Datasheet

CC-RTC-APB Documentation

1.2

Scope

This document describes the CC-RTC-APB IP core. Module features and configuration registers are described. The document contains integration guide that covers synthesis options and instantiation example for easy implementation in customer's environment.

Contents

1. Real Time Counter

1.1 Functionality

- Configurable prescaler,
- configurable period,
- compare register,
- interrupt generation:
  - overflow,
  - compare match,
  - change on WKUP0,
- dedicated host wake-up signal on:
  - overflow (OVF),
  - compare match,
  - change on WKUP0,
- dedicated WKUP0 input to host wake-up,
- configurable SHDN output deactivated on:
  - overflow (OVF),
  - compare match,
  - change on WKUP0,
- four backup registers.

1.2 Overview

1.2.1 Real Time Counter

Real Time Counter counts clock cycles of dedicated slow clock input (Figure 1.1). Additionally, interrupts and wake-up event can be generated when the counted value reaches period value (PER, 1.7.5) or compare value (COMPARE, 1.7.6).

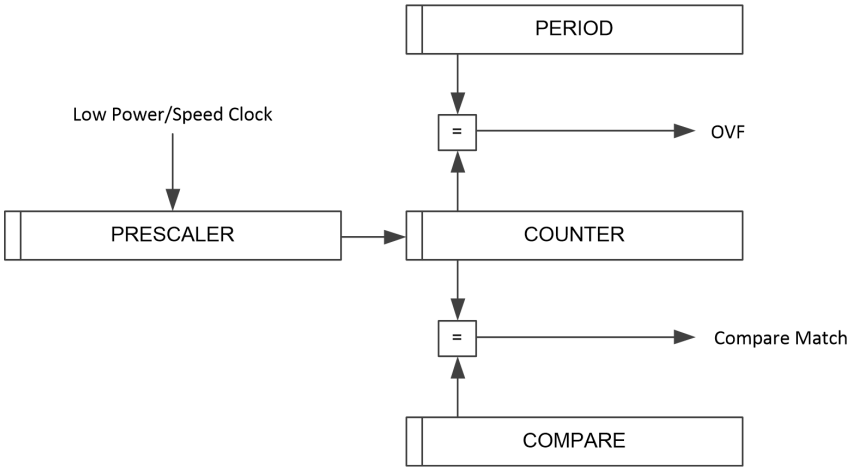


Figure 1.1. Diagram of real time counter.

1.2.2 Shutdown Controller

The shutdown controller (Figure 1.2) manages SHDN output pin. In typical application this pin is connected to the external DC/DC converter or low-drop regulator that supplies power to the host. The host application can program SHDN output to power off the host controller. Shutdown controller will automatically deactivate SHDN pin after configured events:

- counter overflow,
- compare match event,
- change on WKUP0 input.

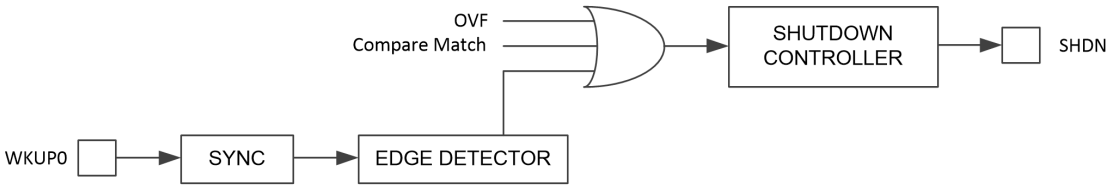


Figure 1.2. Shutdown controller diagram.

1.3 Block Diagram

The Real Time Counter is composed of two modules existing in two power domains: RTC power domain and host/system power domain (Figure 1.3). The low power domain module works in always-on power domain and its main purpose is to count slow clock cycles and to compare its value (COUNT, 1.7.7) against period (PER, 1.7.5) and compare register (COMPARE, 1.7.6). When counter reaches period value, counter is reset and counts from zero. Another part of low power domain module is shutdown controller and set of backup registers that will preserve stored values after system domain power failure.

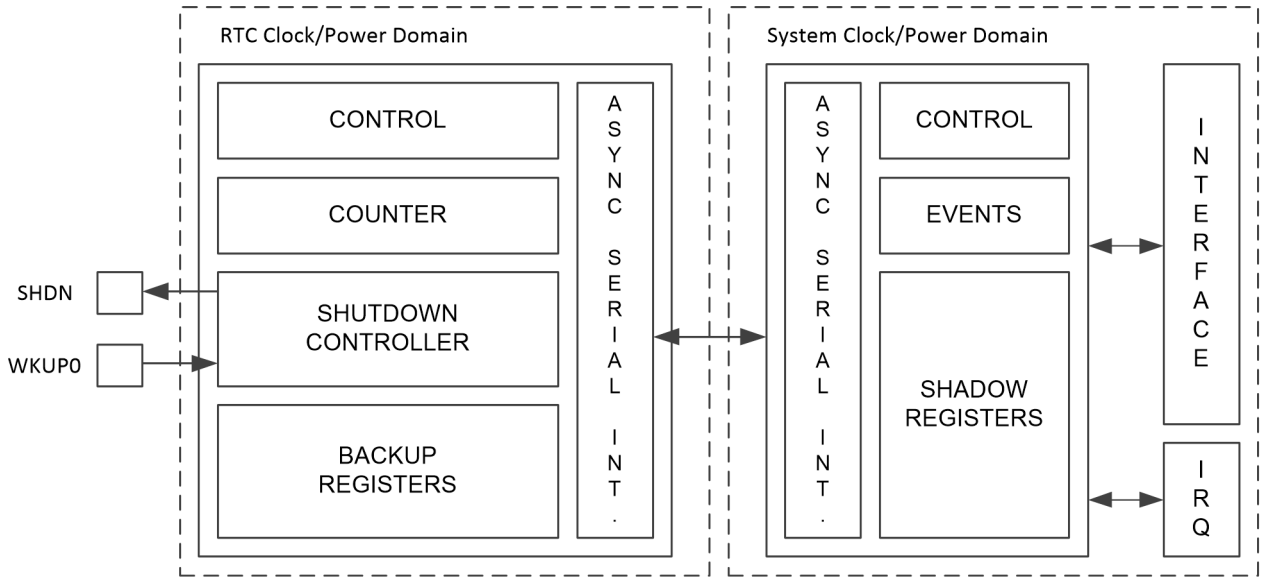


Figure 1.3. RTC block diagram.

1.4 Operation

1.4.1 Counting Mode

Low power domain module counts slow clock cycles. The counting scheme comprises of two levels. The first one is prescaler counter that directly counts slow clock cycles up to the prescaler register (PRES, 1.7.4) value and gets reset. The COUNT (1.7.7) register counts prescaler overflow events and is incremented after every $PRES + 1$ slow clock cycles. The RTC period equals $(PRES + 1) * (PER + 1)$ slow clock cycles. The counter value is continuously compared against period value. After reaching this value, the counter will generate overflow event, get reset and start counting from zero. Additionally, wake-up event can be generated to the host controller. When the counter value matches compare register, compare match event is generated and the counter continues to count. Compare match event can also be used to generate wake-up event to the host controller.

