sending CMD 0. After the card enters idle state, thread 0 can send CMD1. The worst case response time for CMD1 is 1 second. Therefore thread 0 polls for the response till 1 second. Once a valid response is received from the MMC card (thread 2) the MMC card is said to be initialized.

- CMD 9


CS
DataIn
DataOut
H L L L
N_{CS}
X H**H SEND_CSD/CID H H H H H
N_{CR}
N_{CX}
Z Z H H H H ******** H**H Card Response H**H Data Block

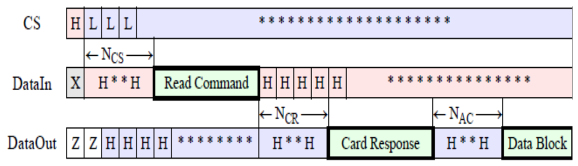
After sending the SEND_CSD command the response is received after time N_{CR} and the data block is received after time N_{CX}.

- CMD 16


CS
DataIN
DataOut
H H L L L
N_{CS}
X X H**H 6 Bytes Command H H H H H
N_{CR}
Z Z Z H H H H ********* H H *** H 1 or 2 Bytes Response

The MMC card responds to the SET_BLOCK_LEN command after time N_{CR}.

- CMD 17


CS
DataIn
DataOut
H L L L
N_{CS}
X H**H Read Command H H H H H
N_{CR}
N_{AC}
Z Z H H H H ******** H**H Card Response H**H Data Block

The card responds to the block read command by sending a response after time N_{CR} followed by the required after time N_{AC}.

Time N_{CS} is not considered as the CS signal is kept asserted (Low) all the while.

Symbol	Min	Max	Unit
N_{CR}	1	8	8
N_{CX}	0	8	8
N_{AC}	1	(10/8) * (TAAC *FOP + 100 * NSAC)	8

For every command thread 1 polls the Readbuffer for response or data from thread 2. In order to have time predictability we poll for the response till the worst case time (i.e. the Max values in the above table).

VIII. Implementation:

```
int get_response (volatile unsigned char buff, int
count) {
   int i;
      int Responseflag=1;
    for (i=0;i<count;i++) // poll for worst case time
    { delay ();
      if (*Readflag==0)
        {
          if ((char)*Readbuffer!=Bus_high)
          {
            if (*Readbuffer== (char)buff){
            printf ("DESIRED RESPONSE RECEIVED \n");
            Responseflag=0;
            }

          }
          *Readflag=1;
        }
    }
    return Responseflag;
   }
```

IX. Problems Faced:

As software models of the MMC card are not available, MMC response was generated by using one of the threads of the PRET processor. As no SPI hardware engine is currently present in the PRET processor, the SPI module has been abstracted keeping in mind the timing of the MMC response for various commands.

X. Conclusion:

An application for reading a block of 512 bytes from an MMC card with time predictability was simulated. As we wait for the worst case time delay in the response, predictability is achieved at cost of performance.

XI. References:

[1] B. Lickly, I. Liu, S. Kim, H. Patel, S. Edwards and E. Lee. Predictable Programming on a Precision Timed Architecture. In Proceedings for Conference on Compliers, Architecture and Synthesis of Embedded Systems (CASES ’08), Atlanta, Georgia,USA, October, 2008.
[2] JEDEC standard MultiMediaCard (MMC) Electrical Standard, Standard Capacity (MMCA, 4.1).
[3] PIC18FXX8 Data Sheet