32-bit AVR® Microcontrollers

Application Note

AVR32108: Peripheral Direct Memory Access Driver

Features Peripheral DMA Controller

- **Controls transfers to and from peripherals such as USART, SSC, SPI, etc.**
- **Supports up to 20 channels (device dependent)**
- **One master clock cycle needed for a transfer from memory to peripheral**
- **Two master clock cycles needed for a transfer from peripheral to memory**
- **Utilizes two independent buffers each way for chained transfers**
- **Interrupts on full/empty buffers**

1 Introduction

The Peripheral DMA Controller (PDC) enables data transfers between on-chip serial peripherals such as the USART, SSC, SPI, TWI and memory controller. Using the PDC increases the processor performance due to less overhead than an interrupt driven data transfer would take. This reduces the need for code and processor speed, which again reduces power consumption.

The PDC channels are implemented in pairs where each pair is dedicated to a particular peripheral. The pair is divided into one channel for receive and one channel for transmit.

Setting up and using the PDC is interfaced through registers in the memory for each peripheral. For each channel these registers contain:

- A 32-bit memory pointer register
- A 16-bit buffer count register
- A 32-bit register for next memory pointer
- A 16-bit register for next buffer count

The PDC is triggered by transmit and receive signals from the peripheral. When buffers reach their limits the PDC can generate interrupts to the corresponding peripheral.

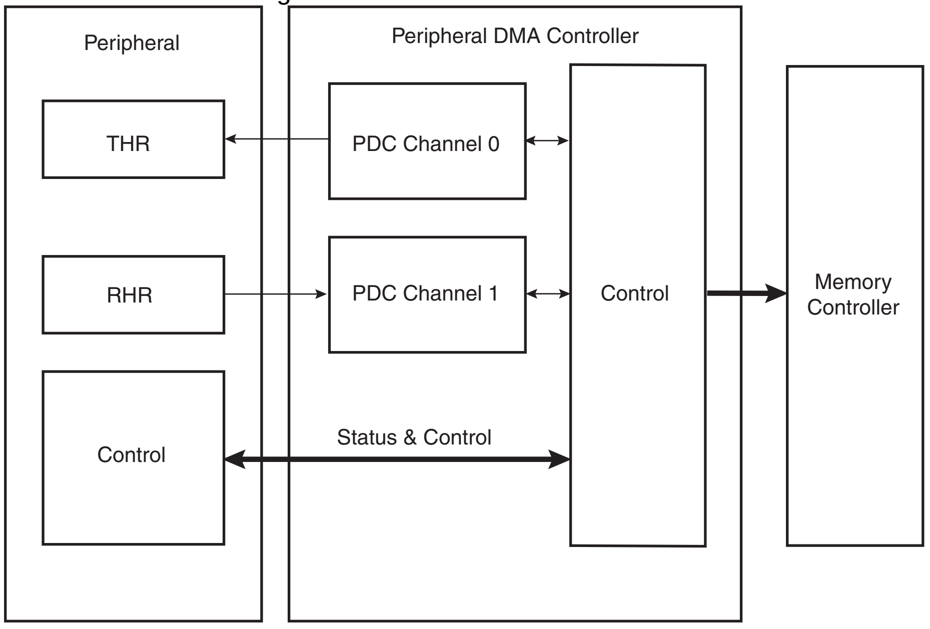
This application note includes an example of using the USART with the PDC, with and without interrupt control. For details regarding interrupt control, application note *AVR®32101 - The AVR32 Interrupt Controller* is suggested for further reading.

Rev. 32016A-AVR32-05/06

2 PDC Functional Description

2.1 Block Diagram

Figure 2-1. PDC Block Diagram



2.2 PDC Registers

Table 2-1. PDC Register Mapping for AP7000

Offset	Register	Register name	Read/Write	Reset
0x100	Receive Pointer Register	RPR	Read/Write	0x0
0x104	Receive Counter Register	RCR	Read/Write	0x0
0x108	Transmit Pointer Register	TPR	Read/Write	0x0
0x10C	Transmit Counter Register	TCR	Read/Write	0x0
0x110	Receive Next Pointer Register	RNPR	Read/Write	0x0
0x114	Receive Next Counter Register	RNCR	Read/Write	0x0
0x118	Transmit Next Pointer Register	TNPR	Read/Write	0x0
0x11C	Transmit Next Counter Register	TNCR	Read/Write	0x0
0x120	PDC Transfer Control Register	PTCR	Write-only	--
0x124	PDC Transfer Status Register	PTSR	Read-only	0x0

2.3 Configuration

The PDC channels are configurable for the user with the 10 registers described in chapter 2.2. Each peripheral capable of DMA transfers have these registers mapped at an offset 0x100 from the peripheral base address.

