Datasheet

CC-WDT-AXI Documentation

1.0

Scope

This document describes the CC-WDT-AXI 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. 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

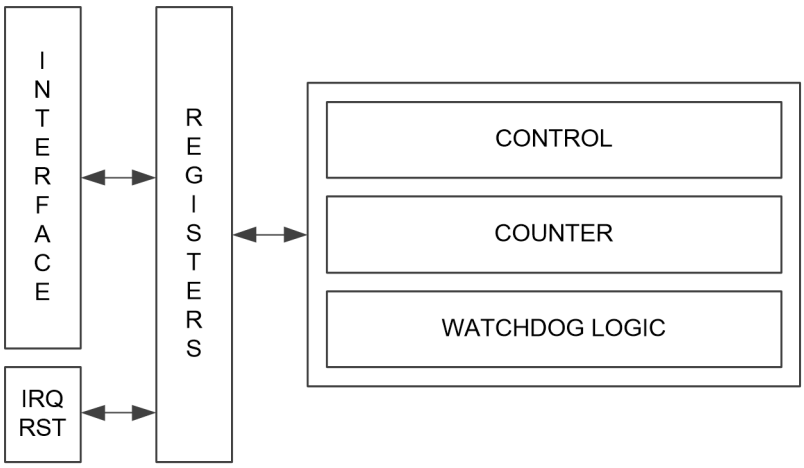


Figure 1.1. Watchdog Timer block diagram.

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

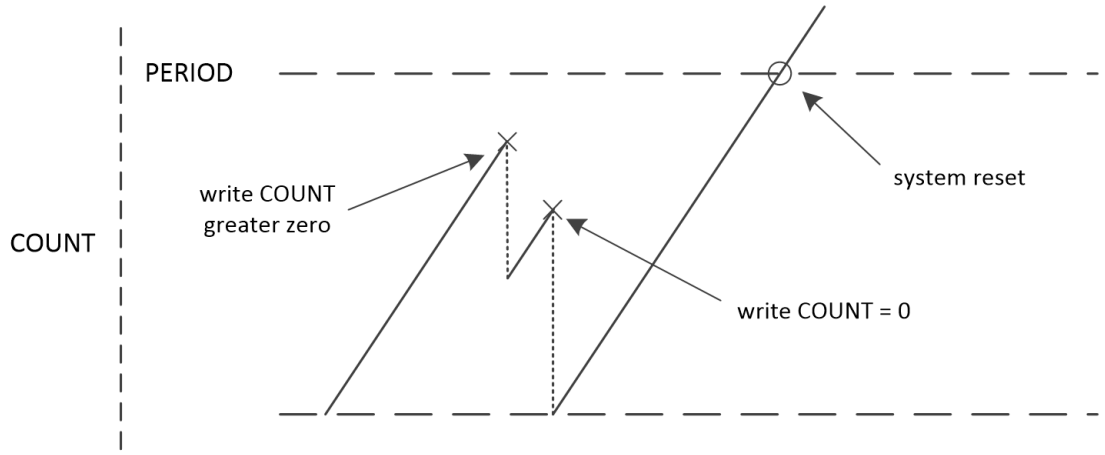


Figure 1.2. Watchdog Timer in STANDARD mode.

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

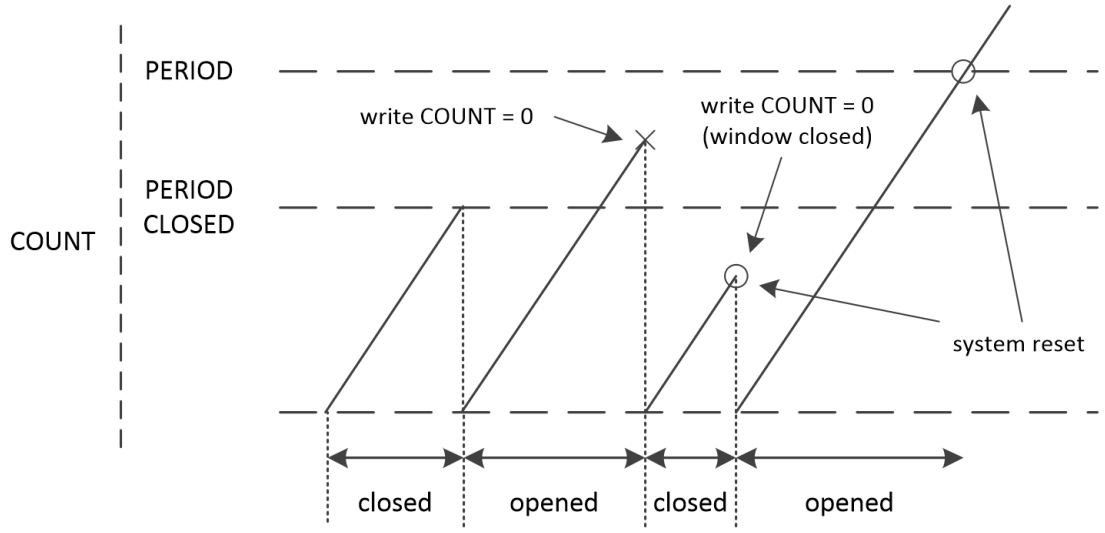


Figure 1.3. Watchdog Timer in WINDOW mode.