1.4.2 Shadow Registers

Some of the configuration registers are placed in the lower power domain. To improve access time to this registers, the shadow copies of their data are stored in system domain. After each write, stored values are firstly written in the system domain. Next, they are copied to the low power domain. Local copy is marked as valid so the read access can be performed immediately. The count register is the exception and is synchronized with the low power domain on every read access. The BUSY bit on status register (STATUS, 1.7.3) is set when the synchronization is in progress. Each write access during which BUSY bit is set will be discarded. Each read access during which BUSY bit is set will return local copy of the register (might be out-of-date). Write progress can be monitored in two ways:

- polling of BUSY bit in status register (STATUS, 1.7.3),
- Ready Interrupt generated when BUSY bit is cleared.

1.4.3 Clock Domain Crossing

The count register has a shadow copy located in the system domain, which is synchronized with the low power domain on every read access. To avoid metastability issues, count register is internally implemented as a gray counter. Additionally, internal logic detects possible non-gray counter transition when count register reaches PER value and will be cleared in the next slow clock cycle. In such situation zero value is forwarded in advance. To ensure proper detector work conditions, system clock should be at least five times faster than the slow clock. To mitigate this limitation, user should set PER register to the value that will allow gray code transition to zero (e.g. 0xFFFFFFFF, 0xFFFF, 0x3FF, etc.)

1.4.4 Backup Registers

Real Time Counter has four backup registers to store user data. Written data will be preserved after system domain power failure.

1.4.5 Timestamp Generation

Communication between low power and system domain is monitored against transmission errors. The timestamp register can be used to store current counter value when a transmission error is detected.

1.4.6 WKUP0 Input

The Real Time Counter has dedicated WKUP0 pin connected to the edge detector. It can be used to:

- generate WKUP0 interrupt,
- generate wake-up event,
- deactivate SHDN output.

WKUP0 input is equipped with deglitch logic controlled by debounce control register (WKUP0DBCN, 1.7.8. After detecting configured signal edge, the new input state has to be stable for $DEBOUNCE + 1$ slow clock cycles.

1.4.7 SHDN Output

The shutdown controller manages SHDN output pin. In typical application this pin is connected to the external DC/DC converter or low-drop regulator that supplies power to the host. Host controller can activate SHDN output (set high or low) by writing non-zero value to EN bit in shutdown control register (SHDNCTRL, 1.7.2). Writing to the shutdown control register is protected with 0xA5 key value. SHDN output can be deactivated by configured events:

- counter overflow,
- compare match event,
- change on WKUP0 input.

Figure 1.4 presents timing of the SHDN output and the corresponding output enable signal. SHDN output enable signal is low after reset to allow user program to set the desired value of SHDNINV bit and, as the result, the idle value of SHDN output. Output enable signal is asserted one slow clock cycle after the first write operation to the SHDNCTRL register. This ensures that the valid output data will be present when the buffer starts to drive. SHDN output reacts in one clock cycle to the SHDNINV bit changes.

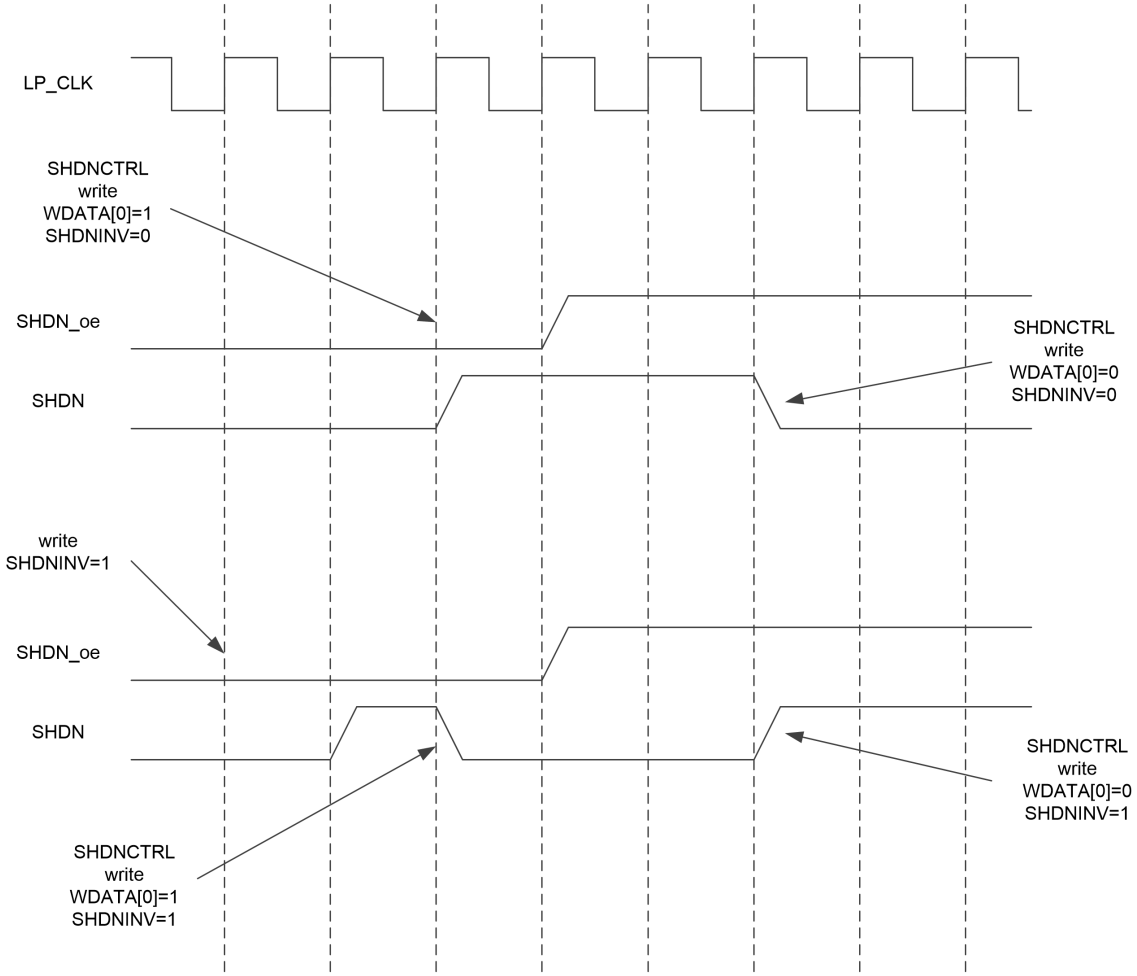


Figure 1.4. SHDN pad timing.

1.5 Interrupts

The Real Timer Counter module has six interrupt sources.

1.5.1 Overflow Interrupt

The Overflow Interrupt is signalized by OVFIF flag in interrupt flag register (IRQF, 1.7.10). Interrupt is generated when COUNT register matches PER register. Interrupt is cleared after reading IRQF register or by writing one in OVFIF bit.

1.5.2 Compare Match Interrupt

The Compare Match Interrupt is signalized by CMPIF flag in interrupt flag register (IRQF, 1.7.10). Interrupt is generated when COUNT register matches COMAPRE register. Interrupt is cleared after reading IRQF register or by writing one in CMPIF bit.

1.5.3 WKUP0 Interrupt

The WKUP0 Interrupt is signalized by WKUP0IF flag in interrupt flag register (IRQF, 1.7.10). Interrupt is generated after detecting configured edge on WKUP0 input. Interrupt is cleared after reading IRQF register or by writing one in WKUP0IF bit.

