2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION JAPAN

Hardware/Software co-design and co-verification of embedded systems

Mayank Nigam

Nikita Gulliya

AGNISYS
SYSTEM DEVELOPMENT WITH CERTAINTY

accellera SYSTEMS INITIATIVE

Agenda

Introduction

- **Digital Peripheral Custom IP:**
  - Structure and Importance of Registers
- **Approach:**
  - Methodologies for Co-Design, Verification, and Validation

Description

- **SystemRDL Format:**
  - Register Specifications and Automated Generation
- **RTL Generation:**
  - HDL Code and Register Logic
- **Register Map:**
  - Address Definitions for Hardware-Software Interface
- **UVM Code:**
  - Verification Environment and Test Sequences
- **wr_data and rd_data Simulation:**
  - Simulation Process and Verification
- **C-API and C-Code**
  - Hardware-Software Interface and Firmware
- **Validation Code in Xilinx SDK Environment**
  - Firmware and Test Code Development in Xilinx SDK

Application

- **Example 1:**
  - Reuse of Validated Registers
- **Example 2:**
  - Receiver Block Diagram and Data Handling

Results

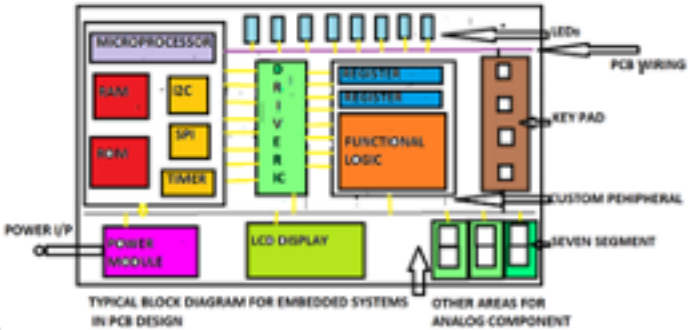
- Efficiency Improvements and Cycle Time Reduction

Conclusion

- Generalized Approach and Industry Impact

Introduction

- **Increasing Complexity:** Hardware chip complexity is on the rise.
- **Initiator-Target Architectures:** Chips used as targets, often custom peripheral IPs in Printed Circuit Boards for Embedded Systems
- **Functionality and Registers:** Provided in datasheets but lack programmer-focused validation.
- **Programmer's Perspective:**
  - Programmers interface Peripherals and write software to manage peripheral pins and internal Registers.
  - Real-world applications require tailored software.
- **Importance for Silicon Vendors:**
  - Need for a comprehensive approach in design, verification, and validation of peripheral IPs.
- **Methodologies:**
  - This paper presents methodologies for co-design, verification, and validation.
  - Demonstrated with detailed examples.



Digital Peripheral Custom IP

Two Main Sections:

- **Logic Circuits:** Designed based on specific functionality.
- **Registers:** Used for configuration and data exchange.

Registers' Role:

- Configure the IP.
- Send/receive data to/from the initiator and external devices.
- Act as inputs/outputs for internal functional circuits.
- Send data to output pins of the IP.

Programmers' Use:

- Use pin/port status to drive external devices in embedded systems.

Need for Co-Design and Co-Verification:

- Essential before releasing Custom complex IPs.
- Prevents escalation of non-recurring engineering (NRE) costs.
- Reduces failure rates in embedded systems.
- Critical for industries such as automotive, avionics, medical, and communication etc

Approach for Hardware-Software Co-Design and Co-Verification

Register Management:

- Programmers write data to IP's registers from the software side.
- Register values must align with expected functionality post-reset or default conditions.
- Correct values ensure IP functionality.

Critical Aspect:

- Accurate writing/reading from correct register after internal Functional behaviour of Custom IC through application programs in point of view of Programer

Methodology:

- **Prototypes:** Created for hardware-software interface.
- **Verification & Validation:**
  - HDL languages for RTL generation.
  - UVM for verification.
  - Embedded C for validation, independent of processors.

Functionality Testing:

- Tests include lock, shadow, aliasing, TMR, trigger buffer, RW pair, FIFO, counters, etc.
- Application in UART, I2S, GPIO, EEPROM, dual-port RAM, or custom IPs for embedded systems.

Components for Co-Design and Co-Verification

Four Crucial Components:

1. **Register Specification:** Defined in RDL format.
2. **RTL Generation:** Hardware description using HDL languages.
3. **Register Map:** Header files for register addresses and definitions.
4. **Verification Environment:**
    - UVM code for verification.
5. **Validation Environment:**
    - C-API and C-code for testing and validation.

Comprehensive Process: Ensures IP functions correctly within the embedded domain.