Buffer size is configured with the counter registers. Reading these registers will give the remaining transfers left before the buffer is full for receive or empty for transmit.

Buffer memory address is configured with the pointer register. Reading these register returns the next location to be transferred.

The PDC have two registers for status and control. Transmit and receive can independently be turned off and on by setting TXTEN/TXTDIS and RXTEN/RXTDIS. Disabling the transfer before reading data from the counter or pointer register guarantees that the register does not change value between reads.

2.4 Memory Pointers

The PDC transfers data by reading from or writing to a buffer located in memory. The pointer to the initial memory location is given by setting the RPR register for receives or the TPR register for transmits.

Depending on the size of the transfer (8-bit, 16-bit or 32-bit), the memory pointers are automatically incremented by 1, 2 or 4.

It is possible to update the pointer register while the PDC is enabled, this will allow the PDC to transfer data from a different memory address.

2.5 Transfer Counters

The counter registers control the size of the transfer buffers, the maximum buffer size is limited by the 16-bit value of these registers. After a transfer the counter is decremented and when a counter reaches zero the PDC stops transferring data using the corresponding Pointer Register.

When the Counter Register reaches zero and the Next Counter Register is not equal to zero the PDC will load the Next Pointer Register into the Pointer Register and load the Next Counter Register into the Counter Register. This allows chaining the buffers, making transfer continuous.

If the Next Counter Register equals zero the related peripheral flag is set in the status register and the PDC stops.

It is possible to update the Counter Register while the PDC is active, this will make the PDC count transfers from the new value.

End flags are automatically cleared if one of the Counter Registers is written.

Writing to the Next Counter Register when the Counter Register is zero and the PDC is enabled will automatically load the values from the Next Registers into the Counter Register and Pointer Register.

The Next Counter Register is set to zero when it is loaded into the Counter Register.

2.6 Data transfers

The transfers are triggered by transmit ready (TXRDY) and receive ready (RXRDY) signals.

When a peripheral receives data, it sends a receive ready signal (RXRDY) to the PDC which then requests access to the system bus. When access is granted, the PDC starts a read of the peripheral receive holding register (RHR) and then triggers a write in memory.

After each receive, the relevant PDC memory pointer is incremented and the number of receives left is decremented. When the memory block size is reached, a signal is sent to the peripheral and the receive operation stops.

When a PDC transmit data, it checks if the transmit ready signal (TXRDY) for the peripheral is valid for a new transmit. Then the PDC requests access to the system bus. When access is granted, the PDC loads data into the peripheral transmit holding register (THR).

After each transfer, the relevant PDC memory pointer is incremented and the number of transmits left is decremented. When the memory block size is reached, a signal is sent to the peripheral and the transmit operation stops.

2.7 Priority of PDC Transfer Requests

Priority is fixed for each peripheral on the part; consult the datasheet for further details. Priority for simultaneous requests of the same type (receive or transmit) for identical peripherals, but different instances, is determined by the numbering of the peripherals. Lower number gives higher priority, for example if USART0 and USART1 receives data simultaneous, USART0 will be handled before USART1.

Requests are normally treated in the order they occurred, and receiver are handled before transmitter requests.

2.8 Interrupts

The PDC does not send interrupt request to the CPU itself. The peripheral devices that utilize the PDC have dedicated bits in their status registers that are controlled by the PDC. The peripheral can be configured to interrupt the CPU when these status bits are asserted.

The PDC will assert a status bit when:

- The first receiver buffer is filled (ENDRX)
- Both receiver buffers are filled (RXBUFF)
- The first transmitter buffer is emptied (ENDTX)
- Both transmitter buffers are emptied (TXBUFE)

For more details concerning the interrupt controller see application note *AVR32101 - The AVR32 interrupt controller*.

3 PDC design considerations

Whenever designing an interrupt configuration or peripheral DMA control for a specific system, care should be taken to get the optimal performance and the best system behavior.

3.1 PDC buffers and caching

It is critical that the PDC buffers in RAM are updated and not just updated in the cache. The peripheral DMA controller accesses the memory directly, without using the memory management unit.

