REC10B0002-0100

Everywhere you imagine. RENESAS

M16C/62P Group
Renesas Embedded
Application Programming Interface
Reference Manual

Reference Manual

Rev.1.00
Revision Date: Nov.1, 2007

Renesas Technology
www.renesas.com

Renesas Embedded
Application Programming Interface
Reference Manual
<M16C/62P Group>

Table of Contents

1. Introduction

The Renesas Embedded Application Programming Interface (API) is a unified API for the microcomputers made by Renesas Technology Corporation.

2. Driver

2.1 Overview

The library described herein provides a peripheral facility control program (peripheral driver) for microcomputers. Use of the Renesas API permits the peripheral driver to be built into a user program.

2.2 Driver Features

The library described herein has the following features available as a peripheral driver.

(1) Serial I/O control feature

It comprises a serial interface driver, which sets or clears the conditions of serial communication, as well as controls and manages the transmission/reception of communication data.

(2) Timer control feature

It comprises a timer driver, which sets or clears the operating conditions of timers, as well as controls the timer operation.

(3) I/O port control feature

It comprises an I/O port driver, which sets or clears the usage conditions of I/O ports, as well as control data read/write operation.

(4) External interrupt control feature

It comprises an external interrupt driver, which sets or clears the usage conditions of external interrupts, as well as controls interrupt operation.

(5) A/D converter control feature

It comprises an A/D converter driver, which sets or clears the usage conditions of A/D converters, as well as controls A/D converter operation.

2.3 Serial Interface Driver

The serial interface driver sets serial communication, clears settings, transmit/receives data, and controls the status of serial communication.

There are two kinds of serial interface driver: a single-data transmission/reception API and a multi-data transmission/reception API.

2.4 Timer Driver

The timer driver sets the timer, clears timer settings, controls timer operation, and acquires a counter value with respect to the following modes:

- Timer mode
- Event counter mode
- Pulse width modulation mode (PWM mode)
- Pulse period measurement mode
- Pulse width measurement mode

2.4.1 Timer Mode

In this mode, the timer counts the internally generated count source. When an underflow or an overflow interrupt occurs, it calls a preset callback function.

2.4.2 Event Counter Mode

In this mode, the timer counts the external signal fed in from an input pin or an overflow or underflow from other timer. When an underflow or an overflow interrupt occurs, it calls a preset callback function.

2.4.3 Pulse Width Modulation Mode (PWM Mode)

In this mode, the timer outputs pulses in a given width successively. When an underflow or an overflow interrupt occurs, it calls a preset callback function.

2.4.4 Pulse Period Measurement Mode

In this mode, the timer measures the pulse period of an external signal fed in from an input pin. When an underflow or an overflow interrupt occurs, it calls a preset callback function.

2.4.5 Pulse Width Measurement Mode

In this mode, the timer measures the pulse width of an external signal fed in from an input pin. When an underflow or an overflow interrupt occurs, it calls a preset callback function.

2.5 I/O Port Driver

The I/O port driver sets the I/O port for input or output, writes data to the I/O port, and reads data from the I/O port.

2.6 External Interrupt Driver

The external interrupt driver sets external interrupts, controls external interrupts, acquires the status of external interrupt flags, and clears external interrupt flags.

2.7 A/D Converter Driver

The A/D converter driver sets the A/D converter, controls the A/D converter, clears settings of the A/D converter, acquires the A/D converter value, acquires the status of the A/D converter, and clears the status of the A/D converter.

3. Standard Types

This section describes the standard types defined in the library. For details about the set values, refer to the description of each API.

Standard type	Description
Boolean	The Boolean type represents the enum-type data that indicates whether successful (RAPI_TRUE (= 1)) or failed (RAPI_FALSE (= 0)).
VoidFuncNotify	The VoidFuncNotify type represents the type of the notification function to be registered.

4. Library Reference

4.1 API List by Peripheral Facility

The table below lists the Renesas Embedded APIs classified by peripheral facility.

NO	Facility classification	API	API operation
1	Single-data serial I/O	R_BasicOpenSerialDriver	Opens serial port
2		R_BasicCloseSerialDriver	Closes serial port
3		R_BasicSetSerialFormat	Sets serial communication
4		R_BasicStartSerialReceiving	Receives 1 data
5		R_BasicStartSerialSending	Transmits 1 data
6		R_BasicReceivingStatusRead	Reads receive status
7		R_BasicSendingStatusRead	Reads transmit status
8		R_BasicStopSerialReceiving	Stops reception
9		R_BasicStopSerialSending	Stops transmission
10	Multi-data serial I/O	R_OpenSerialDriver	Opens serial port
11		R_CloseSerialDriver	Closes serial port
12		R_ConfigSerialDriverNotify	Registers notification function
13		R_SetSerialFormat	Sets serial communication
14		R_SetSerialInterrupt	Sets transmit/receive interrupt
15		R_StartSerialReceiving	Starts reception
16		R_StartSerialSending	Starts transmission
17		R_StopSerialReceiving	Stops reception
18		R_StopSerialSending	Stops transmission
19		R_PollingSerialReceiving	Receives by polling
20		R_PollingSerialSending	Transmits by polling
21	Timer	R_CreateTimer	Sets timer mode
22		R_EnableTimer	Controls timer mode operation
23		R_DestroyTimer	Clears timer mode setting
24		R_CreateEventCounter	Sets event counter mode
25		R_EnableEventCounter	Controls operation of event counter mode
26		R_DestroyEventCounter	Clears setting of event counter mode
27		R_GetEventCounter	Gets event counter mode counter value
28		R_CreatePulseWidthModulationMode	Sets pulse width modulation mode
29		R_EnablePulseWidthModulationMode	Controls operation of pulse width modulation mode
30		R_DestroyPulseWidthModulationMode	Clears setting of pulse width modulation mode
31		R_CreatePulsePeriodMeasurementMode	Sets pulse period measurement mode
32		R_EnablePulsePeriodMeasurementMode	Controls operation of pulse period measurement mode
33		R_DestroyPulsePeriodMeasurementMode	Clears setting of pulse width measurement mode
34		R_GetPulsePeriodMeasurementMode	Acquires measured value of pulse period measurement mode

35		__CreatePulseWidthMeasurementMode	Sets pulse width measurement mode
36		__EnablePulseWidthMeasurementMode	Controls operation of pulse width measurement mode
37	Timer	__DestroyPulseWidthMeasurementMode	Clears setting of pulse width measurement mode
38		__GetPulseWidthMeasurementMode	Acquires measured value of pulse width measurement mode
39		__SetTimerRegister	Sets timer register
40		__EnableTimerRegister	Controls operation of timer register
41		__ClearTimerRegister	Clears timer register
42		__GetTimerRegister	Gets timer register value
43	I/O port	__SetIOPort	Sets I/O port
44		__ReadIOPort	Reads from I/O port
45		__WriteIOPort	Writes to I/O port
46		__SetIOPortRegister	Sets I/O port register
47		__ReadIOPortRegister	Reads from I/O port register
48		__WriteIOPortRegister	Writes to I/O port register
49	External interrupt	__SetInterrupt	Sets external interrupt
50		__EnableInterrupt	Controls external interrupt
51		__GetInterruptFlag	Gets flag status of external interrupt
52		__ClearInterruptFlag	Clears flag of external interrupt
53	A/D converter	__CreateADC	Sets A/D converter
54		__EnableADC	Controls operation of A/D converter
55		__DestroyADC	Discards settings of A/D converter
56		__GetADC	Gets A/D conversion value (register specified)
67		__GetADCAll	Gets A/D conversion value (all registers)

4.2 Description of Each API

This section describes each API and explains how to use them, showing a program example for each.

The description of each API is divided into the following items.

- **Synopsis** : Outlines the content of processing performed by the function. It also shows the syntax of the function, followed by a brief explanation of arguments.
- **Description** : Describes the function and how to use it in detail.
- **Return value** : Explains the returned value of the function.
- **Functionality** : Indicates the functional classification of the function.
- **Reference** : Indicates the related functions.
- **Remark** : Describes the precautions to be taken when using the API.
- **Program example** : Presents a program showing how to use the function.

4.2.1 Serial I/O
__BasicOpenSerialDriver

Synopsis

<Open a serial port>

Boolean __BasicOpenSerialDriver(unsigned long data)

data	Setup data

Description

Opens and initializes a specified serial port.

For data, the following values can be set.

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

Return value

If the serial port specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Serial I/O

Reference

__BasicCloseSerialDriver

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
    /* Open serial driver */
    return __BasicOpenSerialDriver( RAPI_COM1 );
}
```

__BasicCloseSerialDriver

Synopsis

<Close a serial port>

Boolean __BasicCloseSerialDriver(unsigned long data)

data	Setup data

Description

Closes a specified serial port. For data, the following values can be set.

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

Return value

If the serial port specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Serial I/O

Reference

__BasicOpenSerialDriver

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
   /* Close serial driver */
   return __BasicCloseSerialDriver( RAPI_COM1 );
}
```

__BasicSetSerialFormat

Synopsis

<Set serial communication>

Boolean __BasicSetSerialFormat(unsigned long data1, unsigned char data2)

data1	Setup data 1
data2	Setup data 2

Description

Sets serial communication according to specified parameters.

[data1]

For data1, the following values can be set. To set multiple definition values at the same time, use the symbol "|" to separate each specified value.

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

For serial communication mode, the following values can be set.

(UART0, UART1, UART2)

RAPI_SM_SYNC	Clock synchronous serial communication mode
RAPI_SM_ASYNC	Clock asynchronous serial communication mode

(SI/O3, SI/O4)

RAPI_SIO_SM_SYNC	Clock synchronous serial communication mode

For the data length format of clock asynchronous serial communication, the following values can be set.

If the API is used in clock synchronous serial communication mode, do not set these values.

(UART0, UART1, UART2)

RAPI_BIT_7	Transfer data length 7 bits	RAPI_BIT_8	Transfer data length 8 bits
RAPI_BIT_9	Transfer data length 9 bits		

For the clock source of serial communication, the following values can be set.

(UART0, UART1, UART2)

RAPI_CKDIR_INT	Internal clock is used as the clock source of serial communication.
RAPI_CKDIR_EXT	External clock is used as the clock source of serial communication.

(SI/O3, SI/O4)

RAPI_SIO_CKDIR_INT	Internal clock is used as the clock source of serial communication.
RAPI_SIO_CKDIR_EXT	External clock is used as the clock source of serial communication.

For the stop bit length of clock asynchronous serial communication, the following values can be set.

If the API is used in clock synchronous serial communication mode, do not set these values.

(UART0, UART1, UART2)

RAPI_STPB_1	1 stop bit	RAPI_STPB_2	2 stop bits

For the parity bit of clock asynchronous serial communication, the following values can be set.

If the API is used in clock synchronous serial communication mode, do not set these values.

(UART0, UART1, UART2)

RAPI_PARITY_NON	No parity bit	RAPI_PARITY_EVEN	Even parity bit
RAPI_PARITY_ODD	Odd parity bit		

For the clock polarity of serial communication, the following values can be set.

If the API is used in clock asynchronous serial communication mode, do not set these values.

(UART0, UART1, UART2)

RAPI_DPOL_NON	Polarity not inverted	RAPI_DPOL_INV	Polarity inverted

(SI/O3, SI/O4)

RAPI_SIO_DPOL_NON	Polarity not inverted	RAPI_SIO_DPOL_INV	Polarity inverted

For the count source of the built-in baud rate generator, the following values can be set.

(UART0, UART1, UART2)

RAPI_BCSS_F1	f1SIO	RAPI_BCSS_F2	f2SIO
RAPI_BCSS_F8	f8SIO	RAPI_BCSS_F32	f32SIO

(SI/O3, SI/O4)

RAPI_SIO_BCSS_F1	f1SIO	RAPI_SIO_BCSS_F2	f2SIO
RAPI_SIO_BCSS_F8	f8SIO	RAPI_SIO_BCSS_F32	f32SIO

For the _CTS/_RTS function, the following values can be set.

If the internal clock is selected for use in clock synchronous serial communication mode, the _RTS function has no effect.

(UART0, UART1, UART2)

RAPI_CTSRTS_DIS	_CTS/_RTS functions are not used.
RAPI_CTS_SEL	_CTS function is selected.
RAPI_RTS_SEL	_RTS function is selected.

For the transfer format, the following values can be set.

If the data length selected for use in clock asynchronous serial communication mode is 7 or 9 bits long, do not set these values.

(UART0, UART1, UART2)

RAPI_LSB_SEL	LSB first	RAPI_MSB_SEL	MSB first

(SI/O3, SI/O4)

RAPI_SIO_LSB_SEL	LSB first	RAPI_SIO_MSB_SEL	MSB first

For serial data logic switchover, the following values can be set.

(UART0, UART1, UART2)

RAPI_LOGIC_NO_REV	The value written in the transmit buffer register does not have its logic inverted.
RAPI_LOGIC_REV	The value written in the transmit buffer register is inverted before being transmitted.

[data2]

Sets the divide-by-N value of a communication speed.

Return value

If serial communication was successfully set, RAPI_TRUE is returned; if settings failed, RAPI_FALSE is returned.

Functionality Serial I/O

Reference

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include  ”rapi_sif_m16c_62p”

Boolean func( void )
{
   /* Set the data of RAPI_COM1 to serial driver */
   Return _BasicSetSerialFormat(RAPI_COM1 | RAPI_SM_SYNC | RAPI_CKDIR_INT
            | RAPI_BCSS_F1 | RAPI_DPOL_NON | RAPI_LSB_SEL, 20);
}
```

__BasicStartSerialReceiving

Synopsis

<Receive 1 data>

Boolean __BasicStartSerialReceiving(unsigned long data)

data	Setup data

Description

Starts receiving 1 data of serial communication.

[data]

For data, the following values can be set.

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

Return value

If data reception in serial communication was successfully started, RAPI_TRUE is returned; if failed, RAPI_FALSE is returned.

Functionality

Serial I/O

Reference

__BasicReceivingStatusRead, __BasicStopSerialReceiving

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include  ”rapi_sif_m16c_62p”

void func( void )
{
    ..........
    __BasicStartSerialReceiving( RAPI_COM1 );
    ..........
}
```

__BasicStartSerialSending

Synopsis

<Transmit 1 data>

Boolean _BasicStartSerialSending(unsigned long data1, unsigned int data2)

data	Setup data
data	Transmit data

Description

Starts sending 1 data of serial communication.

For data1, the following values can be set.

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

Return value

If data transmission in serial communication was successfully started, RAPI_TRUE is returned; if failed, RAPI_FALSE is returned.