1.5.4 Ready Interrupt

The Ready Interrupt is signalized by READYIF flag in interrupt flag register (IRQF, 1.7.10). Interrupt is generated when BUSY bit is cleared. Interrupt is cleared after reading IRQF register or by writing one in READYIF bit.

1.5.5 Transmission Error Interrupt

The Transmission Error Interrupt is signalized by TRERRIF flag in interrupt flag register (IRQF, 1.7.10). Interrupt is generated when a transmission error occurs during data synchronization between low power and system power domains. Interrupt is cleared after reading IRQF register or by writing one in TRERRIF bit.

1.5.6 Timestamp Captured Interrupt

The Timestamp Captured Interrupt is signalized by TSCAPTIE flag in interrupt flag register (IRQF, 1.7.10). Interrupt is generated when new timestamp value is generated. Interrupt is cleared after reading IRQF register or by writing one in TSCAPTIE bit.

1.6 Configuration Registers

1.7 Registers List

The core is controlled through registers mapped into memory address space. Not implemented locations are read as zeros.

Address Offset	Register	Name
0x00	CTRL	Control Register
0x04	SHDNCTRL	Shutdown Control Register
0x08	STATUS	Status Register
0x0C	PRES	Prescaler Register
0x10	PER	Period Register
0x14	COMPARE	Compare Register
0x18	COUNT	Count Register
0x1C	WKUP0DBCN	Wake-up Debounce Register
0x20	IRQM	Interrupt Mask Register
0x24	IRQF	Interrupt Flags Register
0x28	IRQMAP	Interrupt Mapping Register
0x2C	BACKUP0	Backup 0 Register
0x30	BACKUP1	Backup 1 Register
0x34	BACKUP2	Backup 2 Register
0x38	BACKUP3	Backup 3 Register
0x3C	TMSTMPCTRL	Timestamp Control Register
0x40	TMSTMPSTAT	Timestamp Status Register
0x44	TMSTMP	Timestamp Register

1.7.1 Control Register

Address: 0x00

31	30	...	...	...	10	9	8
DBG_STOP		...	...	...		WKUP0MD[1:0]	
R/W	R	R	R	R	R	R/W	
0	0	0	0	0	0	0	

7	6	5	4	3	2	1	0
SHDNINV	SHDNRWK0	SHDNRCMP	SHDNROVF	WKUPWK0	WKUPCMP	WKUPOVF	EN
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

EN *RTC Enable*

0 RTC module is disabled. Clock to the module is stopped.

1 RTC module is enabled. Clock to the module is supplied.

WKUPOVF *Overflow Wakeup Enable*

Wake-up on overflow event (COUNT = PER):

0 Disabled.

1 Enabled.

WKUPCMP *Compare Match Wakeup Enable*

Wake-up on compare match event (COUNT = COMPARE):

0 Disabled.

1 Enabled.

WKUPWK0 *WKUP0 Wakeup Enable*

Wake-up on WKUP0 event:

0 Disabled.

1 Enabled.

SHDNROVF *SHDN Release on Overflow Enable*

Release SHDN on overflow event (COUNT = PER):

0 Disabled.

1 Enabled.

SHDNRCMP *SHDN Release on Compare Match Enable*

Release SHDN on compare match event (COUNT = COMPARE):

0 Disabled.

1 Enabled.

SHDNRWK0 *SHDN Release on WKUP0 Enable*

Release SHDN on WKUP0 event:

0 Disabled.

1 Enabled.

SHDNINV *SHDN Inverted*

SHDN output polarization:

0 SHDN is set high on activation and low on deactivation.

1 SHDN is set low on activation and high on deactivation.

WKUP0MD[1:0] *WKUP0 Detection Mode*

WKUP0 detection mode:

WKUP0MD[1:0]	WKUP0 mode
00	detection disabled
01	rising edge detection
10	falling edge detection
11	rising and falling edge detection

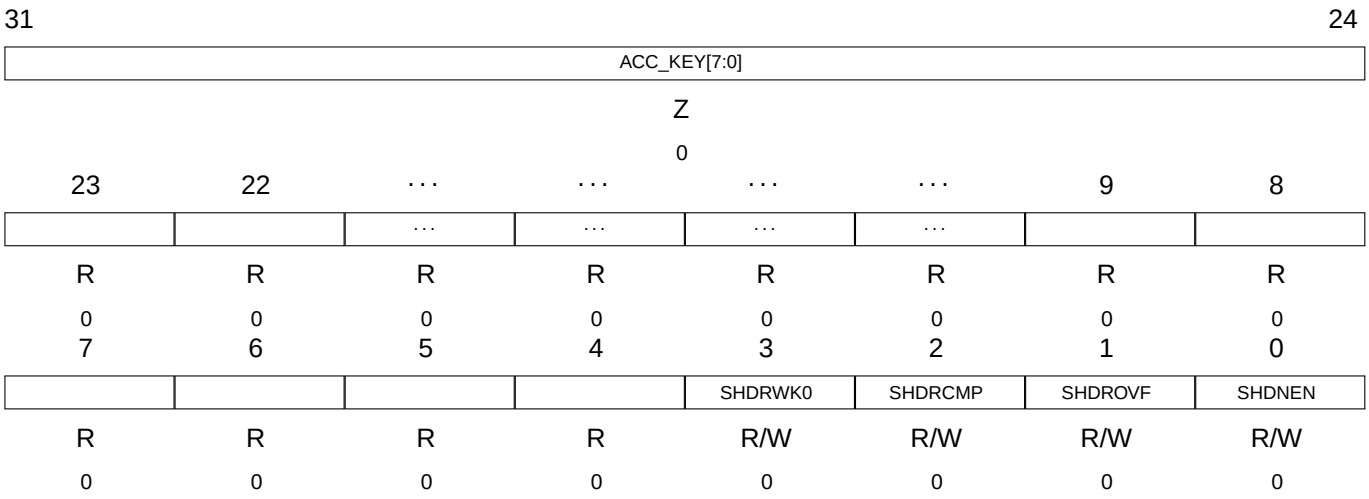
DBG_STOP *RTC stop in debug mode*

0 RTC is counting in debug mode.

1 RTC is not counting in debug mode.

1.7.2 Shutdown Control Register

Address: 0x04



SHUTDOWN *SHDN Enable*

Register write:

0 SHDN deactivation,

1 SHDN activation.

Register read returns actual SHDN state:

0 deactivated,

1 activated.

SHDROVF *SHDN Release on Overflow*

Register write:

0 disable SHDN release on overflow,

1 enable SHDN release on overflow.

Register read:

0 RTC overflow not detected,

1 SHDN was released on RTC overflow.

SHDRCMP *SHDN Release on Compare Match*

Register write:

0 disable SHDN release on compare match event,

1 enable SHDN release on compare match event.

Register read:

0 RTC compare match event not detected,

1 SHDN was released on compare match event.

SHDRWK0 *SHDN Release on WKUP0 event*

Register write:

0 disable SHDN release on WKUP0 event,

1 enable SHDN release on WKUP0 event.