3.1.1 Cache-line alignment

The PDC buffers must be cache-line aligned in memory for correct behavior of the PDC. If it is not cache-line aligned it may invalidate non-related data.

3.1.2 Clean cache

If transmit buffers are placed in cached memory, the cache should be cleaned before register TCR and TNCR are updated. This has to be performed to ensure that the memory and cache is consistent.

For receive buffers the cache should be invalidated before register RCR and RNCR are updated.

4 Implementation

4.1 Driver files

The driver consists of seven files:

- *pdc.c*, PDC driver source file
- *pdc.h*, PDC driver header file
- *interrupt_gcc.c*, interrupt example usage for GNU GCC compiler source file
- *interrupt_iar.c*, interrupt example usage for IAR® System compiler source file
- *interrupt_gcc.h* and *interrupt_iar.h*, interrupt example usage header file
- *settings.h*, header file for adjusting parameters to the example

For adapting this application note to a specific hardware setup the *settings.h* file has to be altered to match the devices and development tools used.

The PDC example application is depending on that the Memory Management Unit (MMU) is not enabled.

4.1.1 PDC driver files

The PDC driver is a general implementation and will work with any peripheral capable of PDC with a memory offset of 0x100 to the PDC registers.

To enable the peripheral DMA controller for a module, the module itself must be initialized as usual. After the module is initialized and enabled, buffers for receive or transmit must be programmed into the PDC registers. Functions are described shortly in Table 4-1 and more detailed in the doxygen documentation (see chapter 4.2).

Table 4-1. The PDC driver function summary

Function name	**Description**
pdc_enable()	Enables both transmit and receive of data.
pdc_enableTx()	Enables transmit of data.
pdc_enableRx()	Enables receive of data.
pdc_disable()	Disables both transmit and receive of data.
pdc_disableTx()	Disables transmit of data.
pdc_disableRx()	Disables receive of data.
pdc_setTxBuf()	Set the Pointer Register, Counter Register, Next Pointer Register and Next Counter Register for transmitting data. Then it enables the PDC.
pdc_setTxNextBuf()	Set the Next Pointer Register and Next Counter Register for transmitting data.

Function name	Description
pdc_setRxBuf()	Set the Pointer Register, Counter Register, Next Pointer Register and Next Counter Register for receiving data. Then it enables the PDC.
pdc_setRxNextBuf()	Set the Next Pointer Register and Next Counter Register for receiving data.
pdc_getTcr()	Get the number of transmits left from Count Register.
pdc_getTxByesLeft()	Get the total number of transmits left from both Counter Register and Next Counter Register.
pdc_getRcr()	Get the number of receives left from Count Register.
pdc_getRxBytesLeft()	Get the total number of receives left from both Counter Register and Next Counter Register.
pdc_flushCache()	Flush cache lines
pdc_translatePtr()	Translates virtual addresses to physical addresses when the memory management unit is not in use.

4.1.2 Interrupt driver files

The interrupt driver is an implementation specific driver for PDC interrupts for a given peripheral, in this example the USART0 module.

For the IAR® Systems compiler the #pragma compiler switch is used.

For GNU GCC compiler the set_interrupts_base() and register_interrupt() are used with an __int_handler.

For more documentation regarding syntax to the functions consult the code documentation (see chapter 4.2). More details concerning the interrupt controller see application note *AVR32101 - The AVR32 Interrupt Controller*.

4.2 Doxygen documentation

All source code is prepared for doxygen automatic documentation generation. Premade doxygen documentation is also supplied with the source to this application note, located in src/doxygen/index.html.

Doxygen is a tool for generating documentation from source code by analyzing the source code and using known keywords. For more details see http://www.stack.nl/~dimitri/doxygen/.

