Overview

The aim of this document is to provide AT697 users a functional example of the AT697 PCI system controller capability. This application note describes the PCI mechanism handled by the AT697 through the use of a bootloader retrieving from an Ethernet BOOTP/TFTP server an IP address and a file to load into RAM.

1. References

   • AT697 datasheet. doc4226 on the Atmel web site.
   • PCI Revision 2.2
   • C-PCI Revision 2.1
2. System Description

The System consists of an AT697 Evaluation board (EvaB V2.0.0), an Ethernet board (L-N100TX based on RHINE III), a C-PCI backplane and a computer running a DHCP/BOOTP/TFTP server.

The aim of this application is to use a bootloader in order to load a program from a TFTP server into EvaB RAM and launch it.

The bootloader is a bootstrap, originally from eCos (open source real-time operating system): RedBoot. A PCI driver and the Rhine Ethernet PCI driver were ported and integrated on AT697.

In this application the AT697 is the HOST on the PCI bus and the PCI System controller (the arbiter is embedded in the AT697). The Ethernet board is a SATELLITE.
3. AT697 PCI initialisation

The ATMEL AT697 processor (with EvaB evaluation board) is able to act as a PCI System Controller and a HOST on a PCI bus. Thus, a PCI arbiter is included on the chip and allows setting up a system with a passive C-PCI backplane, an EvaB board (with AT697 installed) and other various PCI peripherals.

The embedded arbiter needs to be enabled: switch 17 and 18 of the EvaB board.

In order to activate the PCI Interface, the AT697 has to configure itself. Indeed, in a PCI system, a HOST configures other peripherals plugged on the PCI bus and itself. There are two main operations to do the minimum configuration:

1. Map the EvaB memory (RAM) in the PCI space. This is necessary so other PCI peripherals can access data located in the EvaB RAM. Initialise the MEMBAR1, MEMBAR2 and IOBAR3 (set the PCI addresses where other peripherals will read/write when accessing EvaB IO/RAM). Another register is used for memory mapping: the PCITPA register sets the local RAM address corresponding to MEMBAR1 and MEMBAR2 (8 most significant bits of each 16 Mbyte memory page). For example, if MEMBAR1 is set to 0x20000000 and PCITPA to 0x40006000, a PCI peripheral will be able to access the EvaB RAM (from 0x40000000 to 0x41000000 – 16Mbytes) at PCI address 0x20000000 to 0x21000000.

2. The 16 lower bits of the PCISC register correspond to the Command register. Some functionality have to be enabled to run the PCI interface:
   – Enable I/O response (allows AT697 to respond to I/O requests),
   – Enable MEM Response (allows AT697 to respond to MEM requests),
   – Enable PCI Master (allows AT697 to initiate PCI transfers)
4. PCI enumeration

In this application note, the AT697 is the HOST on the PCI bus. Once auto-configured, the AT697 will look for PCI peripherals on the bus and configure them.

The PCI enumeration is the first step to begin PCI transactions. The method used to access to PCI peripherals is to select each slot of the backplane one after the other using the IDSEL signal and try to read the board Vendor ID (offset 0x00).

According to the C-PCI specification, the IDSEL signal is decoded using the Device Number (bit 31 to 16) respectively selecting PCI slot 0 to 16.

**Decomposition of an address used to access configuration space**

<table>
<thead>
<tr>
<th>Device Number</th>
<th>Dword Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 0</td>
</tr>
</tbody>
</table>

If bit 31 of the Device Number is set, the IDSEL of the first slot will be set and so on.

For example, in order to read the Vendor ID and test if a board is present in the second slot, its IDSEL set by reading at 0x40000000 (bit 30 set).
During enumeration, DMA mode has to be used because addresses will always be between 0x80000000 and 0x00010000 and direct addressing mode is only usable between 0xA0000000 and 0xF0000000.

In DMA mode, all accesses are 32 bits. But, 8 and 16 bits commands can be created based on 32 bits access. For reading: reading 32 bits and returning only the requested data and for writing: reading 32 bits, modifying the read value and re-writing it.

