APPLICATION NOTE

AT07893: SAM4L Peripheral DMA Controller (PDCA)

ASF PROGRAMMERS MANUAL

SAM4L Peripheral DMA Controller (PDCA)

The Peripheral DMA Controller (PDCA) transfers data between on-chip peripheral modules such as USART, SPI, and memories (those memories may be on or off chip). Using the PDCA avoids CPU intervention for data transfers, improving the performance of the microcontroller. The PDCA can transfer data from memory to a peripheral or from a peripheral to memory.

The outline of this documentation is as follows:

Table of Contents

Software License

1. Prerequisites

There are no prerequisites for this module.

2. Module Overview

The Peripheral DMA Controller (PDCA) transfers data between on-chip peripheral modules such as USART, SPI, and memories (those memories may be on and off chip). Using the PDCA avoids CPU intervention for data transfers, improving the performance of the microcontroller. The PDCA can transfer data from memory to a peripheral or from a peripheral to memory. The PDCA consists of multiple DMA channels. Each channel has:

- A Peripheral Select Register
- A 32-bit memory pointer
- A 16-bit transfer counter
- A 32-bit memory pointer reload value
- A 16-bit transfer counter reload value

The PDCA communicates with the peripheral modules over a set of handshake interfaces. The peripheral signals the PDCA when it is ready to receive or transmit data. The PDCA acknowledges the request when the transmission has started. When a transmit buffer is empty or a receive buffer is full, an optional interrupt request can be generated.

3. Special Considerations

None.

4. Extra Information

For extra information, see Extra Information for Peripheral DMA Controller Driver. This includes:

- Acronyms
- Dependencies
- Errata
- Module History

5. Examples

For a list of examples related to this driver, see Examples for Peripheral DMA Controller.

6. API Overview

6.1 Variable and Type Definitions

6.1.1 Type pdca_callback_t

```
typedef void(* pdca_callback_t )(enum pdca_channel_status status)
```

PDCA callback type.

6.1.2 Type pdca_channel_interrupt_mask_t

```
typedef uint32_t pdca_channel_interrupt_mask_t
```

PDCA channel interrupt mask type.

6.1.3 Type pdca_channel_num_t

```
typedef uint8_t pdca_channel_num_t
```

PDCA channel number type.

6.2 Structure Definitions

6.2.1 Struct pdca_channel_config_t

PDCA channel configuration structure.

Table 6-1. Members

Type	Name	Description
void *	addr	Memory address.
bool	etrig	Enable/disable the transfer upon event trigger.
uint32_t	pid	Select peripheral ID.
void *	r_addr	Next memory address.
uint32_t	r_size	Next transfer counter.
bool	ring	Ring buffer function.
uint32_t	size	Transfer counter.
uint32_t	transfer_size	Select the size of the transfer (in bytes, half-words or words).

6.3 Function Definitions

6.3.1 Function pdca_channel_clear_error()

Clear transfer error for the given channel.

```
void pdca_channel_clear_error(
  pdca_channel_num_t pdca_ch_number)
```

Table 6-2. Parameters

Data direction	Parameter name	Description
[in]	pdca_ch_number	PDCA channel

6.3.2 Function pdca_channel_disable()

Disable the PDCA for the given channel.

```
void pdca_channel_disable(
  pdca_channel_num_t pdca_ch_number)
```

Table 6-3. Parameters

Data direction	Parameter name	Description
[in]	pdca_ch_number	PDCA channel

6.3.3 Function pdca_channel_disable_interrupt()

Disable PDCA interrupt.

```
void pdca_channel_disable_interrupt(
  pdca_channel_num_t pdca_ch_number,
  const pdca_channel_interrupt_mask_t pdca_channel_interrupt_mask)
```

Table 6-4. Parameters

Data direction	Parameter name	Description
[in]	pdca_ch_number	PDCA channel
[in]	pdca_channel_interrupt_mask	Interrupts to be disabled

