ISSN: 2278 – 7798
*International Journal of Science, Engineering and Technology Research (IJSETR)*
*Volume 5, Issue 11, November 2016*

# Connectivity verification of Processor subsystem of ZynqUltraScale+ MPSoC

**A.RAJANI, ASSISTANT PROFESSOR, JNTUK**

**P.VIJITHA, MTECH VLSI DESIGN, JNTUK**

***Abstract*—verification is very important to validate every design and the primary objective of the verification of a design is to check the correctness and performance of the Register-transfer level design against the specification.The aim of the project is to check the correctness and performance of the design and checking the connectivity of processor subsystem contains low power domain and full power domain.Verification is categorized into the following: "Whole system bring up" which includes booting of each processor and able to do some basic transfers to local memory i.e. Clock settings and some basic operations at system level. "Connectivity checks" which include the basic transaction to all modules at system level and toggle all ports. "Basic Functional Verification" which includes, Data Flow, Data Integrity through the system, address decoding etc., of each module at system level.Here Verification is for the WDT and TTC blocks using advanced extensible interface (AXI) bus.**

***Index Terms*—Specification, Functional Verification,power domain,performance, Connectivity checks.**

## INTRODUCTION

The ZynqUltraScale+ MPSoc is the combination of Field-programmable gate array and Application-specific integrated circuit. The ASIC part known as Processor sub system(PS) and FPGA part known as Programming logic(PL). Project is developed to do connectivity verification for Triple timer counter and Watchdog timer blocks in the ASIC part (Processor sub system).

The processor subsystem contains sub blocks like APU (Application Processing Unit), GPU (Graphics Processing Unit), RPU (Real time Processing Unit), I/O peripherals, and High Speed Bus interfaces for the data transfer between Processor sub system and Programming Logic and within blocks.

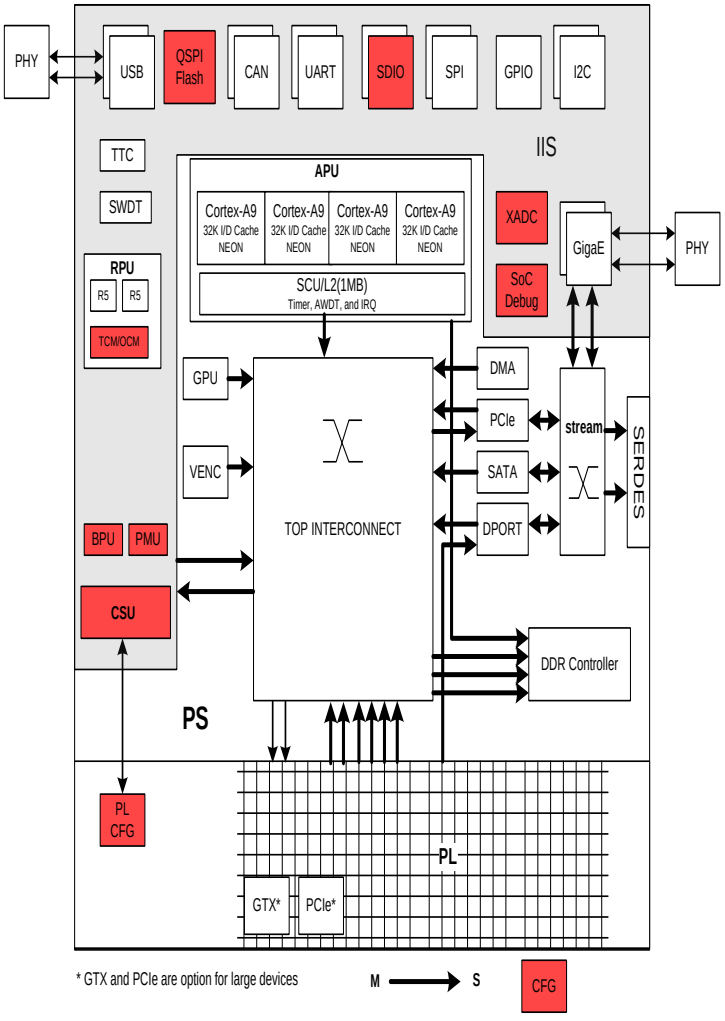



Fig1: Design Architecture

# METHODOLOGY:

### C-BASED SOC VERIFICATION ENVIRONMENT:

The test code will be developed for the required scenario (or functionality) and saved in the on-chip memory(OCM). The CPU executes the instructions from memory, this finally performs register reads and writes to the Design under test (DUT). Checks for register and data correctness are embedded inside the testcase to validate the IP behavior when it is integrated with the full system.