Functionality

Serial I/O

Reference

__BasicSendingStatusRead, __BasicStopSerialSending

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include  ”rapi_sif_m16c_62p”

Void func( void )
{
    ..........
    __BasicStartSerialSending( RAPI_COM1, 0x00AA );
    ..........
}
```

__BasicReceivingStatusRead

Synopsis

<Read receive status>

unsigned int __BasicReceivingStatusRead(unsigned long data)

data	Setup data

Description

Returns the receive status of serial communication.

[data]

For data, the following values can be set.

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

Return value

The receive status of serial communication is returned. The returned value is one of the following.

(UART0, UART1, UART2)

RAPI_RX_INCOMPLETE	Reception not complete yet.
Other than above	Reception complete. The value read from the UARTi receive buffer register (i = 0 to 2).

(SI/O3, SI/O4)

RAPI_RX_INCOMPLETE	Reception not complete yet.
Other than above	Reception complete. Low-order 8 bits: The value read from the SI/Oi transmit/receive register (i = 3, 4).

Functionality

Serial I/O

Reference

__BasicStartSerialReceiving, __BasicStopSerialReceiving

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include  ”rapi_sif_m16c_62p”

void func( void )
{
    unsigned int rcv_data;

    ..........
    rcv_data = __BasicReceivingStatusRead( RAPI_COM1 );
    ..........
}
```

__BasicSendingStatusRead

Synopsis

<Read transmit status>

Boolean __BasicSendingStatusRead(unsigned long data)

data	Setup data

Description

Returns the transmit status of serial communication. For data, the following values can be set.

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

Return value

If no data exists in the transmit buffer, RAPI_TRUE is returned; if data exists, RAPI_FALSE is returned.

Functionality

Serial I/O

Reference

__BasicStartSerialSending, __BasicStopSerialSending

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
    ..........
    if (__BasicSendingStatusRead( RAPI_COM1 ) == RAPI_TRUE) {
       /* Transmission completion */
    }
    ..........
}
```

__BasicStopSerialReceiving

Synopsis

<Stop reception>

Boolean Rapi_BasicStopSerialReceiving(unsigned long data)

data	Setup data

Description

Stops receiving data in serial communication

[data]

For data, the following values can be set.

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2		

Return value

If data reception in serial communication was successfully stopped, RAPI_TRUE is returned; if failed, RAPI_FALSE is returned.

Functionality

Serial I/O

Reference

__BasicStartSerialReceiving

Remark

- For the M16C SI/03 and SI/04, this API cannot be used.
- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
    /* Stop receiving data in serial communication */
    __BasicStopSerialReceiving ( RAPI_COM1 );
}
```

__BasicStopSerialSending

Synopsis

<Stop transmission>

Boolean __BasicStopSerialSending(unsigned long data)

data	Setup data

Description

Stops transmitting data in serial communication.

[data]

For data, the following values can be set.

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2		

Return value

If data transmission in serial communication was successfully stopped, RAPI_TRUE is returned; if failed, RAPI_FALSE is returned.

Functionality

Serial I/O

Reference

__BasicStartSerialSending

Remark

- For the M16C SI/03 and SI/04, this API cannot be used.
- When operating in clock synchronous serial communication mode, data reception is stopped at the same time by this API.
- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
    /* Stop sending data in serial communication */
    __BasicStopSerialSending ( RAPI_COM1 );
}
```

__OpenSerialDriver

Synopsis

<Open a serial port>

Boolean __OpenSerialDriver(unsigned long data)

data	Setup data

Description

Opens and initializes a specified serial port.

[data]

For data, the following values can be set.

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

Return value

If the serial port specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Serial I/O

Reference

__CloseSerialDriver

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
   /* Open serial driver */
   return __OpenSerialDriver( RAPI_COM1 );
}
```

__CloseSerialDriver

Synopsis

<Close a serial port>

Boolean __CloseSerialDriver(unsigned long data)

data	Setup data

Description

Closes a specified serial port.

[data]

For data, the following values can be set.

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

Return value

If the serial port specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Serial I/O

Reference

__OpenSerialDriver

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
   /* Close serial driver */
   return __CloseSerialDriver( RAPI_COM1 );
}
```

__ConfigSerialDriverNotify

Synopsis

<Register a notification function>

Boolean __ConfigSerialDriverNotify(unsigned long data, VoidFuncNotify *func)

data	Setup data
func	Function pointer to be registered

Description

Registers the notification function necessary to get various transmit/receive information of serial communication.

[data]

For data, the following values can be set.

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

[func]

The function to be registered in func must be supplied to the serial I/O driver by the user.

The serial I/O driver calls the function registered in func.

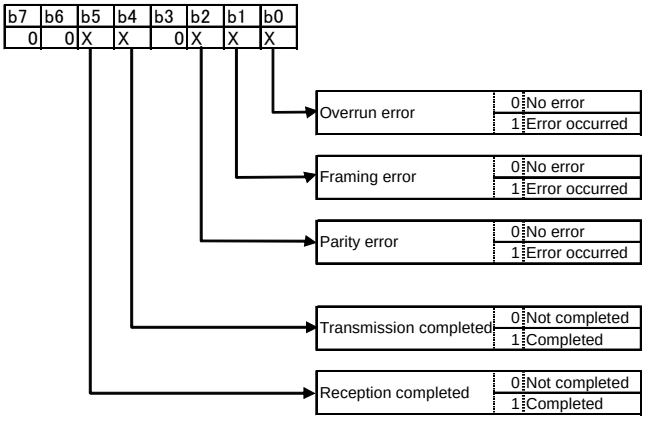
The serial I/O driver notifies the user of the transmit/receive status by an argument.

The type of the function to be registered is shown below.

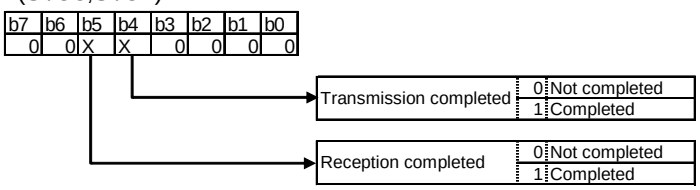
void “any function name” (unsigned char notify);

The argument is detailed below.

(UART0, UART1, UART2)



(SI/O3,SI/O4)



Return value

If the serial port specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Serial I/O

Reference

__StartSerialReceiving, __StartSerialSending

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include ”rapi_sif_m16c_62p”

void Notify(unsigned char result) {
   if ((result&RAPI_OVER_ERR) == RAPI_OVER_ERR) {
      /* Overrun error */
   }
   if ((result&RAPI_FRAMING_ERR) == RAPI_FRAMING_ERR) {
      /* Framing error */
   }
   if ((result&RAPI_PARITY_ERR) == RAPI_PARITY_ERR) {
      /* Parity error */
   }
   if ((result&RAPI_TX_END) == RAPI_TX_END) {
      /* Transmission completion */
   }
   if ((result&RAPI_RX_END) == RAPI_RX_END) {
      /* Reception completion */
   }
}

Boolean func( void )
{
   /* Set callback functions of RAPI_COM1 to serial driver */
   return __ConfigSerialDriverNotify( RAPI_COM1, Notify );
}
```

__SetSerialFormat

Synopsis

<Set serial communication>

Boolean __SetSerialFormat(unsigned long data1, unsigned char data2)

data1	Setup data 1
data2	Setup data 2

Description

Sets serial communication according to specified parameters.

For details about parameters, refer to the description of __BasicSetSerialFormat.

Return value

If serial communication was successfully set, RAPI_TRUE is returned; if settings failed, RAPI_FALSE is returned.

Functionality

Serial I/O

Reference

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include  ”rapi_sif_m16c_62p”

Boolean func( void )
{
    /* Set the data of RAPI_COM1 to serial driver */
    return __SetSerialFormat(RAPI_COM1 | RAPI_SM_SYNC | RAPI_CKDIR_INT |
                          RAPI_BCSS_F1 | RAPI_DPOL_NON | RAPI_LSB_SEL, 20);
}
```

__SetSerialInterrupt

Synopsis

<Set serial interrupts>

Boolean __SetSerialInterrupt(unsigned long data)

data	Setup data

Description

Sets serial interrupts according to specified parameters.

[data]

For data, the following values can be set.

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

For interrupt settings, the following values can set.

(UART0, UART1, UART2)

RAPI_INT_TX_DIS	Transmit interrupt disabled
RAPI_INT_TX_LV_1	Transmit interrupt priority level 1
RAPI_INT_TX_LV_2	Transmit interrupt priority level 2
RAPI_INT_TX_LV_3	Transmit interrupt priority level 3
RAPI_INT_TX_LV_4	Transmit interrupt priority level 4
RAPI_INT_TX_LV_5	Transmit interrupt priority level 5
RAPI_INT_TX_LV_6	Transmit interrupt priority level 6
RAPI_INT_TX_LV_7	Transmit interrupt priority level 7
RAPI_INT_RX_DIS	Receive interrupt disabled
RAPI_INT_RX_LV_1	Receive interrupt priority level 1
RAPI_INT_RX_LV_2	Receive interrupt priority level 2
RAPI_INT_RX_LV_3	Receive interrupt priority level 3
RAPI_INT_RX_LV_4	Receive interrupt priority level 4
RAPI_INT_RX_LV_5	Receive interrupt priority level 5
RAPI_INT_RX_LV_6	Receive interrupt priority level 6
RAPI_INT_RX_LV_7	Receive interrupt priority level 7

(SI/O3, SI/O4)

RAPI_INT_SIO_DIS	SI/O interrupt disabled
RAPI_INT_SIO_LV_1	SI/O interrupt priority level 1
RAPI_INT_SIO_LV_2	SI/O interrupt priority level 2
RAPI_INT_SIO_LV_3	SI/O interrupt priority level 3
RAPI_INT_SIO_LV_4	SI/O interrupt priority level 4
RAPI_INT_SIO_LV_5	SI/O interrupt priority level 5
RAPI_INT_SIO_LV_6	SI/O interrupt priority level 6
RAPI_INT_SIO_LV_7	SI/O interrupt priority level 7

Return value

If the serial port specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality Serial I/O

Reference

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include  ”rapi_sif_m16c_62p”

Boolean func( void )
{
   /* Set interrupt of RAPI_COM1 to serial driver */
   return    __SetSerialInterrupt(    RAPI_COM1    |    RAPI_INT_TX_LV_1    |
                                    RAPI_INT_RX_LV_2 );
}
```

__StartSerialReceiving

Synopsis

<Start reception>

Boolean __StartSerialReceiving(unsigned long data, unsigned char wordNum, unsigned int *RcvDtBuf)

data	Setup data
wordNum	Number of words received
RcvDtBuf	Pointer to the buffer in which received data is stored

Description

Starts reception of serial communication and gets received data by a specified number of words. When acquisition of received data is complete, this API calls a notification function (if a notification function is registered).

[data]

For data, the following values can be set.

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

Return value

If reception of serial communication was successfully started, RAPI_TRUE is returned; if failed, RAPI_FALSE is returned.

Functionality

Serial I/O

Reference

__ConfigSerialDriverNotify, __StopSerialReceiving

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include  ”rapi_sif_m16c_62p”

unsigned int buffer[10];
void func( void )
{
    /* Get 5 word data received in serial communication */
    __StartSerialReceiving( RAPI_COM1, 5, buffer );
}
```

__StartSerialSending

Synopsis

<Start transmission>

Boolean __StartSerialSending(unsigned long data, unsigned char wordNum, unsigned int *SndDtBuf)

data	Setup data
wordNum	Number of words transmitted
SndDtBuf	Pointer to the transmit data

Description

Starts transmission of serial communication and writes transmit data to the transmit buffer by a specified number of words. When transmission of all transmit data is complete, this API calls a notification function (if a notification function is registered).

[data]

For data, the following values can be set.

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

Return value

If transmission of serial communication was successfully started, RAPI_TRUE is returned; if failed, RAPI_FALSE is returned.

Functionality

Serial I/O

Reference

__ConfigSerialDriverNotify, __StopSerialSending

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include ”rapi_sif_m16c_62p”

unsigned int buffer[10];
void func( void )
{
    /* Set 5 word data to transmit buffer of serial communication */
    __StartSerialSending( RAPI_COM1, 5, buffer );
}
```

__StopSerialReceiving

Synopsis

<Stop reception>

Boolean __StopSerialReceiving(unsigned long data)

data	Setup data

Description

Stops reception of serial communication.

[data]

For data, the following values can be set.

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2		

Return value

If reception of serial communication was successfully stopped, RAPI_TRUE is returned; if failed, RAPI_FALSE is returned.

Functionality

Serial I/O

Reference

__StartSerialReceiving

Remark

- For the M16C SI/03 and SI/04, this API cannot be used.
- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
    /* Stop receiving data in serial communication */
    __StopSerialReceiving ( RAPI_COM1 );
}
```

__StopSerialSending

Synopsis

<Stop transmission>

Boolean __StopSerialSending(unsigned long data)

data	Setup data

Description

Stops transmission of serial communication.

[data]

For data, the following values can be set.

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2		

Return value

If transmission of serial communication was successfully stopped, RAPI_TRUE is returned; if failed, RAPI_FALSE is returned.

Functionality

Serial I/O

Reference

__StartSerialReceiving

Remark

- For the M16C SI/03 and SI/04, this API cannot be used.
- When operating in clock synchronous serial communication mode, data reception is stopped at the same time by this API.
- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include ”rapi_sif_m16c_62p”

void func( void )
{
   /* Stop sending data in serial communication */
   __StopSerialSending ( RAPI_COM1 );
}
```

__PollingSerialReceiving

Synopsis

<Polling reception>

Boolean __PollingSerialReceiving(unsigned long data)

data	Setup data

Description

Performs reception of serial communication by polling. This API gets received data by an amount specified by __StartSerialReceiving. When acquisition of received data is complete, it calls a notification function (if a notification function is registered).

[data]

For data, the following values can be set.

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

Return value

If the serial port specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Serial I/O

Reference

__ConfigSerialDriverNotify, __SetSerialInterrupt, __StartSerialReceiving

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include ”rapi_sif_m16c_62p”

unsigned int buffer[10];

void func( void )
{
    /* Reception interrupt disable */
    __SetSerialInterrupt( RAPI_COM1 | RAPI_INT_TX_DIS | RAPI_INT_RX_DIS );
    /* Start reception */
    __StartSerialReceiving( RAPI_COM1, 5, buffer );
    while(1){
        __PollingSerialReceiving( RAPI_COM1 );
    }
}
```

__PollingSerialSending

Synopsis

<Polling transmission>

Boolean __PollingSerialSending(unsigned long data)

data	Setup data

Description

Performs transmission of serial communication by polling. This API sends transmit data by an amount specified by __StartSerialSending from the transmit data buffer specified by __StartSerialSending. When transmission of all transmit data is complete, it calls a notification function (if a notification function is registered).

[data]

For data, the following values can be set.

RAPI_COM1	UART0	RAPI_COM2	UART1
RAPI_COM3	UART2	RAPI_COM4	SI/O3
RAPI_COM5	SI/O4		

Return value

If the serial port specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Serial I/O

Reference

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include ”rapi_sif_m16c_62p”

unsigned int buffer[10];
void func( void )
{
   /* Transmission interrupt disable */
   __SetSerialInterrupt( RAPI_COM1 | RAPI_INT_TX_DIS | RAPI_INT_RX_DIS );
   /* Start transmission */
   __StartSerialSending( RAPI_COM1, 5, buffer );
   while(1){
       __PollingSerialSending( RAPI_COM1 );
   }
}
```

4.2.2 Timer
__CreateTimer

Synopsis

<Set timer mode>

Boolean __CreateTimer(unsigned long data1, unsigned int data2, unsigned int data3, unsigned int data4, void* func)

data1	Setup data 1
data2	Setup data 2
data3	Setup data 3
data4	Setup data 4
func	Callback function pointer (Specify 0 if no callback functions are set.)