6.3.4 Function pdca_channel_enable()

Enable the PDCA for the given channel.

```
void pdca_channel_enable(
  pdca_channel_num_t pdca_ch_number)
```

Table 6-5. Parameters

Data direction	Parameter name	Description
[in]	pdca_ch_number	PDCA channel

6.3.5 Function pdca_channel_enable_interrupt()

Enable PDCA transfer error interrupt.

```
void pdca_channel_enable_interrupt(
  pdca_channel_num_t pdca_ch_number,
  const pdca_channel_interrupt_mask_t pdca_channel_interrupt_mask)
```

Table 6-6. Parameters

Data direction	Parameter name	Description
[in]	pdca_ch_number	PDCA channel
[in]	pdca_channel_interrupt_mask	Interrupts to be enabled

6.3.6 Function pdca_channel_get_handler()

Get the base address of the specified PDCA channel configuration registers.

```
PdcaChannel * pdca_channel_get_handler(
  pdca_channel_num_t pdca_ch_number)
```

Table 6-7. Parameters

Data direction	Parameter name	Description
[in]	pdca_ch_number	PDCA channel (range 0 15)

Returns The channel configuration registers base address.

6.3.7 Function pdca_channel_get_interrupt_mask()

Get PDCA interrupt mask.

```
pdca_channel_interrupt_mask_t pdca_channel_get_interrupt_mask(
  pdca_channel_num_t pdca_ch_number)
```

Table 6-8. Parameters

Data direction	Parameter name	Description
[in]	pdca_ch_number	PDCA channel

6.3.8 Function pdca_channel_is_enabled()

Check if PDCA channel is enabled.

```
bool pdca_channel_is_enabled(
  pdca_channel_num_t pdca_ch_number)
```

Table 6-9. Parameters

Data direction	Parameter name	Description
[in]	pdca_ch_number	PDCA channel number to query

Table 6-10. Return Values

Return value	Description
true	PDCA channel is enabled
false	PDCA channel is disabled

6.3.9 Function pdca_channel_read_load_size()

Read PDCA channel load values from hardware.

```
uint32_t pdca_channel_read_load_size(
  pdca_channel_num_t pdca_ch_number)
```

Each channel has a 16-bit Transfer Counter Register (TCR). This register must be written with the number of transfers to be performed. The TCR register should contain the number of data items to be transferred independently of the transfer size. The TCR can be read at any time during transfer to see the number of remaining transfers.

Table 6-11. Parameters

Data direction	Parameter name	Description
[in]	pdca_ch_number	PDCA channel

Returns Size of the data block to load.

Note The size of a data item is held in the channel config structure. See pdca_channel_set_config and Basic use case for more deatils.

6.3.10 Function pdca_channel_read_reload_size()

Read PDCA channel reload values from hardware.

```
uint32_t pdca_channel_read_reload_size(
  pdca_channel_num_t pdca_ch_number)
```

Table 6-12. Parameters

Data direction	Parameter name	Description
[in]	pdca_ch_number	PDCA channel

Returns Size of the data block to reload.

6.3.11 Function pdca_channel_set_callback()

Set a callback for the specified PDCA channel.

```
void pdca_channel_set_callback(
```

```
  pdca_channel_num_t pdca_ch_number,
  pdca_callback_t callback,
  uint8_t irq_line,
  uint8_t irq_level,
  const pdca_channel_interrupt_mask_t pdca_channel_interrupt_mask)
```

Table 6-13. Parameters

Data direction	Parameter name	Description
[in]	pdca_ch_number	PDCA channel number (range 0 15)
[in]	callback	callback function pointer
[in]	irq_line	interrupt line
[in]	irq_level	interrupt level
[in]	pdca_channel_interrupt_mask	Interrupts to be enabled

