APPLICATION NOTE

AT07334: SAM4 TWI Master Mode Driver

ASF PROGRAMMERS MANUAL

SAM4 TWI Master Mode Driver

This documents describe the usage of the driver for the TWI (Two-Wire Interface) Master on the Atmel® SAM4 family of devices. It describes usage in an I^2C compatible manner and does not provide any guidence for the use of SMBus.

Table of Contents

Software License

1. Prequisites

There are no prequisites for this module.

2. Module Overview

This driver provides access to the main features of the TWIM controller. The two wire interface connects components via a two-wire serial bus, each device, on the bus, has a unique ID.

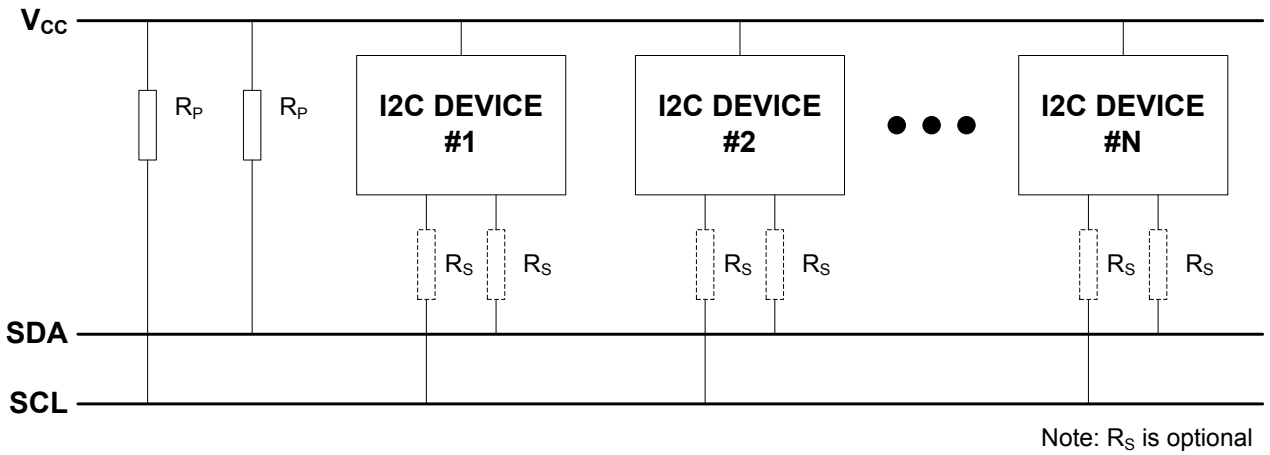
The TWI modules are programmable as masters with sequential or single-byte access. High speed mode capability is also supported.

The TWI bus provides a simple, but efficient method of interconnecting multiple master and slave devices. An arbitration mechanism is provided for resolving bus ownership between masters, as only one master device may own the bus at any given time. The arbitration mechanism relies on the wired-AND connections to avoid bus drivers short-circuiting.

2.1 TWI Bus Topology

Figure 2-1: I2C bus topology on page 6. The pull-up resistors (R_s) will provide a high level on the bus lines when none of the TWI devices are driving the bus. These are optional, and can be replaced with a constant current source.

Figure 2-1. I2C bus topology



Note: R_S is optional

3. Examples

- TWIM Master Example
- Quick Start Guide

4. API Overview

4.1 Variable and Type Definitions

4.1.1 Type twim_transfer_status_t

```
typedef enum twim_transfer_status twim_transfer_status_t
```

See twim_transfer_status for enumeration definitions.

4.1.2 Type twim_callback_t

```
typedef void(* twim_callback_t )(Twim *)
```

4.2 Structure Definitions

4.2.1 Struct twim_config

This is the configuration structure for the TWI Master device. It is used as an argument for twim_set_config to provide the desired configurations for the module.