Register read:

0 WKUP0 event not detected,

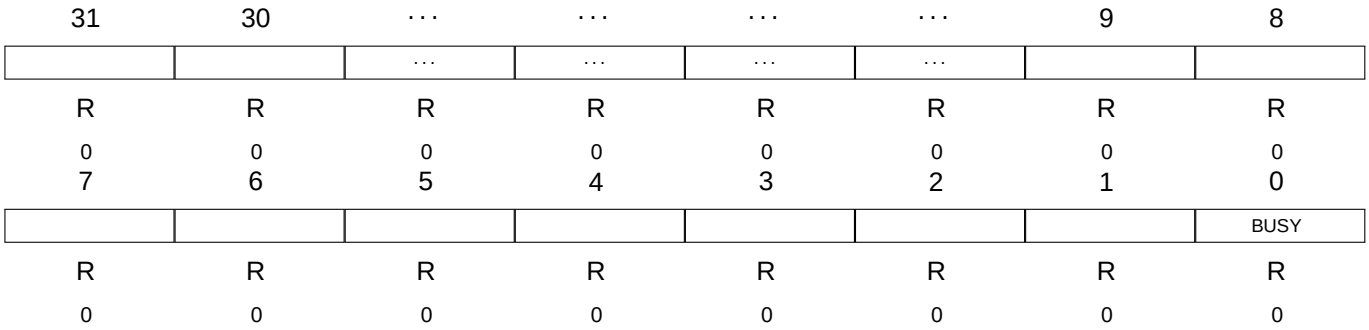
1 SHDN was released on WKUP0 event.

ACC_KEY[7:0] *Access Key*

This bits must be set to 0xA5 for write operation to be successful.

1.7.3 Status Register

Address: 0x08



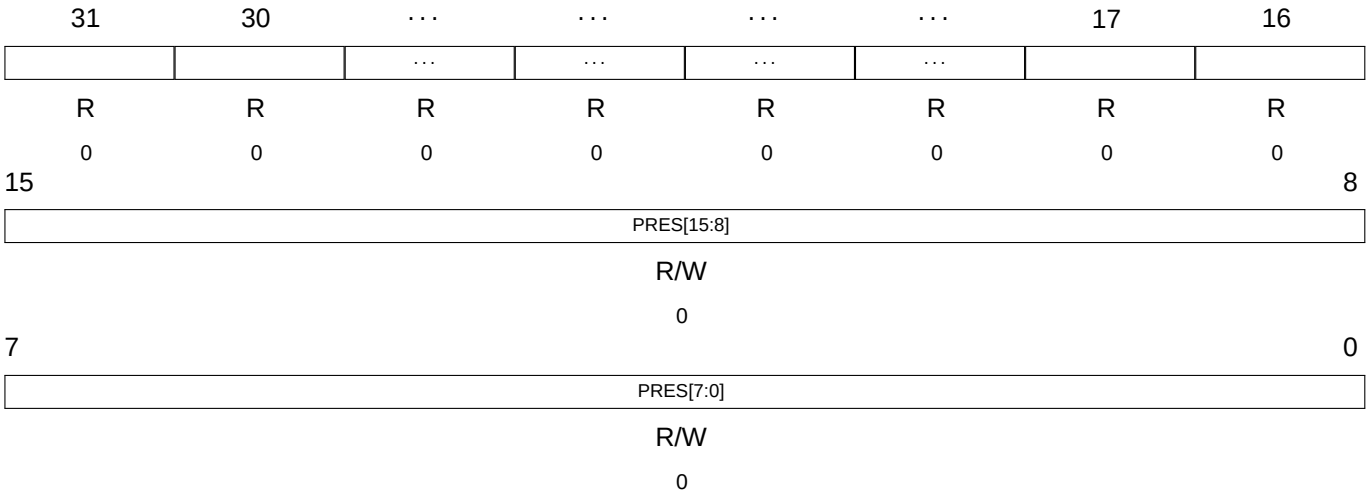
BUSY *RTC Busy*

0 Module is ready for read or write operation.

1 Data synchronization is i progress. Write operation will be discarded.

1.7.4 Prescaler Register

Address: 0x0C



PRES[15:0] *RTC Counter Prescaler*

RTC counter is incremented every $PRESCALER + 1$ slow clock cycles.

1.7.5 Period Register

Address: 0x10

31	0
PER[31:0]	
R/W	
0	

PER[31:0] *RTC Counter Period*

Counting period of RTC module. After reaching this value the timer will cause overflow event and start counting from zero. Can be used to trigger interrupt, wake-up event or release of SHDN line.

1.7.6 Compare Register

Address: 0x14

31	0
COMPARE[31:0]	
R/W	
0	

COMPARE[31:0] *RTC Compare Match*

Trigger value for compare match event. After reaching this value the timer will cause compare match. Can be used to trigger interrupt, wake-up event or release of SHDN line.

1.7.7 Count Register

Address: 0x18

31	0
COUNT[31:0]	
R/W	
0	

COUNT[31:0] *Current COUNT*

Current value of COUNT register.

1.7.8 Wake-up Debounce Register

Address: 0x1C

31	30	...	...	...	...	9	8
		...	...	...	...		
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0

7	0
WKUP0DBCN[7:0]	
R/W	
0	

WKUP0DBCN[7:0] *Debounce Counter Value*

Number of slow clock cycles the WKUP0 line must remain stable after detecting configured signal edge to recognize event.

1.7.9 Interrupt Mask Register

Address: 0x20

31	30	...	...	...	...	9	8
		...	...	...	...		
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
		TSCAPTIE	TRERRIR	READYIE	WKUP0IE	CMPIE	OVFIE
R	R	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

OVFIE *Overflow Interrupt Enable*

0 Overflow interrupt disabled.

1 Overflow interrupt enabled.

CMPIE *Compare Match Interrupt Enable*

0 Compare match interrupt disabled.

1 Compare match interrupt enabled.

WKUP0IE *WKUP0 Interrupt Enable*

0 WKUP0 interrupt disabled.

1 WKUP0 interrupt enabled.

READYIE *Ready Interrupt Enable*

0 Module ready interrupt disable.

1 Module ready interrupt enable.

TRERRIE *Transmission Error Interrupt Enable*

0 Transmission error interrupt disable.

1 Transmission error interrupt enable.

TSCAPTIE *Timestamp Captured Interrupt Enable*

0 Timestamp captured interrupt disable.

1 Timestamp captured interrupt enable.

1.7.10 Interrupt Flags Register

Address: 0x24

31	30	...	...	...	...	9	8
		...	...	...	...		
R	R	R	R	R	R	R	R
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
		TSCAPTIF	TRERRIF	READYIF	WKUP0IF	CMPIF	OVFIF
R	R	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

OVFIF *Overflow Interrupt Flag*

1 Overflow interrupt detected.

Bit is cleared after register readout or by writing one to this position.

CMPIF *Compare Match Interrupt Flag*

1 Compare match interrupt detected.

Bit is cleared after register readout or by writing one to this position.

WKUP0IF *WKUP0 Interrupt Flag*

1 WKUP0 interrupt detected.

Bit is cleared after register readout or by writing one to this position.

READYIF *Ready Interrupt Flag*

1 Interrupt on module ready detected.

Bit is cleared after register readout or by writing one to this position.

TRERRIF *Transmission Error Interrupt Flag*

1 Interrupt on transmission error detected.

Bit is cleared after register readout or by writing one to this position.

