

## Scope

---

This document describes the CC-WDT-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

|  |          |
|--|----------|
| <b>1. Watchdog Timer</b> . . . . .     | <b>3</b> |
| 1.1 Overview . . . . .                 | 3        |
| 1.1.1 Functionality . . . . .          | 3        |
| 1.1.2 Block Diagram . . . . .          | 3        |
| 1.2 Operation . . . . .                | 4        |
| 1.2.1 STANDARD Mode . . . . .          | 4        |
| 1.2.2 WINDOW Mode . . . . .            | 5        |
| 1.2.3 Write Protection . . . . .       | 5        |
| 1.3 Configuration Registers . . . . .  | 6        |
| 1.3.1 Registers List . . . . .         | 6        |
| 1.3.2 Lock Register . . . . .          | 6        |
| 1.3.3 Control Register . . . . .       | 7        |
| 1.3.4 Count Register . . . . .         | 7        |
| 1.3.5 Period Register . . . . .        | 8        |
| 1.3.6 Period Closed Register . . . . . | 8        |
| 1.3.7 Prescaler Register . . . . .     | 8        |
| 1.4 Implementation . . . . .           | 9        |
| 1.4.1 Design Structure . . . . .       | 9        |
| 1.4.2 Simulation Flow . . . . .        | 9        |
| 1.4.3 Clock and Reset . . . . .        | 10       |
| 1.4.4 Constraints . . . . .            | 12       |
| 1.4.5 Configuration Options . . . . .  | 12       |
| 1.4.6 Signals Description . . . . .    | 12       |
| 1.4.7 Instantiation . . . . .          | 13       |
| 1.5 Revision History . . . . .         | 14       |



# 1. Watchdog Timer

## 1.1 Overview

Watchdog timer is used to monitor system behavior, in particular to detect and recover from malfunctions. In STANDARD mode the watchdog timer counts clock cycles up to configured maximum value in period register (PER). In case of reaching the period value the module initiates system reset event. To prevent that, the user program has to regularly reset/modify watchdog COUNT register. In WINDOW mode the watchdog timer COUNT register has to be written in a particular time window. Too early or too late access to COUNT register will result in system reset. All watchdog registers are protected against accidental modification.

### 1.1.1 Functionality

- Configurable prescaler and timeout,
- two operation modes:
  - STANDARD,
  - WINDOW,
- register write protection.

### 1.1.2 Block Diagram

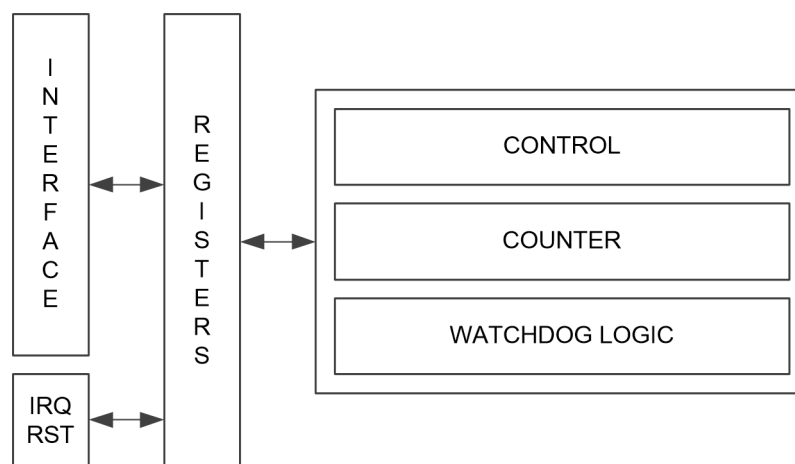


Figure 1.1. Watchdog Timer block diagram.



## 1.2 Operation

The Watchdog Timer counts bus clock cycles. The counting scheme comprises of two levels. The first one is prescaler counter that directly counts bus clock cycles up to the prescaler register (PRES) value and gets reset. The COUNT register counts prescaler overflow events and is incremented after every PRES+1 clock cycles. The internal prescaler register is reset each time a new value is stored in PRES register.

