

1. General description

The chip provides a built-in 8051 CPU with USB interface to handle three types of flash memory: NAND-type, SPI and NextFlash. It can co-operate with Digital Audio processing chip to manage the flash memory and to communicate with the PC.

2. Feature

- Build-in 8032 micro-controller and 1K bytes Data RAM
- Support USB bus with built-in transceiver
- Four USB pipes supported, including default control pipe, bulk-in pipe, bulk-out pipe and interrupt pipe
- Support 2M / 4M / 8M x 8 bit NAND-type Flash memory
- Hardware generated ECC code for flash memory access
- Support SPI and NextFlash serial interface
- Support SPCA751 interface
- 100-pin LQFP

Serial ports

The micro-controller does not implement serial port framing error detection and does not implement slave address comparison for multiprocessor communications.

Timer 2

The micro-controller does not implement timer2 down-counting mode or the down-count enable bit(T2MOD, bit0). The timer2 overflow output is active for one clock cycle. In the DS80C320, the timer2 overflow output is a square wave with a 50% duty cycle.

Watchdog timer

The micro-controller does not implement an internal watchdog timer.

Power fail detector

The micro-controller does not implement an internal power fail detector.

Stop mode

The micro-controller internal cycle counter is reset in stop mode. The micro-controller exits stop mode only when reset.

Timed access protection

The micro-controller does not implement timed access protection.

3.3. USB Pipes

SPCA514A supports the following four USB pipes.

Default pipe (EP0): process the standard and vendor commands.

INTERRUPT-IN pipe (EP1): transmit device events (interface 0)

BULK-IN pipe (EP2): upload data to the PC (interface 0)

BULK-OUT pipe (EP3): download data and ROM code (interface 0)

All standard commands, except the Get Descriptor command, are processed by hardware. For Get Descriptor command and vendor commands, the USB controller latches

the 8-byte commands in the EP0 FIFO and interrupts the micro-controller. The micro-controller then read the 8-byte command, decodes it, and prepares the appropriate data according to the command if necessary. The data, if any, is then sent to the USB bus by the USB controller.

USB Packet Format

USB Vendor Command for Register Read/Write (for example)

Command	bmReqType	BRequest	wValue	WIndex	wLength
Read	0C1	0x00	reserved	address	1
Write	0x41	0x00	High byte : reserved low byte: write value	address	0

USB Bulk-IN Packet Format

The maximum packet size is fixed at 64 bytes. The size of each Bulk-IN packet is either maximum or zero. The zero-padding is applied to the last packet to make the size maximum if necessary.

USB Bulk-OUT Packet Format

The maximum packet size is fixed at 64 bytes. The size of each Bulk-OUT packet must be maximal.

USB Interrupt-IN Packet Format

The maximum packet size is fixed at two bytes. The micro-controller must program the interrupt pipe registers (both register 0x8506 and 0x8507) after it detects a new event. The USB controller sends the interrupt data to the host only after register 0x8507 is written.

3.4. Flash Memory

The flash memory is used to store data. The CPU can transfer command, address and data to the flash memory by the 8-bit I/O port. There are three operation modes for the CPU to read/write the flash memory: one is the direct mode, another is the FIFO mode and the other is pseudo DRAM mode. The ECC that is 22-bit code for every 256 bytes will

be generated in the FIFO mode. The ECC generated by hardware can be read from the registers (3 bytes for 256 bytes/page and 6 bytes for 512 bytes/page). The read/write operation sequence is described as follows.

1. set the flash memory chip enable
2. set the flash memory command enable (0X8400)
3. write command to the flash memory via the flash memory data register (0X8400)
4. clear the flash memory command enable
5. set the flash memory address enable
6. write address to the flash memory via the flash memory data register (0X8400)
7. clear the flash memory address enable
8. wait the flash memory ready

Direct mode:

9. read/write data from/to the flash memory via the flash memory data register (0X8400)
10. read/write additional data from/to the flash memory via the flash memory data register (0X8400)

FIFO mode:

9. read/write data from/to the flash memory via the post buffer data register (0X8300)
10. read the ECC generated by hardware from the ECC registers
11. read/write additional data from/to the flash memory via the flash memory register (0X8400)

Pseudo DMA mode:

9. read the post buffer and write to the flash memory or read the flash memory and write to the post buffer via the post buffer data register (0X8300)
10. read the ECC generated by hardware from the ECC registers
11. read/write additional data from/to the flash memory via the flash memory register (0X8400)

P.S.