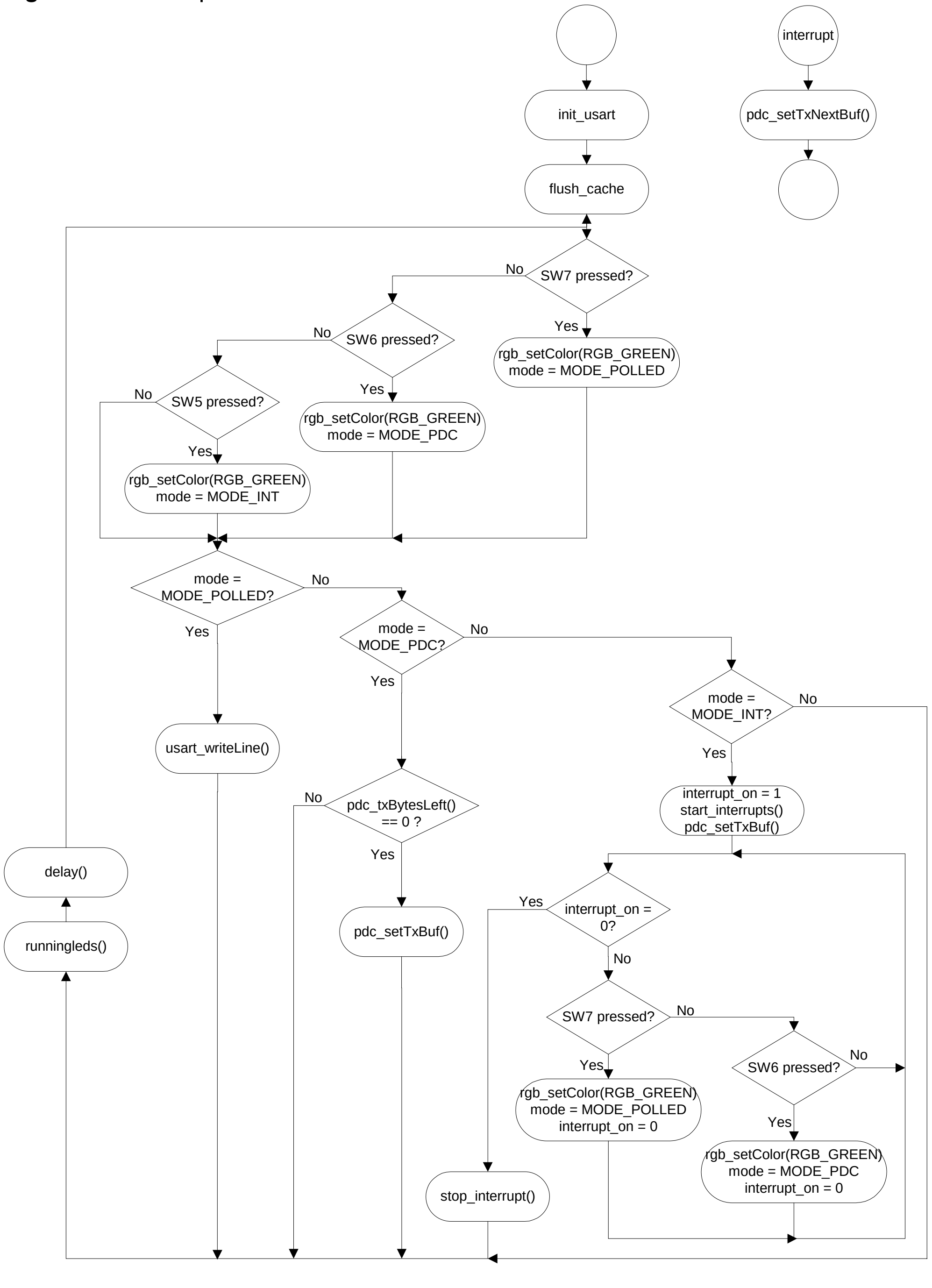
4.3 Example code

Included with the application note is an example source code for two compilers. The example is named *pdc_example.c*, and the different interrupt conventions for each compiler are implemented in two separate files. The compiler includes the correct interrupt functions using macros.

The example code is an application using the USART in different ways. It will start by doing conventional polled usage of the USART, and by user input it is possible to switch to PDC driven USART with and without interrupt control.

Figure 4-1 shows the flow of the example application. For detailed explanation of the functions please see the code documentation (see chapter 4.2).

Figure 4-1. Example code flow chart



Atmel Corporation

2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

Regional Headquarters

Europe
Atmel Sarl
Route des Arsenaux 41
Case Postale 80
CH-1705 Fribourg
Switzerland
Tel: (41) 26-426-5555
Fax: (41) 26-426-5500

Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Japan
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

Memory
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics/Imaging/Hi-Rel MPU/
High Speed Converters/RF Datacom
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-30-00
Fax: (33) 4-76-58-34-80

Literature Requests
www.atmel.com/literature