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
		...	...	...	...		
O	O	O	O	O	O	O	O
0	0	0	0	0	0	0	0
7	6	5	4	3	2	1	0
				PRES_UNL	PERCL_UNL	PER_UNL	CTRL_UNL
O	O	O	O	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

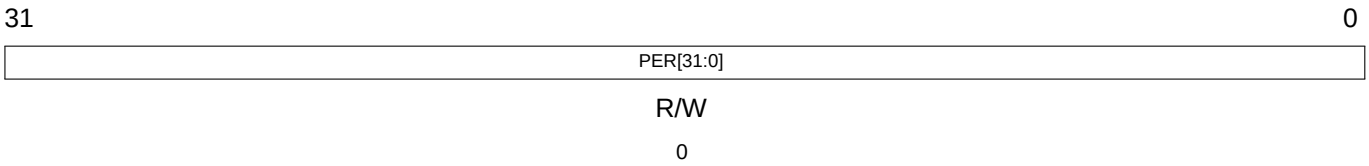
31	0
COUNT[31:0]	
R/W	
0	

COUNT[31:0] *Current Count*

Current count value.

1.3.5 Period Register

Address: 0x0C

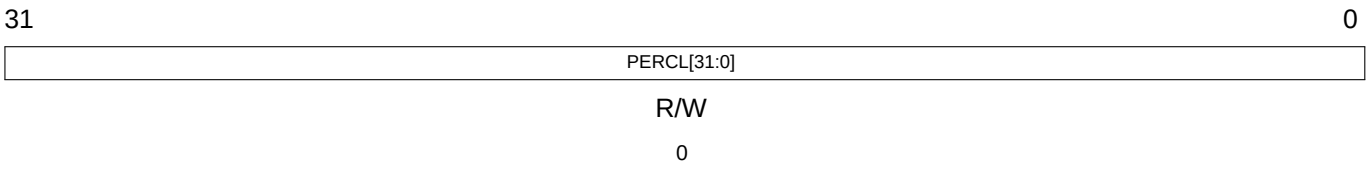


PER[31:0] *Period Value*

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

1.3.6 Period Closed Register

Address: 0x10

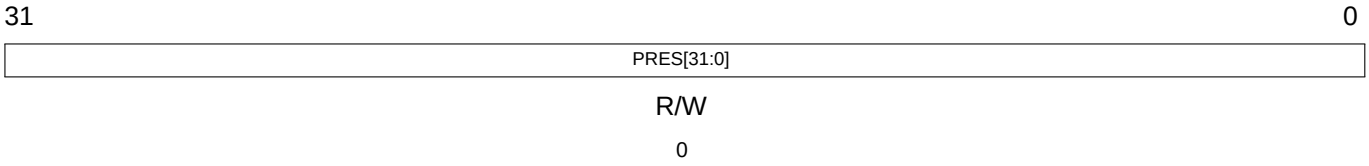


PERCL[31:0] *Period Closed Value*

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 (*AXI/rtl* and *WATCHDOG/rtl* folder) utilizes Verilog 2005 HDL. The testbench part (*AXI/tb* and *WATCHDOG/tb* folder) uses SystemVerilog language.

```
AXI
├── rtl
│   └── AXI_PERIPH
│       └── amba_axilite_apb_bridge.v
└── tb
    └── AXI_PERIPH
        ├── APB
        │   └── virtual_APB_slave.sv
        ├── AXI
        │   └── tb_amba_axilite_tasks.sv
        ├── common
        │   └── timescale.v
        ├── run
        │   └── ncvlog_amba_axilite_apb_bridge.sh
        ├── tests
        │   └── tb_read_write_test.sv
        └── tb_amba_axilite_apb_bridge.sv
WATCHDOG
├── beh
├── rtl
│   ├── APB_WATCHDOG.v
│   ├── AXILITE_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. The testbench is divided into two environments. The first one tests the APB_WATCHDOG module. To run the simulation using Cadence® Incisive® Enterprise Simulator run *ncvlog_apb_watchdog.sh* script located in the *WATCHDOG/tb/run* folder. The simulation should end with reporting no errors. The second environment tests the AXI4-Lite to APB3 converter. To run the simulation using Cadence® Incisive® Enterprise