SMART ARM-based Microcontrollers

AT07912: SAM4L Parallel Capture (PARC) Driver

APPLICATION NOTE

Introduction

This driver for Atmel® | SMART ARM®-based microcontrollers provides an interface for the configuration and management of the Parallel Capture module.

The Parallel Capture module samples an external 8-bit bus with an external input clock. It can be connected to a CMOS digital image sensor, an ADC, a DSP synchronous port, etc.

Devices from the following series can use this module:

- Atmel | SMART SAM4L

The outline of this documentation is as follows:

Table of Contents

1. Software License

2. Prerequisites

There are no prerequisites for this module.

3. Module Overview

The Parallel Capture (PARC) module samples an external 8-bit bus with an external input clock. It can be connected to a CMOS digital image sensor, an ADC, a DSP synchronous port, etc.

3.1. Features

The PARC module offers the following features:

- Captures 8-bit data with an external input clock
- Data captures can be concatenated to form 16-bit and 32-bit values
- Two external data enables PCEN1 and PCEN2
- Four data sample conditions:
  - When PCEN1 is high
  - When PCEN1 and PCEN2 are both high
  - When PCEN1 or PCEN2 are high
  - PC1EN1 and PCEN2 are both inactive i.e. data is sampled without condition
- Peripheral Direct Memory Access (DMA) is supported
- Peripheral events are supported

3.2. Concatenated Data Capture

Captured data bytes are stored in the module's Receive Holding Register (RHR). Concatenated data can also be stored in the RHR to make 16-bit or 32-bit values, with the first byte received in the Least Significant Byte (LSB) position. Refer to the diagram entitled "parallel capture waveforms" in the parc chapter of the device-specific datasheet for more information.

4. Special Considerations

4.1. I/O Lines

The PARC pins are multiplexed with other peripherals. The user application must first configure the I/O Controller to yield control of the pins.

4.2. Power Management

The PARC module stops functioning when the system enters a sleep mode that disables its clock.

4.3. Clocks

The PARC module's clock is generated by the Power Manager. It can be disabled either manually via software or automatically when the system enters a sleep mode that disables the clocks to the peripheral bus modules. For correct behavior, the PARC module's clock frequency must be at least twice the PCCK frequency.

4.4. DMA

The PARC module's DMA handshake interface is connected to the Peripheral DMA Controller (PDCA). Using the PARC module's DMA functionality requires the PDCA to be configured first.

4.5. Interrupt

The PARC interrupt request line is connected to the Nested Vectored Interrupt Controller (NVIC). Using the PARC interrupt requires that the NVIC to be configured first.

5. Extra Information

For extra information, see Extra Information for Parallel Capture. This includes:

- Acronyms
- Dependencies
- Errata
- Module History

6. Examples

For a list of examples related to this driver, see Examples for Parallel Capture.

7. API Overview

7.1. Variable and Type Definitions

7.1.1. Type parc_callback_t

```
typedef void(* parc_callback_t )(const struct parc_module *const
module_inst)
```

PARC interrupt handler callback type.

7.2. Structure Definitions

7.2.1. Struct parc_config

PARC configuration structure

Note: This structure should be initialized by the parc_get_config_defaults() function before being further modified by the user application.

Table 7-1 Members

Type	Name	Description
enum parc_capture_mode	capture_mode	Capture mode.
enum parc_data_size	dsize	Data size.
enum parc_sampling_edge	sampling_edge	Sampling edge select.
enum parc_smode	smode	Sampling mode.

7.2.2. Struct parc_module

PARC driver structure.

Note: This structure should be initialized by the function parc_init().

Table 7-2 Members

Type	Name	Description
parc_callback_t	callback[]	Array to store callback functions (**for PARC driver use only**).
uint8_t	enabled_callback_mask	Bitmask of callbacks enabled/disabled (**for PARC driver use only**).
Parc *	hw	Base address of the PARC module.

Type	Name	Description
struct parc_config *	parc_cfg	Pointer to the PARC configuration structure.
uint8_t	registered_callback_mask	Bitmask of callbacks registered (**for PARC driver use only**).

7.3. Function Definitions

7.3.1. Function parc_clear_status()

Clear PARC interrupt(s) status flag.

```
enum status_code parc_clear_status(
        struct parc_module *const module_inst,
        const uint32_t status)
```

Table 7-3 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Driver structure pointer
[in]	status	A bitmask of interrupt sources to clear

Returns
The clear interrupt operation status.