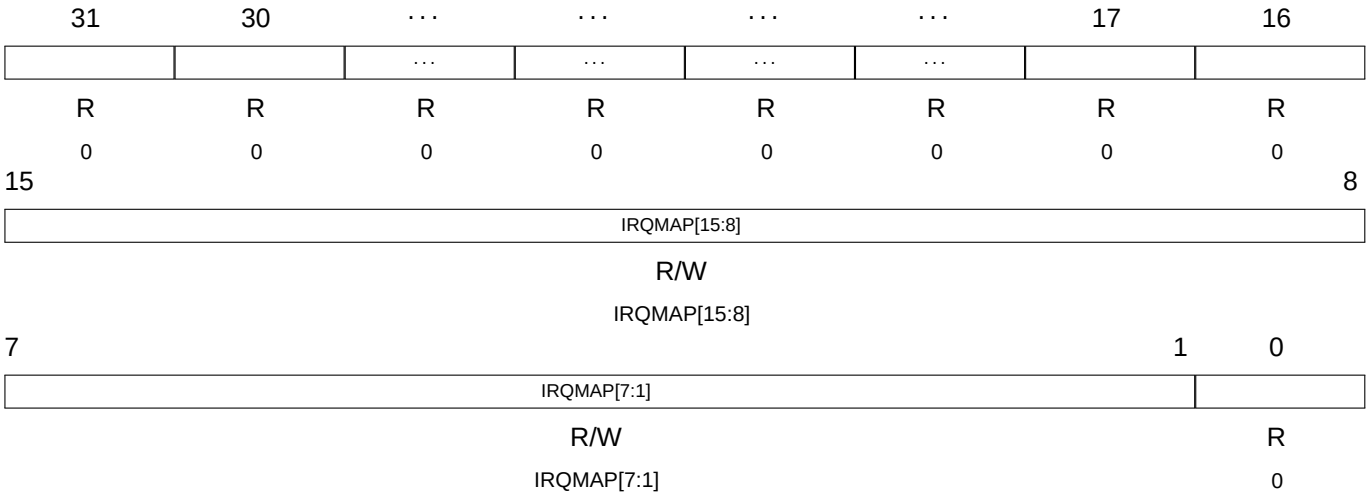
TSCAPTIF *Timestamp Captured Interrupt Flag*

1 Interrupt on new timestamp value was detected.

Bit is cleared after register readout or by writing one to this position.

1.7.11 Interrupt Mapping Register

Address: 0x28

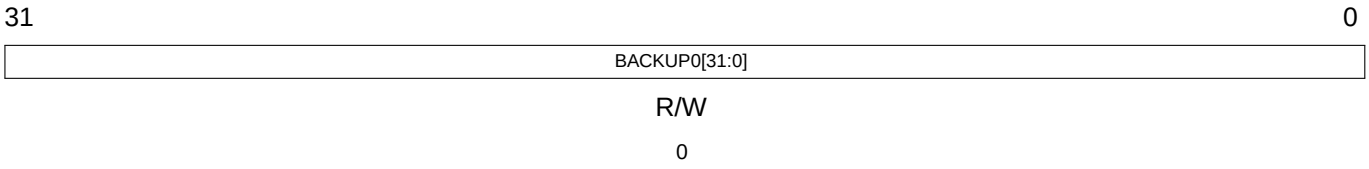


IRQMAP[15:1] *Interrupt Mapping*

Each set bit represents the interrupt number that will be passed to interrupt controller. It is allowed to set more than one bit.

1.7.12 Backup 0 Register

Address: 0x2C

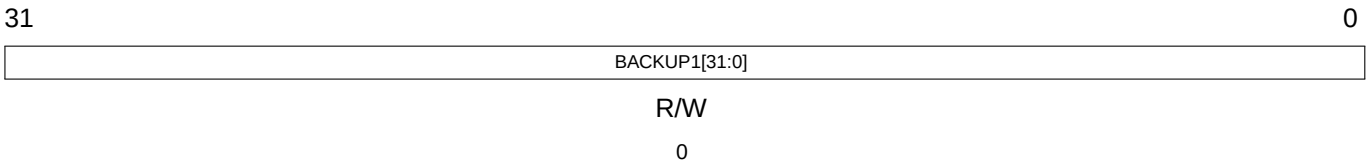


BACKUP0[31:0] *User Value*

User value.

1.7.13 Backup 1 Register

Address: 0x30

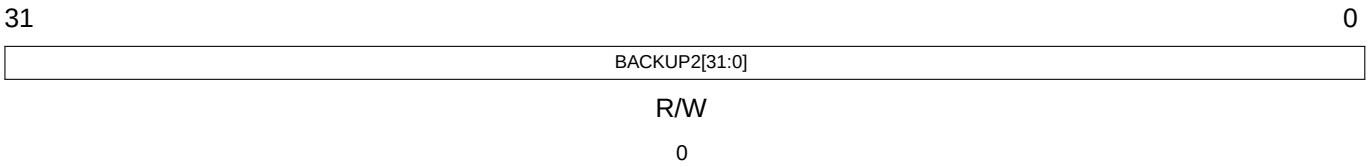


BACKUP1[31:0] *User Value*

User value.

1.7.14 Backup 2 Register

Address: 0x34

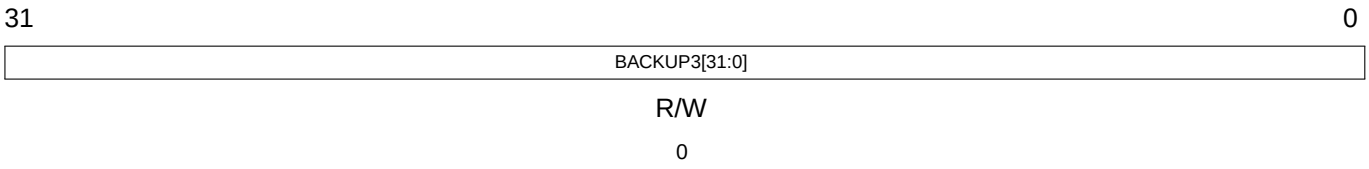


BACKUP2[31:0] *User Value*

User value.

1.7.15 Backup 3 Register

Address: 0x38

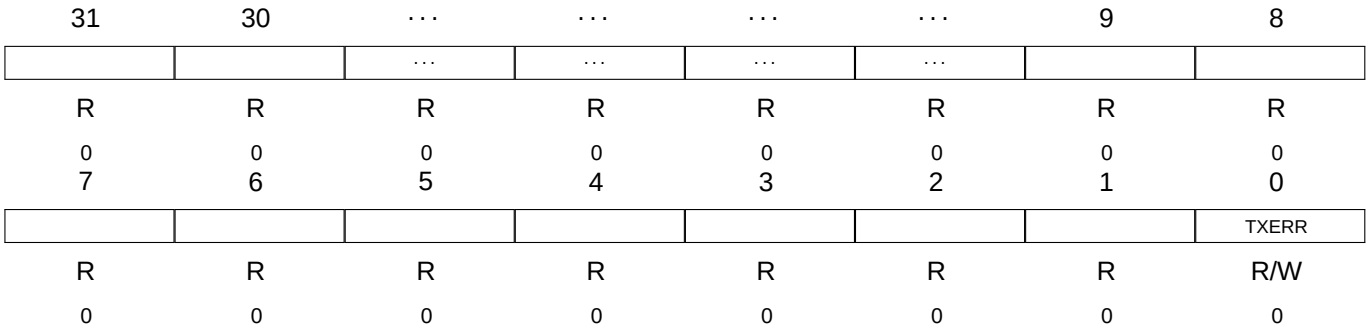


BACKUP3[31:0] *User Value*

User value.

