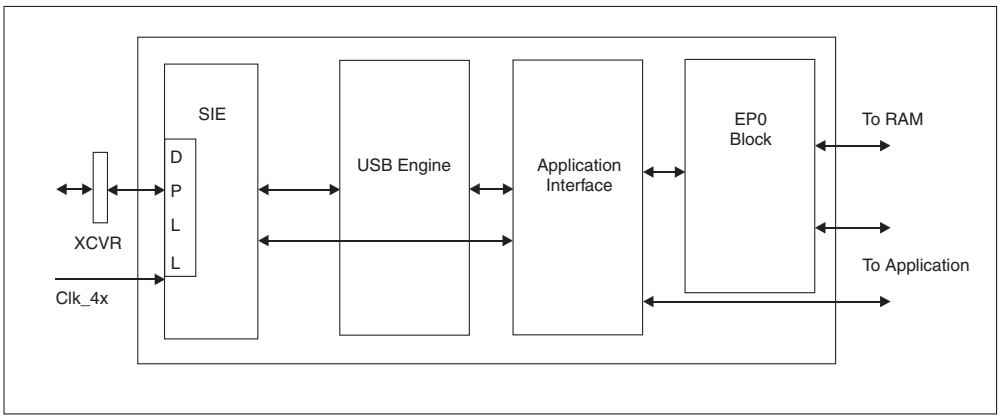
Features

- **USB 1.1-compliant**
- **Approximately 6.5K Gates**
- **Supports 12 Mbps Transfers Only**
- **Functionally Compatible with Open HCI and UHCI**
- **Modular Design for Ease of Integration**
- **Supports Control, Interrupt, Bulk and Isochronous Transfers**
- **Four Configurable Endpoints**
- **Complete Error Handling Capability**
- **Automatic Data Retry in Hardware**
- **Flexible Application Interface**
- **Supports Power Management**

Block Diagram



Overview

The ATUSBFUNC-SS7211 is a fully synthesizable core that can be implemented in any Atmel ASIC library (gate array or standard cell). The core is supported by a comprehensive USB test environment (ATUSBTEST-SS7400) that can be used to verify the entire design, including the application. The USB Function Core can be used in any high-speed (12 Mbps) application, such as a printer, camera or scanner. The core is configured before synthesis to respond to all standard USB hub/host commands. Up to four endpoints are supported. The interface to the application consists of an application interface and a FIFO interface. There are no user programmable registers in this core. The internal state machine and control logic decode the hub/host commands and control the data transfers across the FIFO interface. Control signals are used to transfer error and status information to or from the application.

USB Function Core High-speed, 12 Mbps

ATUSBFUNC-SS7211

Summary

Rev. 1668AS–07/01

Note: This is a summary document. A complete document is available under NDA. For more information, please contact your local Atmel sales office.

Description

There are four major blocks in the USB Function Core: the Serial Interface Engine (SIE), the USB Engine, the Application Interface and the EP0 Block.

SIE

SIE interfaces with a standard USB transceiver on the line side. A DPLL is used to extract the clock from the received data stream. SIE converts the received serial data stream into parallel bytes and delivers it to the USB Engine. In the transmit mode, it converts the parallel bytes from the USB Engine into serial data stream and sends it over to the transceiver. As part of the USB protocol, it performs sync detect, bit stuffing/unstuffing, PID decode, NRZI encoding/decoding, CRC checking/generation for token and data packets. It also monitors the line to detect a reset, End of Packet (EOP), Start of Packet (SOP) and idle condition. In transmit mode, it generates SOP, EOP and Resume signaling. SIE keeps track of the byte boundaries.

USB Engine

The USB Engine keeps track of a transaction from SOP to EOP. On SOP, it checks the validity of the address and endpoint and initiates an appropriate data transaction based on the status of the endpoint FIFOs. It handles the data retry mechanism, using data toggling, and generates appropriate handshakes.