Description

Sets a specified timer to timer mode.

[data1]

For data1, the following definition values can be set. To set multiple definition values at the same time, use the symbol "|" to separate each specified value.

RAPI_TIMER_A0	Uses timer A channel 0.
RAPI_TIMER_A1	Uses timer A channel 1.
RAPI_TIMER_A2	Uses timer A channel 2.
RAPI_TIMER_A3	Uses timer A channel 3.
RAPI_TIMER_A4	Uses timer A channel 4.
RAPI_TIMER_B0	Uses timer B channel 0.
RAPI_TIMER_B1	Uses timer B channel 1.
RAPI_TIMER_B2	Uses timer B channel 2.
RAPI_TIMER_B3	Uses timer B channel 3.
RAPI_TIMER_B4	Uses timer B channel 4.
RAPI_TIMER_B5	Uses timer B channel 5.
RAPI_F1	Selects f_1 for the count source.
RAPI_F2	Selects f_2 for the count source.
RAPI_F8	Selects f_8 for the count source.
RAPI_F32	Selects f_{32} for the count source.
RAPI_FC32	Selects f_{c32} for the count source.
RAPI_TIMER_ON	Sets the timer to start operating in __CreateTimer.
RAPI_TIMER_OFF	Sets the timer to stop operating in __CreateTimer.
RAPI_GATE_L	Selects a gate facility that counts a period during which input at TA_{iIN} pin remains low.
RAPI_GATE_H	Selects a gate facility that counts a period during which input at TA_{iIN} pin remains high.
RAPI_PULSE_ON	Selects that pulses are output from TA_{iIN} pin.
RAPI_PULSE_OFF	Selects that no pulses are output from TA_{iIN} pin.

• Specifiable definition values when timer A is used (RAPI_TIMER_A0 to RAPI_TIMER_A4 specified)

(Count source)	Specify one from { RAPI_F1, RAPI_F2, RAPI_F8, RAPI_F32, RAPI_FC32 }. The default value is RAPI_F2.
(Operating states set)	Specify one from { RAPI_TIMER_ON, RAPI_TIMER_OFF }. The default value is RAPI_TIMER_OFF.
(Pulse output state)	Specify one from { RAPI_PULSE_ON, RAPI_PULSE_OFF }. The default value is RAPI_PULSE_OFF.
(Gate facility)	Specify one from { RAPI_GATE_L, RAPI_GATE_H }. If omitted, "No gate facility" is set.

・ Specifiable definition values when timer B is used (RAPI_TIMER_B0 to RAPI_TIMER_B5 specified)

(Count source)	Specify one from { RAPI_F1, RAPI_F2, RAPI_F8, RAPI_F32, RAPI_FC32 }. The default value is RAPI_F2.
(Operating states set)	Specify one from { RAPI_TIMER_ON, RAPI_TIMER_OFF }. The default value is RAPI_TIMER_OFF.

[data2]

Specify the interrupt priority level (0-7) to be set in the interrupt control register.

[data3]

Specify the value to be set in the timer register in 16 bits.

[data4]

Specify 0.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (timer mode)

Reference

__EnableTimer, __DestroyTimer

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_timer_m16c_62p.h"

void TimerIntFunc( void ){}

void func( void )
{
    /* Set up timer A0 as timer mode */
    __CreateTimer(  RAPI_TIMER_A0|RAPI_TIMER_ON|RAPI_F8,  5,  0x80,  0,
                  TimerIntFunc );
}
```

__EnableTimer

Synopsis

<Control operation of timer mode>

Boolean __EnableTimer(unsigned long data)

data	Setup data

Description

Controls operation of the timer that is set to specified timer mode by starting or stopping it.

[data]

For data, the following definition values can be set. To set multiple definition values at the same time, use the symbol "|" to separate each specified value.

RAPI_TIMER_A0	Selects timer A channel 0.
RAPI_TIMER_A1	Selects timer A channel 1.
RAPI_TIMER_A2	Selects timer A channel 2.
RAPI_TIMER_A3	Selects timer A channel 3.
RAPI_TIMER_A4	Selects timer A channel 4.
RAPI_TIMER_B0	Selects timer B channel 0.
RAPI_TIMER_B1	Selects timer B channel 1.
RAPI_TIMER_B2	Selects timer B channel 2.
RAPI_TIMER_B3	Selects timer B channel 3.
RAPI_TIMER_B4	Selects timer B channel 4.
RAPI_TIMER_B5	Selects timer B channel 5.
RAPI_TIMER_ON	Sets the timer that is set to timer mode to start operating.
RAPI_TIMER_OFF	Sets the timer that is set to timer mode to stop operating.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (timer mode)

Reference

__CreateTimer, __DestroyTimer

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
    /* Disable timer A1 as timer mode */
    __EnableTimer( RAPI_TIMER_A1 | RAPI_TIMER_OFF );
}
```

__DestroyTimer

Synopsis

<Discard settings of timer mode>

Boolean __DestroyTimer(unsigned long data)

data	Setup data

Description

Discards settings of the timer that is set to specified timer mode.

[data]

For data, the following definition values can be set.

RAPI_TIMER_A0	Selects timer A channel 0.
RAPI_TIMER_A1	Selects timer A channel 1.
RAPI_TIMER_A2	Selects timer A channel 2
RAPI_TIMER_A3	Selects timer A channel 3.
RAPI_TIMER_A4	Selects timer A channel 4.
RAPI_TIMER_B0	Selects timer B channel 0.
RAPI_TIMER_B1	Selects timer B channel 1.
RAPI_TIMER_B2	Selects timer B channel 2.
RAPI_TIMER_B3	Selects timer B channel 3.
RAPI_TIMER_B4	Selects timer B channel 4.
RAPI_TIMER_B5	Selects timer B channel 5.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (timer mode)

Reference

__CreateTimer, __EnableTimer

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
   /* Destroy the setting of timer A2 as timer mode */
   __DestroyTimer( RAPI_TIMER_A2 );
}
```

__CreateEventCounter

Synopsis

<Set event counter mode>

Boolean __CreateEventCounter(unsigned long data1, unsigned int data2, unsigned int data3, unsigned int data4, void* func)

data1	Setup data 1
data2	Setup data 2
data3	Setup data 3
data4	Setup data 4
func	Callback function pointer (Specify 0 if no callback functions are set.)

Description

Sets a specified timer to event counter mode.

[data1]

For data1, the following definition values can be set. To set multiple definition values at the same time, use the symbol "|" to separate each specified value.

RAPI_TIMER_A0	Uses timer A channel 0.
RAPI_TIMER_A1	Uses timer A channel 1.
RAPI_TIMER_A2	Uses timer A channel 2.
RAPI_TIMER_A3	Uses timer A channel 3.
RAPI_TIMER_A4	Uses timer A channel 4.
RAPI_TIMER_B0	Uses timer B channel 0.
RAPI_TIMER_B1	Uses timer B channel 1.
RAPI_TIMER_B2	Uses timer B channel 2.
RAPI_TIMER_B3	Uses timer B channel 3.
RAPI_TIMER_B4	Uses timer B channel 4.
RAPI_TIMER_B5	Uses timer B channel 5.
RAPI_EV_EXTERNAL	Selects the external signal input to TA_{iIN} pin (when using timer Ai) or TB_{iIN} pin (when using timer Bi) for the count source.
RAPI_EV_TIMER_AJ	Selects overflow or underflow of timer Aj (j = i-1, however j = 4 if i = 0) for the count source.
RAPI_EV_TIMER_AK	Selects overflow or underflow of timer Ak (k = i+1, however k = 0 if i = 4) for the count source.
RAPI_EV_TIMER_B2	Selects overflow or underflow of timer B2 for the count source.
RAPI_EV_TIMER_BJ	Selects overflow or underflow of timer Bj (j = i – 1, however j = 2 if i = 0, j = 5 if i = 3) for the count source.
RAPI_TIMER_ON	Sets the timer to start operating in __CreateEventCounter.
RAPI_TIMER_OFF	Sets the timer to stop operating in __CreateEventCounter.
RAPI_PULSE_ON	Selects that pulses are output from TA_{iIN} pin.
RAPI_PULSE_OFF	Selects that no pulses are output from TA_{iIN} pin.
RAPI_AUTO_RELOAD	Selects reload type for the count type.
RAPI_FREE_RUN	Selects free-run type for the count type.

RAPI_UP_COUNT	Selects up-count for the count operation.
RAPI_DOWN_COUNT	Selects down-count for the count operation.
RAPI_UDF_REGISTER	Selects the UDF register for the cause of up/down switching.
RAPI_TAIOUT	Selects the input signal at TA_{iOUT} pin for the cause of up/down switching.
RAPI_RISING	Selects the rising edge of count source as active edge.
RAPI_FALLING	Selects the falling edge of count source as active edge.
RAPI_BOTH	Selects both rising and falling edges of count source as active edges.

• Specifiable definition values when timer A is used (RAPI_TIMER_A0 to RAPI_TIMER_A4 specified)

(Count source) Specify one from { RAPI_EV_EXTERNAL, RAPI_EV_TIMER_AJ, RAPI_EV_TIMER_AK, RAPI_EV_TIMER_B2 }. The default value is RAPI_EV_EXTERNAL.

(Operating states set) Specify one from { RAPI_TIMER_ON, RAPI_TIMER_OFF }. The default value is RAPI_TIMER_OFF.

(Pulse output facility) Specify one from { RAPI_PULSE_ON, RAPI_PULSE_OFF }. The default value is RAPI_PULSE_OFF.

(Gate facility) Specify one from { RAPI_GATE_L, RAPI_GATE_H }. If omitted, “No gate facility” is set.

(Count type) Specify one from { RAPI_AUTO_RELOAD, RAPI_FREE_RUN }. The default value is RAPI_AUTO_RELOAD.

(Count direction) Specify one from { RAPI_UP_COUNT, RAPI_DOWN_COUNT }. The default value is RAPI_DOWN_COUNT. The count direction can only be set when the UDF register is used.

(Count direction switching) Specify one from { RAPI_UDF_REGISTER, RAPI_TAIOUT }. The default value is RAPI_UDF_REGISTER.

(Count edge) Specify one from { RAPI_RISING, RAPI_FALLING }. The default value is RAPI_FALLING.

• Specifiable definition values when timer B is used (RAPI_TIMER_B0 to RAPI_TIMER_B4 specified)

(Count source) Specify one from { RAPI_EV_EXTERNAL, RAPI_EV_TIMER_BJ }. The default value is RAPI_EV_EXTERNAL.

(Operating states set) Specify one from { RAPI_TIMER_ON, RAPI_TIMER_OFF }. The default value is RAPI_TIMER_OFF.

(Count edge) Specify one from { RAPI_RISING, RAPI_FALLING, RAPI_BOTH }. The default value is RAPI_FALLING.

[data2]

Specify the interrupt priority level (0-7) to be set in the interrupt control register.

[data3]

Specify the value to be set in the timer register in 16 bits.

[data4]

Specify 0.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (event counter mode)

Reference

__EnableEventCounter, __DestroyEventCounter, __GetEventCounter

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_timer_m16c_62p.h"

void TimerIntFunc( void ){}

void func( void )
{
    /* Set up timer B0 as event counter mode */
    __CreateEventCounter( RAPI_TIMER_B0|RAPI_TIMER_ON|RAPI_FALLING, 5,
                          0x80, 0, TimerIntFunc );
}
```

__EnableEventCounter

Synopsis

<Control operation of event counter mode>

Boolean __EnableEventCounter(unsigned long data)

data	Setup data

Description

Controls operation of the timer that is set to specified timer mode by starting or stopping it.

[data]

For data, the following definition values can be set. To set multiple definition values at the same time, use the symbol "|" to separate each specified value.

RAPI_TIMER_A0	Selects timer A channel 0.
RAPI_TIMER_A1	Selects timer A channel 1.
RAPI_TIMER_A2	Selects timer A channel 2.
RAPI_TIMER_A3	Selects timer A channel 3.
RAPI_TIMER_A4	Selects timer A channel 4.
RAPI_TIMER_B0	Selects timer B channel 0.
RAPI_TIMER_B1	Selects timer B channel 1.
RAPI_TIMER_B2	Selects timer B channel 2.
RAPI_TIMER_B3	Selects timer B channel 3.
RAPI_TIMER_B4	Selects timer B channel 4.
RAPI_TIMER_B5	Selects timer B channel 5.
RAPI_TIMER_ON	Sets the timer that is set to event counter mode to start operating.
RAPI_TIMER_OFF	Sets the timer that is set to event counter mode to stop operating.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (event counter mode)

Reference

__CreateEventCounter, __DestroyEventCounter, __GetEventCounter

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
    /* Disable timer B1 as event counter mode */
    __EnableEventCounter( RAPI_TIMER_B1|RAPI_TIMER_OFF );
}
```

__DestroyEventCounter

Synopsis

<Discard settings of event counter mode>

Boolean __DestroyEventCounter(unsigned long data)

data	Setup data

Description

Discards settings of the timer that is set to specified timer mode.

[data]

For data, the following definition values can be set.

RAPI_TIMER_A0	Selects timer A channel 0.
RAPI_TIMER_A1	Selects timer A channel 1.
RAPI_TIMER_A2	Selects timer A channel 2.
RAPI_TIMER_A3	Selects timer A channel 3.
RAPI_TIMER_A4	Selects timer A channel 4.
RAPI_TIMER_B0	Selects timer B channel 0.
RAPI_TIMER_B1	Selects timer B channel 1.
RAPI_TIMER_B2	Selects timer B channel 2.
RAPI_TIMER_B3	Selects timer B channel 3.
RAPI_TIMER_B4	Selects timer B channel 4.
RAPI_TIMER_B5	Selects timer B channel 5.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (event counter mode)

Reference

__CreateEventCounter, __EnableEventCounter, __GetEventCounter

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
    /* Destroy the setting of timer B2 as event counter mode */
    __DestroyEventCounter( RAPI_TIMER_B2 );
}
```

__GetEventCounter

Synopsis

<Get event counter mode counter value>

Boolean __GetEventCounter(unsigned long data1, unsigned int *data2)

data1	Setup data 1
data2	Pointer to the buffer in which counter value is stored

Description

Gets the counter value of the timer that is set to specified event counter mode.

[data1]

For data, the following definition values can be set.