1.7.16 Timestamp Control Register

Address: 0x3C



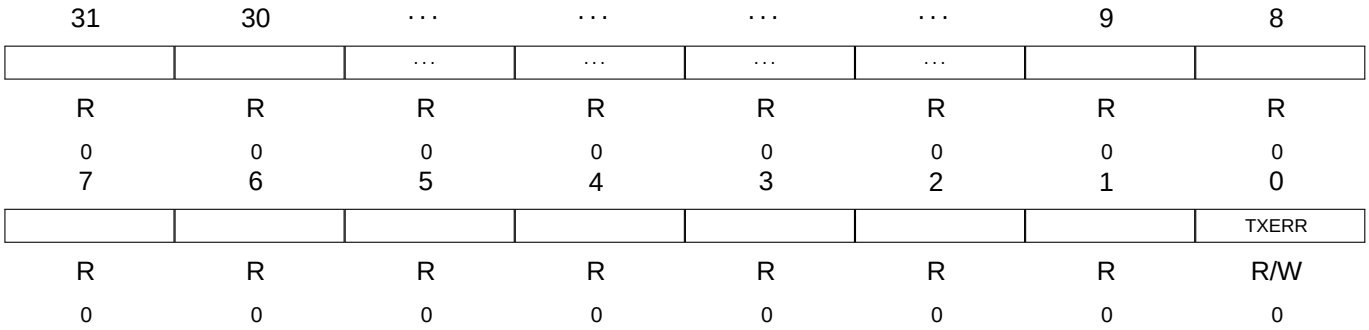
TXERR *Transmission Error Timestamp Capture*

0 Disable timestamp capture on transmission error.

1 Enable timestamp capture on transmission error.

1.7.17 Timestamp Status Register

Address: 0x40

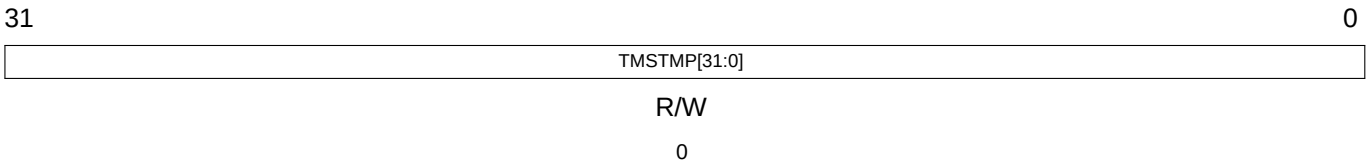


TXERR *Transmission Error Timestamp Captured*

1 Current timestamp value refers to the transmission error event.

1.7.18 Timestamp Register

Address: 0x44



TMSTMP[31:0] *Timestamp Value*

Current timestamp value.

1.8 Implementation

1.8.1 Design Structure

The synthesible RTL IP core part (*COMMON/rtl* and *RTC/rtl* folder) utilizes Verilog 2005 HDL. The testbench part (*RTC/tb* folder) uses SystemVerilog language.

```
COMMON
├── rtl
│   ├── DFF_en.v
│   ├── edge_detector.v
│   └── synchronizer.v
RTC
├── beh
├── rtl
│   ├── APB_RTC.v
│   ├── LP_RTC.v
│   ├── RTC_config.v
│   └── RTC_defines.v
├── tb
│   ├── APB
│   │   ├── tb_APB_RTC_init.v
│   │   └── tb_APB_RTC_reg_access_tasks.v
│   ├── common
│   │   ├── tb_RTC_other_tasks.v
│   │   ├── tb_RTC_read_config_tasks.v
│   │   ├── tb_RTC_write_config_tasks.v
│   │   └── timescale.v
│   ├── run
│   │   └── ncvlog_apb_rtc.sh
│   ├── tests
│   │   ├── tb_interrupt_MAPPING_test.sv
│   │   ├── tb_RTC_clock_uncertainty_test.sv
│   │   ├── tb_RTC_count_test_with_clock_uncertainty.sv
│   │   ├── tb_RTC_count_test.sv
│   │   ├── tb_RTC_POWER_DOWN_test.sv
│   │   ├── tb_RTC_register_access_test.sv
│   │   ├── tb_RTC_SHDN_test.sv
│   │   ├── tb_RTC_transmission_errors_test.sv
│   │   └── tb_RTC_WKUP0_test.sv
│   └── tb_APB_RTC.sv
└── compile.list
```

1.8.2 Simulation Flow

The IP Core is provided with self-checking testbench to verify the correct operation of the IP prior to use in a design. To run the simulation using Cadence® Incisive® Enterprise Simulator run *ncvlog_apb_rtc.sh* script located in *RTC/tb/run* folder. The simulation should end with reporting no errors.

1.8.3 Clock and Reset

The CC-RTC-APB IP core is composed of two modules existing in two power domains: RTC power domain and host/system power domain. Both modules communicate with each other using asynchronous protocol with error detection.

APB_RTC module exists in host/system power domain and utilizes a fully synchronous positive edge clocking domain and negative asynchronous reset assertion. External reset synchronizer has to be used to ensure synchronous reset deassertion.

LP_RTC module exists in low power RTC domain and contains two positive edge clock domains with asynchronous resets assertion. No synchronization logic is implemented to handle single bit communication between two clock domains as source signals are stable long before they are used. Multibit COUNT register is implemented as a gray counter to mitigate data corruption during clock domain crossing. External reset synchronizers are necessary to ensure synchronous reset deassertion.

LP_CLK input is used as a clock source for the RTC counter. PCLK clock is forwarded from the host/system power domain and is used to speed-up data synchronization between two power domains.

1.8.4 Constraints

In most cases only module output ports are registered. Therefore, it is a good practice to reserve the entire clock cycle for module inputs combinational logic and set minimal input delay (*set_input_delay* command). Registered outputs leave the entire clock cycle for external logic (*set_output_delay* command).

Nets crossing LP_CLK and PCLK domains should be marked as false paths (*set_false_path* command). Max delay (*set_max_delay* command) of one destination clock cycle should be set between nets crossing LP_CLK and PCLK clock domains in both APB_RTC and LP_RTC modules.

External module inputs and clock domain crossing logic are implemented using Synchronizer module located in the synchronizer.v file. If possible, they should be replaced with integrated 2FF synchronizers from the target technology library. Otherwise, max delay (*set_max_delay* command) of 10% to 20% of one destination clock cycle should be set between synchronizer stages. Do not use dynamic FFs to implement synchronizer module.

1.8.5 Wake-up Output

LP_WAKE_UP pin can be used by external power management module to wake-up host processor from sleep mode.