### 1.2.1 STANDARD Mode

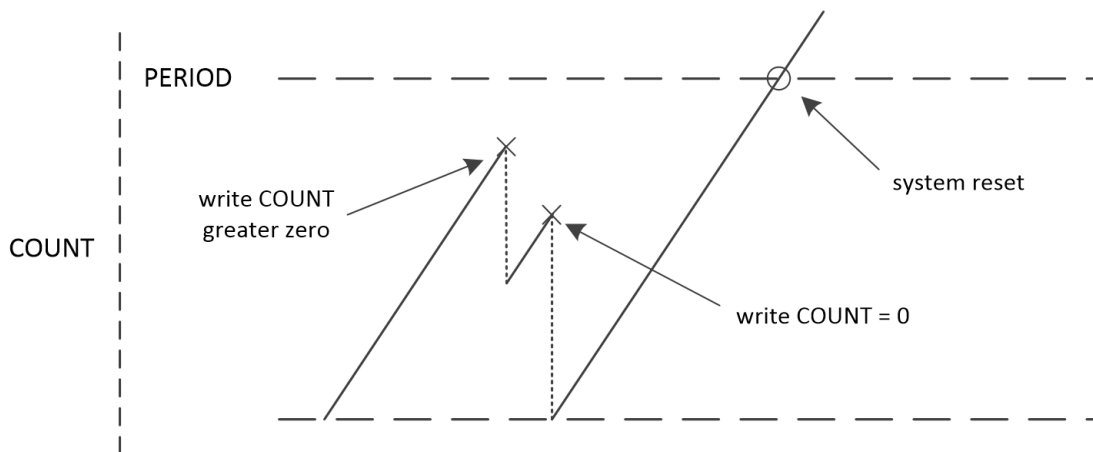


Figure 1.2. Watchdog Timer in STANDARD mode.

In STANDARD mode (Figure 1.2) Watchdog Timer counts prescaled clock cycles up to the defined value in period register (PER). After reaching that value the module initiates system reset event. User program can modify both COUNT and PER registers to adjust timeout value depending on the current needs.



## 1.2.2 WINDOW Mode

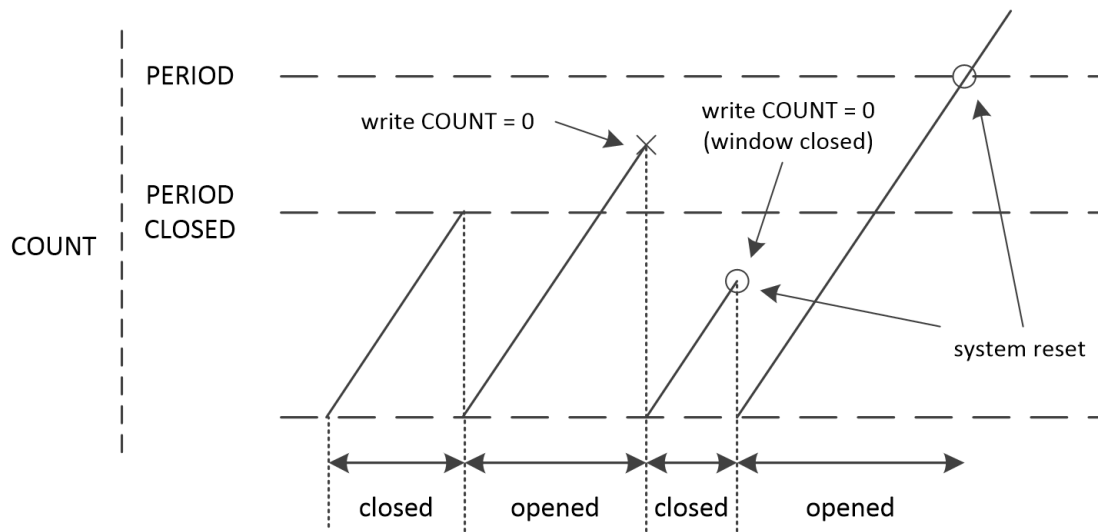


Figure 1.3. Watchdog Timer in WINDOW mode.

In WINDOW mode (Figure 1.3) Watchdog Timer uses two values stored in period closed (PERCL) and period (PER) registers. The first one defines the time of closed window, when modifying COUNT register will lead to system reset. After that time the opened window lasts for the time defined in period register. Timer overflow during opened window will cause system reset. User application should reset COUNT register before that moment. Writing to COUNT register during opened window will switch to closed window counting again.