A generic SoC verification testbench is illustrated in Figure. Here Bus Matrix is developed with Advanced Extensible Interface (AXI) and Advanced Peripheral Bus (APB) protocols.

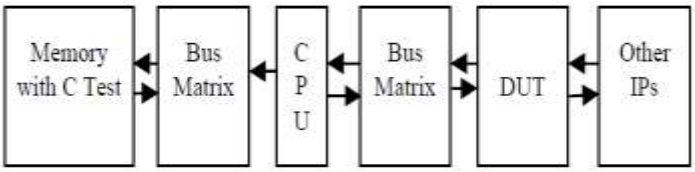



Fig2: Soc Verification testbench Architecture

Project to check the connectivity of WDT and TTC blocksin the Design. Here every module in the design contains registers. Hence the verification is done by performing the read and write operations from one register to another.

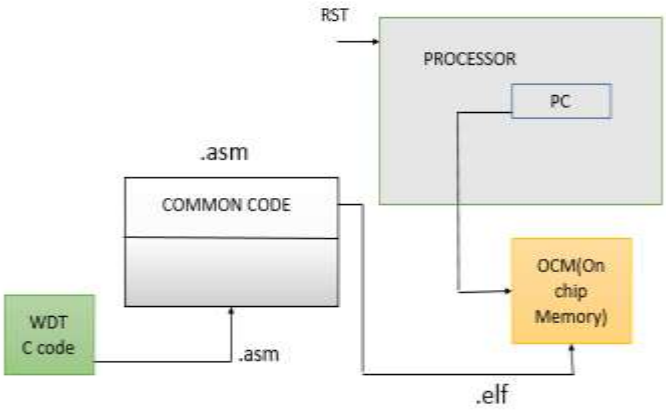



Fig3: Initial booting Process

In the initial booting process the basic testcode is append to the common code in the .asm format. And this code will be run on the Particular processor by loading code into the Boot Rom of every processor(For initial Booting) in the .elf format.

In a SoC Design it contains large number of blocks. For the communication of the each block we use Bus Matrix. In this design it is developed with protocols like AXI and APB.

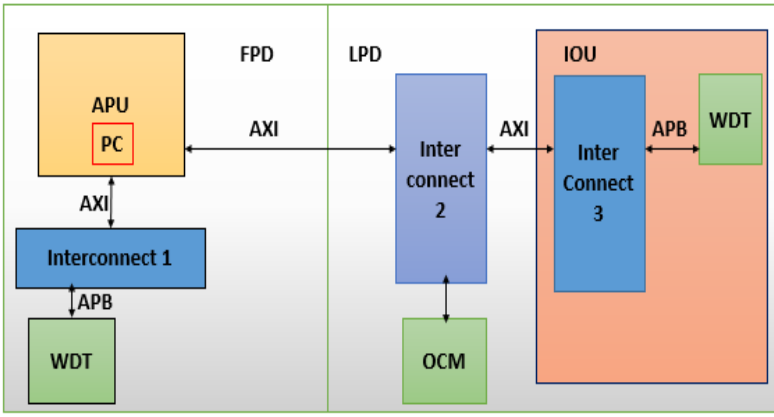



Fig4: Data Transfer Through interconnections

## Verification Blocks:

### WDT:

WDT(Watch dog timer) module to be programmed as part of a SoC design.The WDT can be used to prevent system lockup, for example if software becomes trapped ina deadlock. In normal operation the user restarts the watchdog at regular intervals before the timercounts down to zero.

If the timer does reach zero and the watchdog is enabled, one or a combination of the following signals is generated: a system reset, an interrupt or an external signal.

The watchdog time-out period and the duration of any output signals are variable.

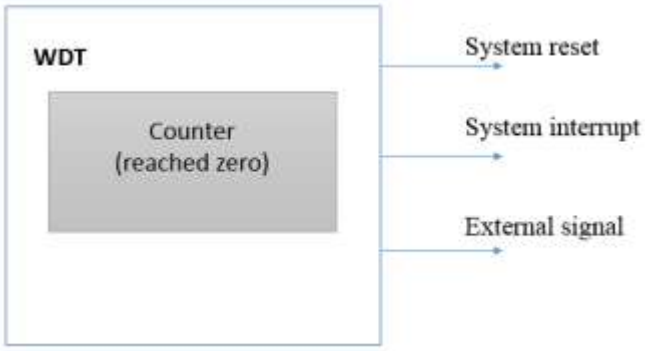



Fig5: WDT Block with generated interrupts

**TTC (TRIPLE TIMER COUNTER):**

The TTC module provides three independent timer/counter modules that can each be clocked. Counters can be set to decrement or increment. An interrupt is generated if the Counter overflows. Each of the counters can be programmed to generate interrupt pulses:

· At a regular, predefined period, that is on a timed interval

· When the counter registers overflow

· When the count matches any one of three 'match' registers

Therefore, up to six different events can trigger a timer counter interrupt. There are three match interrupts, an overflow interrupt, an interval interrupt. Note that the overflow interrupt and interval interrupt are mutually exclusive.