1. The additional data means the data stored in the last 8 or 16 bytes of a page in the flash memory. The size is 8 or 16 bytes based on that the page size is 256/512 bytes.

2. The flash memory data access must be through by the CPU because the flash memory data bus is directly connected with the CPU data bus. Therefore there are two type pseudo DMA operations to access flash memory data to post buffer:

- (1) The CPU reads the data register of the post buffer (0X8300) and the hardware generates a write pulse to flash memory in the same time. The data flow is from post buffer to flash memory.
- (2) The CPU reads the data register of the post buffer and the hardware generates a read pulse to flash memory in the same time. The data flow is from flash memory to post buffer.

3. The CPU accesses flash memory data via the register (0X8400) in the direct mode. The ECC is not generated by hardware in this mode. Therefore all accesses for the flash memory that do not want to disturb the ECC calculation must be in this mode, for example the additional data access.

3.5. Post buffer

3.5.1. Buffer Control

There are two 1k-byte deep FIFOs in the device to concurrently handle both incoming and outgoing data stream in various operation modes and to easily handle data for the USB host controller. Some detailed information about the FIFO is shown as follows.

OprMode	Source	Destination	FIFO Size	Note
Idle	None	None		Idle mode
Upload 1	CPU (0X8300)	Flash memory	The page size of flash memory	FIFO mode
Upload 2	Flash Memory	CPU (0X8300)	The page size of flash memory	FIFO mode
Upload 3	Flash Memory	USB	The page size of flash memory	Pseudo DMA (2)
Upload 4	CPU (0X8300)	USB	64 bytes	Bulk IN pipe
Upload 5	USB	CPU (0X8300)	64 bytes	Bulk OUT pipe

Upload 6	USB	Flash memory	The page size of flash memory	Bulk OUT pipe Pseudo DMA (1)
Upload 8	CPU (0X8300)	CPU (0X8300)	64 bytes	Test mode

3.5.2. Upload Mode (1):

When SPCA514A wants to record sound into flash memory, the “OprMode” field in the register (0X8301) must be set to the value of 0X1. In this mode, the CPU will read and process the audio data and then write into flash memory by the FIFO mode. The deep of post buffer is the same as the page size of flash memory in this mode.

3.5.3. Upload Mode (2):

When SPCA514A wants to read data from flash memory and to transfer through the CPU, for example RS-232 port, the “OprMode” field in the register (0X8301) must be set to the value of 0X2. In this mode, the CPU will read and process data from flash memory by the FIFO mode. The deep of post buffer is the same as the page size of flash memory in this mode.

3.5.4. Upload Mode (3):

When SPCA514A wants to upload data from flash memory to the PC through the USB bus, the “OprMode” field in the register (0X8301) must be set to the value of 0X3. In this mode, The data in flash memory will be read by the CPU and written into post buffer by the pseudo DMA mode (2). The data in the post buffer will be read through the Bulk-IN pipe by the USB host controller. The deep of post buffer is the same as the page size of flash memory in this mode.

3.5.5. Upload Mode (4):

When the USB host wants to do loop-back test, the “OprMode” field in the register (0X8301) must be set to the value of 0X4. The forward path is from the PC through Bulk-OUT pipe to the CPU and the backward path is from the CPU through Bulk-IN pipe to the PC. The backward path is supported in this mode and the forward path is supported in the next mode, Upload Mode (5). The deep of post buffer is 64 bytes in this mode.