## 1.2.3 Write Protection

CTRL, PER, PERCL and PRES registers are protected against accidental modification. Each register has an unlock bit stored in LOCK register that unlocks it for modification for 32 clock cycles. User application has to ensure that it successfully modified the desired register before they were closed for modification again.



## 1.3 Configuration Registers

### 1.3.1 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           | LOCK     | Lock Register          |
| 0x04           | CTRL     | Control Register       |
| 0x08           | COUNT    | Count Register         |
| 0x0C           | PER      | Period Register        |
| 0x10           | PERCL    | Period Closed Register |
| 0x14           | PRES     | Prescaler Register     |

### 1.3.2 Lock Register

Address: 0x00

| 31 | 30 | ... | ... | ...      | ...       | 9       | 8        |
|----|----|-----|-----|----------|-----------|---------|----------|
| 0  | 0  | 0   | 0   | 0        | 0         | 0       | 0        |
| 7  | 6  | 5   | 4   | 3        | 2         | 1       | 0        |
|    |    |     |     | PRES_UNL | PERCL_UNL | PER_UNL | CTRL_UNL |
| 0  | 0  | 0   | 0   | R/W      | R/W       | R/W     | R/W      |
| 0  | 0  | 0   | 0   | 0        | 0         | 0       | 0        |

#### **CTRL\_UNL** *CTRL Register Unlock*

Unlocks modification of CTRL register for 32 clock cycles.

#### **PER\_UNL** *PER Register Unlock*

Unlocks modification of PER register for 32 clock cycles.

#### **PERCL\_UNL** *PERCL Register Unlock*

Unlocks modification of PERCL register for 32 clock cycles.

#### **PRES\_UNL** *PRES Register Unlock*

Unlocks modification of PRES register for 32 clock cycles.



### 1.3.3 Control Register

Address: 0x04

|          |    |     |     |     |     |      |     |
|----------|----|-----|-----|-----|-----|------|-----|
| 31       | 30 | ... | ... | ... | ... | 9    | 8   |
| DBG_STOP |    | ... | ... | ... | ... |      |     |
| R/W      | O  | O   | O   | O   | O   | O    | O   |
| 0        | 0  | 0   | 0   | 0   | 0   | 0    | 0   |
| 7        | 6  | 5   | 4   | 3   | 2   | 1    | 0   |
|          |    |     |     |     |     | MODE | EN  |
| O        | O  | O   | O   | O   | O   | R/W  | R/W |
| 0        | 0  | 0   | 0   | 0   | 0   | 0    | 0   |

**EN** *Watchdog Enable*

- 0 Watchdog Timer disabled. Clock to the module is stopped.
- 1 Watchdog Timer enabled. Clock to the module is supplied.

**MODE** *Operation Mode*

- 0 STANDARD mode.
- 1 WINDOW mode.

**DBG\_STOP** *Stop in debug mode*

- 0 Watchdog is counting in debug mode.
- 1 Watchdog is not counting in debug mode.

### 1.3.4 Count Register

Address: 0x08

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

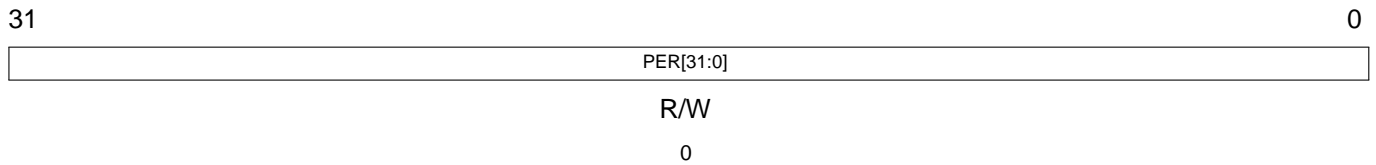
**COUNT[31:0]** *Current Count*

Current count value.