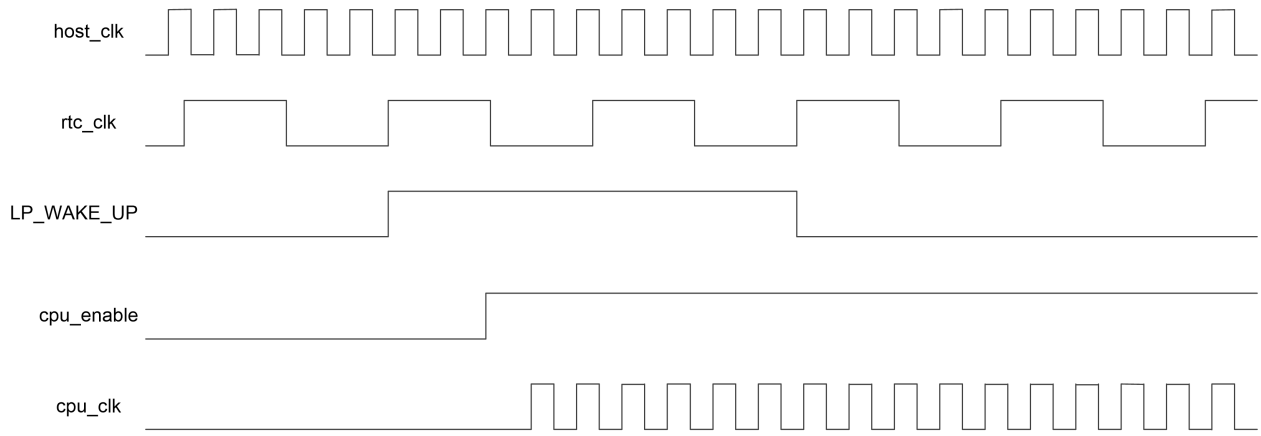


Figure 1.5. Host processor wake-up from sleep mode.

1.8.6 Configuration Options

The table below shows the generic parameters of the core.

LP_RTC module			
Generic name	Description	Range	Default
RTC_width	Configure width of RTC count register[1]	1:32	32
prescaler_width	Configure width of RTC prescaler register[1]	1:32	16
debounce_counter_width	Configure width of RTC debounce counter[1]	1:32	8

APB_RTC module			
Generic name	Description	Range	Default
RTC_width	Configure width of RTC count register[1]	1:32	32
prescaler_width	Configure width of RTC prescaler register[1]	1:32	16
debounce_counter_width	Configure width of RTC debounce counter[1]	1:32	8
default_interrupt_MAPPING	Reset value of interrupt_MAPPING register	0:32767	0

The table below shows the define parameters of the core (RTC_config.v file).

Define name	Description	Default
WKUP0_INPUT_SYNCHRONIZATION	Comment to remove WKUP0 input synchronizing flip flops	defined

[1] RTC_width, prescaler_width and debounce_counter_width parameters must be set equally in LP_RTC and APB_RTC modules.

1.8.7 Signals Description

LP_RTC module				
Signal name	Description	I/O	Active	Type
LP_CLK	Synchronous low power clock	I	rising	clock
PCLK	Synchronous host clock	I	rising	clock
LP_nRST	Asynchronous low power reset	I	low	reset (LP_CLK)
PRESETn	Asynchronous host reset	I	low	reset (PCLK)
HS_DATA_OUTPUT[3:0]	Communication data output	O	data	reg. (PCLK)
HS_DATA_INPUT[3:0]	Communication data input	I	data	comb. (PCLK)
HS_DATA_WRITE	Communication data write	I	high	reg. (PCLK)
HS_DATA_READ	Communication data read	I	high	reg. (PCLK)
HS_RDY	Communication data ready	I	high	reg. (PCLK)
HS_ACK	Communication data acknowledge	O	high	reg. (PCLK)
HS_ERR	Communication data error	O	high	reg. (PCLK)
HS_BUSY	Communication data busy	O	high	reg. (PCLK)
HS_debug_mode	Debug mode input	I	high	reg. (LP_CLK)
LP_OVF_event	Overflow event output	O	high	reg. (LP_CLK)
LP_COMPARE_match_event	Compare match event output	O	high	reg. (LP_CLK)
LP_WKUP0_event	WKUP0 event output	O	high	reg. (LP_CLK)
LP_TIMESTAMP_ CAPTURED_event	Timestamp captured event output	O	high	reg. (LP_CLK)
LP_WAKE_UP	Host wake-up signal	O	pos. pulse	reg. (LP_CLK)
LP_clock_request	Clock request signal	O	high	comb.[2]
WKUP0	WKUP0 pad input	I	data	reg./comb.[3]
SHDN	SHDN pad output	O	data	reg.
SHDN_oe	SHDN pad output enable	O	high	reg.

APB_RTC module				
Signal name	Description	I/O	Active	Type
PCLK	Synchronous clock	I	rising	clock
PRESETn	Asynchronous reset	I	low	reset
PSEL	APB peripheral select	I	high	comb.
PENABLE	APB bus enable	I	high	comb.
PADDR[6:2]	APB bus address	I	data	comb.
PWRITE	APB bus write	I	high	comb.
PWDATA[31:0]	APB bus write data	I	data	comb.
PREADY	APB bus ready	O	high	const.
PRDATA[31:0]	APB bus read data	O	data	reg.
LP_DATA_OUTPUT[3:0]	Communication data output	O	data	reg.
LP_DATA_INPUT[3:0]	Communication data input	I	data	comb.
LP_DATA_WRITE	Communication data write	O	high	reg.
LP_DATA_READ	Communication data read	O	high	reg.
LP_RDY	Communication data ready	O	high	reg.
LP_ACK	Communication data acknowledge	I	high	reg.
LP_ERR	Communication data error	I	high	reg.
LP_BUSY	Communication data busy	I	high	reg.
LP_OVF_event	Overflow event output	I	high	reg.
LP_COMPARE_match_event	Compare match event output	I	high	reg.
LP_WKUP0_event	WKUP0 event output	I	high	reg.
LP_TIMESTAMP_ CAPTURED_event	Timestamp captured event output	I	high	reg.
interrupt_OVF	Overflow interrupt	O	high	reg.
interrupt_COMPARE_MATCH	Compare match interrupt	O	high	reg.
interrupt_READY	Ready interrupt	O	high	reg.
interrupt_WKUP0	WKUP0 interrupt	O	high	reg.
interrupt_TIMESTAMP_CAPTURED	Timestamp captured interrupt	O	high	reg.
interrupt_TRANSMISSION_ERROR	Transmission error interrupt	O	high	reg.
interrupt_MAPPING[15:1]	Interrupt mapping vector	O	data	reg.
clock_request	Clock request signal	O	high	reg.

[2] Signal must be externally synchronized to LP_CLK.
[3] Depending on WKUP0_INPUTS_SYNCHRONIZATION setting. Defined value will insert input synchronization flip-flops.