3.5.6. Upload Mode (5):

When the USB host wants to update the ROM code for the CPU on flash memory, the “OprMode” field in the register (0X8301) must be set to the value of 0X5. The new ROM code is transmitted into post buffer from the PC through the Bulk-OUT pipe. Then the CPU reads the post buffer and update the ROM code on the flash memory. The deep of post buffer is 64 bytes in this mode.

3.5.7. Upload Mode (6):

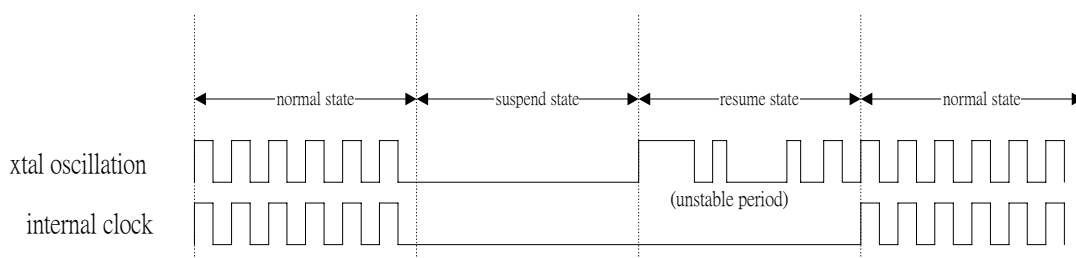
When the USB host wants to pseudo write into flash memory by the CPU, the “OprMode” field in the register (0X8301) must be set to the value of 0X6. The data is transmitted into post buffer from the PC through the Bulk-OUT pipe. Then the CPU reads the post buffer and write into the flash memory. The deep of post buffer is 256 bytes in this mode.

3.5.8. Upload Mode (8):

This is a test mode. In this mode, the “OprMode” field in the register (0X8301) must be set to the value of 0X8. The CPU will write, read back and compare data through post buffer and the deep of post buffer is 64 bytes in this mode.

3.6. Power Control

To reduce power-consumption during suspend period, the clock pad is disabled during suspend state. All the internal clocks stop and the SPCA514 can only accept asynchronous events during the suspend state. Before the SPCA514 returns to normal operational state, there is an additional state called resume state. During the resume state the clock pad start oscillation. However, the internal clocks are still masked out to prevent chip malfunction since during this period the clock is not stable. The following diagram depicts the sequence for the SCAP514 to enter and get out of the suspend/resume state.



From normal to suspend state:

To enter suspend state, the firmware must program register 0X8002 bit 0. The SPCA514 will stop the crystal oscillation if this bit is set.

From the application's point of view, there are two possible causes to put the chip into suspend state. The first one is the USB suspend. After the USB bus is idle for 6 ms, the SPCA514 will issue an interrupt to inform the micro-controller. Then the micro-controller must program the bit 0 of register 0X8002.

The other cause to enter suspend state is that when the UI is idle longer than a period of time. This part is completely implemented by the firmware. The SPCA514 hardware has no interrupt for this. Like the USB suspend, if the firmware decide the UI idle time is long enough, it must program bit 0 of register 0X8002 to put the chip into suspend state.

From suspend to normal state

As stated above, before the SPCA514 returns to normal state from suspend state, it must goes through the resume state. There are two possible causes for the SPCA514 to enter resume state from suspend state.