RAPI_TIMER_A0	Selects timer A channel 0.
RAPI_TIMER_A1	Selects timer A channel 1.
RAPI_TIMER_A2	Selects timer A channel 2.
RAPI_TIMER_A3	Selects timer A channel 3.
RAPI_TIMER_A4	Selects timer A channel 4.
RAPI_TIMER_B0	Selects timer B channel 0.
RAPI_TIMER_B1	Selects timer B channel 1.
RAPI_TIMER_B2	Selects timer B channel 2.
RAPI_TIMER_B3	Selects timer B channel 3.
RAPI_TIMER_B4	Selects timer B channel 4.
RAPI_TIMER_B5	Selects timer B channel 5.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (event counter mode)

Reference

__CreateEventCounter, __EnableEventCounter, __DestroyEventCounter

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
    unsigned int data[2];

    /* Get the counter of timer B3 as event counter mode */
    __GetEventCounter(RAPI_TIMER_B3, data );
}
```

__CreatePulseWidthModulationMode

Synopsis

<Set pulse width modulation mode>

Boolean __CreatePulseWidthModulationMode(unsigned long data1, unsigned int data2, unsigned int* data3, void* data4)

data1	Setup data 1
data2	Setup data 2
data3	Setup data 3
func	Callback function pointer (Specify 0 if no callback functions are set.)

Description

Sets a specified timer to pulse width modulation mode.

[data1]

For data1, the following definition values can be set. To set multiple definition values at the same time, use the symbol "|" to separate each specified value.

RAPI_TIMER_A0	Uses timer A channel 0.
RAPI_TIMER_A1	Uses timer A channel 1.
RAPI_TIMER_A2	Uses timer A channel 2.
RAPI_TIMER_A3	Uses timer A channel 3.
RAPI_TIMER_A4	Uses timer A channel 4.
RAPI_F1	Selects f_1 for the count source.
RAPI_F2	Selects f_2 for the count source.
RAPI_F8	Selects f_8 for the count source.
RAPI_F32	Selects f_{32} for the count source.
RAPI_FC32	Selects f_{C32} for the count source.
RAPI_TIMER_ON	Sets the timer to start operating in __CreatePulseWidthModulationMode.
RAPI_TIMER_OFF	Sets the timer to stop operating in __CreatePulseWidthModulationMode.
RAPI_TG_TAIIN	Selects external trigger input from TA_{iIN} pin for the count start condition.
RAPI_EV_ TIMER_AJ	Selects overflow or underflow of timer Aj (j = i – 1, however j = 4 if i = 0) as the trigger for the timer to start counting.
RAPI_EV_TIMER_ AK	Selects overflow or underflow of timer Ak (k = i + 1, however k = 0 if i = 4) as the trigger for the timer to start counting.
RAPI_EV_TIMER_ B2	Selects overflow or underflow of timer B2 as the trigger for the timer to start counting.
RAPI_TG_TAIS	Only writing 1 to the TA_iS bit of the TABSR register causes the timer to start counting.
RAPI_PULSE_ON	Selects that pulses are output from TA_{iIN} pin. Selectable only when timer Ai is used.
RAPI_PULSE_OFF	Selects that no pulses are output from TA_{iIN} pin. Selectable only when timer Ai is used.
RAPI_PWM_16	Selects operation as a 16-bit pulse width modulator.
RAPI_PWM_8	Selects operation as an 8-bit pulse width modulator.
RAPI_RISING	Selects the rising edge of TA_{iIN} pin input signal as active edge.

RAPI_FALLING	Selects the falling edge of TA_{iIN} pin input signal as active edge.

• Specifiable definition values when timer A is used (RAPI_TIMER_A0 to RAPI_TIMER_A4 specified)

(Count source)	Specify one from { RAPI_F1, RAPI_F2, RAPI_F32, RAPI_FC32 }. The default value is RAPI_F2.
(Operating states set)	Specify one from { RAPI_TIMER_ON, RAPI_TIMER_OFF }. The default value is RAPI_TIMER_OFF.
(Count start condition)	Specify one from { RAPI_TG_TAIS, RAPI_TG_TAIIN, RAPI_EV_TIMER_AJ, RAPI_EV_TIMER_AK, RAPI_EV_TIMER_B2 }. The default value is RAPI_TG_TAIIN.
(Pulse output facility)	Specify one from { RAPI_PULSE_ON, RAPI_PULSE_OFF }. The default value is RAPI_PULSE_OFF.
(Modulator)	Specify one from { RAPI_PWM_16, RAPI_PWM_8 }. The default value is RAPI_PWM_16.
(TA_{iIN} pin input)	Specify one from { RAPI_RISING, RAPI_FALLING }. The default value is RAPI_FALLING. The active edge of TA_{iIN} pin input can only be set when RAPI_TG_TAIIN is selected.

[data2]

Specify the interrupt priority level (0-7) to be set in the interrupt control register.

[data3]

Specify a pointer to the 16-bit variable in which the set value for the timer register is stored.

For 16-bit PWM, specify the value of 'n' in "high-level width n/f_j, period 65535/f_j" in 16 bits.

For 8-bit PWM, specify the values of 'n' and 'm' in "high-level width n (m + 1)/f, period 255 (m + 1)/f_j" in the 8 high-order bits and the 8 low-order bits, respectively.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (pulse width modulation mode (PWM mode))

Reference

__EnablePulseWidthModulationMode, __DestroyPulseWidthModulationMode

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_timer_m16c_62p.h"

void TimerIntFunc( void ){}
```

```
void func( void )
{
    unsigned int p_tim[] =  {0xAA, 0xBB, 0xCC};

    /* Set up timer A3 as pulse width modulation mode */

__CreatePulseWidthModulationMode( RAPI_TIMER_A3|RAPI_TIMER_ON|RAPI_F8,
                                  5, p_tim, TimerIntFunc);
}
```

__EnablePulseWidthModulationMode

Synopsis

<Control operation of pulse width modulation mode>

Boolean __EnablePulseWidthModulationMode(unsigned long data)

data	Setup data

Description

Controls operation of the timer that is set to specified pulse width modulation mode by starting or stopping it.

[data]

For data, the following definition values can be set. To set multiple definition values at the same time, use the symbol "|" to separate each specified value.

RAPI_TIMER_A0	Selects timer A channel 0.
RAPI_TIMER_A1	Selects timer A channel 1.
RAPI_TIMER_A2	Selects timer A channel 2.
RAPI_TIMER_A3	Selects timer A channel 3.
RAPI_TIMER_A4	Selects timer A channel 4.
RAPI_TIMER_ON	Sets the timer that is set to pulse width modulation mode to start operating.
RAPI_TIMER_OFF	Sets the timer that is set to pulse width modulation mode to stop operating.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (pulse width modulation mode (PWM mode))

Reference

__CreatePulseWidthModulationMode, __DestroyPulseWidthModulationMode

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
    /* Enable timer A2 as pulse width modulation mode */
    __EnablePulseWidthModulationMode( RAPI_TIMER_A2|RAPI_TIMER_ON );
}
```

__DestroyPulseWidthModulationMode

Synopsis

<Discard settings of pulse width modulation mode>

Boolean __DestroyPulseWidthModulationMode(unsigned long data)

data	Setup data

Description

Discards settings of the timer that is set to specified pulse width modulation mode.

[data]

For data, the following definition values can be set.

RAPI_TIMER_A0	Selects timer A channel 0.
RAPI_TIMER_A1	Selects timer A channel 1.
RAPI_TIMER_A2	Selects timer A channel 2.
RAPI_TIMER_A3	Selects timer A channel 3.
RAPI_TIMER_A4	Selects timer A channel 4.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (pulse width modulation mode (PWM mode))

Reference

__CreatePulseWidthModulationMode, __EnablePulseWidthModulationMode

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
    /* Destroy the setting of timer A1 as pulse width modulation mode */
    __DestroyPulseWidthModulationMode( RAPI_TIMER_A1 );
}
```

__CreatePulsePeriodMeasurementMode

Synopsis

<Set pulse period measurement mode>

Boolean __CreatePulsePeriodMeasurementMode(unsigned long data1, unsigned int data2, unsigned int data3, unsigned int data4, void* func)

data1	Setup data 1
data2	Setup data 2
data3	Setup data 3
data4	Setup data 4
func	Callback function pointer (Specify 0 if no callback functions are set.)

Description

Sets a specified timer to pulse period measurement mode.

[data1]

For data1, the following definition values can be set. To set multiple definition values at the same time, use the symbol "|" to separate each specified value.

RAPI_TIMER_B0	Uses timer B channel 0.
RAPI_TIMER_B1	Uses timer B channel 1.
RAPI_TIMER_B2	Uses timer B channel 2.
RAPI_TIMER_B3	Uses timer B channel 3.
RAPI_TIMER_B4	Uses timer B channel 4.
RAPI_TIMER_B5	Uses timer B channel 5.
RAPI_F1	Selects f_1 for the count source.
RAPI_F2	Selects f_2 for the count source.
RAPI_F8	Selects f_8 for the count source.
RAPI_F32	Selects f_{32} for the count source.
RAPI_FC32	Selects f_{C32} for the count source.
RAPI_TIMER_ON	Sets the timer to start operating in __CreatePulsePeriodMeasurementMode.
RAPI_TIMER_OFF	Sets the timer to stop operating in __CreatePulsePeriodMeasurementMode.
RAPI_RISING_RISING	Selects measurement of an interval from the rise to the next rise of a measurement pulse.
RAPI_FALLING_FALLING	Selects measurement of an interval from the fall to the next fall of a measurement pulse.

• Specifiable definition values when timer B is used (RAPI_TIMER_B0 to RAPI_TIMER_B5 specified)

(Count source) Specify one from { RAPI_F1, RAPI_F2, RAPI_F8, RAPI_F32, RAPI_FC32 }. The default value is RAPI_F2.

(Operating states set) Specify one from { RAPI_TIMER_ON, RAPI_TIMER_OFF }. The default value is RAPI_TIMER_OFF.

(Measurement pulse) Specify one from { RAPI_RISING_RISING, RAPI_FALLING_FALLING }. The default value is RAPI_FALLING_FALLING.

[data2]

Specify the interrupt priority level (0-7) to be set in the interrupt control register.

[data3]

Specify 0.

[data4]

Specify 0.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (pulse period measurement mode)

Reference

__EnablePulsePeriodMeasurementMode, __DestroyPulsePeriodMeasurementMode, __GetPulsePeriodMeasurementMode

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_timer_m16c_62p.h"

void TimerIntFunc( void ){}

void func( void )
{
    /* Set up timer B0 as pulse period measurement mode */
    __CreatePulsePeriodMeasurementMode(
        RAPI_TIMER_B0|RAPI_TIMER_ON|RAPI_FALLING_FALLING|RAPI_F8,
        5, 0, 0, TimerIntFunc);
}
```

__EnablePulsePeriodMeasurementMode

Synopsis

<Control operation of pulse period measurement mode>

Boolean __EnablePulsePeriodMeasurementMode(unsigned long data)

data	Setup data

Description

Controls operation of the timer that is set to specified pulse period measurement mode by starting or stopping it.

[data]

For data, the following definition values can be set. To set multiple definition values at the same time, use the symbol "|" to separate each specified value.

RAPI_TIMER_B0	Selects timer B channel 0.
RAPI_TIMER_B1	Selects timer B channel 1.
RAPI_TIMER_B2	Selects timer B channel 2.
RAPI_TIMER_B3	Selects timer B channel 3.
RAPI_TIMER_B4	Selects timer B channel 4.
RAPI_TIMER_B5	Selects timer B channel 5.
RAPI_TIMER_ON	Sets the timer that is set to pulse period measurement mode to start operating.
RAPI_TIMER_OFF	Sets the timer that is set to pulse period measurement mode to stop operating.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (pulse period measurement mode)

Reference

__CreatePulsePeriodMeasurementMode, __DestroyPulsePeriodMeasurementMode, __GetPulsePeriodMeasurementMode

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
    /* Enable timer B1 as pulse period measurement mode */
    __EnablePulsePeriodMeasurementMode( RAPI_TIMER_B1|RAPI_TIMER_ON );
}
```

__DestroyPulsePeriodMeasurementMode

Synopsis

<Discard settings of pulse period measurement mode>

Boolean __DestroyPulsePeriodMeasurementMode(unsigned long data)

data	Setup data

Description

Discards settings of the timer that is set to specified pulse period measurement mode.

[data]

For data, the following definition values can be set.

RAPI_TIMER_B0	Uses timer B channel 0.
RAPI_TIMER_B1	Uses timer B channel 1.
RAPI_TIMER_B2	Uses timer B channel 2.
RAPI_TIMER_B3	Uses timer B channel 3.
RAPI_TIMER_B4	Uses timer B channel 4.
RAPI_TIMER_B5	Uses timer B channel 5.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (pulse period measurement mode)

Reference

__CreatePulsePeriodMeasurementMode, __EnablePulsePeriodMeasurementMode, __GetPulsePeriodMeasurementMode

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
    /* Destroy the setting of timer B2 as pulse period measurement mode */
    __DestroyPulsePeriodMeasurementMode( RAPI_TIMER_B2 );
}
```

__GetPulsePeriodMeasurementMode

Synopsis

<Get measured value in pulse period measurement mode>

Boolean __GetPulsePeriodMeasurementMode(unsigned long data1, unsigned int *data2)

data1	Setup data 1
data2	Pointer to the buffer in which counter value is stored

Description

Gets the counter value of the timer that is set to specified pulse period measurement mode.

[data1]

For data, the following definition values can be set.

RAPI_TIMER_B0	Uses timer B channel 0.
RAPI_TIMER_B1	Uses timer B channel 1.
RAPI_TIMER_B2	Uses timer B channel 2.
RAPI_TIMER_B3	Uses timer B channel 3.
RAPI_TIMER_B4	Uses timer B channel 4.
RAPI_TIMER_B5	Uses timer B channel 5.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (pulse period measurement mode)

Reference

__CreatePulsePeriodMeasurementMode, __EnablePulsePeriodMeasurementMode, __DestroyPulsePeriodMeasurementMode

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
    unsigned int data[2];

    /* Get the measured value of timer B3 as pulse period measurement mode
*/
    __GetPulsePeriodMeasurementMode( RAPI_TIMER_B3, data );
}
```

__CreatePulseWidthMeasurementMode

Synopsis

<Set pulse width measurement mode>

Boolean __CreatePulseWidthMeasurementMode(unsigned long data1, unsigned int data2, unsigned int data3, unsigned int data4, void* func)

data1	Setup data 1
data2	Setup data 2
data3	Setup data 3
data4	Setup data 4
func	Callback function pointer (Specify 0 if no callback functions are set.)

Description

Sets a specified timer to pulse with measurement mode.

[data1]

For data1, the following definition values can be set. To set multiple definition values at the same time, use the symbol "|" to separate each specified value.

RAPI_TIMER_B0	Uses timer B channel 0.
RAPI_TIMER_B1	Uses timer B channel 1.
RAPI_TIMER_B2	Uses timer B channel 2.
RAPI_TIMER_B3	Uses timer B channel 3.
RAPI_TIMER_B4	Uses timer B channel 4.
RAPI_TIMER_B5	Uses timer B channel 5.
RAPI_F1	Selects f_1 for the count source.
RAPI_F2	Selects f_2 for the count source.
RAPI_F8	Selects f_8 for the count source.
RAPI_F32	Selects f_{32} for the count source.
RAPI_FC32	Selects f_{C32} for the count source.
RAPI_TIMER_ON	Sets the timer to start operating in __CreatePulseWidthMeasurementMode.
RAPI_TIMER_OFF	Sets the timer to stop operating in __CreatePulseWidthMeasurementMode.