The software implementation to do the enumeration consists in the following steps:

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter DMA mode</td>
<td>Set the bit 0 of PCIIC (0x80000158)</td>
</tr>
<tr>
<td>Wait until no transaction by PCI core is pending</td>
<td>Wait until bit 7 of PCIIS is null</td>
</tr>
<tr>
<td>Write start address</td>
<td>Write 0x40000000 in the PCISA register</td>
</tr>
<tr>
<td>Reset the initiator error</td>
<td>Set the bit 6 of PCIITP register</td>
</tr>
<tr>
<td>Starts the DMA transfer</td>
<td>Write in the PCIDMAA register the address of the local variable which will receive the read data</td>
</tr>
<tr>
<td>Wait for termination of DMA or initiator error occurs</td>
<td>Wait until bit 6 or 7 of PCIITP register is set</td>
</tr>
<tr>
<td>If an initiator error occurs</td>
<td>If an initiator error occurs, it means that the board is not accessible (no board in the selected slot).</td>
</tr>
<tr>
<td>Else, the DMA transfer is completed and your data (VendorID in this case) has been stored in the local variable.</td>
<td>Else, the DMA transfer is completed and your data (VendorID in this case) has been stored in the local variable.</td>
</tr>
</tbody>
</table>

Next, all information contained in the PCI configuration registers of the detected peripheral board can be retrieved (Vendor ID, Device ID, Status register, Command register...).

A few limitations appears when operating in DMA mode:

- It only works with 32 bits requests (not 16 or 8). There is a work-around, we can emulate 16 and 8 bits reading and writing by software
- Reading and writing in DMA mode must be done on 32 bit-word aligned addresses
5. PCI peripheral initialisation

The detected board’s PCI interface needs to be configured by the AT697.

The first step is the BAR configuration consisting in the following steps:

<table>
<thead>
<tr>
<th>Description</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieve the nature of the first BAR</td>
<td>Write 0xFFFFFFF to the first BAR (0x10)</td>
</tr>
<tr>
<td></td>
<td>Read the first BAR value (0x10)</td>
</tr>
<tr>
<td></td>
<td>The returned value is (for example) 0xFFFFFFFF01: it means we have an IOBAR</td>
</tr>
<tr>
<td></td>
<td>(byte 0 is set) of 256 bytes (0x100)</td>
</tr>
<tr>
<td>Map the IO space at 0xA0000000 in PCI space</td>
<td>Write 0xA0000000 in the first BAR (0x10)</td>
</tr>
<tr>
<td>Retrieve the nature of the second BAR</td>
<td>Write 0xFFFFFFFF to the second BAR (0x14)</td>
</tr>
<tr>
<td></td>
<td>Read the second BAR value (0x14)</td>
</tr>
<tr>
<td></td>
<td>The returned value is (for example) 0xFFFFF000: it means we have a MEMBAR</td>
</tr>
<tr>
<td></td>
<td>(byte 0 is not set) of 4096 bytes (0x1000)</td>
</tr>
<tr>
<td>Map the MEM space at 0x800000000 in PCI space</td>
<td>Write 0x80000000 in the second BAR (0x14)</td>
</tr>
<tr>
<td>Continue these steps up to 0x24 configuration</td>
<td></td>
</tr>
</tbody>
</table>

IOBAR and MEMBAR from different peripherals must not overlap in the PCI space mapping.

The last configuration space action done on the peripheral is its enabling using the 16 lower bits of the PCISC register (Command register):

- Enable I/O response (allows peripheral to respond to I/O requests),
- Enable MEM Response (allows peripheral to respond to MEM requests),
- Enable PCI Master (allows peripheral to initiate PCI transfers)
6. HOST-initiated access to the SATELLITE

Once the detected peripheral is configured and its PCI interface enabled, I/O and memory spaces access are done by reading/writing in the PCI space. The HOST is in this case an initiator on the PCI bus.