### 1.3.5 Period Register

**Address:** 0x0C

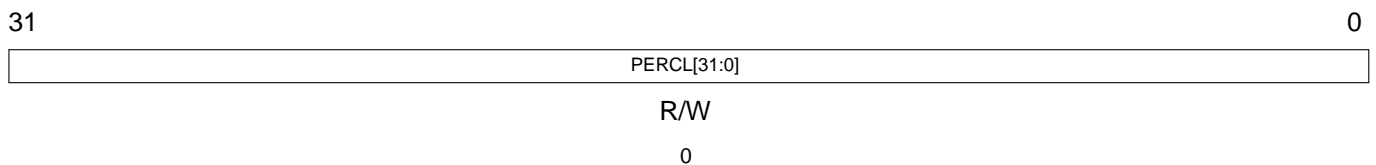


#### **PER[31:0]** *Period Value*

When COUNT register reaches this value, watchdog will initiate system reset.

### 1.3.6 Period Closed Register

**Address:** 0x10

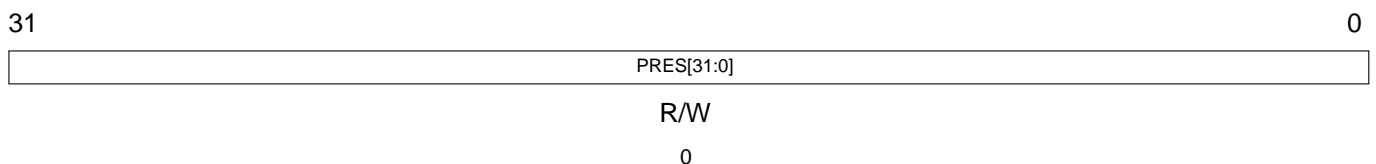


#### **PERCL[31:0]** *Period Closed Value*

In WINDOW mode, this value indicates the time of closed window. Write to COUNT register during closed window will initiate system reset.

### 1.3.7 Prescaler Register

**Address:** 0x14



#### **PRSC[31:0]** *Watchdog Prescaler*

Prescaler counter directly counts bus clock cycles up to the prescaler register (PRES) value and gets reset.





## 1.4 Implementation

### 1.4.1 Design Structure

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

```
WATCHDOG
├── beh
├── rtl
│   ├── APB_WATCHDOG.v
│   ├── WATCHDOG_config.v
│   ├── WATCHDOG_defines.v
│   └── WATCHDOG.v
├── tb
│   ├── APB
│   │   ├── tb_APB_WATCHDOG_init.v
│   │   └── tb_APB_WATCHDOG_reg_access_tasks.v
│   ├── common
│   │   ├── tb_WATCHDOG_other_tasks.v
│   │   ├── tb_WATCHDOG_read_config_tasks.v
│   │   ├── tb_WATCHDOG_write_config_tasks.v
│   │   └── timescale.v
│   ├── run
│   │   └── ncvlog_apb_watchdog.sh
│   ├── tests
│   │   ├── tb_WATCHDOG_NORMAL_MODE_test.sv
│   │   └── tb_WATCHDOG_WINDOW_MODE_test.sv
│   └── tb_APB_WATCHDOG.sv
└── compile.list
```

### 1.4.2 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\_watchdog.sh* script located in *WATCHDOG/tb/run* folder. The simulation should end with reporting no errors.



### 1.4.3 Clock and Reset

The CC-WDT-APB utilizes a fully synchronous design with one positive edge clocking domain and negative asynchronous reset assertion. External reset synchronizer has to be used to ensure synchronous reset deassertion.

The system\_reset output should be connected to the main system reset synchronizer input.