• Specifiable definition values when timer B is used (RAPI_TIMER_B0 to RAPI_TIMER_B2 specified)

(Count source) Specify one from { RAPI_F1, RAPI_F2, RAPI_F8, RAPI_F32, RAPI_FC32 }. The default value is RAPI_F2.

(Operating states set) Specify one from { RAPI_TIMER_ON, RAPI_TIMER_OFF }. The default value is RAPI_TIMER_OFF.

[data2]

Specify the interrupt priority level (0-7) to be set in the interrupt control register.

[data3]

Specify 0.

[data4]

Specify 0.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (pulse width measurement mode)

Reference

__EnablePulseWidthMeasurementMode, __DestroyPulseWidthMeasurementMode, __GetPulseWidthMeasurementMode

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_timer_m16c_62p.h"

void TimerIntFunc( void ){}

void func( void )
{
    /* Set up timer B4 as pulse width measurement mode */
    __CreatePulseWidthMeasurementMode(
        RAPI_TIMER_B4|RAPI_TIMER_ON|RAPI_RISING_FALLING|RAPI_F8,
        5, 0, 0, TimerIntFunc);
}
```

__EnablePulseWidthMeasurementMode

Synopsis

<Control operation of pulse width measurement mode>

Boolean __EnablePulseWidthMeasurementMode(unsigned long data)

data	Setup data

Description

Controls operation of the timer that is set to specified pulse width measurement mode.

[data]

For data, the following definition values can be set. To set multiple definition values at the same time, use the symbol "|" to separate each specified value.

RAPI_TIMER_B0	Selects timer B channel 0.
RAPI_TIMER_B1	Selects timer B channel 1.
RAPI_TIMER_B2	Selects timer B channel 2.
RAPI_TIMER_B3	Selects timer B channel 3.
RAPI_TIMER_B4	Selects timer B channel 4.
RAPI_TIMER_B5	Selects timer B channel 5.
RAPI_TIMER_ON	Sets the timer that is set to pulse width measurement mode to start operating.
RAPI_TIMER_OFF	Sets the timer that is set to pulse width measurement mode to stop operating.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (pulse width measurement mode)

Reference

__CreatePulseWidthMeasurementMode, __DestroyPulseWidthMeasurementMode, __GetPulseWidthMeasurementMode

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
   /* Disable timer B5 as pulse width measurement mode */
   __EnablePulseWidthMeasurementMode( RAPI_TIMER_B5|RAPI_TIMER_OFF );
}
```

__DestroyPulseWidthMeasurementMode

Synopsis

<Discard settings of pulse width measurement mode>

Boolean __DestroyPulseWidthMeasurementMode(unsigned long data)

data	Setup data

Description

Discards settings of the timer that is set to specified pulse width measurement mode.

[data]

For data, the following definition values can be set.

RAPI_TIMER_B0	Uses timer B channel 0.
RAPI_TIMER_B1	Uses timer B channel 1.
RAPI_TIMER_B2	Uses timer B channel 2.
RAPI_TIMER_B3	Uses timer B channel 3.
RAPI_TIMER_B4	Uses timer B channel 4.
RAPI_TIMER_B5	Uses timer B channel 5.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (pulse width measurement mode)

Reference

__CreatePulseWidthMeasurementMode, __EnablePulseWidthMeasurementMode, __GetPulseWidthMeasurementMode

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
    /* Destroy the setting of timer B0 as pulse width measurement mode */
    __DestroyPulseWidthMeasurementMode( RAPI_TIMER_B0 );
}
```

__GetPulseWidthMeasurementMode

Synopsis

<Get measured value in pulse width measurement mode>

Boolean __GetPulseWidthMeasurementMode(unsigned long data1, unsigned int *data2)

data1	Setup data 1
data2	Pointer to the buffer in which counter value is stored

Description

Gets the counter value of the timer that is set to specified pulse width measurement mode.

[data1]

For data, the following definition values can be set.

RAPI_TIMER_B0	Uses timer B channel 0.
RAPI_TIMER_B1	Uses timer B channel 1.
RAPI_TIMER_B2	Uses timer B channel 2.
RAPI_TIMER_B3	Uses timer B channel 3.
RAPI_TIMER_B4	Uses timer B channel 4.
RAPI_TIMER_B5	Uses timer B channel 5.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (pulse width measurement mode)

Reference

__CreatePulseWidthMeasurementMode, __EnablePulseWidthMeasurementMode, __DestroyPulseWidthMeasurementMode

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_timer_m16c_62p.h"

void func( void )
{
    unsigned int data[2];

    /* Get the measured value of timer B1 as pulse width measurement mode
*/
    __GetPulseWidthMeasurementMode( RAPI_TIMER_B1, data );
}
```

__SetTimerRegister

Synopsis

<Set timer register>

Boolean __SetTimerRegister(unsigned long data1, unsigned int *data2)

data1	Setup data 1
data2	Pointer to the buffer in which register value is stored

Description

Sets the registers of a specified timer.

[data1]

For data, the following definition values can be set.

RAPI_TIMER_A0	Selects timer A channel 0.
RAPI_TIMER_A1	Selects timer A channel 1.
RAPI_TIMER_A2	Selects timer A channel 2.
RAPI_TIMER_A3	Selects timer A channel 3.
RAPI_TIMER_A4	Selects timer A channel 4.
RAPI_TIMER_B0	Selects timer B channel 0.
RAPI_TIMER_B1	Selects timer B channel 1.
RAPI_TIMER_B2	Selects timer B channel 2.

[data2]

The content of a pointer to the buffer in which the register value is stored must be specified as described below. The value is set in each register in order of buffer pointer elements.

- **When using timer A (RAPI_TIMER_A0 to RAPI_TIMER_A4 specified)**

[0]: Specify the set value for the timer Ai mode register (i = 0–4).
[1]: Specify the set value for the timer Ai register (i = 0–4).
[2]: Specify the set value for the up/down flag register.
[3]: Specify the set value for the one-shot start flag register.
[4]: Specify the set value for the trigger select register.
[5]: Specify the set value for the time-clock prescaler reset register.
[6]: Specify the set value for the count start flag register.

- **When using timer B (RAPI_TIMER_B0 to RAPI_TIMER_B5)**

[0]: Specify the set value for the timer Bi mode register (i = 0–5).
[1]: Specify the set value for the timer Bi register (i = 0–5).
[3]: Specify the set value for the time-clock prescaler reset register.
[4]: Specify the set value for the count start flag register.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (register manipulation)

Reference

__EnableTimerRegister, __ClearTimerRegister, __GetTimerRegister

Remark

•

If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

- The specifiable timers differ with each CPU used.

Program example

```
#include " rapi_timer_m16c_62p.h"

void func( void )
{
   unsigned char data[] =  {0,0,0,0,0,0,0};

   /* Set up timer A0 register */
   __SetTimerRegister( RAPI_TIMER_A0, data );
}
```

__EnableTimerRegister

Synopsis

<Control operation of timer register>

Boolean __EnableTimerRegister(unsigned long data)

data	Setup data

Description

Controls operation of a specified timer by starting or stopping it.

[data]

For data, the following definition values can be set. To set multiple definition values at the same time, use the symbol "|" to separate each specified value.

RAPI_TIMER_A0	Selects timer A channel 0.
RAPI_TIMER_A1	Selects timer A channel 1.
RAPI_TIMER_A2	Selects timer A channel 2.
RAPI_TIMER_A3	Selects timer A channel 3.
RAPI_TIMER_A4	Selects timer A channel 4.
RAPI_TIMER_B0	Selects timer B channel 0.
RAPI_TIMER_B1	Selects timer B channel 1.
RAPI_TIMER_B2	Selects timer B channel 2.
RAPI_TIMER_B3	Selects timer B channel 3.
RAPI_TIMER_B4	Selects timer B channel 4.
RAPI_TIMER_B5	Selects timer B channel 5.
RAPI_TIMER_ON	Sets the selected timer to start operating.
RAPI_TIMER_OFF	Sets the selected timer to stop operating.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (register manipulation)

Reference

__SetTimerRegister, __ClearTimerRegister, __GetTimerRegister

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include " rapi_timer_m16c_62p.h"

void func( void )
{
    /* Activate timer A1 */
    __EnableTimerRegister( RAPI_TIMER_A1|RAPI_TIMER_ON );
}
```

__ClearTimerRegister

Synopsis

<Clear timer register>

Boolean __ClearTimerRegister(unsigned long data)

data	Setup data

Description

Sets the timer register of a specified timer to its initial value after reset.

[data]

For data, the following definition values can be set.

RAPI_TIMER_A0	Selects timer A channel 0.
RAPI_TIMER_A1	Selects timer A channel 1.
RAPI_TIMER_A2	Selects timer A channel 2.
RAPI_TIMER_A3	Selects timer A channel 3.
RAPI_TIMER_A4	Selects timer A channel 4.
RAPI_TIMER_B0	Selects timer B channel 0.
RAPI_TIMER_B1	Selects timer B channel 1.
RAPI_TIMER_B2	Selects timer B channel 2.
RAPI_TIMER_B3	Selects timer B channel 3.
RAPI_TIMER_B4	Selects timer B channel 4.
RAPI_TIMER_B5	Selects timer B channel 5.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (register manipulation)

Reference

__SetTimerRegister, __EnableTimerRegister, __GetTimerRegister

Remark

- If an undefined value is specified in the argument, operation of the API cannot be guaranteed.

Program example

```
#include " rapi_timer_m16c_62p.h"

void func( void )
{
    /* Clear the setting of timer A2 */
    __ClearTimerRegister( RAPI_TIMER_A2 );
}
```

__GetTimerRegister

Synopsis

<Get timer register value>

Boolean __GetTimerRegister(unsigned long data1, unsigned int *data2)

data1	Setup data
data2	Pointer to the buffer in which register value is stored

Description

Gets the counter value of a specified timer.

[data]

For data, the following definition values can be set.

RAPI_TIMER_A0	Selects timer A channel 0.
RAPI_TIMER_A1	Selects timer A channel 1.
RAPI_TIMER_A2	Selects timer A channel 2.
RAPI_TIMER_A3	Selects timer A channel 3.
RAPI_TIMER_A4	Selects timer A channel 4.
RAPI_TIMER_B0	Selects timer B channel 0.
RAPI_TIMER_B1	Selects timer B channel 1.
RAPI_TIMER_B2	Selects timer B channel 2.
RAPI_TIMER_B3	Selects timer B channel 3.
RAPI_TIMER_B4	Selects timer B channel 4.
RAPI_TIMER_B5	Selects timer B channel 5.

[data2]

Specify a pointer to the array in which the acquired register value is stored.

The content of the array is described below.

- **When using timer A (RAPI_TIMER_A0 to RAPI_TIMER_A4 specified)**

[0]: Store the value of timer Ai mode register (i = 0–4).

[1]: Store the value of timer Ai register (i = 0–4).

[2]: Store the value of the up/down flag register.

[3]: Store the value of the one-shot start flag register.

[4]: Store the value of the trigger select register.

[5]: Store the value of the time-clock prescaler reset flag register.

[6]: Store the value of the count start flag register.

- **When using timer B (RAPI_TIMER_B0 to RAPI_TIMER_B5 specified)**

[0]: Store the value of timer Bi mode register (i = 0–5).

[1]: Store the value of timer Bi register (i = 0–5).

[2]: Store the value of the time-clock prescaler reset flag register.

[3]: Store the value of the count start flag register.

Return value

If the timer specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

Timer (register manipulation)

Reference

__SetTimerRegister, __EnableTimerRegister, __ClearTimerRegister

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include " rapi_timer_m16c_62p.h"

void func( void )
{
   unsigned int data[7];

   /* Get the value of timer A3 registers */
    __GetTimerRegister( RAPI_TIMER_A3, data );
}
```

4.2.3 I/O Port __SetIOPort

Synopsis

<Set I/O port>

Boolean __SetIOPort(unsigned long data1, unsigned int data2)

data1	Setup data 1
data2	Setup data 2

Description

Sets the operating conditions of a specified I/O port.

[data1]

For data1, the following definition values can be set. To set multiple definition values at the same time, use the symbol "|" to separate each specified value. Note, however, that multiple ports cannot be specified at the same time.

The definition values corresponding to each I/O port are listed below.

RAPI_PORT_0_0	Port $P0_0$	RAPI_PORT_0_1	Port $P0_1$
RAPI_PORT_0_2	Port $P0_2$	RAPI_PORT_0_3	Port $P0_3$
RAPI_PORT_0_4	Port $P0_4$	RAPI_PORT_0_5	Port $P0_5$
RAPI_PORT_0_6	Port $P0_6$	RAPI_PORT_0_7	Port $P0_7$
RAPI_PORT_1_0	Port $P1_0$	RAPI_PORT_1_1	Port $P1_1$
RAPI_PORT_1_2	Port $P1_2$	RAPI_PORT_1_3	Port $P1_3$
RAPI_PORT_1_4	Port $P1_4$	RAPI_PORT_1_5	Port $P1_5$
RAPI_PORT_1_6	Port $P1_6$	RAPI_PORT_1_7	Port $P1_7$
RAPI_PORT_2_0	Port $P2_0$	RAPI_PORT_2_1	Port $P2_1$
RAPI_PORT_2_2	Port $P2_2$	RAPI_PORT_2_3	Port $P2_3$
RAPI_PORT_2_4	Port $P2_4$	RAPI_PORT_2_5	Port $P2_5$
RAPI_PORT_2_6	Port $P2_6$	RAPI_PORT_2_7	Port $P2_7$
RAPI_PORT_3_0	Port $P3_0$	RAPI_PORT_3_1	Port $P3_1$
RAPI_PORT_3_2	Port $P3_2$	RAPI_PORT_3_3	Port $P3_3$
RAPI_PORT_3_4	Port $P3_4$	RAPI_PORT_3_5	Port $P3_5$
RAPI_PORT_3_6	Port $P3_6$	RAPI_PORT_3_7	Port $P3_7$
RAPI_PORT_4_0	Port $P4_0$	RAPI_PORT_4_1	Port $P4_1$
RAPI_PORT_4_2	Port $P4_2$	RAPI_PORT_4_3	Port $P4_3$
RAPI_PORT_4_4	Port $P4_4$	RAPI_PORT_4_5	Port $P4_5$
RAPI_PORT_4_6	Port $P4_6$	RAPI_PORT_4_7	Port $P4_7$
RAPI_PORT_5_0	Port $P5_0$	RAPI_PORT_5_1	Port $P5_1$
RAPI_PORT_5_2	Port $P5_2$	RAPI_PORT_5_3	Port $P5_3$
RAPI_PORT_5_4	Port $P5_4$	RAPI_PORT_5_5	Port $P5_5$
RAPI_PORT_5_6	Port $P5_6$	RAPI_PORT_5_7	Port $P5_7$
RAPI_PORT_6_0	Port $P6_0$	RAPI_PORT_6_1	Port $P6_1$
RAPI_PORT_6_2	Port $P6_2$	RAPI_PORT_6_3	Port $P6_3$
RAPI_PORT_6_4	Port $P6_4$	RAPI_PORT_6_5	Port $P6_5$
RAPI_PORT_6_6	Port $P6_6$	RAPI_PORT_6_7	Port $P6_7$
RAPI_PORT_7_0	Port $P7_0$	RAPI_PORT_7_1	Port $P7_1$