6.3.12 Function pdca_channel_set_config()

Configure the specified PDCA channel.

```
void pdca_channel_set_config(
  pdca_channel_num_t pdca_ch_number,
  const pdca_channel_config_t * cfg)
```

Table 6-14. Parameters

Data direction	Parameter name	Description
[in]	pdca_ch_number	PDCA channel (range 0 15)
[in]	cfg	Pointer to a PDCA channel configuration structure

6.3.13 Function pdca_channel_write_load()

Write PDCA channel load values to hardware.

```
void pdca_channel_write_load(
  pdca_channel_num_t pdca_ch_number,
  volatile void * addr,
  uint32_t size)
```

Table 6-15. Parameters

Data direction	Parameter name	Description
[in]	pdca_ch_number	PDCA channel (range 0 15)
[in]	addr	Address where data to load is stored
[in]	size	Size of the data block to load

6.3.14 Function pdca_channel_write_reload()

Write PDCA channel reload values to hardware.

```
void pdca_channel_write_reload(
  pdca_channel_num_t pdca_ch_number,
  volatile void * addr,
  uint32_t size)
```

Table 6-16. Parameters

Data direction	Parameter name	Description
[in]	pdca_ch_number	PDCA channel
[in]	addr	Address where data to load is storedr
[in]	size	Size of the data block to load

6.3.15 Function pdca_disable()

Disable the PDCA module.

```
void pdca_disable(
  Pdca * pdca)
```

Table 6-17. Parameters

Data direction	Parameter name	Description
[in]	pdca	Base address of the PDCA module

6.3.16 Function pdca_enable()

Disable the PDCA module.

```
void pdca_enable(
  Pdca * pdca)
```

Table 6-18. Parameters

Data direction	Parameter name	Description
[in]	pdca	Base address of the PDCA module

6.3.17 Function pdca_get_channel_status()

Get the PDCA channel transfer enable status.

```
enum pdca_channel_status pdca_get_channel_status(
  pdca_channel_num_t pdca_ch_number)
```

Table 6-19. Parameters

Data direction	Parameter name	Description
[in]	pdca_ch_number	PDCA channel

Returns

1 if channel transfer is enabled.

0 if channel transfer is disabled.

6.4 Enumeration Definitions

6.4.1 Enum pdca_channel_status

PDCA channel status.

Table 6-20. Members

Enum value	Description
PDCA_CH_FREE	PDCA channel disabled.
PDCA_CH_BUSY	PDCA channel enabled but transfer is on-going.
PDCA_CH_COUNTER_RELOAD_IS_ZERO	PDCA channel counter reload is zero.
PDCA_CH_TRANSFER_COMPLETED	PDCA channel has completed a block transfer.
PDCA_CH_TRANSFER_ERROR	PDCA channel failed to complete a block transfer.

7. Extra Information for Peripheral DMA Controller Driver

7.1 Acronyms

Below is a table listing the acronyms used in this module, along with their intended meanings.

Acronym	Definition
DMA	Direct Memory Access
QSG	Quick Start Guide

7.2 Dependencies

This driver has the following dependencies:

- System Clock Management (Sysclock)
- Sleep Manager (Sleepmgr)

7.3 Errata

There are no errata related to this driver.

7.4 Module History

An overview of the module history is presented in the table below, with details on the enhancements and fixes made to the module since its first release. The current version of this corresponds to the newest version in the table.

Changelog
Initial document release

8. Examples for Peripheral DMA Controller

This is a list of the available Quick Start Guides (QSGs) and example applications for SAM4L Peripheral DMA Controller (PDCA). QSGs are simple examples with step-by-step instructions to configure and use this driver in a selection of use cases. Note that QSGs can be compiled as a standalone application or be added to the user application.

- Peripheral DMA Controller Example

8.1 Peripheral DMA Controller Example

8.1.1 Purpose

The ASF example in pdca/pdca_usart_example demonstrates how to use PDCA driver to send data to the USART.