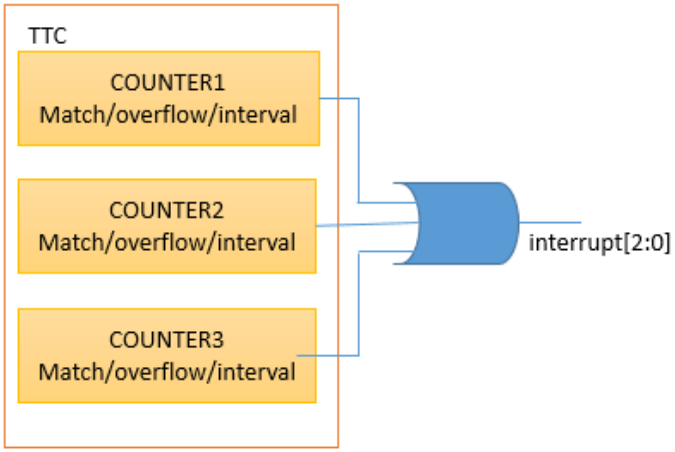



Fig6: TTC Block with generated interrupts

## INTERRUPT SYSTEM:

An **interrupt** is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing. The processor responds by suspending its current activities, saving its state, and executing a function called an *interrupt handler* (or an interrupt service routine, ISR) to deal with the event. This interruption is temporary, and, after the interrupt handler finishes, the processor resumes normal activities.

Generating interrupt in TTC/WDT prevents from generating system reset when fault condition occurs.

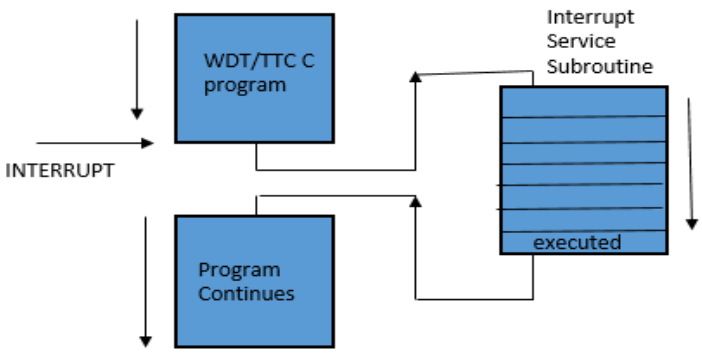
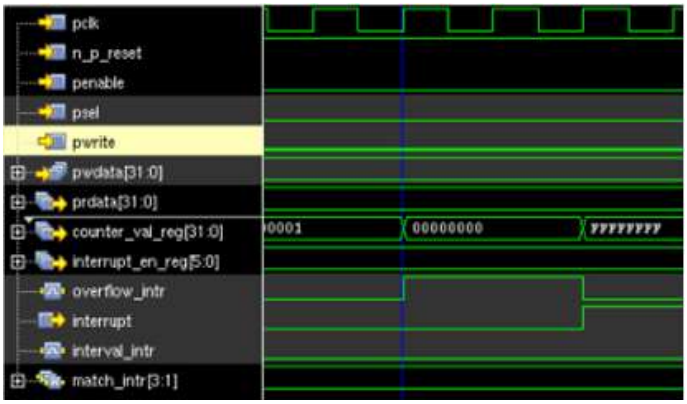