Table 4-1. Members

Type	Name	Description
uint8_t	clock_drive_strength_low	Pull-down drive strength of the TWCK output buffer
uint8_t	clock_slew_limit	Slew limit of the TWCK output buffer
uint8_t	data_drive_strength_low	Pull-down drive strength of the TWD output buffer
uint8_t	data_setup_cycles	Clock cycles for data setup count
uint8_t	data_slew_limit	Slew limit of the TWD output buffer
uint8_t	hs_clock_drive_strength_high	Pull-up drive strength of the TWCK output buffer in high speed mode
uint8_t	hs_clock_drive_strength_low	Pull-down drive strength of the TWCK output buffer in high speed mode
uint8_t	hs_clock_slew_limit	Slew limit of the TWCK output buffer in high speed mode
uint8_t	hs_data_drive_strength_low	Pull-down drive strength of the TWD output buffer in high speed mode
uint8_t	hs_data_slew_limit	Slew limit of the TWD output buffer in high speed mode
uint8_t	hsmode_data_setup_cycles	Clock cycles for data setup count in high speed mode
uint32_t	hsmode_speed	The baudrate of the TWI bus in high speed mode

Type	Name	Description
bool	smbus	SMBUS mode
uint32_t	speed	The baudrate of the TWI bus
uint32_t	twim_clk	The TWIM clock frequency

4.2.2 Struct twim_package

Table 4-2. Members

Type	Name	Description
uint8_t	addr[]	TWI address/commands to issue to the other chip (node)
uint8_t	addr_length	Length of the TWI data address segment (1-3 bytes)
void *	buffer	Where to find the data to be written
uint32_t	chip	TWI chip address to communicate with
bool	high_speed	Indicate if it is a high-speed transfer
uint8_t	high_speed_code	High speed mode master code, valid if high_speed is true
uint32_t	length	How many bytes do we want to write
bool	ten_bit	Indicate if it is 10-bit addressing

4.3 Macro Definitions

4.3.1 TWI Driver Compatibility

These codes are defined for SAM devices to simplify the porting of existing code from other Atmel devices. For the definition of technical terms and acronyms refer to Acronyms and the appropriate device datasheet.

4.3.1.1 Macro twi_options_t

```
#define twi_options_t
```

4.3.1.2 Macro twi_package_t

```
#define twi_package_t
```

4.3.1.3 Macro twi_master_init

```
#define twi_master_init
```

4.3.1.4 Macro twi_probe

```
#define twi_probe
```

4.3.2 Macro TWI_FAST_MODE_PLUS_SPEED

```
#define TWI_FAST_MODE_PLUS_SPEED
```

TWI Fast Mode Plus mode.

4.3.3 Macro TWI_FAST_MODE_SPEED

```
#define TWI_FAST_MODE_SPEED
```

Value to set TWI to Fast speed mode.

4.3.4 Macro TWI_HIGH_SPEED_MODE_SPEED

```
#define TWI_HIGH_SPEED_MODE_SPEED
```

Value to set TWI to High speed plus mode.

4.3.5 Macro TWI_STD_MODE_SPEED

```
#define TWI_STD_MODE_SPEED
```

Value to set TWI to Standard speed mode.

4.3.6 Macro TWIM_IER_NAK_MASK

```
#define TWIM_IER_NAK_MASK
```

Interrupt Enable Register mask to enable interupts for NAKs. Used to set the register.

4.3.7 Macro TWIM_IER_STD_MASK

```
#define TWIM_IER_STD_MASK
```

Interrupt Enable Register mask to enable interupts on ANAK or ARBLST Used to set the register.

4.3.8 Macro TWIM_LOW_POWER_ENABLE

```
#define TWIM_LOW_POWER_ENABLE
```

Enable TWIM Low Power Transfer by default.

4.3.9 Macro TWIM_SCR_NAK_MASK

```
#define TWIM_SCR_NAK_MASK
```

Mask to clear the DNAK and ANAK bits in the Status Clear register.

4.3.10 Macro TWIM_SR_NAK_MASK

```
#define TWIM_SR_NAK_MASK
```

Status Register mask to test for NAKs in data or address phase.

4.3.11 Macro TWIM_SR_STD_MASK

```
#define TWIM_SR_STD_MASK
```

Status Register mask to test for conditions ANAK or ARBLST.

4.4 Function Definitions

4.4.1 Function twi_master_read()

Read multiple bytes from a TWI compatible slave device.

```
status_code_t twi_master_read(
  Twim * twim,
  struct twim_package * package)
```

Table 4-3. Parameters

Data direction	Parameter name	Description
[in]	twim	Base address of the TWIM
[in]	package	Package information and data

Table 4-4. Return Values

Return value	Description
STATUS_OK	If all bytes were read successfully
ERR_IO_ERROR	NACK received or Bus Arbitration lost

4.4.2 Function twi_master_write()

Write multiple bytes to a TWI compatible slave device.

```
status_code_t twi_master_write(
  Twim * twim,
  struct twim_package * package)
```

Table 4-5. Parameters

Data direction	Parameter name	Description
[in]	twim	Base address of the TWIM
[in]	package	Package information and data

Table 4-6. Return Values

Return value	Description
STATUS_OK	If all bytes were send successfully
ERR_IO_ERROR	NACK received or Bus Arbitration lost

4.4.3 Function twim_clear_status()

Clear the current status of the TWIM.

```
void twim_clear_status(
  Twim * twim,
  uint32_t clear_status)
```

Write the value passed in clear_status to the Status Clear Register. Bits set in clear_status result in setting to zero the corresponding position in the Status Register. See the appropriate datasheet for more detail.