8.1.2 Requirements

This example can be used on SAM4L series.

8.1.3 Description

The SAM4L controller sends data in ascii_anim1.h and ascii_anim2.h from USART to the terminal.

8.1.4 Usage

1. Build the program and download it into the evaluation board.
2. On the computer, open, and configure a terminal application (e.g., HyperTerminal on Microsoft® Windows®) with these settings:
   - 115200 bauds
   - 8 bits of data
   - No parity
   - 1 stop bit
   - No flow control
3. In the terminal window, the following text should appear (values depend on the board and chip used):

```
-- PDCA_USART Example xxx --
-- xxxxxx-xx
-- Compiled: xxx xx xxxx xx:xx:xx --
```

4. The sent text should appear.

9. Quickstart guide for SAM PDCA driver

This is the quickstart guide for the SAM PDCA driver, with step-by-step instructions on how to configure and use the driver in a selection of use cases.

The use cases contain several code fragments. The code fragments in the steps for setup can be copied into a custom initialization function, while the steps for usage can be copied into, e.g., the main application function.

9.1 Basic use case

In this basic use case, the PDCA module and channel are configured for:

- Select USART2 as peripheral
- Interrupt-based handling

9.1.1 Prerequisites

- System Clock Management (Sysclock)

9.2 Setup steps

9.2.1 Example code

Add to application C-file:

```
void pdca_callback(void)
{
   //Get PDCA RX channel status and check if PDCA transfer complete
   if (status == PDCA_CH_TRANSFER_COMPLETED) {
       pdca_channel_write_load(PDCA_RX_CHANNEL, g_uc_pdc_buffer, BUFFER_SIZE);
       pdca_channel_write_load(PDCA_TX_CHANNEL, g_uc_pdc_buffer, BUFFER_SIZE);
   }
}
void pdca_setup(void)
{
   pdca_enable(PDCA);

   pdca_channel_write_config(PDCA_RX_CHANNEL, &PDCA_RX_CONFIGS);
   pdca_channel_write_config(PDCA_TX_CHANNEL, &PDCA_TX_CONFIGS);

   pdca_channel_set_callback(PDCA_RX_CHANNEL, pdca_tranfer_done, PDCA_0_IRQn, 1, PDCA_IER_TRC);

   pdca_channel_enable(PDCA_RX_CHANNEL);
   pdca_channel_enable(PDCA_TX_CHANNEL);
}
```

9.2.2 Workflow

1. Define the interrupt callback function in the application:

```
void pdca_callback(void)
{
   //Get PDCA RX channel status and check if PDCA transfer complete
   if (status == PDCA_CH_TRANSFER_COMPLETED) {
       pdca_channel_write_load(PDCA_RX_CHANNEL, g_uc_pdc_buffer, BUFFER_SIZE);
       pdca_channel_write_load(PDCA_TX_CHANNEL, g_uc_pdc_buffer, BUFFER_SIZE);
   }
}
```

2. Enable PDCA module:

```
pdca_enable(PDCA);
```

Note

Includes enabling the module clock and clock sleep mode.

3. Configure PDCA channel with specified mode:

```
pdca_channel_write_config(PDCA_RX_CHANNEL, &PDCA_RX_CONFIGS);
                pdca_channel_write_config(PDCA_TX_CHANNEL, &PDCA_TX_CONFIGS);
```

4. Set the PDCA callback function and enable PDCA interrupt.

```
pdca_channel_set_callback(PDCA_RX_CHANNEL, pdca_tranfer_done, PDCA_0_IRQn, 1, PDCA_IER_TRC);
```

5. Enable PDCA channel:

```
pdca_channel_enable(PDCA_RX_CHANNEL);
                pdca_channel_enable(PDCA_TX_CHANNEL);
```

Index

Document Revision History

Doc. Rev.	Date	Comments
42313A	05/2014	Initial document release

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | www.atmel.com