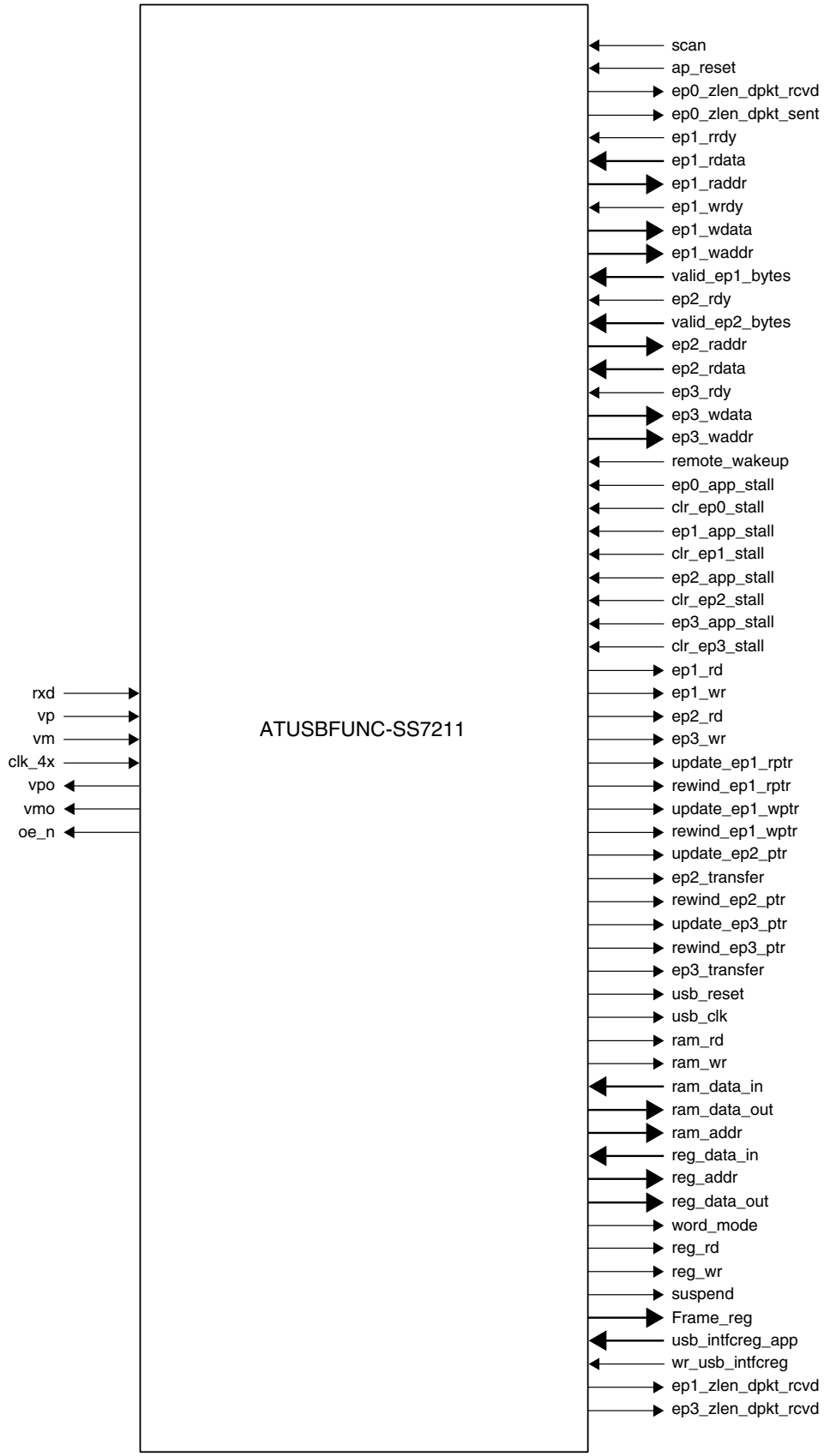
Application Interface

The Application Interface provides a simple mechanism to interface to the user logic. This interface allows direct access to all the endpoint FIFOs except for the control endpoint EP0. This block generates the signals for all other endpoint transfers and provides access to their respective FIFOs. For example, a bulk in transfer can be done to EP2 and setting, clearing of stall conditions are controlled by the application. EP1, EP2 and EP3 endpoints are controlled independently. This allows simultaneous access to these endpoints.

EP0 Block

All the transfers to the EP0 endpoint (control transfers) are handled by this block. The USB commands are decoded and either the memory (external RAM) or registers are accessed. Standard-, class- and vendor-specific commands are decoded, and a simple bus is provided to access all the registers. A RAM interface in this block accesses the configuration data.