Simulator run *ncvlog_amba_axilite_apb_bridge.sh* script located in the *AXI/tb/AXI_PERIPH/run* folder. The simulation should end with reporting no errors. The AXILITE_WATCHDOG top module is composed of the APB_WATCHDOG core and the amba_axilite_apb_bridge AXI4-Lite to APB3 converter.

1.4.3 Clock and Reset

The CC-WDT-AXI 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.

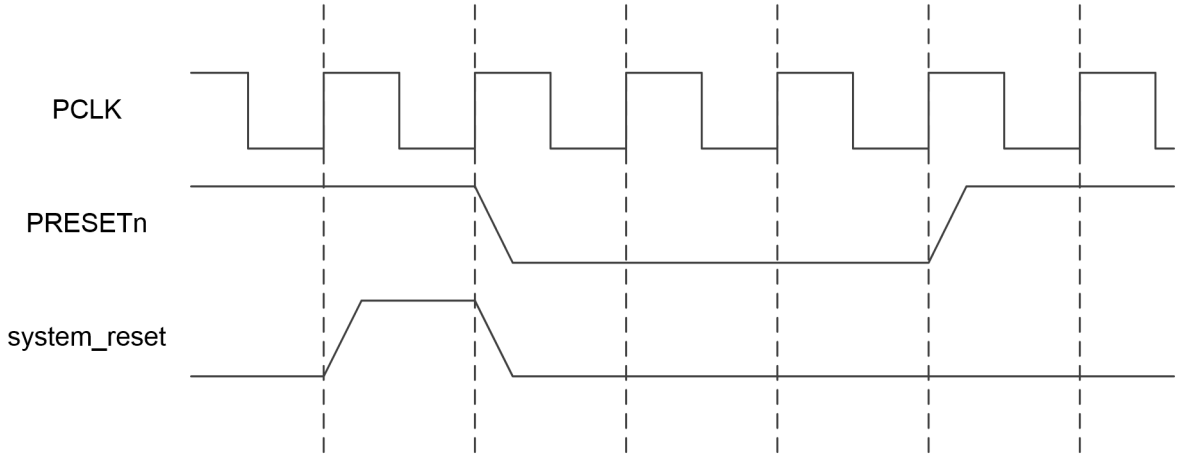


Figure 1.4. Watchdog timeout system reset.

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.

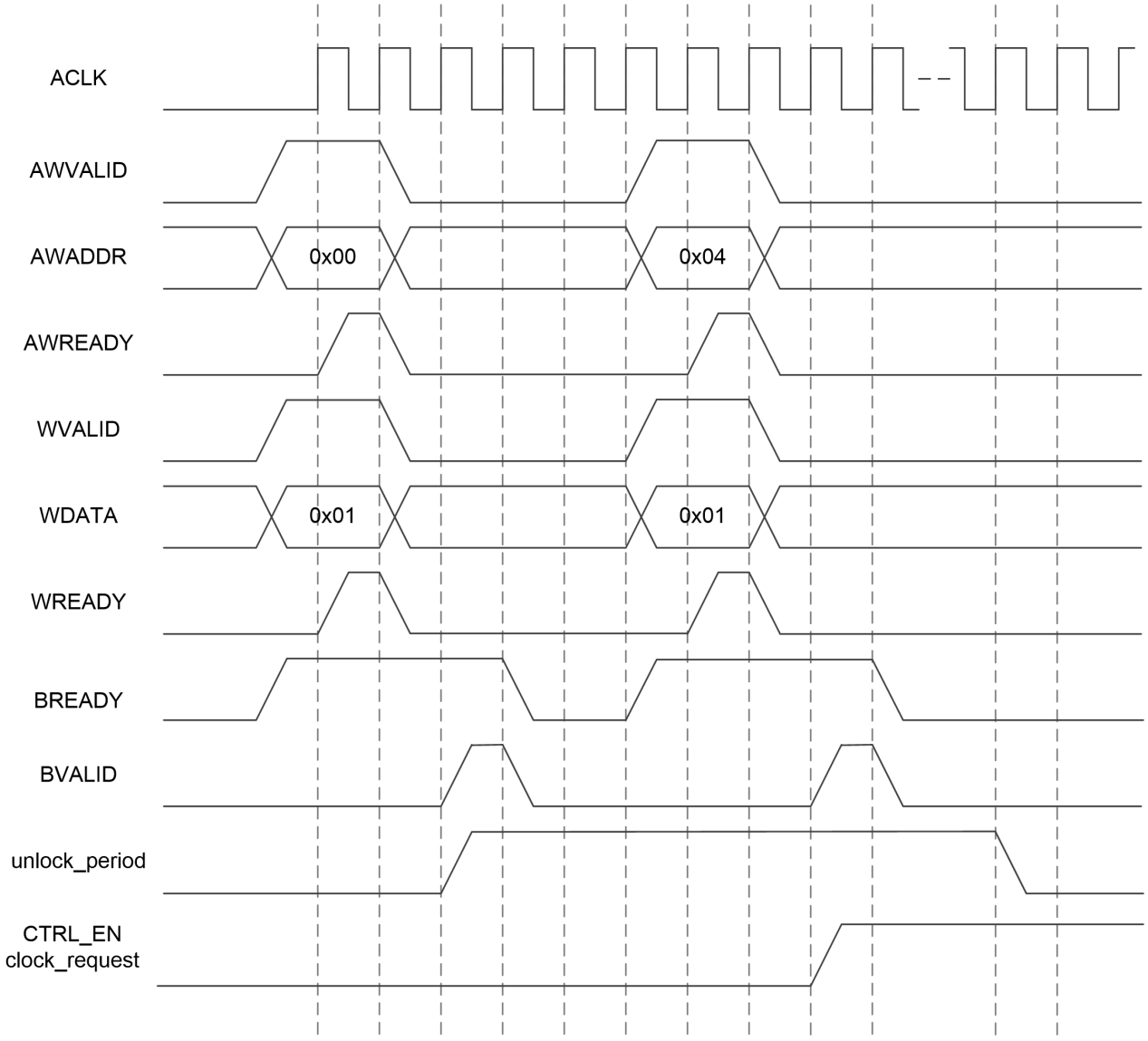


Figure 1.5. Watchdog clock enable procedure.

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
ACLK	Synchronous clock	I	rising	clock
ARESETn	Asynchronous reset	I	low	reset
AWADDR[4:2]	AXI4-Lite write address	I	data	comb.
AWPROT[2:0][1]	AXI4-Lite write address protection type	I	data	comb.
AWVALID	AXI4-Lite write address valid	I	high	comb.
AWREADY	AXI4-Lite write address ready	O	high	reg.
WDATA[31:0]	AXI4-Lite write data	I	data	comb.
WSTRB[3:0]	AXI4-Lite write strobe	I	high	comb.
WVALID	AXI4-Lite write valid	I	high	comb.
WREADY	AXI4-Lite write ready	O	high	reg.
BRESP[1:0]	AXI4-Lite write response	O	data	reg.
BVALID	AXI4-Lite write response valid	O	high	reg.
BREADY	AXI4-Lite write response ready	I	high	comb.
ARADDR[4:2]	AXI4-Lite read address	I	data	comb.
ARPROT[2:0][1]	AXI4-Lite read address protection type	I	data	comb.
ARVALID	AXI4-Lite read address valid	I	high	comb.
ARREADY	AXI4-Lite read address ready	O	high	reg.
RDATA[31:0]	AXI4-Lite read data	O	data	reg.
RRESP[1:0]	AXI4-Lite read response	O	data	reg.
RVALID	AXI4-Lite read valid	O	high	reg.
RREADY	AXI4-Lite read ready	I	high	comb.
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] Signal is not used in the design.

1.4.7 Instantiation