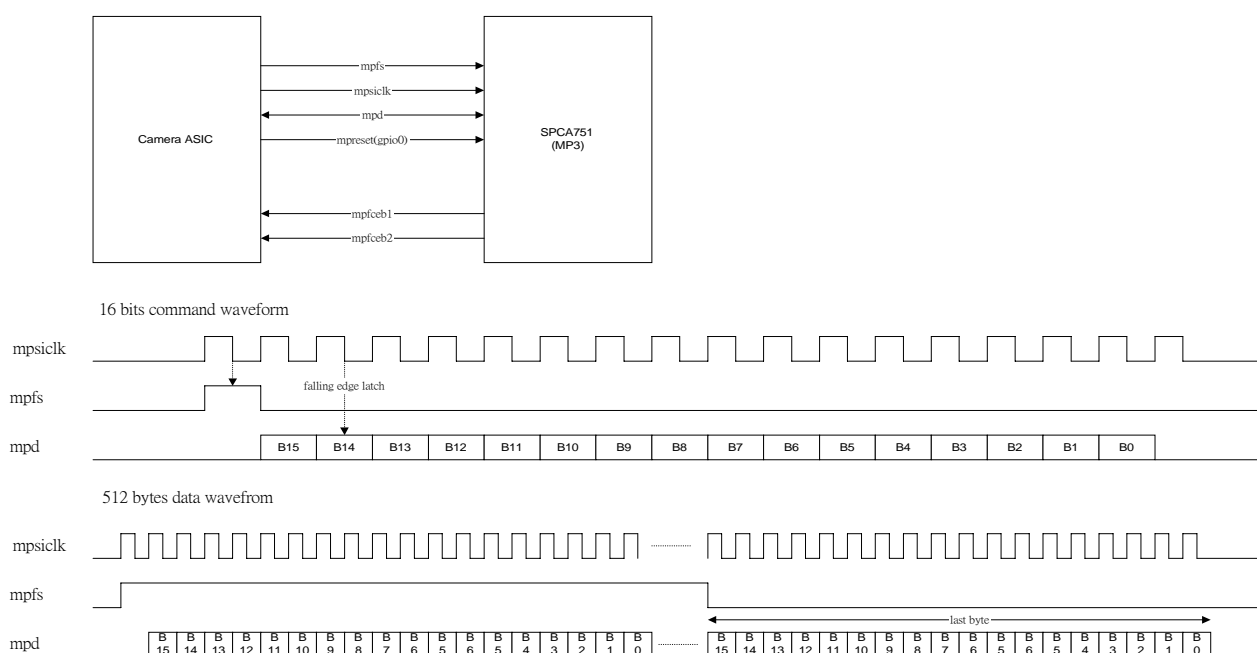
- (1) When bit 4 of register 0X8003 is set and the USB controller detects a bus activity on the USB bus.
- (2) When bit 5 of register 0X8003 is set and the SPCA514 detects a key status change via GPIO pins.

All the two causes are processed by SPCA514 automatically, no firmware intervention needed. Since the clock is stopped during suspend, it's not possible for the micro-controller to do anything about it. The length of internal resume period can be programmed from 1 ms to 20 ms via bit[3:0] of register 0X8003.