Table 7-4 Return Values

Return value	Description
STATUS_OK	The interrupt was cleared successfully

7.3.2. Function parc_disable()

Disable the PARC module.

```
enum status_code parc_disable(
        struct parc_module *const module_inst)
```

Table 7-5 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Driver structure pointer

Returns
The disable procedure status.

Table 7-6 Return Values

Return value	Description
STATUS_OK	The disable procedure was successful

7.3.3. Function parc_disable_callback()

Disable a PARC interrupt callback function.

```
enum status_code parc_disable_callback(
        struct parc_module *const module_inst,
        enum parc_callback_type type)
```

Table 7-7 Parameters

Data direction	Parameter name	Description
[in, out]	module_inst	Driver structure pointer
[in]	type	Interrupt callback to disable

Returns
The status of the interrupt callback disable operation.

Table 7-8 Return Values

Return value	Description
STATUS_OK	Interrupt callback disabled successfully

7.3.4. Function parc_disable_events()

Disable the PARC events mode.

```
enum status_code parc_disable_events(
        struct parc_module *const module_inst)
```

Table 7-9 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Driver structure pointer

Returns
The status of the event disable operation.

Table 7-10 Return Values

Return value	Description
STATUS_OK	The event was disabled successfully

7.3.5. Function parc_disable_interrupts()

Disable PARC interrupt(s).

```
void parc_disable_interrupts(
        struct parc_module *const module_inst,
        enum parc_interrupt_source source)
```

Table 7-11 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Driver structure pointer
[in]	source	Bitmask of interrupt sources to disable

7.3.6. Function parc_enable()

Enable the PARC module.

```
enum status_code parc_enable(
        struct parc_module *const module_inst)
```

Table 7-12 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Driver structure pointer

Returns
The enable procedure status.

Table 7-13 Return Values

Return value	Description
STATUS_OK	The enable procedure was successful

7.3.7. Function parc_enable_callback()

Enable a PARC interrupt callback function.

```
enum status_code parc_enable_callback(
        struct parc_module *const module_inst,
        enum parc_callback_type type)
```

Note: The callback function will be called from the PARC interrupt handler when the callback's corresponding interrupt is asserted.

Table 7-14 Parameters

Data direction	Parameter name	Description
[in, out]	module_inst	Driver structure pointer
[in]	type	Interrupt callback to enable

Returns
The status of the interrupt callback enable operation.

Table 7-15 Return Values

Return value	Description
STATUS_OK	Interrupt callback enabled successfully

7.3.8. Function parc_enable_events()

Enable the PARC events mode.

```
enum status_code parc_enable_events(
        struct parc_module *const module_inst)
```

Table 7-16 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Driver structure pointer

Returns
The status of the event enable operation.

Table 7-17 Return Values

Return value	Description
STATUS_OK	The event was enabled successfully

7.3.9. Function parc_enable_interrupts()

Enable PARC interrupt(s).

```
void parc_enable_interrupts(
        struct parc_module *const module_inst,
        enum parc_interrupt_source source)
```

Table 7-18 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Driver structure pointer
[in]	source	Bitmask of interrupt sources to enable

7.3.10. Function parc_get_config_defaults()

Initialize a PARC configuration structure to default values.

```
void parc_get_config_defaults(
        struct parc_config *const config)
```

Initializes the supplied PARC configuration structure to a set of known default values.

Note: This function should be called to populate new instances of the configuration structure before being further modified by the user application.

The default configurations are:

- Capture every byte
- Sample the data bus on the rising edge of the PCCK input clock
- Sample data regardless of the levels of PCEN1 and PCEN2
- 8-bit data width

Table 7-19 Parameters

Data direction	Parameter name	Description
[out]	config	Pointer to a PARC configuration structure

7.3.11. Function parc_get_status()

Get the PARC module's current status.

```
uint32_t parc_get_status(
        struct parc_module *const module_inst)
```

Table 7-20 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Driver structure pointer

Returns
The PARC module's current status as a bitmask.

Table 7-21 Return Values

Return value	Description
PARC_STATUS_EN	PARC is enabled
PARC_STATUS_CS	PARC capture status
PARC_STATUS_DRDY	PARC data ready
PARC_STATUS_OVR	PARC overrun

7.3.12. Function parc_get_version()

Get the PARC module version.

```
uint32_t parc_get_version(
        struct parc_module *const module_inst)
```

Table 7-22 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Driver structure pointer

Returns
The PARC module version.