Pinout


ATUSBFUNC-SS7211
rxd
vp
vm
clk_4x
vpo
vmo
oe_n
scan
ap_reset
ep0_zlen_dpkt_rcvd
ep0_zlen_dpkt_sent
ep1_rrdy
ep1_rdata
ep1_raddr
ep1_wrdy
ep1_wdata
ep1_waddr
valid_ep1_bytes
ep2_rdy
valid_ep2_bytes
ep2_raddr
ep2_rdata
ep3_rdy
ep3_wdata
ep3_waddr
remote_wakeup
ep0_app_stall
clr_ep0_stall
ep1_app_stall
clr_ep1_stall
ep2_app_stall
clr_ep2_stall
ep3_app_stall
clr_ep3_stall
ep1_rd
ep1_wr
ep2_rd
ep3_wr
update_ep1_rptr
rewind_ep1_rptr
update_ep1_wptr
rewind_ep1_wptr
update_ep2_ptr
ep2_transfer
rewind_ep2_ptr
update_ep3_ptr
rewind_ep3_ptr
ep3_transfer
usb_reset
usb_clk
ram_rd
ram_wr
ram_data_in
ram_data_out
ram_addr
reg_data_in
reg_addr
reg_data_out
word_mode
reg_rd
reg_wr
suspend
Frame_reg
usb_intfcreg_app
wr_usb_intfcreg
ep1_zlen_dpkt_rcvd
ep3_zlen_dpkt_rcvd

Transceiver Signals

vp **In**

This signal along with vm is used to identify signaling on the bus. (See Note 1.)

vm **In**

This signal along with vp is used to identify signaling on the bus. (See Note 1.)

rxd **In**

Receive data. The receive data from the upstream port is seen on this pin.

vpo **Out**

This signal along with vmo is used to represent logic “1” or logic “0”. (See Note 2.)

vmo **Out**

This signal along with vpo is used to represent logic “1” or logic “0”. (See Note 2.)

oe_n **Out**

Asserted when transceiver is in the transmit mode.

Note 1: Decode for vp and vm

vp	vm	Signaling
0	0	SE0
0	1	Full Speed
1	0	Low Speed
1	1	Error

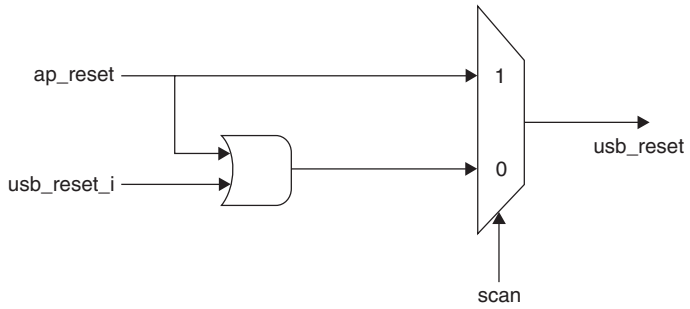
Note 2: Decode for vpo and vmo

vpo	vmo	Signaling
0	0	SE0
0	1	Logic 0
1	0	Logic 1
1	1	Illegal

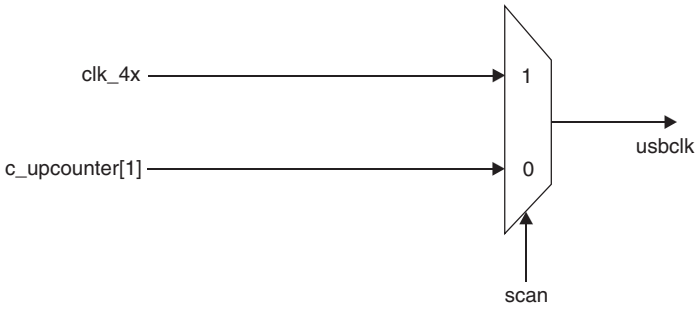
Application Signals

scan **In**

A scan signal is used to select between the application and full scan mode. Logic “1” selects the scan mode and logic “0” selects the application mode. For scan mode, the signal ap_reset is the global reset. In scan mode, the signal ap_reset is connected to usb_reset. This logic is implemented in the *rx_signaling.v* file. The signal usb_reset is used as a reset in all other modules.