1.8.8 Instantiation

```
//-------------------------------------------------------------------------------
// HOST POWER DOMAIN
//-------------------------------------------------------------------------------

icg
    icg_rtc_u ( .E(rtc_PSEL|rtc_clock_request),
                .clk(PCLK),
                .gclk(rtc_hs_pclk),
                .scan_enable(scan_enable));

APB_RTC     #(  .RTC_width(CFG_RTC_WIDTH),
                .prescaler_width(CFG_TMR_PRES),
                .debounce_counter_width(CFG_DBNC_WIDTH),
                .default_interrupt_MAPPING(CFG_DEF_INT_MAPPING)

APB_RTC_u   (   .PCLK(rtc_hs_pclk),
                .PRESETn(rtc_hs_rst),
                .PSEL(rtc_PSEL),
                .PENABLE(PENABLE),
                .PADDR(PADDR[6:2]),
                .PWRITE(PWRITE),
                .PWDATA(PWDATA),
                .PREADY(rtc_PREADY),
                .PRDATA(rtc_PRDATA),
                .LP_DATA_OUTPUT(rtc_hs_data_output),
                .LP_DATA_INPUT(rtc_hs_data_input),
                .LP_DATA_WRITE(rtc_hs_data_write),
                .LP_DATA_READ(rtc_hs_data_read),
                .LP_RDY(rtc_hs_rdy),
                .LP_ACK(rtc_hs_ack),
                .LP_ERR(rtc_hs_err),
                .LP_OVF_event(rtc_hs_ovf_event),
                .LP_WKUP0_event(rtc_hs_wkup0_event),
                .LP_COMPARE_MATCH_event(rtc_hs_compare_match_event),
                .LP_TIMESTAMP_CAPTURED_event(rtc_hs_timestamp_captured_event),
                .LP_BUSY(rtc_hs_busy),
                .interrupt_OVF(rtc_interrupt_OVF),
                .interrupt_COMPARE_MATCH(rtc_interrupt_COMPARE_MATCH),
                .interrupt_READY(rtc_interrupt_READY),
```


```
                .interrupt_MAPPING(rtc_interrupt_MAPPING),
                .interrupt_WKUP0(rtc_interrupt_WKUP0),
                .interrupt_TRANSMISSION_ERROR(rtc_interrupt_TRANS_ERROR),
                .interrupt_TIMESTAMP_CAPTURED(rtc_interrupt_TIMESTAMP),
                .clock_request(rtc_clock_request));

assign rtc_irq        = rtc_interrupt_OVF | rtc_interrupt_COMPARE_MATCH |
                        rtc_interrupt_READY | rtc_interrupt_MAPPING |
                        rtc_interrupt_WKUP0 | rtc_interrupt_TRANS_ERROR |
                        rtc_interrupt_TIMESTAMP;

assign rtc_irq_vector = rtc_interrupt_MAPPING & {15{rtc_irq}};

//-------------------------------------------------------------------------------
// LOW POWER ISOLATION CELLS PCLK -> LP_CLK
//-------------------------------------------------------------------------------

wire [9:0] from_pclk_to_lpclk;
wire [9:0] to_lpclk_from_pclk;

assign from_pclk_to_lpclk = {    rtc_hs_rst,
                                 rtc_hs_data_output,
                                 rtc_hs_rdy,
                                 rtc_hs_data_write,
                                 rtc_hs_data_read,
                                 rtc_hs_pclk,
                                 rtc_hs_debug_mode };

assign to_lpclk_from_pclk = rtc_hs_power_good ? from_pclk_to_lpclk : 10'd0;

assign { rtc_lp_rst,
         rtc_lp_data_output,
         rtc_lp_rdy,
         rtc_lp_data_write,
         rtc_lp_data_read,
         rtc_lp_pclk,
         rtc_lp_debug_mode } = to_lpclk_from_pclk;

//-------------------------------------------------------------------------------
// LOW POWER ISOLATION CELLS LP_CLK -> PCLK
//-------------------------------------------------------------------------------
```

```
wire [11:0] from_lpclk_to_pclk;
wire [11:0] to_pclk_from_lpclk;

assign from_lpclk_to_pclk = {    rtc_lp_data_input,
                                 rtc_lp_ack,
                                 rtc_lp_err,
                                 rtc_lp_ovf_event,
                                 rtc_lp_wkup0_event,
                                 rtc_lp_compare_match_event,
                                 rtc_lp_timestamp_captured_event,
                                 rtc_lp_busy,
                                 rtc_lp_wake_up };

assign to_pclk_from_lpclk = rtc_lp_power_good ? from_lpclk_to_pclk : 12'd0;

assign { rtc_hs_data_input,
         rtc_hs_ack,
         rtc_hs_err,
         rtc_hs_ovf_event,
         rtc_hs_wkup0_event,
         rtc_hs_compare_match_event,
         rtc_hs_timestamp_captured_event,
         rtc_hs_busy,
         rtc_hs_wake_up } = to_pclk_from_lpclk;

//-------------------------------------------------------------------------------
// LOW POWER RTC DOMAIN
//-------------------------------------------------------------------------------

rstgen       #( .WIDTH(4))
rstgen_lpclk  ( .clk(LP_CLK),
                .rst(rst),
                .scan_mode(scan_mode),
                .in(1'b1),
                .out(LP_RST));

rstgen       #( .WIDTH(4))
rstgen_pclk   ( .clk(rtc_lp_pclk),
                .rst(rtc_lp_rst),
                .scan_mode(scan_mode),
                .in(LP_RST),
                .out(PRESETn));
```


```
dffhsynchnr0 #( .N(1))
dff_req_synch ( .clk(LP_CLK),
                .rst(LP_RST),
                .D(rtc_lp_clock_request),
                .Q(rtc_lp_clock_request_synch));

icg
    icg_lprtc ( .E(rtc_lp_clock_request_synch),
                .clk(LP_CLK),
                .gclk(gated_LP_CLK),
                .scan_enable(scan_enable));

LP_RTC       #( .RTC_width(CFG_RTC_WIDTH),
                .prescaler_width(CFG_TMR_PRES),
                .debounce_counter_width(CFG_DBNC_WIDTH))

    LP_RTC_u  ( .LP_CLK(gated_LP_CLK),
                .PCLK(rtc_lp_pclk),
                .PRESETn(PRESETn),
                .LP_nRST(LP_RST),
                .HS_DATA_INPUT(rtc_lp_data_output),
                .HS_DATA_OUTPUT(rtc_lp_data_input),
                .HS_DATA_WRITE(rtc_lp_data_write),
                .HS_DATA_READ(rtc_lp_data_read),
                .HS_RDY(rtc_lp_rdy),
                .HS_ACK(rtc_lp_ack),
                .HS_ERR(rtc_lp_err),
                .LP_OVF_event(rtc_lp_ovf_event),
                .LP_COMPARE_MATCH_event(rtc_lp_compare_match_event),
                .LP_TIMESTAMP_CAPTURED_event(rtc_lp_timestamp_captured_event),
                .LP_WKUP0_event(rtc_lp_wkup0_event),
                .HS_BUSY(rtc_lp_busy),
                .LP_WAKE_UP(rtc_lp_wake_up),
                .LP_clock_request(rtc_lp_clock_request),
                .HS_debug_mode(rtc_lp_debug_mode),
                .WKUP0(WKUP0),
                .SHDN(SHDN),
                .SHDN_oe(SHDN_oe));

io_pad_model      #( .IO_NUM(1))
  wkup_pad_model   ( .core_input(WKUP0),
```

```
                    .core_output(1'b0),
                    .IO_pad(WKUP0_pad),
                    .output_enable(1'b0));

io_pad_model      #( .IO_NUM(1))
  shdn_pad_model   ( .core_input(1'b0),
                    .core_output(SHDN),
                    .IO_pad(SHDN_pad),
                    .output_enable(SHDN_oe));
```


1.9 Revision History

Doc. Rev.	Date	Comments
1.2	11-2017	Corrected 1.8.8 Instantiation section.
1.1	08-2017	Updated 1.8.3 Clock and Reset section. Updated 1.8.4 Constraints section.
1.0	05-2017	First Issue.

ChipCraft Sp. z o.o. Dobrzańskiego 3 lok. BS073, 20-262 Lublin, POLAND www.chipcraft-ic.com

 CC-RTC-APB-Doc_112017.