Documentation for SCSI controller project
PIA (**P**arallel **I**nterface **A**gent) module
Hardware V2.0 / Firmware V1.0

2006−06−18 / Michael Bäuerle <micha@hilfe−fuer−linux.de>

Preamble
The goal of this project is a general purpose parallel SCSI controller[1] for hobby usage. It was designed to be modular and SCSI3 compliant.

Description
The **P**arallel **I**nterface **A**gent handles the SCSI Layer2 (Link layer) protocol. It controls bus access and data transfer as described in the SPI (**S**CSI **P**arallel **I**nterface) specification. It does not deal with high level functions like the SIP protocol and the device model (=> This is the job of the "Target" module).

More and more of the (good old) simple SCSI chips from NCR, AMD and WD that are suitable for this task are discontinued and no longer available. This module "PIA" uses a microcontroller and 74xxx logic instead of an ASIC.

Hardware Features
Hardware V2.x is based on an Atmel ATmega165−16 AVR microcontroller. SCSI parity is generated and checked in hardware by a 74280 logic chip. Because common microcontrollers like the used AVR have not enough pin current to drive the SCSI bus directly, external transceivers (=> Implemented as module "PHY") must be used. This has the advantage that the PIA can not only be used for SE but also for HVD or LVD SCSI with matching PHYs. Hardware V2.0 uses a LVD/SE multimode PHY. A jumper on Pins 1−2 of JP1 will force the PHY to SE mode. The PHY must be configured to "Inverting mode" by a jumper on Pins 1−2 of JP10 for PIA firmware V1.x!

Firmware Features
Firmware V1.x can only be used for SCSI targets, the initiator mode is not supported. Asynchronous transfers are used to move data via the SCSI bus.

SCSI bus phases are handled completely by the PIA firmware. Illegal phase sequences are rejected so that a base functionality is provided by an abstract interface to the "Target" module.

All mandatory features required by the SCSI standard are implemented. The following features may be useful but are currently not supported:
- Arbitration support (required for disconnect/reconnect and AER)
- Reselection support in target mode (required for disconnect/reconnect)
- Selection support in initiator mode (required for AER)
- Synchronous data transfer support on the SCSI bus (to reduce bus load)

[1]) The circuit that controls a SCSI device not the Host adapter

Content

0. Abstract

The PIA firmware uses register, bit and pin names that are defined in ’include/pia.h’. Register mapping and interrupt vector table can be found in ’pia.s’.

Unexpected interrupts

If an unexpected interrupt occur (this is a fatal error), the PIA will freeze and flash the Busy−LED with 2Hz. Additionally the /ACK line is held asserted to prevent any register access. A hardware reset must be executed to recover from this state!

FIFO buffer

The PIA uses a 768Byte FIFO ringbuffer in its SRAM for data transfer from and to the target controller. This buffer must be accessed with the routines provided in ’buffer.s’ only! The size of the buffer is defined by the constant BUFFERSIZE in ’pia.s’. The following 16Bit registerpairs are used exclusively for FIFO management:

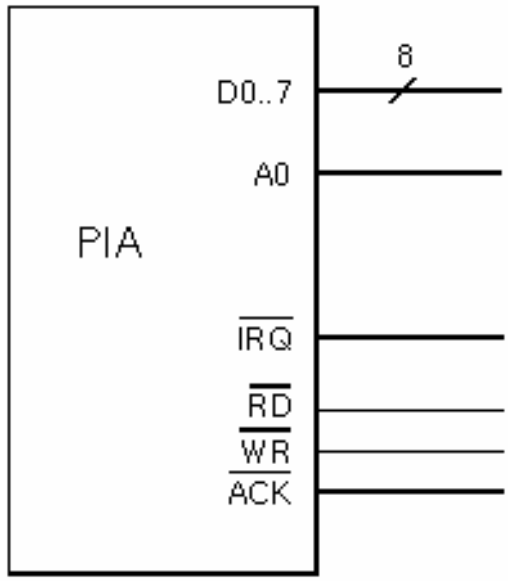
BS:	Buffersize (Initialized with value of constant BUFFERSIZE)
LPL:	Lower Pointer Limit (Lowest valid address inside buffer)
UPL:	Upper Pointer Limit (Highest valid address inside buffer)
X:	Number of bytes in FIFO
Y:	Pointer to FIFO read end
Z:	Pointer to FIFO write end