RAPI_PORT_7_2	Port $P7_2$	RAPI_PORT_7_3	Port $P7_3$
RAPI_PORT_7_4	Port $P7_4$	RAPI_PORT_7_5	Port $P7_5$
RAPI_PORT_7_6	Port $P7_6$	RAPI_PORT_7_7	Port $P7_7$
RAPI_PORT_8_0	Port $P8_0$	RAPI_PORT_8_1	Port $P8_1$
RAPI_PORT_8_2	Port $P8_2$	RAPI_PORT_8_3	Port $P8_3$
RAPI_PORT_8_4	Port $P8_4$	RAPI_PORT_8_5	Port $P8_5$
RAPI_PORT_8_6	Port $P8_6$	RAPI_PORT_8_7	Port $P8_7$
RAPI_PORT_9_0	Port $P9_0$	RAPI_PORT_9_1	Port $P9_1$
RAPI_PORT_9_2	Port $P9_2$	RAPI_PORT_9_3	Port $P9_3$
RAPI_PORT_9_4	Port $P9_4$	RAPI_PORT_9_5	Port $P9_5$
RAPI_PORT_9_6	Port $P9_6$	RAPI_PORT_9_7	Port $P9_7$
RAPI_PORT_10_0	Port $P10_0$	RAPI_PORT_10_1	Port $P10_1$
RAPI_PORT_10_2	Port $P10_2$	RAPI_PORT_10_3	Port $P10_3$
RAPI_PORT_10_4	Port $P10_4$	RAPI_PORT_10_5	Port $P10_5$
RAPI_PORT_10_6	Port $P10_6$	RAPI_PORT_10_7	Port $P10_7$
RAPI_PORT_11_0	Port $P11_0$	RAPI_PORT_11_1	Port $P11_1$
RAPI_PORT_11_2	Port $P11_2$	RAPI_PORT_11_3	Port $P11_3$
RAPI_PORT_11_4	Port $P11_4$	RAPI_PORT_11_5	Port $P11_5$
RAPI_PORT_11_6	Port $P11_6$	RAPI_PORT_11_7	Port $P11_7$
RAPI_PORT_12_0	Port $P12_0$	RAPI_PORT_12_1	Port $P12_1$
RAPI_PORT_12_2	Port $P12_2$	RAPI_PORT_12_3	Port $P12_3$
RAPI_PORT_12_4	Port $P12_4$	RAPI_PORT_12_5	Port $P12_5$
RAPI_PORT_12_6	Port $P12_6$	RAPI_PORT_12_7	Port $P12_7$
RAPI_PORT_13_0	Port $P13_0$	RAPI_PORT_13_1	Port $P13_1$
RAPI_PORT_13_2	Port $P13_2$	RAPI_PORT_13_3	Port $P13_3$
RAPI_PORT_13_4	Port $P13_4$	RAPI_PORT_13_5	Port $P13_5$
RAPI_PORT_13_6	Port $P13_6$	RAPI_PORT_13_7	Port $P13_7$
RAPI_PORT_14_0	Port $P14_0$	RAPI_PORT_14_1	Port $P14_1$

The definition values related to port settings are described below.

RAPI_PORT_INPUT	Sets a selected port for input.
RAPI_PORT_OUTPUT	Sets a selected port for output.
RAPI_PULLED_HIGH	Sets a selected port to be pulled high.
RAPI_NOT_PULLED _HIGH	Sets a selected port not to be pulled high.
RAPI_LATCH	Sets a selected port to read the port latch regardless of whether it is set for input or output. Specifiable only when port P1 is used.

[data2]

Specify 0.

Return value

If the I/O port specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality I/O port

Reference __ReadIOPort, __WriteIOPort, __SetIOPortRegister, __ReadIOPortRegister, __WriteIOPortRegister

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include " rapi_io_port_m16c_62p.h"

void func( void )
{
   /* Set up port P03 as input port */
   __SetIOPort(RAPI_PORT_0_3| RAPI_PORT_INPUT| RAPI_PULLED_HIGH, 0 );
}
```

__ReadIOPort

Synopsis

<Read from I/O port>

Boolean __ReadIOPort(unsigned long data1, unsigned int *data2)

data1	Setup data 1
data2	Pointer to the variable in which the value read from I/O port is stored.

Description

Gets the value of a specified I/O port.

[data1]

Specify an I/O port from which data is read. The definition values corresponding to each I/O port are listed below.

RAPI_PORT_0_0	Port $P0_0$	RAPI_PORT_0_1	Port $P0_1$
RAPI_PORT_0_2	Port $P0_2$	RAPI_PORT_0_3	Port $P0_3$
RAPI_PORT_0_4	Port $P0_4$	RAPI_PORT_0_5	Port $P0_5$
RAPI_PORT_0_6	Port $P0_6$	RAPI_PORT_0_7	Port $P0_7$
RAPI_PORT_1_0	Port $P1_0$	RAPI_PORT_1_1	Port $P1_1$
RAPI_PORT_1_2	Port $P1_2$	RAPI_PORT_1_3	Port $P1_3$
RAPI_PORT_1_4	Port $P1_4$	RAPI_PORT_1_5	Port $P1_5$
RAPI_PORT_1_6	Port $P1_6$	RAPI_PORT_1_7	Port $P1_7$
RAPI_PORT_2_0	Port $P2_0$	RAPI_PORT_2_1	Port $P2_1$
RAPI_PORT_2_2	Port $P2_2$	RAPI_PORT_2_3	Port $P2_3$
RAPI_PORT_2_4	Port $P2_4$	RAPI_PORT_2_5	Port $P2_5$
RAPI_PORT_2_6	Port $P2_6$	RAPI_PORT_2_7	Port $P2_7$
RAPI_PORT_3_0	Port $P3_0$	RAPI_PORT_3_1	Port $P3_1$
RAPI_PORT_3_2	Port $P3_2$	RAPI_PORT_3_3	Port $P3_3$
RAPI_PORT_3_4	Port $P3_4$	RAPI_PORT_3_5	Port $P3_5$
RAPI_PORT_3_6	Port $P3_6$	RAPI_PORT_3_7	Port $P3_7$
RAPI_PORT_6_0	Port $P6_0$	RAPI_PORT_6_1	Port $P6_1$
RAPI_PORT_6_2	Port $P6_2$	RAPI_PORT_6_3	Port $P6_3$
RAPI_PORT_6_4	Port $P6_4$	RAPI_PORT_6_5	Port $P6_5$
RAPI_PORT_6_6	Port $P6_6$	RAPI_PORT_6_7	Port $P6_7$
RAPI_PORT_7_0	Port $P7_0$	RAPI_PORT_7_1	Port $P7_1$
RAPI_PORT_7_2	Port $P7_2$	RAPI_PORT_7_3	Port $P7_3$
RAPI_PORT_7_4	Port $P7_4$	RAPI_PORT_7_5	Port $P7_5$
RAPI_PORT_7_6	Port $P7_6$	RAPI_PORT_7_7	Port $P7_7$
RAPI_PORT_8_0	Port $P8_0$	RAPI_PORT_8_1	Port $P8_1$
RAPI_PORT_8_2	Port $P8_2$	RAPI_PORT_8_3	Port $P8_3$
RAPI_PORT_8_4	Port $P8_4$	RAPI_PORT_8_5	Port $P8_5$
RAPI_PORT_8_6	Port $P8_6$	RAPI_PORT_8_7	Port $P8_7$
RAPI_PORT_9_0	Port $P9_0$	RAPI_PORT_9_1	Port $P9_1$
RAPI_PORT_9_2	Port $P9_2$	RAPI_PORT_9_3	Port $P9_3$
RAPI_PORT_9_4	Port $P9_4$	RAPI_PORT_9_5	Port $P9_5$

RAPI_PORT_9_6	Port $P9_6$	RAPI_PORT_9_7	Port $P9_7$
RAPI_PORT_10_0	Port $P10_0$	RAPI_PORT_10_1	Port $P10_1$
RAPI_PORT_10_2	Port $P10_2$	RAPI_PORT_10_3	Port $P10_3$
RAPI_PORT_10_4	Port $P10_4$	RAPI_PORT_10_5	Port $P10_5$
RAPI_PORT_10_6	Port $P10_6$	RAPI_PORT_10_7	Port $P10_7$
RAPI_PORT_11_0	Port $P11_0$	RAPI_PORT_11_1	Port $P11_1$
RAPI_PORT_11_2	Port $P11_2$	RAPI_PORT_11_3	Port $P11_3$
RAPI_PORT_11_4	Port $P11_4$	RAPI_PORT_11_5	Port $P11_5$
RAPI_PORT_11_6	Port $P11_6$	RAPI_PORT_11_7	Port $P11_7$
RAPI_PORT_12_0	Port $P12_0$	RAPI_PORT_12_1	Port $P12_1$
RAPI_PORT_12_2	Port $P12_2$	RAPI_PORT_12_3	Port $P12_3$
RAPI_PORT_12_4	Port $P12_4$	RAPI_PORT_12_5	Port $P12_5$
RAPI_PORT_12_6	Port $P12_6$	RAPI_PORT_12_7	Port $P12_7$
RAPI_PORT_13_0	Port $P13_0$	RAPI_PORT_13_1	Port $P13_1$
RAPI_PORT_13_2	Port $P13_2$	RAPI_PORT_13_3	Port $P13_3$
RAPI_PORT_13_4	Port $P13_4$	RAPI_PORT_13_5	Port $P13_5$
RAPI_PORT_13_6	Port $P13_6$	RAPI_PORT_13_7	Port $P13_7$
RAPI_PORT_14_0	Port $P14_0$	RAPI_PORT_14_1	Port $P14_1$

Return value

If the I/O port specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

I/O port

Reference

__SetIOPort, __WriteIOPort, __SetIOPortRegister, __ReadIOPortRegister, __WriteIOPortRegister

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include " rapi_io_port_m16c_62p.h"

void func( void )
{
   /* Get the value of port P12 */
   __ReadIOPort(RAPI_PORT_1_2, &data );
}
```

__WriteIOPort

Synopsis

<Write to I/O port>

Boolean __WriteIOPort(unsigned long data1, unsigned int data2)

data1	Setup data 1
data2	Data to be written to I/O port

Description

Writes data to a specified I/O port.

[data1]

Specify an I/O port to which data is written. The definition values corresponding to each I/O port are listed below.

RAPI_PORT_0_0	Port $P0_0$	RAPI_PORT_0_1	Port $P0_1$
RAPI_PORT_0_2	Port $P0_2$	RAPI_PORT_0_3	Port $P0_3$
RAPI_PORT_0_4	Port $P0_4$	RAPI_PORT_0_5	Port $P0_5$
RAPI_PORT_0_6	Port $P0_6$	RAPI_PORT_0_7	Port $P0_7$
RAPI_PORT_1_0	Port $P1_0$	RAPI_PORT_1_1	Port $P1_1$
RAPI_PORT_1_2	Port $P1_2$	RAPI_PORT_1_3	Port $P1_3$
RAPI_PORT_1_4	Port $P1_4$	RAPI_PORT_1_5	Port $P1_5$
RAPI_PORT_1_6	Port $P1_6$	RAPI_PORT_1_7	Port $P1_7$
RAPI_PORT_2_0	Port $P2_0$	RAPI_PORT_2_1	Port $P2_1$
RAPI_PORT_2_2	Port $P2_2$	RAPI_PORT_2_3	Port $P2_3$
RAPI_PORT_2_4	Port $P2_4$	RAPI_PORT_2_5	Port $P2_5$
RAPI_PORT_2_6	Port $P2_6$	RAPI_PORT_2_7	Port $P2_7$
RAPI_PORT_3_0	Port $P3_0$	RAPI_PORT_3_1	Port $P3_1$
RAPI_PORT_3_2	Port $P3_2$	RAPI_PORT_3_3	Port $P3_3$
RAPI_PORT_3_4	Port $P3_4$	RAPI_PORT_3_5	Port $P3_5$
RAPI_PORT_3_6	Port $P3_6$	RAPI_PORT_3_7	Port $P3_7$
RAPI_PORT_6_0	Port $P6_0$	RAPI_PORT_6_1	Port $P6_1$
RAPI_PORT_6_2	Port $P6_2$	RAPI_PORT_6_3	Port $P6_3$
RAPI_PORT_6_4	Port $P6_4$	RAPI_PORT_6_5	Port $P6_5$
RAPI_PORT_6_6	Port $P6_6$	RAPI_PORT_6_7	Port $P6_7$
RAPI_PORT_7_0	Port $P7_0$	RAPI_PORT_7_1	Port $P7_1$
RAPI_PORT_7_2	Port $P7_2$	RAPI_PORT_7_3	Port $P7_3$
RAPI_PORT_7_4	Port $P7_4$	RAPI_PORT_7_5	Port $P7_5$
RAPI_PORT_7_6	Port $P7_6$	RAPI_PORT_7_7	Port $P7_7$
RAPI_PORT_8_0	Port $P8_0$	RAPI_PORT_8_1	Port $P8_1$
RAPI_PORT_8_2	Port $P8_2$	RAPI_PORT_8_3	Port $P8_3$
RAPI_PORT_8_4	Port $P8_4$	RAPI_PORT_8_5	Port $P8_5$
RAPI_PORT_8_6	Port $P8_6$	RAPI_PORT_8_7	Port $P8_7$
RAPI_PORT_9_0	Port $P9_0$	RAPI_PORT_9_1	Port $P9_1$
RAPI_PORT_9_2	Port $P9_2$	RAPI_PORT_9_3	Port $P9_3$
RAPI_PORT_9_4	Port $P9_4$	RAPI_PORT_9_5	Port $P9_5$

RAPI_PORT_9_6	Port $P9_6$	RAPI_PORT_9_7	Port $P9_7$
RAPI_PORT_10_0	Port $P10_0$	RAPI_PORT_10_1	Port $P10_1$
RAPI_PORT_10_2	Port $P10_2$	RAPI_PORT_10_3	Port $P10_3$
RAPI_PORT_10_4	Port $P10_4$	RAPI_PORT_10_5	Port $P10_5$
RAPI_PORT_10_6	Port $P10_6$	RAPI_PORT_10_7	Port $P10_7$
RAPI_PORT_11_0	Port $P11_0$	RAPI_PORT_11_1	Port $P11_1$
RAPI_PORT_11_2	Port $P11_2$	RAPI_PORT_11_3	Port $P11_3$
RAPI_PORT_11_4	Port $P11_4$	RAPI_PORT_11_5	Port $P11_5$
RAPI_PORT_11_6	Port $P11_6$	RAPI_PORT_11_7	Port $P11_7$
RAPI_PORT_12_0	Port $P12_0$	RAPI_PORT_12_1	Port $P12_1$
RAPI_PORT_12_2	Port $P12_2$	RAPI_PORT_12_3	Port $P12_3$
RAPI_PORT_12_4	Port $P12_4$	RAPI_PORT_12_5	Port $P12_5$
RAPI_PORT_12_6	Port $P12_6$	RAPI_PORT_12_7	Port $P12_7$
RAPI_PORT_13_0	Port $P13_0$	RAPI_PORT_13_1	Port $P13_1$
RAPI_PORT_13_2	Port $P13_2$	RAPI_PORT_13_3	Port $P13_3$
RAPI_PORT_13_4	Port $P13_4$	RAPI_PORT_13_5	Port $P13_5$
RAPI_PORT_13_6	Port $P13_6$	RAPI_PORT_13_7	Port $P13_7$
RAPI_PORT_14_0	Port $P14_0$	RAPI_PORT_14_1	Port $P14_1$