Taking the same example as in the previous section, I/O space is mapped at 0xA0000000 and memory space at 0x80000000.

All the following examples are specific to the L-N100TX board (Ethernet board used in our application) but methods are generic. By now, the internal architecture of the peripheral you are working on (internal registers and mechanisms) must be known.

Two modes can be used to access I/O or memory:

Direct addressing:

This kind of access can be done by setting the mode to AHB_SLAVE in the application before using the read/write commands. This mode is advocated when PCI peripherals on the bus don’t contain large amount of memory and/or I/O because it can only be used on PCI addresses between 0xA0000000 and 0xFFFF0000. It fits most of applications.

DMA:
This one can be done by setting the mode to AHB_MASTER in the application before using the read/write commands. This mode can be used to read any address in the PCI space. It’s the answer when many peripherals with large amount of memory and/or I/O are present on the bus.

In our example, we can access to the I/O space memory in DMA and direct addressing mode but the memory space can only be accessed by DMA (< 0xA0000000).

In the BOOTP application, only I/O accesses are used because the Ethernet board doesn’t need memory accesses. Moreover, in order to have the choice of addressing mode I/O space is mapped at 0xA0000000.

I/O space is used to read Ethernet specific data (MAC address of the board), read status of Ethernet interface (MII status), configure TxRing and RxRing and send commands to the board.

Direct addressing mode consists of the following steps:

<table>
<thead>
<tr>
<th>Initialise the wanted command (io read, io write, mem read…)</th>
<th>Write the wanted command in the PCIIC: 0x00 in the 2 upper bits (I/O read/write)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read a value</td>
<td>*local_variable_addr = pci_addr</td>
</tr>
<tr>
<td>Write a value</td>
<td>*pci_addr = *local_variable_addr</td>
</tr>
</tbody>
</table>
7. **SATELLITE-initiated access to the HOST**

In some cases, a SATELLITE can be initiator on the PCI bus too. In the BOOTP application, the Ethernet board acts like this and the AT697 becomes the TARGET.

The Ethernet board (L-N100TX) has no embedded memory to store TX and RX buffers. So AT697 provides pointers on its memory to the Ethernet board.

Then, when the AT697 sends a transmit command to the Ethernet board, the Ethernet board read its internal register containing the address (PCI) of transmit buffer (located on the EvaB) and read into PCI at this address.

Indeed, the PCI address where the Ethernet board is reading correspond to the transmit buffer located into the EvaB RAM. So, the Ethernet board read in the EvaB RAM in DMA mode (the only one accessible when AT697 is TARGET) through the PCI.
8. PCI CORE Architecture

The AT697 PCI CORE is described in the scheme below:

The AT697 can be an INITIATOR or a TARGET on the PCI bus.

The AT697 PCI CORE allows 2 different addressing modes in the INITIATOR case (AT697 request the PCI transaction): DMA or direct addressing. In TARGET case (another peripheral request the transaction), only the DMA mode is accessible.

All PCI transactions are based on polling in the BOOTP application. Interrupts aren't used because of original eCos PCI upper-layer based on polling.
9. Memory mapping

The mapping of the system is represented in the scheme below:

In the left part, there is the internal AT697 address space:
- I/O (can be mapped in the PCI address space but not used in our application)
- RAM (mapped in the PCI address space – middle part - in our application at 0x00000000)
- Internal registers (can’t be mapped in the PCI space)

In the right part, there is the internal RHINE III address space:
- I/O (mapped in the PCI address space – middle part - in our application at 0xA0000000)
- MEM (mapped in the PCI address space – middle part - in our application at 0xC0000000 but we don’t use it)

AT697 is accessing the RHINE III by reading/writing to PCI space (I/O between 0xA0000000 and 0xA0000100). MEM is not used in our application.