Selection Abort Timeout

SCSI3 specify a selection abort time of 200us (Maximum time for the target to take over bus control), the PIA supports 192us (after this time, the ACCEPT SELECTION command reports an error). This time may be too short for the target module if the debug system is enabled. For historically reasons, the selection timeout delay (the time the initiator should wait for the target to take over bus control) is recommended to be 250ms. Most initiators wait at least 64ms so this value can be increased in ’init.s’ up to 16ms.
Note: PIA firmware is configured to 16ms by default! Use 192us to be SCSI3 compliant.

1. PIA interface

The PIA is controlled by the Target MCU via a bidirectional 8Bit parallel interface using 8 data, 1 address and 4 control lines:



PIAs local Interface

Name	***Direction***	***Description***
D0 .. D7	Bidirectional	Data bus (has internal weak pull−up resistors enabled)
A0	Input	Address (Selects Control/Status or Data register)
/RD	Input	Read strobe (Read from PIA registers)
/WR	Input	Write strobe (Write to PIA registers)
/ACK	Output	Acknoledge (Access completed)
/IRQ	Output	Interrupt request (Target should read status register)

The /IRQ line is asynchronous and can become active at any time.
The A0 address line must be stable during the complete read or write access.
/RD and /WR are not allowed to be both active (low) at any time.

Protocol:

- Verify that PIA is ready by checking /ACK to be inactive (high)
- Drive A0 low to access the Control/Status register or drive A0 high to access the data register.
- Write access only: Drive the Dx lines with data byte.
- Activate /RD (drive low) for a read access or activate /WR (drive low) for a write access.
- Wait until the PIA drives /ACK active (low).
- Read access only: Read data from Dx lines.
- Drive /RD and /WR inactive (high).
  Write access only: Release data bus Dx.

2. PIA register set

The PIA contains three registers visible via its local interface. Because the PIA has only one address line, two registers are mapped on address zero:

Address	***Read***	***Write***
0	STATUS	CONTROL
1	DATA	DATA

STATUS

The status register contains the current state of the PIA:

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
FLOW2	FLOW	SELECT	ATN	RESET	COMPLETE	ERROR	ENABLE

ENABLE: PIA is configured and running (The CONFIGURE command sets this bit)
ERROR: Command was not successful (The RECOVER command clears this bit)
COMPLETE: Command completed (successfully if ERROR is not set)
RESET: SCSI bus reset detected
ATN: ATTENTION condition detected (GET_MESSAGE command request)
SELECT: An initiator has selected the PIA. The target must execute an ACCEPT_SELECTION command within 200µs or the PIA will time out
FLOW: Data is available or expected (Look at command description for details)
FLOW2: Used for flow control by some commands to transfer large data blocks

If one of Bits 2−6 become set or Bit 7 changes state, the /IRQ line becomes active (low) to inform the Target. Reading the status register will deactivate the /IRQ line.

CONTROL

To execute a PIA command, the command code must be written to the control register. The PIA can execute the following commands:

- ABORT
- CONFIGURE
- RECOVER
- ACCEPT_SELECTION
- GET_MESSAGE
- PUT_MESSAGE
- GET_COMMAND
- PUT_STATUS
- GET_DATA
- PUT_DATA
- BUSFREE

See next section for command details.

DATA

The data register is used to move data from and to the PIA. Also used to transfer additional bytes of multibyte commands.

3. PIA Commands

General:
After sending the command to the PIA, the command is in progress (this can be verified by checking the COMPLETE flag to be cleared in the status register). When the command is completed, the COMPLETE flag is set and an interrupt is generated. A cleared ERROR flag indicates success.