For scan mode, the signal clk_4x is global clock. In scan mode, the signal clk_4x is connected to usb clk. This logic is implemented in the *dpll.v* file. The signal usb clk is used as a clock in all other modules. The usb_clock and ap_clk are the same.



clk_4x **In**

A 4x clock is used by DPLL to extract the clock from the receive data stream. This clock is 48 MHz for full-speed, 12-Mbps USB transfer.

ap_reset **In**

Application reset. All the application-specific logic is reset when this signal is asserted.

ep0_zlen_dpkt_rcvd Out

This signal gives the status of the EP0 zero-length data packet received by the application.

ep0_zlen_dpkt_sent Out

This signal gives the status of the EP0 zero-length data packet sent by the application.

ep1_zlen_dpkt_rcvd Out

This signal gives the status of the EP1 zero-length data packet received by the application.

ep3_zlen_dpkt_rcvd Out

This signal gives the status of the EP3 zero-length data packet received by the application.

ep1_rrdy In

When asserted, the EP1 endpoint (FIFO) has been filled with the new application data. The data will be provided to the host during the next host-in transfer to EP1.

ep1_wrdy In

Assertion of this signal indicates the EP1 endpoint is available for the host data. The data from the previous host-out transfer to EP1 has been successfully consumed.

ep2_rdy In

When asserted, the EP2 endpoint (FIFO) has been filled with the new application data. The data will be provided to the host during the next host-in transfer to EP2.

ep3_rdy In

Assertion of this signal indicates the EP3 endpoint is available for the host data. The data from the previous host-out transfer to EP3 has been successfully consumed.

ep1_rdata[7:0] In

The EP1 endpoint read data bus. When the ep1_rd signal is asserted, the EP1 FIFO is read using this bus. The endpoint is addressed by ep1_raddr[2:0].

ep1_wdata[7:0] Out

The EP1 endpoint write data bus. When the ep1_wr signal is asserted, the EP1 FIFO is written using this bus. The endpoint is addressed by ep1_waddr[2:0].

valid_ep1_bytes[3:0] In

The EP1 endpoint valid number of bytes bus. This gives the valid number of bytes in EP1 FIFO. The maximum value of valid_ep1_bytes is eight. If the value of the valid_ep1_bytes is greater than eight, the fifo_ctl block treats valid number of bytes as eight only. If the valid number of bytes is zero, ap_interface sends the zero length data packet to host.

valid_ep2_bytes[10:0] In

The EP2 endpoint valid number of bytes bus. This gives the number of valid bytes in EP2 FIFO, as well as the valid number of bytes that are to be read from EP2 FIFO. The maximum value of valid_ep2_bytes in bulk mode is 64, in isochronous mode 1024. If the value of the valid_ep2_bytes is greater than 64 in bulk mode, the fifo_ctl block treats valid number of bytes as 64 only. If the value of the valid_ep2_bytes is greater than 1024 in isochronous mode, the fifo_ctl block treats valid number of bytes as 1024 only. If the valid number of bytes is zero, ap_interface sends the zero length data packet to host. The mode is specified with the signal usbintfc_reg[0].

ep2_rdata[7:0] In

The EP2 endpoint read data bus. When the ep2_rd signal is asserted, the EP2 FIFO is read using this bus. The endpoint is addressed by ep2_raddr[5:0].

remote_wakeup In

The application asserts this signal to wake up the device from the suspend mode. When SIE detects idle for more than 3 ms, the suspend signal is asserted. The application, which has the remote wake-up capability, asserts this signal; otherwise, it is deasserted.

ep0_app_stall In

When asserted, the application stalls the EP0 endpoint transactions. Any host transfers to this endpoint will be returned with STALL. This condition needs host intervention. A CLEAR STALL command to EP0 clears the stall condition.

clr_ep0_stall In

When the host issues a CLEAR STALL command for EP0, the application clears the EP0 endpoint stall by asserting this signal.

ep1_app_stall In

When asserted, the application stalls the EP1 endpoint transactions. Any host transfers to this endpoint will be returned with STALL. This condition needs host intervention. A CLEAR STALL command to EP1 clears the stall condition.

clr_ep1_stall In

When the host issues a CLEAR STALL command for EP1, the application clears the EP1 endpoint stall by asserting this signal.

ep2_app_stall In

When asserted, the application stalls the EP2 endpoint transactions. Any host transfers to this endpoint will be returned with STALL. This condition needs host intervention. A CLEAR STALL command to EP2 clears the stall condition.

clr_ep2_stall In

When the host issues a CLEAR STALL command for EP2, the application clears the EP2 endpoint stall condition by asserting this signal.