SystemRDL Format and RTL Generation

SystemRDL Format:

- **Purpose:** Describes register structures, behavior, and constraints.
- **Benefits:** Automated RTL generation, documentation, and software drivers.
- Example Syntax:

```
addrmap block_name {
    reg {
        field {
            hw=rw;
            sw=rw;
        } fld[31:0];
        regwidth=32;
    } reg_name;
};:
```

RTL Generation:

- **Description:** Written in HDL (VHDL/Verilog) to define circuit logic.
- Example Code

```
    always @(posedge clk) begin
    if (!reset_l) begin
        reg_name_fld_q <= 32'bx;
    end else begin
        if (reg_name_fld_in_enb) begin
            reg_name_fld_q <=
reg_name_fld_in;
        end else begin
            if (reg_name_wr_valid) begin
                reg_name_fld_q <= (wr_data
[31 : 0] & reg_enb [31 : 0]) |
(reg_name_fld_q & (~reg_enb [31 : 0]));
            end
        end
    end
end
```

Register Map:
Purpose: Defines register addresses for hardware-software interface.

```
#define block_name_s_ADDRESS 0x0
#define block_name_reg_name_ADDRESS 0x0
#define BLOCK_NAME_REG_NAME_FLD_OFFSET 0
#define BLOCK_NAME_REG_NAME_FLD_MASK
0xFFFFFFFF
#define BLOCK_NAME_REG_NAME_FLD_DEFAULT 0
```

UVM Code:
Purpose: Provides high-level representation of registers for verification.

```
`ifndef CLASS_block_name_reg_name
`define CLASS_block_name_reg_name
class block_name_reg_name extends uvm_reg;
    `uvm_object_utils(block_name_reg_name)

    rand uvm_reg_field fld;
    function new(string name =
"block_name_reg_name");
        super.new(name, 32,
build_coverage(UVM_NO_COVERAGE));
    endfunction
    virtual function void build();
        this.fld =
uvm_reg_field::type_id::create("fld");
        this.fld.configure(.parent(this), .size(32),
.lsb_pos(0), .access("RW"), .volatile(0),
.reset(32'd0));
    endfunction
endclass
`endif
```