Table 4-7. Parameters

Data direction	Parameter name	Description
[in]	twim	Base address of the TWIM
[in]	clear_status	The TWIM status

4.4.4 Function twim_default_callback()

TWIM default callback.

```
void twim_default_callback(
  Twim * twim)
```

Each time a TWI packet is transmitted a callback function is executed. This provides default functionality and is activated by calling twim_set_callback .

Table 4-8. Parameters

Data direction	Parameter name	Description
[in]	twim	Base address of the TWIM

4.4.5 Function twim_disable()

Sets the Master Disable bit of the Control Register, which disables master mode.

```
void twim_disable(
```

```
  Twim * twim)
```

Table 4-9. Parameters

Data direction	Parameter name	Description
[in]	twim	Base address of the TWIM instance

4.4.6 Function twim_disable_interrupt()

Disable the TWIM interrupts and clear their status.

```
void twim_disable_interrupt(
  Twim * twim,
  uint32_t interrupt_source)
```

Write the value passed in interrupt_source into the Interrupt Disable Register. Each bit clears the corresponding bit in the Interrupt Mask Register and the Status Clear Register. For more detail see the appropriate datasheet.

Table 4-10. Parameters

Data direction	Parameter name	Description
[in]	twim	Base address of the TWIM
[in]	interrupt_source	The TWI interrupt to be disabled

4.4.7 Function twim_enable()

Sets the Master Enable bit of the Control Register, which enables TWI master mode.

```
void twim_enable(
  Twim * twim)
```

Table 4-11. Parameters

Data direction	Parameter name	Description
[in]	twim	Base address of the TWIM instance

4.4.8 Function twim_enable_interrupt()

Enable the TWIM interrupts.

```
void twim_enable_interrupt(
  Twim * twim,
  uint32_t interrupt_source)
```

Write the value passed in interrupt_source into the Interrupt Enable Register. Each bit sets the corresponding bit in the Interrupt Mask Register. For more detail see the appropriate datasheet.

Table 4-12. Parameters

Data direction	Parameter name	Description
[in]	twim	Base address of the TWIM

Data direction	Parameter name	Description
[in]	interrupt_source	The TWI interrupt to be enabled

4.4.9 Function twim_get_interrupt_mask()

Return the current content of the Interrupt Mask Register.

```
uint32_t twim_get_interrupt_mask(
  Twim * twim)
```

Table 4-13. Parameters

Data direction	Parameter name	Description
[in]	twim	Base address of the TWIM

Returns TWIM interrupt mask.

4.4.10 Function twim_get_status()

Return the contents of Status Register.

```
uint32_t twim_get_status(
  Twim * twim)
```

Table 4-14. Parameters

Data direction	Parameter name	Description
[in]	twim	Base address of the TWIM

Returns Contents of TWI Status Register.

4.4.11 Function twim_pdca_transfer_prepare()

Setup the TWI master for a PDCA (Peripheral DMA Controler) transfer.

```
void twim_pdca_transfer_prepare(
  Twim * twim,
  twi_package_t * package,
  bool read)
```

Setup a DMA transfer. See twim_package

Table 4-15. Parameters

Data direction	Parameter name	Description
[in]	twim	Base address of the TWIM
[in]	package	Package information and data, see twim_package

Data direction	Parameter name	Description
[in]	read	True if it's a read trasnfer

4.4.12 Function twim_probe()

Test if a device answers for a given TWI address.

```
status_code_t twim_probe(
  Twim * twim,
  uint32_t chip_addr)
```

Table 4-16. Parameters

Data direction	Parameter name	Description
[in]	twim	Base address of the TWIM
[in]	chip_addr	Address of the device being probed for

Table 4-17. Return Values

Return value	Description
STATUS_OK	Slave Found
ERR_IO_ERROR	ANAK received or Bus Arbitration lost

4.4.13 Function twim_set_callback()

Set callback for TWIM.

```
void twim_set_callback(
  Twim * twim,
  uint32_t interrupt_source,
  twim_callback_t callback,
  uint8_t irq_level)
```

For a specfied interrupt source and interrupt level, register an interrupt service routine.