Attention:
The first byte of every command (command code, grey shaded) must be written to the CONTROL register, all further bytes must be written to the DATA register!

Error handling:
If the ERROR flag is set after a command completes, an error code is placed in the DATA register:

- ABORTED (0x00): Command aborted by target
- INVALID (0x01): Invalid command or parameter
- SEQUENCE (0x02): Illegal SCSI bus phase sequence
- PARITY (0x03): SCSI parity error detected
- PROTOCOL (0x04): SCSI parallel interface protocol error detected
- DATA_LENGTH (0x05): Data length error

The ERROR flag can only be cleared by a RECOVER command.

ABORT
This command try to abort any other currently running command. If the running command was aborted by an ABORT command, the ERROR flag in the status register is set and the error code is ABORTED. If you see another error code or no error, the ABORT command was ignored (this may be the case if the ABORT command was received while another command already exits but is not finished yet).

Note:
If an ABORT command is executed while no other command is in progress, the ABORT command is ignored (neither an error code nor a command complete interrupt is generated). You have to use a timeout while waiting for the interrupt!

Syntax:

	7	6	5	4	3	2	1	0
Byte 0	ABORT (0x00)							

The command has no response data.

CONFIGURE
This must be the first command after a PIA reset. This command configures the SCSI address and mode of the PIA. All other commands except ABORT and RECOVER are invalid until the PIA is not configured.

Syntax:

	7	6	5	4	3	2	1	0
Byte 0	CONFIGURE (0x01)							
Byte 1	SCSI address for PIA (0x00 to 0x07)							
Byte 2	Reserved							TAR

TAR: This bit must be set for Target mode (currently the only supported mode)
Reserved: Set all bits to zero to be compatible with future firmware versions

The command has no response data.

RECOVER
This command must be executed after a command has terminated with error. Executing RECOVER clears the data buffer and the Bits ERROR, FLOW and FLOW2 in the status register STAT. This command never fails.

Syntax:

	7	6	5	4	3	2	1	0
Byte 0	RECOVER (0x02)							

The command has no response data.

ACCEPT_SELECTION
This command accepts the selection request from an initiator when the PIA is configured for target mode. If the command is successful, the PIA have taken over SCSI bus control from the initiator.

Syntax:

	7	6	5	4	3	2	1	0
Byte 0	ACCEPT_SELECTION (0x03)							

The command response is the SCSI address of the Initiator that have selected the PIA:

	7	6	5	4	3	2	1	0
Byte 0	SCSI address of Initiator (0x00 to 0x07)							

The FLOW Bit in the status register indicates that the response data is ready to read. The command completes after the target have read the response.

GET_MESSAGE
This command transfers a SCSI message from the initiator to the target.

Syntax:

	7	6	5	4	3	2	1	0
Byte 0	GET_MESSAGE (0x04)							

The command response is the message from the initiator. The first byte of the message specifies how many bytes follow.

One byte message:

	7	6	5	4	3	2	1	0
Byte 0	Message code (0x00, 0x02 to 0x1F, 0x80 to 0xFF)							

Two byte message:

	7	6	5	4	3	2	1	0
Byte 0	Message code (0x20 to 0x2F)							
Byte 1	Message data							

Extended message:

	7	6	5	4	3	2	1	0
Byte 0	Message code (0x01)							
Byte 1	Extended message length (0x00 => 256 Byte)							
Byte 2	Extended message code (First byte that counts for length)							
Byte n	Extended message data							

After the PIA have received the complete message, the FLOW Bit in the status register is set. Now the target can read the message. The command completes after the target have read the last message byte.

PUT_MESSAGE
This command transfers a SCSI message from the target to the initiator.

Syntax:

	7	6	5	4	3	2	1	0
Byte 0	PUT_MESSAGE (0x05)							
Byte n	Message data							

The PIA waits until the complete message is written to its internal data buffer (The PIA knows the message syntax). Then the message is transferred to the initiator. If the data buffer inside the PIA is not empty after the message is sent, the message is oversized and the command terminates with a DATA_LENGTH error. If a message is larger than the maximum legal size of 258 Bytes, the PIA may generate protocol errors but return success.