ep3_app_stall In

When asserted, the application stalls the EP3 endpoint transactions. Any host transfers to this endpoint will be returned with STALL. This condition needs host intervention. A CLEAR STALL command to EP3 will clear the stall condition.

clr_ep3_stall In

When the host issues a CLEAR STALL command for EP3, the application clears the EP3 endpoint stall by asserting this signal.

ep1_rd Out

Endpoint EP1 read strobe. When asserted, the data from the ep1_rdata[7:0] bus is sent to the host. The endpoint is addressed by ep1_raddr[2:0].

ep1_wr Out

Endpoint EP1 write strobe. When asserted, the data is written to the EP1 FIFO using the ep1_wdata[7:0] bus. The endpoint is addressed by ep1_waddr[5:0].

ep2_rd Out

Endpoint EP2 read strobe. The data from the ep2_rdata [7:0] bus is sent to the host when this signal is asserted. The endpoint is addressed by ep2_raddr[5:0].

ep3_wr Out

Endpoint EP3 write strobe. When asserted, the data is written to the EP3 FIFO using the ep3_wdata[7:0] bus. The endpoint is addressed by ep3_waddr[5:0].

reg_data_in[7:0] In

Register data in bus. All the application-specific registers are read using this bus. Information like endpoint status, configuration and interface is stored in the application registers. The register address and the read strobe are generated to the application by the USB Function Core; the application will provide the data in the same cycle.

usb_reset Out

Asserted when the reset signaling is detected on the USB line. Deasserted when the reset signaling is complete.

ep3_wdata[7:0] Out

Endpoint EP3 write data bus. Host data to EP3 endpoint is written from this bus. On ep3_rd, the USB Function Core will load the data to the EP3 FIFO. The FIFO is addressed by ep3_waddr[5:0].

reg_addr[7:0] Out

Register address. This bus addresses all the application registers. For example, the status, configuration and interface registers in the application register address space are addressed.

reg_data_out[15:0] Out

Register data out. On a control out transfer, the EP0 block decodes the setup data. Control accesses to the application will result in either an EPROM access to get and set DESCRIPTORS or application register accesses. Reg_data_out contains the data for the command, reg_addr has the corresponding address for this data. If word_mode is asserted, all the 16 bits contain valid information; otherwise, [7:0] is valid.

word_mode Out

In this mode, the data written out is 16 bits wide. When the USB Function Core writes registers in the application, it can write in 16-bit mode or 8-bit mode. For 16-bit mode, this signal is asserted. During 8-bit mode, this signal is deasserted, and the reg_data_out[7:0] is used to write the data.

reg_rd Out

Register read strobe. This signal is asserted during register read operations.

reg_wr Out

Register write strobe. Asserted during register write operations.

suspend Out

Asserted when SIE detects idle on the USB line for more than 3 ms. Deasserted when the line is active or when the device wants to send resume transfers.

ep1_raddr[2:0] Out

During the EP1 endpoint read access, the FIFO is addressed using this bus. The ep1_rd strobe is asserted when the data from the location address by ep1_raddr [2:0] is sent to the host on ep1_rdata[7:0].

ep1_waddr[2:0] Out

Endpoint EP1 is addressed using this bus during EP1 write access. The ep1_wr strobe is asserted when the data from the ep1_wdata[7:0] bus is written into the location addressed by this bus.

ep2_raddr[9:0] Out

Endpoint EP2 is addressed using this bus during EP2 access. The ep2_rd strobe is asserted when the data from the location address by ep2_raddr[9:0] is sent to the host on ep2_rdata[7:0]. In bulk mode, ep2_raddr should not be greater than 63 (0 to 63 address depth).

ep3_waddr[9:0] Out

Endpoint EP3 is addressed using this bus during EP3 access. The ep3_wr strobe is asserted when the data from the ep3_wdata[7:0] bus is written into the location addressed by this bus. In bulk mode, ep3_waddr is between 0 and 63.

update_ep1_ptr Out

A host-in transfer to EP1 endpoint sends the data from the EP1 FIFO to the host. Upon receiving an ack from the host, this signal is asserted to indicate that the previous interrupt data has been successfully sent to the host.

rewind_ep1_ptr Out

A host-in transfer to EP1 endpoint sends the data from the EP1 FIFO to the host. If the host does not send an ack, this signal is asserted to indicate that the previous interrupt data has not been successfully sent to the host.