3.7. SPCA751 Interface

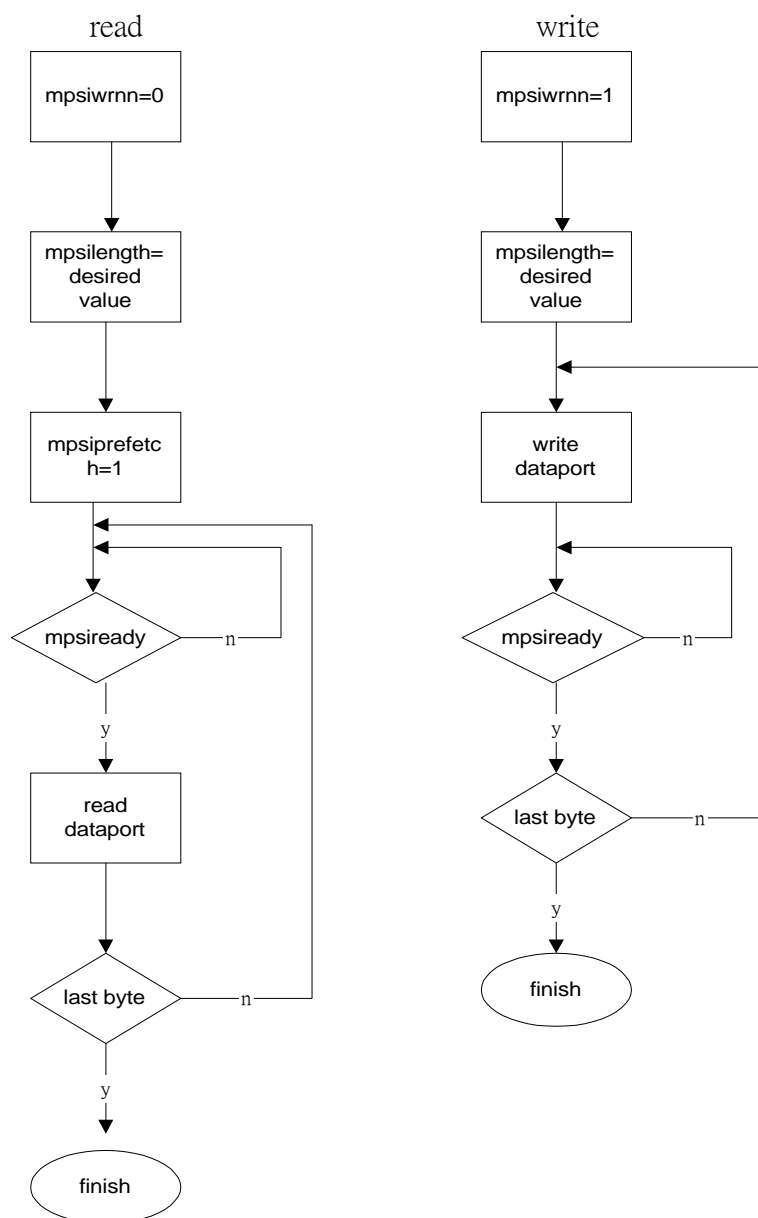
The SPCA514 has a serial interface to communicate with SPCA751. SPCA751 is an Digital Audio processor (DAP). It also supports audio recording and compression. Integrating SPCA751 with SPCA514 results in a complete Digital Audio player and a digital recorder. While operating as an Digital Audio player, the SPCA514 fetch the Digital Audio bit stream from the flash memory and send it to the SPCA751 for playback. While operating as an digital recorder, the SPCA751 record and compress the audio data and send the data to SPCA514. Then, the SPCA514 stores the compressed audio data into the flash memory. Both of the operation is done via the serial interface between SPCA514 and SPCA751. In both cases, the SPCA514 acts as a master. It sends commands, and data if necessary, to the SPCA751. It also requests data from the SPCA751.

The following diagram shows how the SPCA514 communicate with SPCA751. Each data transaction to the SPCA751 starts with a TX frame sync signal (mptxf), and the de-assertion of the frame sync signal indicates the transfer of the last word. Each data transaction from the SPCA751A starts with an RX frame sync (mprxf), and the signal is de-asserted at the last word of data transfer. Notes that the communication between SPCA514 and SPCA751 is based on word unit. The serial clock is driven by the SPCA514. Both SPCA514 and SPCA751 sample the data at the falling edge of the serial clock.



Programming flow of the Digital Audio processor serial interface

Communication with the MP3 processor



3.8. SPI interface

3.8.1 SPI interface to the Serial flash memory

The SPCA514A supports an SPI serial interface to access the SPI-type serial flash memory. Both SPI mode 0 and SPI mode 3 are supported. The serial clock frequency is adjustable from 12 MHz to 3 MHz. The CPU writes data to the serial flash memory via the TXport (register 8230) and read data from the flash memory via the RXport and PRXport. Reading data from the PRXport will invoke a sequence of serial clocks to pre-fetch the next byte of data. Reading data from the RXport merely gets the data that is already received and latched in the SPCA514A. Each read sequence starts with a dummy read to the PRXport, followed by multiple read from the PRXport, and finally end up with a read to the RXport. Note that the data read by the first dummy read to the PRXport should be discarded. Each time before the CPU write to the TXport or read from the PRXport, the CPU must wait until the SPI interface is ready. This could be achieved by polling the status bit (FMSIbusy, register 0X823B bit 0) each time before read or write.