Table 4-18. Parameters

Data direction	Parameter name	Description
[in]	twim	Base address of the TWIM
[in]	interrupt_source	TWIM interrupt source
[in]	callback	Callback function pointer
[in]	irq_level	Interrupt level

4.4.14 Function twim_set_config()

Initialize the TWIM module.

```
status_code_t twim_set_config(
  Twim * twim,
  struct twim_config * config)
```

Table 4-19. Parameters

Data direction	Parameter name	Description
[in]	twim	Base address of the TWIM
[in]	config	Options for initializing the TWIM module

Table 4-20. Return Values

Return value	Description
STATUS_OK	Transaction is successful
ERR_INVALID_ARG	Invalid arg resulting in wrong CWGR Exponential

4.4.15 Function twim_set_hsmode_speed()

Set the TWI bus speed in conjunction with the clock frequency in high speed mode.

```
status_code_t twim_set_hsmode_speed(
  Twim * twim,
  uint32_t speed,
  uint32_t clk,
  uint8_t cycles)
```

Table 4-21. Parameters

Data direction	Parameter name	Description
[in]	twim	Base address of the TWIM
[in]	speed	The desired twim bus speed
[in]	clk	The current running system clock frequency
[in]	cycles	Clock cycles for data setup count

Table 4-22. Return Values

Return value	Description
STATUS_OK	Transaction is successful
ERR_INVALID_ARG	Invalid arg resulting in wrong CWGR Exponential

4.4.16 Function twim_set_speed()

Set the TWI bus speed in conjunction with the clock frequency.

```
status_code_t twim_set_speed(
  Twim * twim,
  uint32_t speed,
  uint32_t clk,
  uint8_t cycles)
```

Table 4-23. Parameters

Data direction	Parameter name	Description
[in]	twim	Base address of the TWIM
[in]	speed	The desired twim bus speed

Data direction	Parameter name	Description
[in]	clk	The current running system clock frequency
[in]	cycles	Clock cycles for data setup count

Table 4-24. Return Values

Return value	Description
STATUS_OK	Transaction is successful
ERR_INVALID_ARG	Invalid arg resulting in wrong CWGR Exponential

4.5 Enumeration Definitions

4.5.1 Enum twim_transfer_status

Table 4-25. Members

Enum value	Description
TWI_SUCCESS	TWI Transaction Success
TWI_INVALID_ARGUMENT	Invalid Argument Passed
TWI_ARBITRATION_LOST	Bus Arbitration Lost
TWI_NO_CHIP_FOUND	Slave Not Found
TWI_RECEIVE_NACK	Data No Acknowledgement Received
TWI_SEND_NACK	Data No Acknowledgement Send
TWI_INVALID_CLOCK_DIV	Invalid Clock Divider Value

5. Special Considerations

NONE.

6. Extra Information

6.1 Acronyms

Term	Definition
SCR	Status Clear Register
NAK	Negative Acknowledgement
DNAK	Negative Acknowledgement int TWI data phase
ANAK	Negative Acknowledgement int TWI address phase
ARBLST	Arbitration lost to a higher priority bus master in TWI multi-master mode
IER	Interupt Enable Register

7. TWIM Master Example

7.1 Purpose

This is an example of how to use the TWIM driver to access an EEPROM.

7.2 Requirements

The program needs a TWI-compatible EEPROM connected as shown below:

7.3 Connections for Board: SAM4L Xplained Pro

Board	Pin	EEPROM
TWIMS3	TWD(PB14,EXT2/PIN11	SDA
TWIMS3	TWCK(PB15,EXT2/PIN12)	SCL
VCC		VCC
GND		GND

7.4 Connections for Board: SAM4L-EK

Board	Pin	EEPROM
TWIMS3	TWD(PB0)	SDA
TWIMS3	TWCK(PB01)	SCL
VCC		VCC
GND		GND

7.5 Description

1. Write a data pattern to the EEPROM device.
2. Read data back from the EEPROM.
3. Compare the two. If they match then it worked.