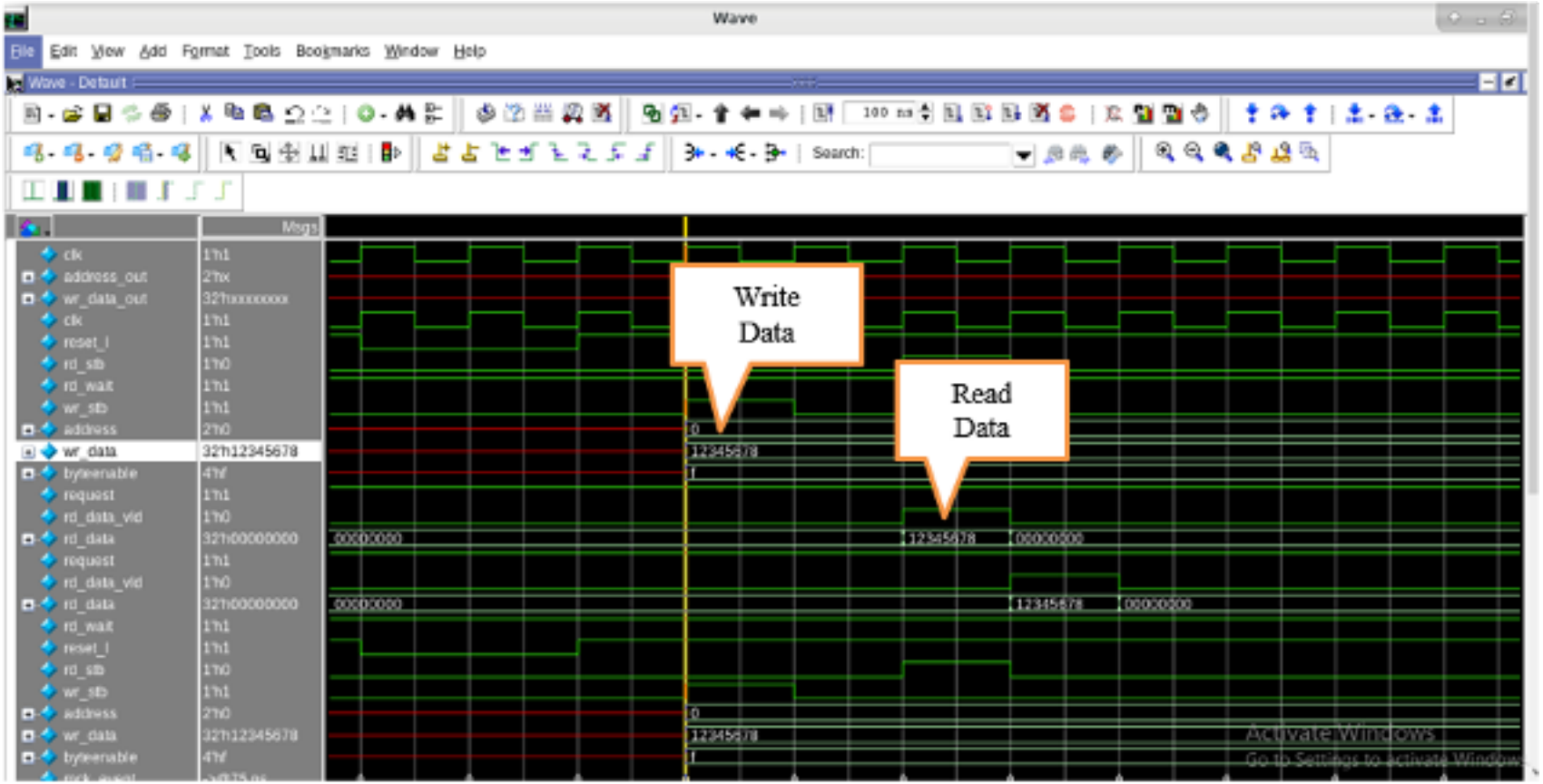
Simulation of Write and Read Operations

Verification Process:

- **Tool Used:** Questa-Sim for simulation.
- **Objective:** Validate write (wr_data) and read (rd_data) operations.

Simulation Details:

- **Testbenches:** Created to generate various data patterns.
- **Write Operations:** Data patterns are written to registers.
- **Read Operations:** Data is read back from registers.
- **Checks Performed:**
  - **Integrity:** Ensure data written can be accurately read back.
  - **Functionality:** Verify correctness of register operations.
  - **Sequences and Corner Cases:** Ensure all scenarios are handled properly.


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
clk 1'h1
address_out 2'hx
wr_data_out 32'hxxxxxxxx
clk 1'h1
reset_l 1'h1
rd_stb 1'h0
rd_wait 1'h1
wr_stb 1'h1
address 2'h0
wr_data 32'h12345678
byteenable 4'hf
request 1'h1
rd_data_vld 1'h0
rd_data 32'h00000000
request 1'h1
rd_data_vld 1'h0
rd_data 32'h00000000
rd_wait 1'h1
reset_l 1'h1
rd_stb 1'h0
wr_stb 1'h1
address 2'h0
wr_data 32'h12345678
byteenable 4'hf
Write Data
Read Data
0
12345678
f
00000000
12345678
00000000
00000000
12345678
00000000
0
12345678
f
Activate Windows
Go to Settings to activate Windows.

C-API and C-Code:

- **Purpose:** Facilitates register interaction and validation.

```
#include "write_read.h"
#include "block_name_seq_name_iss.h"
int block_name_seq_name(int baseAddress) {
    int reg_name;
    REG_WRITE(BLOCK_NAME_REG_NAME_ADDRESS(baseAddress), 0x12345678);
    reg_name = REG_READ(BLOCK_NAME_REG_NAME_ADDRESS(baseAddress));
    return 0;
}
```

Validation in Xilinx SDK

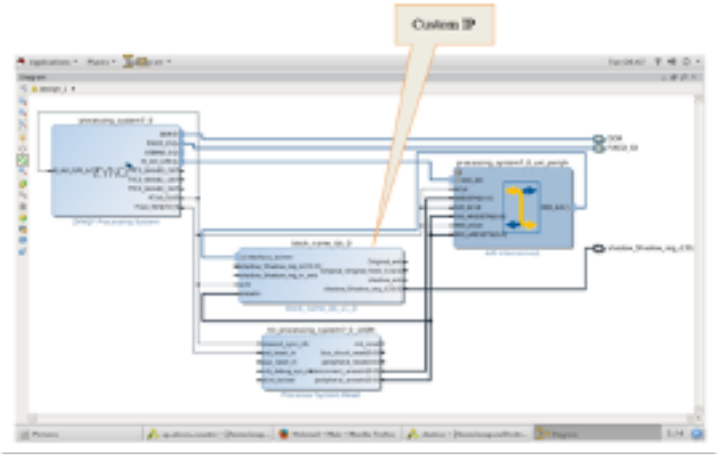
- **Validation Code Development:**
  - **Environment:** Xilinx Software Development Kit (SDK).
  - **Activities:**
    - **Writing Firmware:** Develop firmware for custom IPs.
    - **Test Code:** Create test code to verify functionality.
  - **Tools and Libraries:**
    - Provided by Xilinx SDK.
    - Facilitate development and debugging of software.
  - **Objective:**
    - Ensure custom peripherals operate correctly and efficiently within the target embedded system.