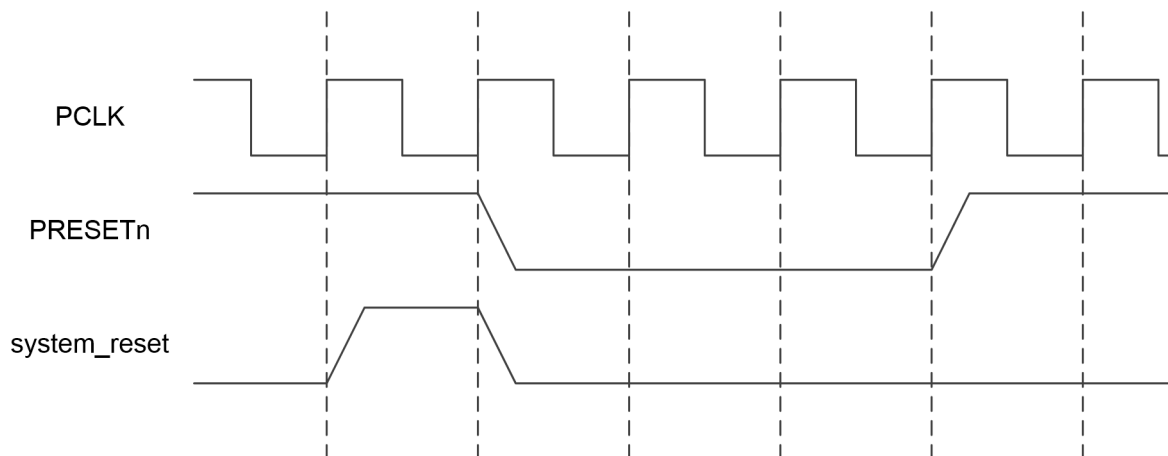


Figure 1.4. Watchdog timeout system reset.



When module clock is supplied with clock gating cell, the proper enable procedure must be carried out. The first write to the LOCK register will hold the clock gating cell on for the 32 PLCK clock cycles. The second write should set EN bit in CTRL to hold the clock\_request signal high.