7.6 Usage

1. Build the program and download it into the evaluation board.
2. Connect a serial cable to the UART port for each evaluation kit.
3. On the computer, open, and configure a terminal application (e.g. HyperTerminal on Microsoft® Windows®) with these settings:
   - 115200 bauds
   - 8 data bits
   - No parity
   - 1 stop bit
   - No flow control
4. Start the application. The following traces shall appear on the terminal:

```
-- TWIM Master Example --
-- xxxxxx-xx
-- Compiled: xxx xx xxxx xx:xx:xx --
```

8. Quick Start Guide

This is the quick start guide for the SAM4 TWI Master Mode Driver, with step-by-step instructions on how to configure and use the driver for a specific use case.The code examples can be copied into the main application loop or any other function that will need to control the TWIM module.

8.1 Use Cases

- TWIM Basic Usage

8.2 TWIM Basic Usage

This use case will demonstrate how to initialize the TWIM module.

8.3 Setup Steps

8.3.1 Prerequisites

This module requires the following service register

- clk_group

8.3.2 Basic Setup Workflow

Setup TWIM options, including TWIM clock frequency, the desired TWI bus speed, the target chip slave address (optional) and being in SMBus mode or not. See twim_config for more information.

```
struct twim_config twim_conf;
```

Get the system clock frequency, in Hertz.

```
twim_conf.twim_clk = sysclk_get_cpu_hz();
```

Select the desired TWI speed. These are defined by a number of macros e.g. TWI_STD_MODE_SPEED, TWI_FAST_MODE_SPEED for the complete list see the Macro definitions section.

```
twim_conf.speed = DESIRED_TWI_BUS_SPEED;
```

```
twim_conf.smbus = false;   // Not SMBus mode
```

Set other settings to defaults.

```
twim_conf.hsmode_speed = 0;
twim_conf.data_setup_cycles = 0;
twim_conf.hsmode_data_setup_cycles = 0;
```

Now make the configuration active:

```
twim_set_config(TWIM0, &twim_conf);
```

Note

The TWIM driver supports I^2C standard speed, fast speed, fast speed plus, and high speed.
The default callback function must be set before calling read/write functions.

```
twim_set_callback(TWIM0, 0, twim_default_callback, 1);
```

Note The read/write functions will enable and disable the corresponding interrupt sources.

8.4 Usage Steps

8.4.1 twim_basic_usage_code

We can send data to the target slave device. Firstly, the data package should be prepared. In one data package, several items should be set; the target slave address, the internal address (if needed), the length of the internal address (if needed), the data buffer to be written, and the length of the data buffer.

```
twi_package_t packet_tx;
packet_tx.chip = TARGET_SLAVE_ADDRESS;  // The address of the TWI Slave device.
packet_tx.addr[0] = (INTERNAL_ADDRESS >> 16) & 0xFF;
packet_tx.addr[1] = (INTERNAL_ADDRESS >> 8) & 0xFF;
packet_tx.addr_length = INTERNAL_ADDRESS_LENGTH;
packet_tx.buffer = (void *) data_buf_tx;
packet_tx.length = DATA_BUF_TX_LENGTH;
```

Note The TWIM driver supports 1-3 bytes of internal address.

After the data package is ready, we can call twi_master_write() to send the package to the slave address. The callback set before will be handled in ISR.

```
twi_master_write(TWIM1, &packet_tx);
```

Note If the function returns STATUS_OK, the package has been sent to the target slave device successfully. Otherwise, the transmission fails.

We can receive data from the target slave device. Firstly, the data package should be prepared. In one data package, several items should be set; the target slave address, the internal address (if needed), the length of the internal address (if needed), the data buffer used to store received data and the length of the data to be received.

```
twi_package_t packet_rx;
packet_rx.chip = TARGET_SLAVE_ADDRESS;
packet_rx.addr[0] = (INTERNAL_ADDRESS >> 16) & 0xFF;
packet_rx.addr[1] = (INTERNAL_ADDRESS >> 8) & 0xFF;
packet_rx.addr_length = INTERNAL_ADDRESS_LENGTH;
packet_rx.buffer = (void *) data_buf_rx;
packet_rx.length = DATA_BUF_RX_LENGTH;
```

Note The TWIM driver supports 1-3 bytes of internal address.

Once the data package is ready, we can call twi_master_read() to receive a data package, in responde, from the slave device. The callback set before will be handled in ISR.

```
twi_master_read(TWIM1, &packet_rx);
```

Note If the function returns STATUS_OK, the package has been received from the target slave device and the data has been stored in the data buffer successfully. Otherwise, the transmission failed.

Index

Document Revision History

Doc. Rev.	Date	Comments
42274A	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