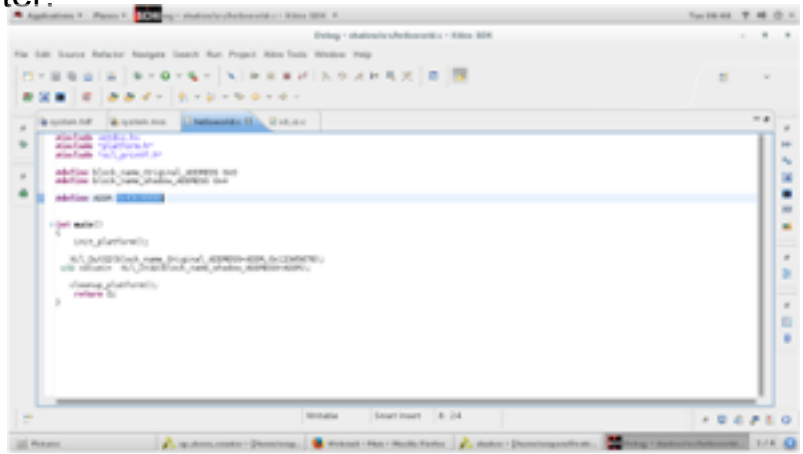
```c
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#define block_name_reg_name_ADDRESS  0x0
#define ADDR 0x43c00000
int main()
{
    init_platform();
   Xil_Out32(block_name_reg_name_ADDRESS+ADDR,0x12345678);
   u32 value1= Xil_In32(block_name_reg_name_ADDRESS+ADDR);
   cleanup_platform();
    return 0;
}
```

Examples of Register Validation

SHADOW Register:

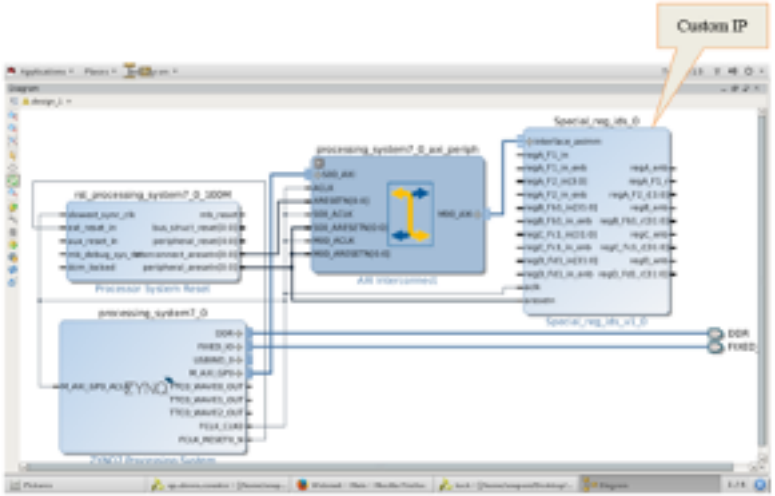
- **Vivado HW Design:**
  - SHADOW REGISTER IP with Zynq processor.
- **SDK Sequences:**
  - C-Program validating SHADOW Register.