The command has no response data.

GET_COMMAND
This command reads a SCSI command from an initiator.

Syntax:

	7	6	5	4	3	2	1	0
Byte 0	GET_COMMAND (0x06)							

The command response is a CDB (**C**ommand **D**escriptor **B**lock):

	7	6	5	4	3	2	1	0
Byte 0	Length			Command code				
Byte n	Command data							

Bits 5, 6 and 7 of Byte 0 specify the length of the CDB:
000: 6 Bytes
001: 10 Bytes
010: 10 Bytes
011: Reserved
100: 16 Bytes
101: 12 Bytes
110: Not supported
111: Not supported

If a reserved or unsupported length code is detected, the command terminates with PROTOCOL error.

The FLOW Bit in the status register indicates that the CDB is ready to read. The command completes after the target have read the CDB.

PUT_STATUS
This command transfers the status of a SCSI command to the initiator.

Syntax:

	7	6	5	4	3	2	1	0
Byte 0	PUT_STATUS (0x07)							
Byte 1	Status							

The command has no response data.

GET_DATA
This command reads data from the initiator.

Syntax:

	7	6	5	4	3	2	1	0
Byte 0	GET_DATA (0x08)							
Byte 1	(MSB)			Data length				
Byte 2								(LSB)

The command response is a data block with the requested number of bytes from the initiator:

	7	6	5	4	3	2	1	0
Byte n	Data							

A data length of 0x0000 reads 64KiByte of data.
If there is more data to transfer than this command can handle (64 KiByte), multiple GET_DATA commands must be used.

The FLOW Bit in the status register indicates that data is available to read. After every 512 Bytes (or the rest needed to complete the requested number of bytes) the PIA has written to the data buffer, the FLOW2 Bit is toggled. If the last IRQ is not serviced at the time the PIA want to toggle FLOW2, it waits until the STATUS register is read.

Note:
Polling the FLOW Bit in status register for flow control after every byte creates CPU load on the PIA and therefore slow down the SCSI transfers. The status register should only be read after the PIA generates an IRQ using the following scheme:
To minimize overhead, the target should wait for the first change of FLOW2 after FLOW was set by the PIA. Now the Target can consecutively read up to 512 Bytes without checking for buffer underruns and then wait for the next change of FLOW2.

The command completes after the target have read the requested number of bytes.

PUT_DATA
This command sends data to the initiator.

Syntax:

	7	6	5	4	3	2	1	0
Byte 0	PUT_DATA (0x09)							
Byte 1	(MSB)			Data length				
Byte 2								(LSB)
Byte n	Data							

A data length of 0x0000 sends 64KiByte of data.
The command transfers the specified number of bytes to the initiator. If there is more data to transfer than this command can handle (64 KiByte), multiple PUT_DATA commands must be used.

The FLOW Bit in the status register indicates that the data buffer is not full and data can be written to the PIA. If there is space available in the data buffer for at least 512 Bytes, the PIA toggles the FLOW2 Bit (the first 512Bytes can always be written).

Note:
Polling the FLOW Bit in status register for flow control after every byte creates CPU load on the PIA and therefore slow down the SCSI transfers. The status register should only be read after the PIA generates an IRQ using the following scheme:
To minimize overhead, the target should poll FLOW2 after writing the first 512 Bytes of data. When FLOW2 is toggled by the PIA, the space for the next 512 Bytes is available. Now the Target can consecutively write the next 512 Bytes without checking for buffer overflows. When the PIA toggles FLOW2 again, the space for the next 512 Bytes is available in the data buffer and so on.

If the data buffer inside the PIA is not empty after the specified number of bytes are sent, a DATA_LENGTH error is returned.

The command has no response data.

BUSFREE
This command can be executed at any time to release the SCSI bus. This command never fails.

Syntax:

	7	6	5	4	3	2	1	0
Byte 0	BUSFREE (0x0A)							

The command has no response data.

EOF