7.3.13. Function parc_init()

Initialize the PARC module.

```
enum status_code parc_init(
        struct parc_module *const module_inst,
```

```
        Parc *const hw,
        struct parc_config *const config)
```

Initialize the PARC driver structure and the hardware module based on the given configuration structure's contents.

Table 7-23 Parameters

Data direction	Parameter name	Description
[in, out]	module_inst	Pointer to the PARC driver structure
[in]	hw	Pointer to the PARC module hardware register structure
[in]	config	Pointer to the PARC configuration structure

Returns
The initialization status.

Table 7-24 Return Values

Return value	Description
STATUS_OK	The initialization was successful

7.3.14. Function parc_is_data_ready()

Check the data ready status of PARC.

```
bool parc_is_data_ready(
        struct parc_module *const module_inst)
```

Table 7-25 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Driver structure pointer

Returns
The PARC data ready status.

Table 7-26 Return Values

Return value	Description
false	Captured data is not ready
true	Captured data is ready

7.3.15. Function parc_read()

Get PARC data.

```
enum status_code parc_read(
        struct parc_module *const module_inst,
        uint32_t * data)
```

Table 7-27 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Driver structure pointer
[out]	data	Pointer to store the captured data in

Returns
The status of the PARC read data request.

Table 7-28 Return Values

Return value	Description
STATUS_OK	Captured data was read successfully
STATUS_ERR_BUSY	Captured data is not ready

7.3.16. Function parc_read_interrupt_mask()

Get the PARC interrupt mask.

```
uint32_t parc_read_interrupt_mask(
        struct parc_module *const module_inst)
```

Table 7-29 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Driver structure pointer

Returns
The PARC interrupt mask.

7.3.17. Function parc_register_callback()

Register a PARC interrupt callback function.

```
enum status_code parc_register_callback(
        struct parc_module *const module,
        parc_callback_t const callback_func,
        enum parc_callback_type callback_type)
```

Note: The callback must be enabled by using parc_enable_callback.

Table 7-30 Parameters

Data direction	Parameter name	Description
[in, out]	module	Driver structure pointer
[in]	callback_func	Pointer to a callback function
[in]	callback_type	Interrupt callback type

Returns
The status of the interrupt callback register operation.

Table 7-31 Return Values

Return value	Description
STATUS_OK	PARC interrupt callback was registered successfully

7.3.18. Function parc_set_config()

Configure the PARC module.

```
enum status_code parc_set_config(
        struct parc_module *const module_inst,
        struct parc_config * config)
```

Configure the PARC module from the supplied configuration structure.

Table 7-32 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Pointer to the PARC driver structure
[in]	config	Pointer to the PARC configuration structure

Returns
The configuration status.

Table 7-33 Return Values

Return value	Description
STATUS_OK	The configuration was successful

7.3.19. Function parc_start_capture()

Start a PARC conversion.

```
void parc_start_capture(
        struct parc_module *const module_inst)
```

Table 7-34 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Driver structure pointer

7.3.20. Function parc_stop_capture()

Stop a PARC conversion.

```
void parc_stop_capture(
        struct parc_module *const module_inst)
```

Table 7-35 Parameters

Data direction	Parameter name	Description
[in]	module_inst	Driver structure pointer

7.3.21. Function parc_unregister_callback()

Unregister a PARC interrupt callback.

```
enum status_code parc_unregister_callback(
        struct parc_module *const module,
        enum parc_callback_type callback_type)
```

Table 7-36 Parameters

Data direction	Parameter name	Description
[in, out]	module	Driver structure pointer
[in]	callback_type	Interrupt callback type

Returns
The status of the interrupt callback unregister operation.

Table 7-37 Return Values

Return value	Description
STATUS_OK	PARC interrupt callback was unregistered successfully

7.4. Enumeration Definitions

7.4.1. Enum parc_callback_type

PARC callback type.

Table 7-38 Members

Enum value	Description
PARC_CALLBACK_DATA_READY	Data ready callback.
PARC_CALLBACK_OVERRUN	Data overrun callback.

7.4.2. Enum parc_capture_mode

PARC captured byte.