Return value

If the I/O port specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

I/O port

Reference

__SetIOPort, __ReadIOPort, __SetIOPortRegister, __ReadIOPortRegister, __WriteIOPortRegister

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include " rapi_io_port_m16c_62p.h"

void func( void )
{
    unsigned int data;

    /* Set the data to port P05 */
    __WriteIOPort( RAPI_PORT_0_5, 0 );
}
```

__SetIOPortRegister

Synopsis

<Set I/O port register>

Boolean __SetIOPortRegister(unsigned long data1, unsigned int data2, unsigned int data3, unsigned int data4)

data1	Setup data 1
data2	Setup data 2
data3	Setup data 3
data4	Setup data 4

Description

[data1]

Set the operating condition of a specified I/O port in each relevant register.

The definition values corresponding to each I/O port register are listed below.

RAPI_PORT_0	Port P0 register	RAPI_PORT_1	Port P1 register
RAPI_PORT_2	Port P2 register	RAPI_PORT_3	Port P3 register
RAPI_PORT_4	Port P4 register	RAPI_PORT_5	Port P5 register
RAPI_PORT_6	Port P6 register	RAPI_PORT_7	Port P7 register
RAPI_PORT_8	Port P8 register	RAPI_PORT_9	Port P9 register
RAPI_PORT_10	Port P10 register	RAPI_PORT_11	Port P11 register
RAPI_PORT_12	Port P12 register	RAPI_PORT_13	Port P13 register
RAPI_PORT_14	Port P14 register		

The definition values related to port settings are described below.

RAPI_LATCH	Set to read the port latch regardless of whether the port is set for input or output. Specifiable only when port P1 is used.

[data2]

Specify the set value for the port direction register corresponding to a selected port.

[data3]

Specify the set value for the pullup control register corresponding to a selected port.

[data4]

Specify 0.

Return value

If the I/O port register specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

I/O port

Reference

__SetIOPort, __ReadIOPort, __WriteIOPort, __ReadIOPortRegister, __WriteIOPortRegister

Remark

•

If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include " rapi_io_port_m16c_62p.h"

void func( void )
{
    /* Set inputs/outputs of port P1 register */
    __SetIOPortRegister(RAPI_PORT_1, 0xAA, 0, 0 );
}
```

__ReadIOPortRegister

Synopsis

<Read from I/O port register>

Boolean __ReadIOPortRegister(unsigned long data1, unsigned int *data2)

data1	Setup data 1
data2	Pointer to the variable in which the value read from I/O port register is stored.

Description

Gets the value of a specified I/O port from each relevant register.

[data1]

Specify an I/O port register from which data is read. The definition values corresponding to each I/O port register are listed below.

RAPI_PORT_0	Port P0 register	RAPI_PORT_1	Port P1 register
RAPI_PORT_2	Port P2 register	RAPI_PORT_3	Port P3 register
RAPI_PORT_4	Port P4 register	RAPI_PORT_5	Port P5 register
RAPI_PORT_6	Port P6 register	RAPI_PORT_7	Port P7 register
RAPI_PORT_8	Port P8 register	RAPI_PORT_9	Port P9 register
RAPI_PORT_10	Port P10 register	RAPI_PORT_11	Port P11 register
RAPI_PORT_12	Port P12 register	RAPI_PORT_13	Port P13 register
RAPI_PORT_14	Port P14 register		

Return value

If the I/O port register specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

I/O port

Reference

__SetIOPort, __ReadIOPort, __WriteIOPort, __SetIOPortRegister, __WriteIOPortRegister

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include " rapi_io_port_m16c_62p.h"

void func( void )
{
    unsigned int data;

    /* Get the value of port P1 register */
    __ReadIOPortRegister( RAPI_PORT_1, &data );
}
```

__WriteIOPortRegister

Synopsis

<Write to I/O port register>

Boolean __WriteIOPortRegister(unsigned long data1, unsigned int data2)

data1	Setup data 1
data2	Data to be written to I/O port register

Description

Writes the value for a specified I/O port to each relevant register.

[data1]

Specify an I/O port register to which data is written. The definition values corresponding to each I/O port register are listed below.

RAPI_PORT_0	Port P0 register	RAPI_PORT_1	Port P1 register
RAPI_PORT_2	Port P2 register	RAPI_PORT_3	Port P3 register
RAPI_PORT_4	Port P4 register	RAPI_PORT_5	Port P5 register
RAPI_PORT_6	Port P6 register	RAPI_PORT_7	Port P7 register
RAPI_PORT_8	Port P8 register	RAPI_PORT_9	Port P9 register
RAPI_PORT_10	Port P10 register	RAPI_PORT_11	Port P11 register
RAPI_PORT_12	Port P12 register	RAPI_PORT_13	Port P13 register
RAPI_PORT_14	Port P14 register		

Return value

If the I/O port register specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

I/O port

Reference

__SetIOPort, __ReadIOPort, __WriteIOPort, __SetIOPortRegister, __ReadIOPortRegister

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include " rapi_io_port_m16c_62p.h"

void func( void )
{
    /* Set the data to port P2 register */
    __WriteIOPortRegister( RAPI_PORT_2, 0xFF );
}
```

4.2.4 External interrupt
__SetInterrupt

Synopsis

<Set external interrupt>

Boolean __SetInterrupt(unsigned long data1, unsigned int data2, void* func)

data1	Setup data 1
data2	Setup data 2
func	Callback function pointer (Specify 0 if no callback functions are set.)

Description

Sets a specified external interrupt.

[data1]

For data1, the following definition values can be set. To set multiple definition values at the same time, use the symbol "|" to separate each specified value. Note, however, that multiple external interrupts cannot be specified at the same time.

RAPI_INT0	Uses _INT0 interrupt.
RAPI_INT1	Uses _INT1 interrupt.
RAPI_INT2	Uses _INT2 interrupt.
RAPI_INT3	Uses _INT3 interrupt.
RAPI_INT4	Uses _INT4 interrupt.
RAPI_INT5	Uses _INT5 interrupt.
RAPI_KEY	Uses key input interrupt.
RAPI_INT_RISING	Specifies a rising edge for the active edge of a selected external interrupt.
RAPI_INT_FALLING	Specifies a falling edge for the active edge of a selected external interrupt.
RAPI_INT_BOTH	Specifies both edges for the active edge of a selected external interrupt.
RAPI_KI0_ENABLE	Uses _KI0 pin input.
RAPI_KI1_ENABLE	Uses _KI1 pin input.
RAPI_KI2_ENABLE	Uses _KI2 pin input.
RAPI_KI3_ENABLE	Uses _KI3 pin input.

• Specifiable definition values when _INT0-5 interrupts are used (RAPI_INT0 to RAPI_INT5 specified)

(Polarity) Specify one from { RAPI_INT_RISING, RAPI_INT_FALLING, RAPI_INT_BOTH }. The default value is RAPI_INT_FALLING.

• Specifiable definition values when key input interrupt is used (RAPI_KEY specified)

(Input pin) To use _KI0, _KI1, _KI2, or _KI3 pin input, specify RAPI_KI0_ENABLE, RAPI_KI1_ENABLE, RAPI_KI2_ENABLE, or RAPI_KI3_ENABLE, respectively. The default value is RAPI_INT_FALLING.

[data2]

Specify the interrupt priority level (0-7) to be set in the interrupt control register.

Return value

If the external interrupt specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

External interrupt

Reference

__EnableInterrupt, __GetInterruptFlag, __ClearInterruptFlag

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include " rapi_interrupt_m16c_62p.h"

void IntFunc( void ){}

void func( void )
{
   /* Set up _INT0 interrupt */
   __SetInterrupt( RAPI_INT0|RAPI_INT_FALLING, 0, IntFunc );
}
```

__EnableInterrupt

Synopsis

<Control external interrupt>

Boolean __EnableInterrupt(unsigned long data1, unsigned int data2)

data1	Setup data 1
data2	Setup data 2

Description

Changes the operating condition of a specified external interrupt.

[data1]

RAPI_INT0	Uses _INT0 interrupt.
RAPI_INT1	Uses _INT1 interrupt.
RAPI_INT2	Uses _INT2 interrupt.
RAPI_INT3	Uses _INT3 interrupt.
RAPI_INT4	Uses _INT4 interrupt.
RAPI_INT5	Uses _INT5 interrupt.
RAPI_KEY	Uses key input interrupt.

[data2]

Specify the interrupt priority level (0-7) to be set in the interrupt control register.

Return value

If the external interrupt specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

External interrupt

Reference

__SetInterrupt, __GetInterruptFlag, __ClearInterruptFlag

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include " rapi_interrupt_m16c_62p.h"

void func( void )
{
    /* Activate _INT1 interrupt ( interrupt priority level 5 ) */
    __EnableInterrupt( RAPI_INT1, 5 );
}
```

__GetInterruptFlag

Synopsis

<Get the status of external interrupt flag>

Boolean __GetInterruptFlag(unsigned long data1, unsigned int *data2))

data1	Setup data 1
data2	Pointer to the buffer in which the acquired flag data is stored

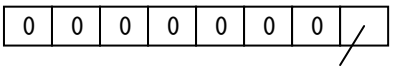
Description

Gets the value of interrupt request flag of a specified external interrupt.

[data1]

RAPI_INT0	Uses _INT0 interrupt.
RAPI_INT1	Uses _INT1 interrupt.
RAPI_INT2	Uses _INT2 interrupt.
RAPI_INT3	Uses _INT3 interrupt.
RAPI_INT4	Uses _INT4 interrupt.
RAPI_INT5	Uses _INT5 interrupt.
RAPI_KEY	Uses key input interrupt.

[data2]



Value of interrupt request flag (0: Interrupt not requested; 1: Interrupt requested)

Return value

If the external interrupt specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

External interrupt

Reference

__SetInterrupt, __EnableInterrupt, __ClearInterruptFlag

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include " rapi_interrupt_m16c_62p.h"

void func( void )
{
    unsigned char data;

    /* Get flag of _INT2 interrupt */
    __GetInterruptFlag( RAPI_INT2, &data );
}
```

__ClearInterruptFlag

Synopsis

<Clear external interrupt flag>

Boolean __ClearInterruptFlag(unsigned long data)

data	Setup data

Description

Clears the interrupt request flag of a specified external interrupt.

[data]

RAPI_INT0	Uses _INT0 interrupt.
RAPI_INT1	Uses _INT1 interrupt.
RAPI_INT2	Uses _INT2 interrupt.
RAPI_INT3	Uses _INT3 interrupt.
RAPI_INT4	Uses _INT4 interrupt.
RAPI_INT5	Uses _INT5 interrupt.
RAPI_KEY	Uses key input interrupt.

Return value

If the external interrupt specification is incorrect, RAPI_FALSE is returned; otherwise, RAPI_TRUE is returned.

Functionality

External interrupt

Reference

__SetInterrupt, __EnableInterrupt, __GetInterruptFlag

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include " rapi_interrupt_m16c_62p.h"

void func( void )
{
    /* Clear status of _INT0 interrupt */
    __ClearInterruptFlag( RAPI_INT0 );
}
```

4.2.5 A/D converter
__CreateADC

Synopsis

<Set A/D converter>

Boolean __CreateADC(unsigned long data1, unsigned int data2, unsigned int data3, void* func)

data1	Setup data 1
data2	Number of analog input pins used by A/D converter
data3	Setup data 3
func	Callback function pointer (Specify 0 if no callback functions are set.)

Description

Sets the A/D converter to specified mode and operating condition.

[data1]

For data1, the following definition values can be set. To set multiple definition values at the same time, use the symbol "|" to separate each specified value. Note, however, that multiple analog input pin symbols cannot be specified at the same time.

RAPI_ONE_SHOT	Selects one-shot mode.
RAPI_REPEAT	Selects repeat mode.
RAPI_SINGLE_SWEEP	Selects single sweep mode.
RAPI_REPEAT_SWEEP0	Selects repeat sweep mode 0.
RAPI_REPEAT_SWEEP1	Selects repeat sweep mode 1.
RAPI_AN0	Uses AN_0 pin for the analog input pin.
RAPI_AN1	Uses AN_1 pin for the analog input pin.
RAPI_AN2	Uses AN_2 pin for the analog input pin.
RAPI_AN3	Uses AN_3 pin for the analog input pin.
RAPI_AN4	Uses AN_4 pin for the analog input pin.
RAPI_AN5	Uses AN_5 pin for the analog input pin.
RAPI_AN6	Uses AN_6 pin for the analog input pin.
RAPI_AN7	Uses AN_7 pin for the analog input pin.
RAPI_AN00	Uses AN_{00} pin for the analog input pin.
RAPI_AN01	Uses AN_{01} pin for the analog input pin.
RAPI_AN02	Uses AN_{02} pin for the analog input pin.
RAPI_AN03	Uses AN_{03} pin for the analog input pin.
RAPI_AN04	Uses AN_{04} pin for the analog input pin.
RAPI_AN05	Uses AN_{05} pin for the analog input pin.
RAPI_AN06	Uses AN_{06} pin for the analog input pin.
RAPI_AN07	Uses AN_{07} pin for the analog input pin.
RAPI_AN20	Uses AN_{20} pin for the analog input pin.
RAPI_AN21	Uses AN_{21} pin for the analog input pin.
RAPI_AN22	Uses AN_{22} pin for the analog input pin.
RAPI_AN23	Uses AN_{23} pin for the analog input pin.
RAPI_AN24	Uses AN_{24} pin for the analog input pin.
RAPI_AN25	Uses AN_{25} pin for the analog input pin.
RAPI_AN26	Uses AN_{26} pin for the analog input pin.