update_ep2_ptr Out

A host-in transfer to EP2 endpoint sends the data from the EP2 FIFO to the host. Upon receiving an ack from the host, this signal is asserted to indicate that the application data has been successfully sent to the host.

ep2_transfer Out

This signal is asserted before EP2 data transfer takes place. This is an indication of EP2 data transfer. This signal is generated after verification of the token PID, CRC check for the token. If the token belongs to EP2 in endpoint and no CRC error, then this signal is generated for one clock cycle. This signal is issued before checking the data PID of that data packet.

rewind_ep2_ptr Out

A host-in transfer to EP2 endpoint sends the data from the EP2 FIFO to the host. If the host does not send an ack, this signal is asserted to indicate that the application data has not been successfully sent to the host.

update_ep3_ptr Out

A host-out transfer to EP3 endpoint fills the EP3 FIFO. Upon successfully receiving the data from the host, this signal is asserted to indicate that the application can use the host data.

rewind_ep3_ptr Out

A host-out transfer to EP3 endpoint fills the EP3 FIFO. If the receive data from the host has an error, this signal is asserted to indicate to the application to ignore the data.

ep3_transfer Out

This signal is asserted before EP3 data transfer takes place. This is an indication of EP3 data transfer. This signal is generated after verification of the token PID, CRC check for the token. If the token belongs to EP3 in endpoint and no CRC error, this signal is generated for one clock cycle. This signal is issued before checking the data PID of that data packet.

frame_reg [10:0] Out

This is an 11-bit frame register bus.

usb_clk Out

This is the USB extracted clock.

usb_intfcreg_app[1..0] In

When the wr_usb_intfcreg is asserted, usb_intfcreg (this signal is in the core) gets the value on usb_intfcreg_app.The usb_intfcreg bits are mode bits for EP2, EP3. If the bit is zero, the corresponding endpoint is in isochronous mode; otherwise, it is in bulk mode. usb_intfcreg_app[0] is for EP2 endpoint (BULK IN or ISO IN) and usb_intfcreg_app[1] is for EP3 endpoint (BULK OUT or ISO OUT). On reset, the EP2 and EP3 are in bulk mode.

wr_usb_intfcreg In

wr_usb_intfcreg write strobe. When asserted, the usb_intfcreg_app is sent to the usb_intfcreg, which decides the mode of the endpoint EP2, EP3.

Memory Interface

ram_wr Out

Memory write strobe. When this signal is asserted, the external RAM is in the write cycle.

ram_rd Out

Memory read strobe. When this signal is asserted, the external RAM is in the read cycle.

ram_data_in[7:0] In

During read cycle, the data from the memory is read from this bus.

ram_data_out[7:0] Out

During a write cycle, the data to the memory is written from this bus.

ram_addr[7:0] Out

The external RAM is addressed using this bus.

Atmel Headquarters

Corporate Headquarters
2325 Orchard Parkway
San Jose, CA 95131
TEL (408) 441-0311
FAX (408) 487-2600

Europe
Atmel SarL
Route des Arsenaux 41
Casa Postale 80
CH-1705 Fribourg
Switzerland
TEL (41) 26-426-5555
FAX (41) 26-426-5500

Asia
Atmel Asia, Ltd.
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimhatsui
East Kowloon
Hong Kong
TEL (852) 2721-9778
FAX (852) 2722-1369

Japan
Atmel Japan K.K.
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
TEL (81) 3-3523-3551
FAX (81) 3-3523-7581

Atmel Operations

Atmel Colorado Springs
1150 E. Cheyenne
Mtn. Blvd.
Colorado Springs, CO
80906
TEL (719) 576-3300
FAX (719) 540-1759

Atmel Rousset
Zone Industrielle
13106 Rousset Cedex
France
TEL (33) 4-4253-6000
FAX (33) 4-4253-6001

Atmel Smart Card ICs
Scottish Enterprise
Technology Park
East Kilbride, Scotland
G75 0QR
TEL (44) 1355-357-
000
FAX (44) 1355-242-
743

Atmel Grenoble
Avenue de Rochep-
leine
BP 123
38521 Saint-Egreve
Cedex
France
TEL (33) 4-7658-3000
FAX (33) 4-7658-3480

Fax-on-Demand
North America:
1-(800) 292-8635
International:
1-(408) 441-0732

e-mail
literature@atmel.com

Web Site
http://www.atmel.com

BBS
1-(408) 436-4309

1668AS–07/01