Table 7-39 Members

Enum value	Description
PARC_BOTH_CAPTURE	Bytes are captured when data is captured every cycle.
PARC_EVEN_CAPTURE	Even bytes are captured when data is captured every two cycles.
PARC_ODD_CAPTURE	Odd bytes are captured when data is captured every two cycles.

7.4.3. Enum parc_data_size

PARC data Size.

Table 7-40 Members

Enum value	Description
PARC_DSIZE_BYTE	PARC data transfer size is 8-bits.
PARC_DSIZE_HALFWORD	PARC data transfer size is 16-bits.
PARC_DSIZE_WORD	PARC data transfer size is 32-bits.

7.4.4. Enum parc_interrupt_source

PARC interrupt source.

Table 7-41 Members

Enum value	Description
PARC_INTERRUPT_DRDY	Data ready interrupt.
PARC_INTERRUPT_OVR	Data overrun interrupt.

7.4.5. Enum parc_sampling_edge

PARC sampling edge.

Table 7-42 Members

Enum value	Description
PARC_RISING_EDGE	Data capture occurs on the rising edge of PCCK.
PARC_FALLING_EDGE	Data capture occurs on the falling edge of PCCK.

7.4.6. Enum parc_smode

PARC sample mode.

Table 7-43 Members

Enum value	Description
PARC_SMODE_PCEN1_H	Data capture occurs with PCEN1 high.
PARC_SMODE_PCEN1_AND_PCEN2_H	Data capture occurs with PCEN1 and PCEN2 high.

Enum value	Description
PARC_SMODE_PCEN1_OR_PCEN2_H	Data capture occurs with either PCEN1 or PCEN2 high.
PARC_SMODE_ALWAYS	Data capture always occurs.

7.4.7. Enum parc_status

PARC status.

Table 7-44 Members

Enum value	Description
PARC_STATUS_EN	PARC module enable/disable status.
PARC_STATUS_CS	PARC Capture mode enable/disable status.
PARC_STATUS_DRDY	Data is ready in the receive holding register.
PARC_STATUS_OVR	Overrun error.

8. Extra Information for Parallel Capture

8.1. Acronyms

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

Acronym	Definition
ADC	Analog to Digital Converter
CMOS	Complementary Metal-Oxide-Semiconductor
DMA	Direct Memory Access
DSP	Digital Signal Processor
I/O	Input/Output
LSB	Least Significant Byte
NVIC	Nested Vectored Interrupt Controller
PCCK	Parallel Capture Clock
PDCA	Peripheral DMA Controller
RHR	Receive Holding Register
QSG	Quick Start Guide

8.2. Dependencies

This driver has the following dependencies:

- None

8.3. Errata

There are no errata related to this driver.

8.4. 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

9. Examples for Parallel Capture

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

- Quick Start Guide for the PARC driver
- Parallel Capture (PARC) - Example Using GPIO as Stimuli

9.1. Quick Start Guide for the PARC driver

This is the quick start guide for the SAM4L Parallel Capture (PARC) Driver, with step-by-step instructions on how to configure and use the driver for a specific use case.

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.

9.1.1. Use Cases

- PARC Basic Use Case

9.1.2. PARC Basic Use Case

This use case demonstrates how to use the PARC module on SAM4L. In this use case the PARC module is configured as:

- Capture every byte
- Sample the data bus on the rising edge of the PCCK input clock
- Sample data regardless of the levels of PCEN1 and PCEN2
- 8-bit data width

9.1.3. Setup Steps

9.1.3.1. Prerequisites

This module requires the following service:

- System clock (sysclk)

9.1.3.2. Setup Code Example

Add the following to the main loop or a setup function:

```
struct parc_module module_inst;
struct parc_config config;
```

```
// Get default configuration
parc_get_config_defaults(&config);
```

```
// Initialize PARC.
parc_init(&module_inst, PARC, &config);

// Enable the PARC
parc_enable(&module_inst);
```

```
// Start capture.
parc_start_capture(&module_inst);
```

9.1.3.3. Basic Setup Workflow

1. Initialize and configure PARC:

```
// Get default configuration
parc_get_config_defaults(&config);
```

```
// Initialize PARC.
parc_init(&module_inst, PARC, &config);
```

2. Enable the PARC module and start capture:

```
// Enable the PARC
parc_enable(&module_inst);

// Start capture.
parc_start_capture(&module_inst);
```