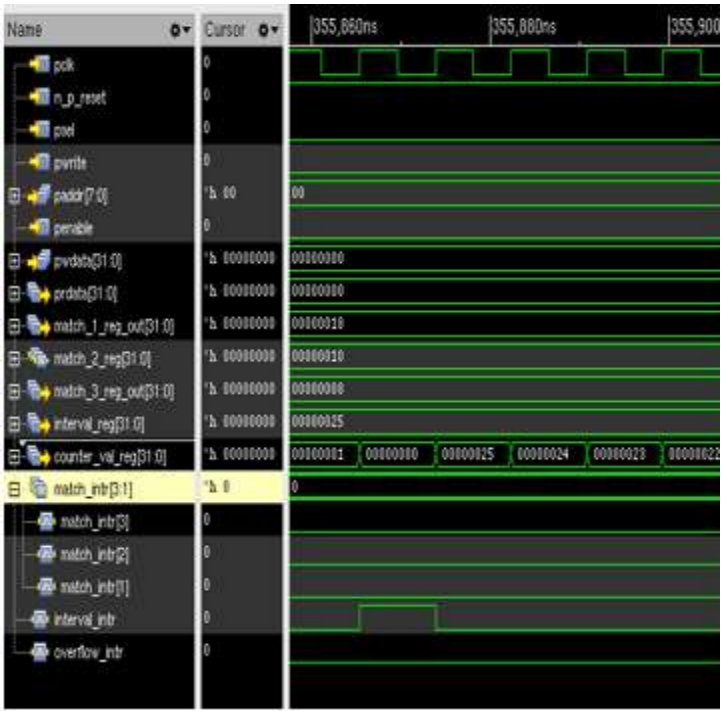

Fig7: Interrupt Handling

## RESULTS:

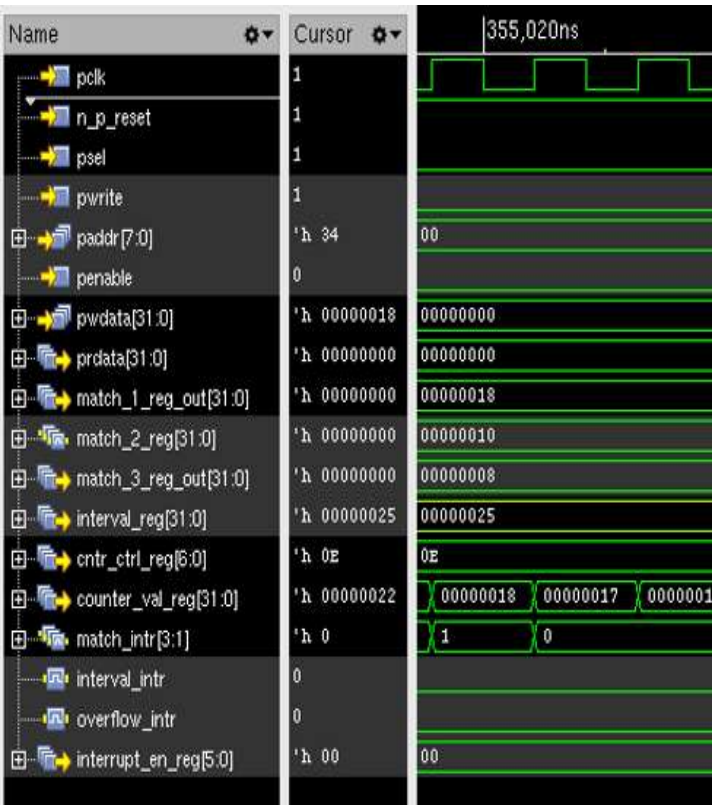
**Overflow interrupt:**



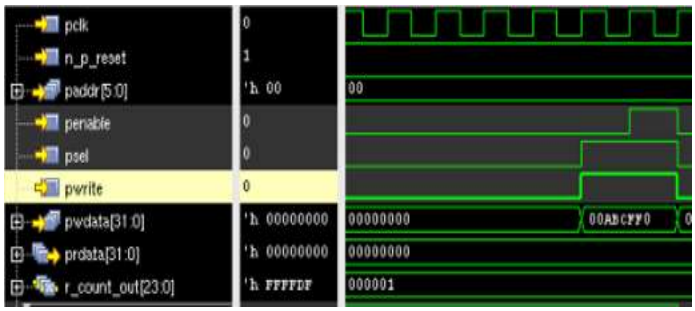
**For Interval interrupt:**




**Match interrupt:**




**Register write using Advanced Peripheral Bus protocol:**




# CONCLUSION:

We have presented the basic functional verification of TTC and WDT blocks with performing write and read operations (connectivity checks) to the appropriate registers by using the advanced extensible interface (AXI) in the whole architecture.

# REFERENCES:


- http://www.accellera.org/images/downloads/standards/uvm/uvm_users_guide_1.1.pdf
- http://www.xilinx.com/support/documentation/ip_documentation/ug761_axi_reference_guide.pdf
- http://www.xilinx.com/support/documentation/product-briefs/zynq-ultrascale-plus-product-brief.pdf
- https://www.xilinx.com/support/documentation/ip_documentation/axi_timebase_wdt/v3_0/pg128-axi-timebase-wdt.pdf