RAPI_AN27	Uses AN_{27} pin for the analog input pin.
RAPI_P0_GROUP	Uses port P_0 group for the analog input pin.
RAPI_P10_GROUP	Uses port P_{10} group for the analog input pin.
RAPI_P2_GROUP	Uses port P_2 group for the analog input pin.
RAPI_FAD	Sets the AD converter's operating frequency to f_{AD}.
RAPI_FAD2	Sets the AD converter's operating frequency to f_{AD} divided by 2.
RAPI_FAD3	Sets the AD converter's operating frequency to f_{AD} divided by 3.
RAPI_FAD4	Sets the AD converter's operating frequency to f_{AD} divided by 4.
RAPI_FAD6	Sets the AD converter's operating frequency to f_{AD} divided by 6.
RAPI_FAD12	Sets the AD converter's operating frequency to f_{AD} divided by 12.
RAPI_10BIT	Sets the AD converter's resolution to 10 bits.
RAPI_8BIT	Sets the AD converter's resolution to 8 bits.
RAPI_AD_ON	Sets the AD converter to start operating.
RAPI_AD_OFF	Sets the AD converter to stop operating.
RAPI_WITH_SAMPLE_HOLD	Specifies that sample-and-hold action is applied.
RAPI_WITHOUT_SAMPLE_ HOLD	Specifies that sample-and-hold action is not applied.
RAPI_SOFTWARE_TRIGGER	Selects software trigger.
RAPI_EXTERNAL_TRIGGER	Selects external trigger.
RAPI_ANEX_UNUSED	ANEX0 and ANEX1 are not used
RAPI_ANEX0	ANEX0 input is A/D converted
RAPI_ANEX1	ANEX1 input is A/D converted
RAPI_EXTERNAL_OP_AMP	External op-amp connection mode

• Specifiable definition values when one-shot mode is used (RAPI_ONE_SHOT specified)

(Input pin) Specify one from { RAPI_AN0, RAPI_AN1, RAPI_AN2, RAPI_AN3, RAPI_AN4, RAPI_AN5, RAPI_AN6, RAPI_AN7, RAPI_AN00, RAPI_AN01, RAPI_AN02, RAPI_AN03, RAPI_AN04, RAPI_AN05, RAPI_AN06, RAPI_AN07, RAPI_AN20, RAPI_AN21, RAPI_AN22, RAPI_AN23, RAPI_AN24, RAPI_AN25, RAPI_AN26, RAPI_AN27, RAPI_ANEX0, RAPI_ANEX1 }. The default is RAPI_AN0.

(Operating frequency) Specify one from { RAPI_FAD, RAPI_FAD2, RAPI_FAD3, RAPI_FAD4, RAPI_FAD6, RAPI_FAD12 }. The default value is RAPI_FAD4.

(Resolution) Specify one from { RAPI_10BIT, RAPI_8BIT }. The default value is RAPI_8BIT.

(Operating states set) Specify one from { RAPI_AD_ON, RAPI_AD_OFF }. The default value is RAPI_AD_OFF.

(Conversion method) Specify one from { RAPI_WITH_SAMPLE_HOLD, RAPI_WITHOUT_SAMPLE_HOLD }. The default value is RAPI_WITHOUT_SAMPLE_HOLD.

(Trigger) Specify one from { RAPI_SOFTWARE_TRIGGER, RAPI_EXTERNAL_TRIGGER }.
The default value is RAPI_SOFTWARE_TRIGGER.

(External Op-Amp Connection) Specify one from { RAPI_ANEX_UNUSED, RAPI_EXTERNAL_OP_AMP }.
The default value is RAPI_ANEX_UNUSED.

• Specifiable definition values when repeat mode is used (RAPI_REPEAT specified)

(Input pin) Specify one from { RAPI_AN0, RAPI_AN1, RAPI_AN2, RAPI_AN3, RAPI_AN4, RAPI_AN5, RAPI_AN6, RAPI_AN7, RAPI_AN00, RAPI_AN01, RAPI_AN02, RAPI_AN03, RAPI_AN04, RAPI_AN05, RAPI_AN06, RAPI_AN07, RAPI_AN20, RAPI_AN21, RAPI_AN22, RAPI_AN23, RAPI_AN24, RAPI_AN25, RAPI_AN26, RAPI_AN27, RAPI_INEX0, RAPI_ANEX1 }. The default is RAPI_AN0.

(Operating frequency) Specify one from { RAPI_FAD, RAPI_FAD2, RAPI_FAD3, RAPI_FAD4, RAPI_FAD6, RAPI_FAD12 }. The default value is RAPI_FAD4.

(Resolution) Specify one from { RAPI_10BIT, RAPI_8BIT }. The default value is RAPI_8BIT.

(Operating states set) Specify one from { RAPI_AD_ON, RAPI_AD_OFF }. The default value is RAPI_AD_OFF.

(Conversion method) Specify one from { RAPI_WITH_SAMPLE_HOLD, RAPI_WITHOUT_SAMPLE_HOLD }.
The default value is RAPI_WITHOUT_SAMPLE_HOLD.

(Trigger) Specify one from { RAPI_SOFTWARE_TRIGGER, RAPI_EXTERNAL_TRIGGER }.
The default value is RAPI_SOFTWARE_TRIGGER.

(External Op-Amp Connection) Specify one from { RAPI_ANEX_UNUSED, RAPI_EXTERNAL_OP_AMP }.
The default value is RAPI_ANEX_UNUSED.

• Specifiable definition values when single sweep mode is used (RAPI_SINGLE_SWEEP specified)

(Input pin) Specify one from { RAPI_P0_GROUP, RAPI_P10_GROUP, RAPI_P2_GROUP }. The default value is RAPI_P10_GROUP.

(Operating frequency) Specify one from { RAPI_FAD, RAPI_FAD2, RAPI_FAD3, RAPI_FAD4, RAPI_FAD6, RAPI_FAD12 }. The default value is RAPI_FAD4.

(Resolution) Specify one from { RAPI_10BIT, RAPI_8BIT }. The default value is RAPI_8BIT.

(Operating states set) Specify one from { RAPI_AD_ON, RAPI_AD_OFF }. The default value is RAPI_AD_OFF.

(Conversion method) Specify one from { RAPI_WITH_SAMPLE_HOLD, RAPI_WITHOUT_SAMPLE_HOLD }.
The default value is RAPI_WITHOUT_SAMPLE_HOLD.

(Trigger) Specify one from { RAPI_SOFTWARE_TRIGGER, RAPI_EXTERNAL_TRIGGER }.
The default value is RAPI_SOFTWARE_TRIGGER.

(External Op-Amp Connection) Specify one from { RAPI_ANEX_UNUSED, RAPI_EXTERNAL_OP_AMP }.
The default value is RAPI_ANEX_UNUSED.

• Specifiable definition values when repeat sweep mode 0 is used (RAPI_REPEAT_SWEEP0 specified)

(Input pin) Specify one from { RAPI_P0_GROUP, RAPI_P10_GROUP, RAPI_P2_GROUP }. The default value is RAPI_P10_GROUP.

(Operating frequency) Specify one from { RAPI_FAD, RAPI_FAD2, RAPI_FAD3, RAPI_FAD4, RAPI_FAD6, RAPI_FAD12 }. The default value is RAPI_FAD4.

(Resolution) Specify one from { RAPI_10BIT, RAPI_8BIT }. The default value is RAPI_8BIT.

(Operating states set)	Specify one from { RAPI_AD_ON, RAPI_AD_OFF }. The default value is RAPI_AD_OFF.
(Conversion method)	Specify one from { RAPI_WITH_SAMPLE_HOLD, RAPI_WITHOUT_SAMPLE_HOLD }. The default value is RAPI_WITHOUT_SAMPLE_HOLD.
(Trigger)	Specify one from { RAPI_SOFTWARE_TRIGGER, RAPI_EXTERNAL_TRIGGER }. The default value is RAPI_SOFTWARE_TRIGGER.
(External Op-Amp Connection)	Specify one from { RAPI_ANEX_UNUSED, RAPI_EXTERNAL_OP_AMP }. The default value is RAPI_ANEX_UNUSED.

• Specifiable definition values when repeat sweep mode 1 is used (RAPI_REPEAT_SWEEP1 specified)

(Input pin)	Specify one from { RAPI_P0_GROUP, RAPI_P10_GROUP, RAPI_P2_GROUP }. The default value is RAPI_P10_GROUP.
(Operating frequency)	Specify one from { RAPI_FAD, RAPI_FAD2, RAPI_FAD3, RAPI_FAD4, RAPI_FAD6, RAPI_FAD12 }. The default value is RAPI_FAD4.
(Resolution)	Specify one from { RAPI_10BIT, RAPI_8BIT }. The default value is RAPI_8BIT.
(Operating states set)	Specify one from { RAPI_AD_ON, RAPI_AD_OFF }. The default value is RAPI_AD_OFF.
(Conversion method)	Specify one from { RAPI_WITH_SAMPLE_HOLD, RAPI_WITHOUT_SAMPLE_HOLD }. The default value is RAPI_WITHOUT_SAMPLE_HOLD.
(Trigger)	Specify one from { RAPI_SOFTWARE_TRIGGER, RAPI_EXTERNAL_TRIGGER }. The default value is RAPI_SOFTWARE_TRIGGER.
(External Op-Amp Connection)	Specify one from { RAPI_ANEX_UNUSED, RAPI_EXTERNAL_OP_AMP }. The default value is RAPI_ANEX_UNUSED.

[data2]

The set value differs with the A/D conversion mode used.

One-shot mode Repeat mode	Specify 1.
Single sweep mode Repeat sweep mode 0	Specify 2, 4, 6, or 8.
Repeat sweep mode1	Specify 1, 2, 3, or 4.

[data3]

Specify the interrupt priority level (0-7) to be set in the interrupt control register.

Return value

If A/D converter was successfully set, RAPI_TRUE is returned; if failed, RAPI_FALSE is returned.

Functionality

A/D converter

Reference

__EnableADC, __DestroyADC, __GetADC, __GetADCAll

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include " rapi_ad_m16c_62p.h"

void AdIntFunc( void ){}

void func( void )
{
   /* Set up A/D converter as one short mode */
   __CreateADC( RAPI_ONE_SHOT|RAPI_AN2|RAPI_FAD2| RAPI_WITH_SAMPLE_HOLD
       | RAPI_EXTERNAL_TRIGGER |RAPI_AD_ON|RAPI_10BIT| RAPI_ANEX_UNUSED,
       1, 5, AdIntFunc );
}
```

__EnableADC

Synopsis

<Control operation of A/D converter>

Boolean __EnableADC (unsigned long data1, unsigned int data2)

data1	Setup data 1
data2	Number of analog input pins used by A/D converter

Description

Controls operation of the A/D converter by starting or stopping it.

[data1]

RAPI_AN0	Uses AN_0 pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN1	Uses AN_1 pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN2	Uses AN_2 pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN3	Uses AN_3 pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN4	Uses AN_4 pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN5	Uses AN_5 pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN6	Uses AN_6 pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN7	Uses AN_7 pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN00	Uses AN_{00} pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN01	Uses AN_{01} pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN02	Uses AN_{02} pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN03	Uses AN_{03} pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN04	Uses AN_{04} pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN05	Uses AN_{05} pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN06	Uses AN_{06} pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN07	Uses AN_{07} pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN20	Uses AN_{20} pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.

RAPI_AN21	Uses AN_{21} pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN22	Uses AN_{22} pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN23	Uses AN_{23} pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN24	Uses AN_{24} pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN25	Uses AN_{25} pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN26	Uses AN_{26} pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_AN27	Uses AN_{27} pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_ANEX0	Uses ANEX0 pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_ANEX1	Uses ANEX1 pin for the analog input pin. Selectable only when one-shot mode or repeat mode is used.
RAPI_P0_GROUP	Uses port P_0 group for the analog input pin. Selectable only when sweep mode, repeat sweep mode 0, or repeat sweep mode 1 is used.
RAPI_P10_GROUP	Uses port P_{10} group for the analog input pin. Selectable only when sweep mode, repeat sweep mode 0, or repeat sweep mode 1 is used.
RAPI_P2_GROUP	Uses port P_2 group for the analog input pin. Selectable only when sweep mode, repeat sweep mode 0, or repeat sweep mode 1 is used.
RAPI_AD_ON	Sets the A/D converter to start operating.
RAPI_AD_OFF	Sets the A/D converter to stop operating.

[data2]

The set value differs with the A/D conversion mode used.

One-shot mode Repeat mode	Specify 1.
Single sweep mode Repeat sweep mode 0	Specify 2, 4, 6, or 8.
Repeat sweep mode 1	Specify 1, 2, 3, or 4.

Return value

If A/D converter was successfully controlled, RAPI_TRUE is returned; if failed, RAPI_FALSE is returned.

Functionality

A/D converter

Reference

CreateADC, DestroyADC, GetADC, GetADCAll

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_ad_m16c_62p.h"

void func( void )
{
    /* Disable A/D converter */
    __EnableADC( RAPI_AN0|RAPI_AD_OFF, 1 );
}
```

__DestroyADC

Synopsis

<Discard settings of A/D converter>

Boolean __DestroyADC(void)

Description

Discards settings of a specified A/D converter.

Return value

If converter setting was successfully discarded, RAPI_TRUE is returned; if failed, RAPI_FALSE is returned.

Functionality

A/D converter

Reference

__CreateADC, __EnableADC, __GetADC, __GetADCAll

Program example

```
#include "rapi_ad_ml6c_62p.h"

void func( void )
{
    /* Destroy A/D converter */
    __DestroyADC();
}
```

__GetADC

Synopsis

<Get A/D converted value (register specified)>

Boolean __GetADC(unsigned long data1, unsigned int *data2)

data1	Setup data 1
data2	Pointer to the buffer in which A/D converted value is stored.

Description

Gets the A/D converted value from a specified A/D register.

For data1, the following values can be set.

RAPI_AD0	Selects A/D register 0.
RAPI_AD1	Selects A/D register 1.
RAPI_AD2	Selects A/D register 2.
RAPI_AD3	Selects A/D register 3.
RAPI_AD4	Selects A/D register 4.
RAPI_AD5	Selects A/D register 5.
RAPI_AD6	Selects A/D register 6.
RAPI_AD7	Selects A/D register 7.

Return value

If A/D converted value was successfully acquired, RAPI_TRUE is returned; if failed, RAPI_FALSE is returned.

Functionality

A/D converter

Reference

__CreateADC, __EnableADC, __DestroyADC, __GetADCAll

Remark

- If an undefined value is specified in the first argument, operation of the API cannot be guaranteed.

Program example

```
#include "rapi_ad_m16c_62p.h"

void func( void )
{
   unsigned int data;

   /* Get an A/D convertered data of A/D register 0 */
   __GetADC( RAPI_AD0, &data );
}
```

__GetADCAll

Synopsis

<Get A/D converted value (all registers)>

Boolean __GetADCAll(unsigned int *data)

data	Pointer to the buffer in which A/D converted value is stored.

Description

Gets the A/D converted value from all A/D registers.

The A/D registers from which A/D converted values are acquired are listed below.

[M16C]	:	[0] A/D register 0
(16 bytes)		[1] A/D register 1
		[2] A/D register 2
		[3] A/D register 3
		[4] A/D register 4
		[5] A/D register 5
		[6] A/D register 6
		[7] A/D register 7

Return value

If A/D converted values were successfully acquired, RAPI_TRUE is returned; if failed, RAPI_FALSE is returned.

Functionality

A/D converter

Reference

__CreateADC, __EnableADC, __DestroyADC, __GetADC

Program example

```
#include "rapi_ad_m16c_62p.h"

void func( void )
{
    unsigned int data[8];

    /* Get A/D convertered datas of A/D register */
    __GetADCAll( data );
}
```

M16C/62P Group
Renesas Embedded
Application Programming Interface
Reference Manual

Rev. 1.00
Issued: November 2007