9.1.4. PARC Basic Usage

9.1.4.1. Basic Usage Code

We can poll the data ready status and then read it once capture has finished:

```
uint32_t captured_data;

while (!parc_is_data_ready(&module_inst)) {
    }
parc_read(&module_inst, &captured_data);
```

We can enable the data ready interrupt and install a callback function to perform a custom task:

```
parc_register_callback(&module_inst,
    (parc_callback_t)parc_complete_callback, PARC_CALLBACK_DATA_READY);
parc_enable_interrupts(&module_inst, PARC_INTERRUPT_DRDY);
parc_enable_callback(&module_inst, PARC_CALLBACK_DATA_READY);
parc_start_capture(&module_inst);

// The callback function example.
static void parc_complete_callback(
    struct parc_module *const module)
{
  callback_data_ready = true;
  parc_read(module, &captured_data);
}
```

9.2. Parallel Capture (PARC) - Example Using GPIO as Stimuli

9.2.1. Purpose

This example demonstrates the data capture function provided by the PARC module.

9.2.2. Requirements

This example can be used with the following evaluation kits with PARC and PDCA modules:

- SAM4L Xplained Pro
- SAM4L8 Xplained Pro

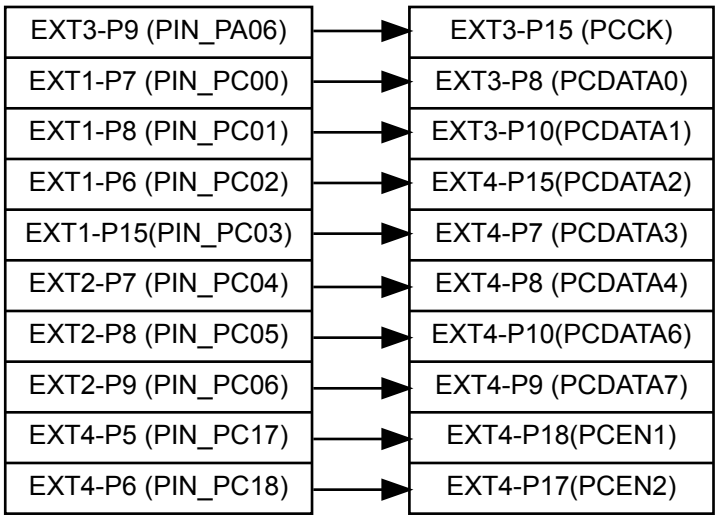
9.2.3. Description

In this example the GPIO pins on the same evaluation board act as the signal source which provides the PARC module with data, clock, and EN signals.

The GPIO pins should be connected to the PARC pins through the development kit's on-board connectors. These pins can be connected easily by using interconnect wires on the SAM4L Xplained Pro and SAM4L8 Xplained Pro evaluation kits.

The connection list on SAM4L Xplained Pro or SAM4L8 Xplained Pro is shown in Figure 9-1 Description on page 25.

Figure 9-1 Description



Note: The PCDATA5 signal is only connected to the LCD connector (EXT5) which cannot be connected to easily using interconnect wires. In this example PCDATA5 is *not required* so can be left unconnected.

9.2.4. Main Files

- parc.c: Parallel Capture driver
- parc.h: Parallel Capture driver header file
- parc_example.c: Parallel Capture example application

9.2.5. Compilation Information

This software is written for GNU GCC and IAR Embedded Workbench® for Atmel®. Other compilers may or may not work.

9.2.6. Usage

1. Connect the GPIO pins and PARC port according to the above connection list.
2. Build the program and download it into the evaluation kit.
3. On the computer, open and configure a terminal application (e.g., HyperTerminal on Microsoft® Windows®) with these settings:
   - 115200 baud

- 8 bits of data
- No parity
- 1 stop bit
- No flow control

4. Start the application.
5. In the terminal window, the following text should appear:

```
    -- SAM PARC Example --
    -- xxxxxx-xx
    -- Compiled: xxx xx xxxx xx:xx:xx --
```

6. Select the PARC configuration by inputing 'y' or 'n' when the following information is displayed on the terminal:

```
  Press y to sample the data when both data enable pins are enabled.
  Press n to sample the data, don't care the status of the data enable
pins.

  Press y to sample all the data.
  Press n to sample the data only one out of two.
```

7. PARC captures data and sends the captured data to the terminal.

10. Document Revision History

Doc. Rev.	Date	Comments
42297B	07/2015	Updated title of application note and added list of supported devices
42297A	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**