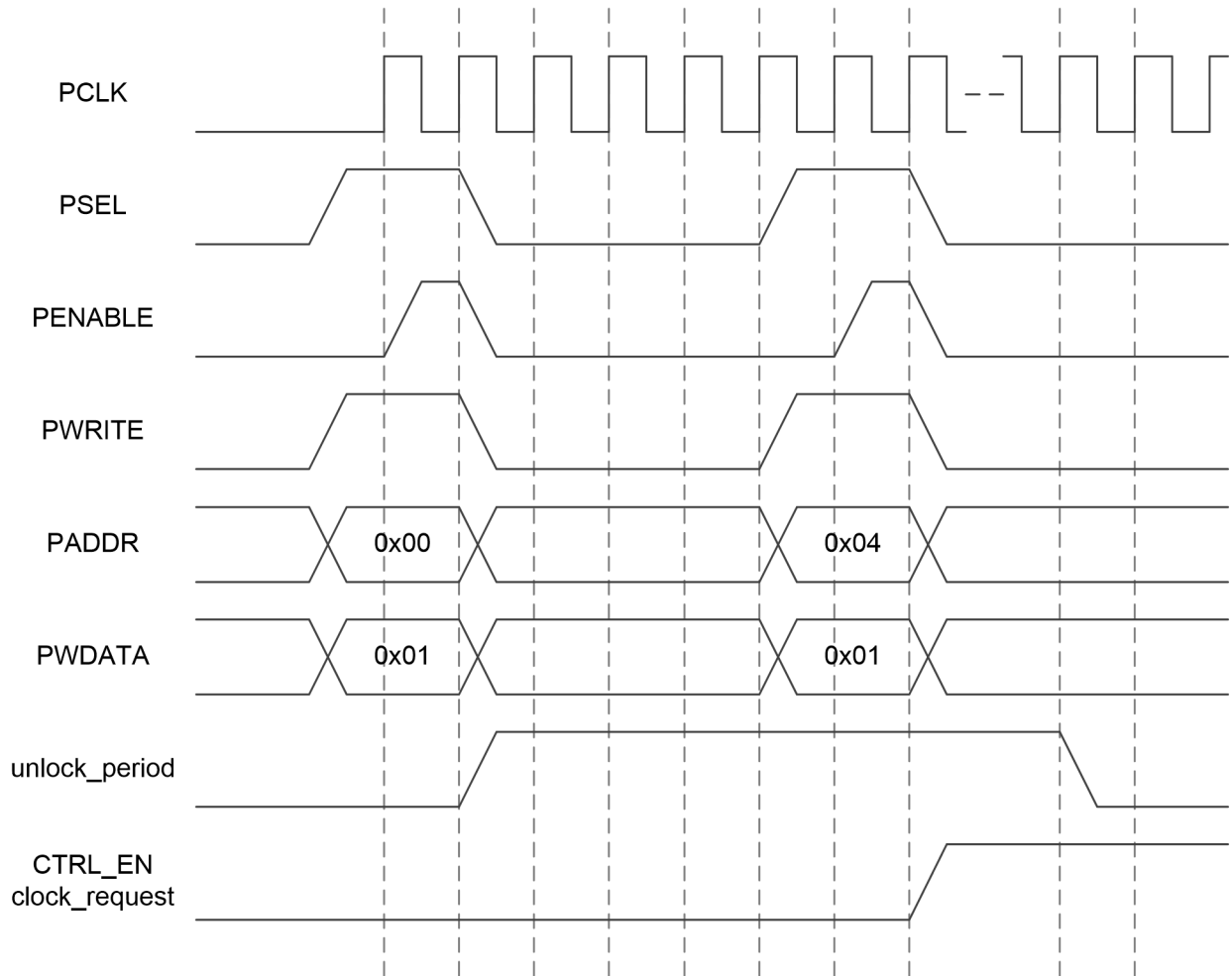


Figure 1.5. Watchdog clock enable procedure.



## 1.4.4 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).

## 1.4.5 Configuration Options

The table below shows the generic parameters of the core.

| Generic name    | Description                                    | Range | Default |
|-----------------|--|-------|---------|
| watchdog_width  | Configure width of watchdog count register     | 1:32  | 32      |
| prescaler_width | Configure width of watchdog prescaler register | 1:32  | 8       |

The table below shows the define parameters of the core (WATCHDOG\_config.v file).

| Define name                   | Description  | Default |
|-------------------------------|--|---------|
| WATCHDOG_UNLOCKED_NUM_CYCLES  | Number of clock cycles the watchdog counter remains unlocked.                                  | 32      |
| WATCHDOG_UNLOCK_COUNTER_WIDTH | Width of watchdog unlock counter. Must allow to store full WATCHDOG_UNLOCKED_NUM_CYCLES value. | 6       |

## 1.4.6 Signals Description

| 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[4:2]    | 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.   |
| system_reset  | Watchdog reset signal | O   | high   | reg.   |
| clock_request | Clock request signal  | O   | high   | reg.   |
| debug_mode    | Debug mode input      | I   | high   | comb.  |



## 1.4.7 Instantiation

```
rstgen #(
    .WIDTH(3))
rstgen_u (
    .clk(PCLK),
    .rstn(rstn),
    .scan_mode(scan_mode),
    .in(~system_reset),
    .out(PRESETn));

icg
icg_wdt_u (
    .E(wdt_PSEL|wdt_clock_request),
    .clk(PCLK),
    .gclk(wdt_clk),
    .scan_enable(scan_enable));

APB_WATCHDOG #(
    .watchdog_width(32),
    .prescaler_width(8))
APB_WATCHDOG_u (
    .PCLK(wdt_clk),
    .PRESETn(PRESETn),
    .PSEL(wdt_PSEL),
    .PENABLE(PENABLE),
    .PADDR(PADDR[4:2]),
    .PWRITE(PWRITE),
    .PDATA(PWDATA),
    .PREADY(wdt_PREADY),
    .PRDATA(wdt_PRDATA),
    .system_reset(system_reset),
    .debug_mode(debug_mode),
    .clock_request(wdt_clock_request));
```



## 1.5 Revision History

| Doc. Rev. | Date    | Comments  |
|-----------|---------|---|
| 1.1       | 11-2018 | Editorial corrections in 1.4.7 Instantiation section. |
| 1.0       | 05-2017 | First Issue.  |





**ChipCraft Sp. z o.o.**

Dobrzańskiego 3 lok. BS073, 20-262 Lublin, POLAND

[www.chipcraft-ic.com](http://www.chipcraft-ic.com)

©2018 ChipCraft Sp. z o.o.

CC-WDT-APB-Doc\_112018.

ChipCraft<sup>®</sup>, ChipCraft logo and combination of thereof are registered trademarks or trademarks of ChipCraft Sp. z o.o. All other names are the property of their respective owners.

Disclaimer: ChipCraft makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. ChipCraft does not make any commitment to update the information contained herein.