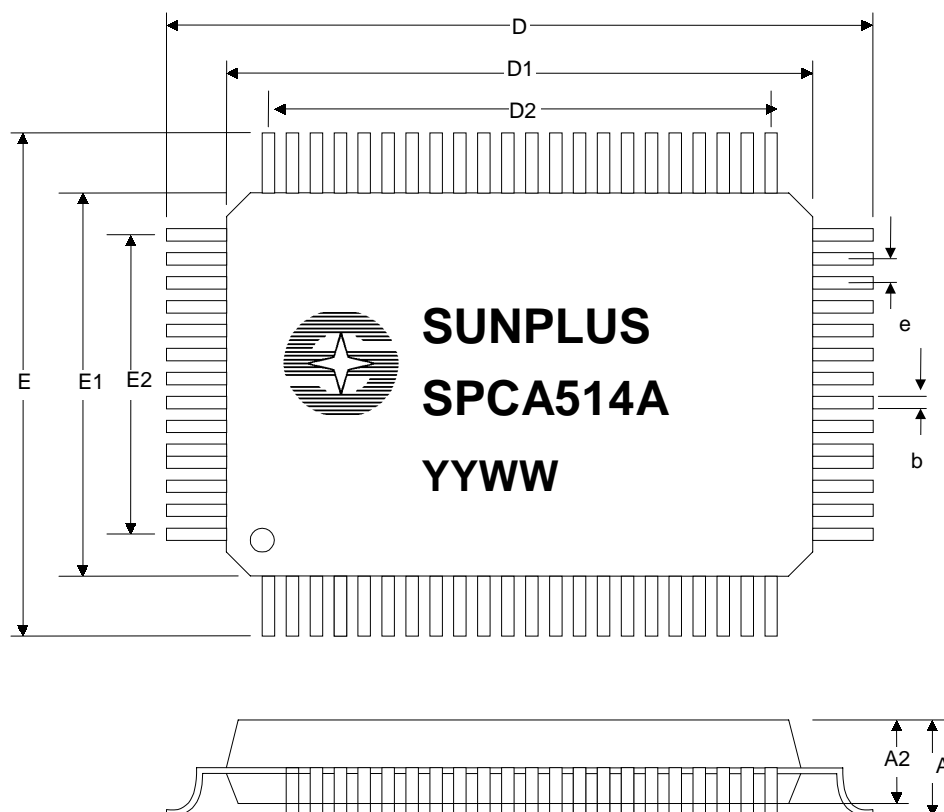
When data is written into the spi-interface serial flash memory or read from them, a 16 bit CRC codes are generated automatically by the SPCA514A. For every 256 bytes of data, 16 bits of CRC codes are generated (access inverse of low byte CRC code via register 0x82b6, inverse of high byte CRC code via register 0x82b7). The CRC codes are generated automatically when the CPU read/write the data port. The CRC code can be cleared each time a new page is written to or read from the spi-interface serial flash memory. To clear the CRC code, write 1 to register 0X82a5 bit0.

3.8.2 Next flash serial interface control

For the Next flash serial interface, the CPU has to control the output enable of the serial data bit because the data pin is bi-directional. Also, the CPU has to explicitly start and stop the transfer in order to control the status of the clock pin. There are 2 extra ports for the CPU to read. The first one is PRX1 port. After this port is read, the SPCA514A assert a signal clock and force the clock pin stay at high state. The second port is PRX7 port. After this port is read, the SPCA514A will assert the next seven clocks to complete the byte read. This special timing is required by the Next flash memory. The CPU should wait for 30 to 100 us between reading PRX1 and PRX7. Please reference to the Next flash data sheet for the timing requirement.

Preliminary 2000-11-21

Package dimension



Symbol	Min.	Nom.	Max.
A	-	-	3.4
A2	2.5	2.72	2.9
E	17.20	17.20	17.20
E1	14.00	14.00	14.00
E2	12.35	12.35	12.35
D	23.20	23.20	23.20
D1	20.00	20.00	20.00
D2	18.85	18.85	18.85
e	0.65	0.65	0.65
b	0.20	0.30	0.38

unit : millimeter