RHINE III is accessing the transmit and receive Ethernet buffers located into EvaB RAM by reading/writing to the PCI space (between 0x00000000 and 0x20000000).
10. **Application scheduling**

The scheduling of the global system (BOOTP application) is the following:

11. **Source entries**

<table>
<thead>
<tr>
<th>Utility</th>
<th>Source entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCI initialisation</td>
<td><code>dv_pci_at697_init (dv_pci_at697.c)</code></td>
</tr>
<tr>
<td>PCI enumeration</td>
<td><code>cyg_pci_find_next (pci.c)</code></td>
</tr>
<tr>
<td>Reading and writing in configuration space</td>
<td><code>dv_pci_at697_conf_b, dv_pci_at697_conf_w and dv_pci_at697_conf_dw (dv_pci_at697.c), respectively 8, 16 and 32 bits</code></td>
</tr>
<tr>
<td>BARs configuration</td>
<td><code>cyg_pci_configure_device (pci.c)</code></td>
</tr>
<tr>
<td>Configure memory access mode</td>
<td><code>dv_pci_at697_set_config (dv_pci_at697.c)</code></td>
</tr>
<tr>
<td>Reading and writing in I/O space</td>
<td><code>dv_pci_at697_io_b, dv_pci_at697_io_w and dv_pci_at697_io_dw (dv_pci_at697.c), respectively 8, 16 and 32 bits</code></td>
</tr>
</tbody>
</table>
12. References

eCos Porting Guide
http://ecos.sourceforge.net/ecos/docs-latest/ref/hal-porting-guide.html

Leon Porting Project by De Naayer Instituut
http://emsys.denayer.wenk.be/?project=empro&page=cases&id=14

eCos Reference Manual
http://ecos.sourceforge.net/ecos/docs-latest/ref/ecos-ref.html

The ecos-discuss Mailing List Archive
http://ecos.sourceforge.net/ml/ecos-discuss/

RedBoot User’s Guide

eCos Latest Documents
http://ecos.sourceforge.net/ecos/docs-latest/

A Linux Ethernet device driver for VIA Rhine family chips

Information about the TFTP server
http://tftp32.jounin.net/

Gaisler website (compiler, information about SPARC architecture, about eCos...)
http://www.gaisler.com/

ATMEL AT697 datasheet and other documents

PCI Revision 2.2 (legal ?)
List of PCI Vendors and PCI Device ID
http://pcidatabase.com

PCI System Architecture (Fourth Edition) by Tom Shanley and Don Anderson

13. AT697 registers used

PCISC (0x80000104)
AT697 is configuring itself his PCI CORE with the lower 16 bits and read the upper 16 bits to retrieve its status

PCIBHLC (0x8000010C) – Not studied
Initialisation of « latency timer » which correspond to allocation time of each initiator

MBAR1 (0x80000110)
Configure the address where the memory page 1 (its local address is specified by the PCITPA register) is mapped in the PCI space (initialised to 0x0000000 in our application).
This register is also used to know the size and the kind of each BAR by writing 0xFFFFFFFF and reading it.

PCIRT (0x80000140) – Not studied
Allows configuring « Retry » parameters during PCI transactions

PCICW (0x80000144) - Not studied
Activate or not wanted byte during AT697 auto-configuration

PCISA (0x80000148)
Used to set the starting address for DMA transactions

PCIDMA (0x80000150)
Allows configuring the DMA command and the number of word to transmit/receive; Also configure the back2back mode.

PCIIS (0x80000154)
Allows to know the PCI CORE state: PCI CORE ready or not, DMA state, transmission and reception fifo state
**PCIIC (0x80000158)**

Used to select the addressing mode (AHB mandatory) and the command used by direct addressing.

**PCITPA (0x8000015C)**

Used to define which address in the local memory corresponds to the MEMBAR1 and MEMBAR2; We write the most significant bits of each MEMBAR.

**PCITSC (0x80000160) – Not studied**

Allows configuring the « Fifo parity error »

**PCITTP (0x80000168)**

Used to know occurring errors during a PCI transfer; Each bit is resetted by setting it.