```
rstgen            #( .WIDTH(3))
    rstgen_u(         .clk(ACLK),
                      .rst(rst),
                      .scan_mode(scan_mode),
                      .in(~system_reset),
                      .out(ARESETn));

icg
    icg_wdt_u      ( .E(ARVALID|AWVALID|wdt_clock_request),
                      .clk(ACLK),
                      .gclk(wdt_clk),
                      .scan_enable(scan_enable));

AXILITE_WATCHDOG  #( .watchdog_width(32),
                      .prescaler_width(8))

AXILITE_WATCHDOG_u ( .ACLK(wdt_clk),
                      .ARESETn(ARESETn),
                      .AWADDR(AWADDR[4:2]),
                      .AWPROT(AWPROT),
                      .AWVALID(AWVALID),
                      .AWREADY(AWREADY),
                      .WDATA(WDATA),
                      .WSTRB(WSTRB),
                      .WVALID(WVALID),
                      .WREADY(WREADY),
                      .BRESP(BRESP),
                      .BVALID(BVALID),
                      .BREADY(BREADY),
                      .ARADDR(ARADDR[4:2]),
                      .ARPROT(ARPROT),
                      .ARVALID(ARVALID),
                      .ARREADY(ARREADY),
                      .RDATA(RDATA),
                      .RRESP(RRESP),
                      .RVALID(RVALID),
                      .RREADY(RREADY),
                      .system_reset(system_reset),
                      .debug_mode(debug_mode),
```


```
    .clock_request(wdt_clock_request));
```

1.5 Revision History

Doc. Rev.	Date	Comments
1.0	11-2017	First Issue.

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

 CC-WDT-AXI-Doc_112017.