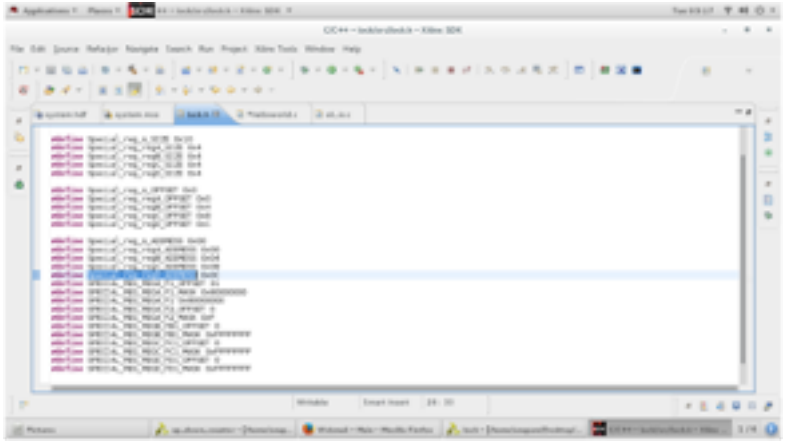


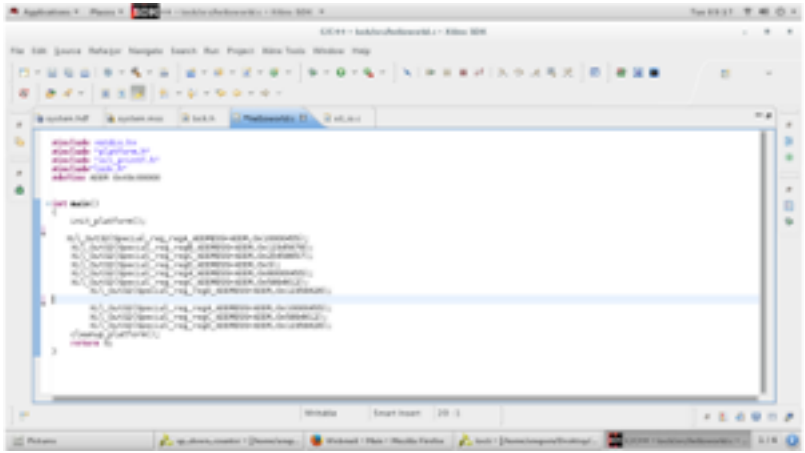


LOCK Register:

- **Vivado HW Design:**
  - LOCK REGISTER IP with Zynq processor.
- **Software SDK:**
  - C-Headers for LOCK Register validation.
  - C-Program validating LOCK Register.

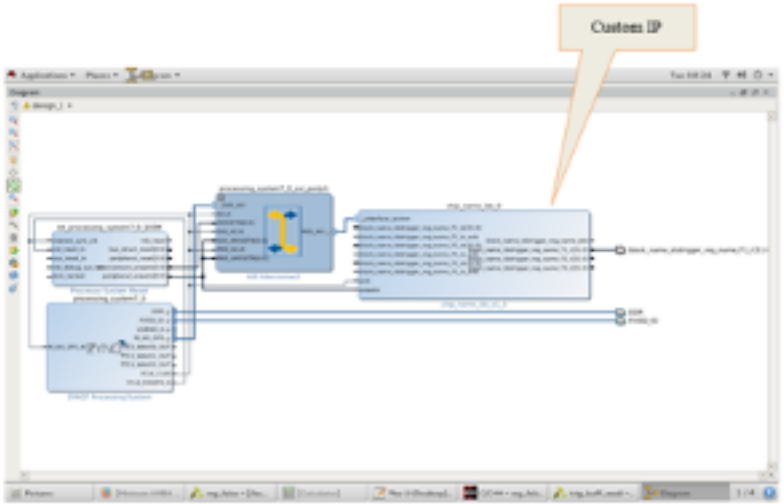


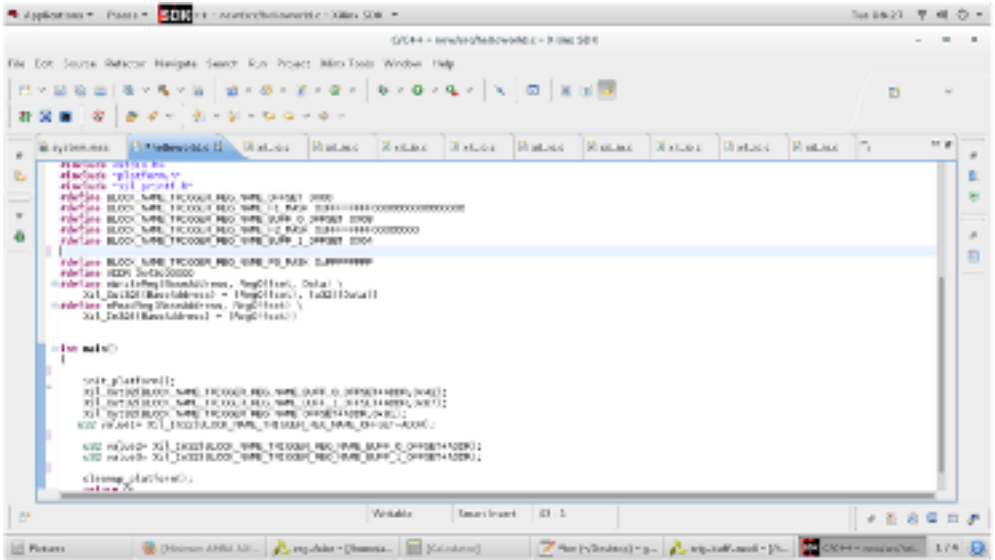




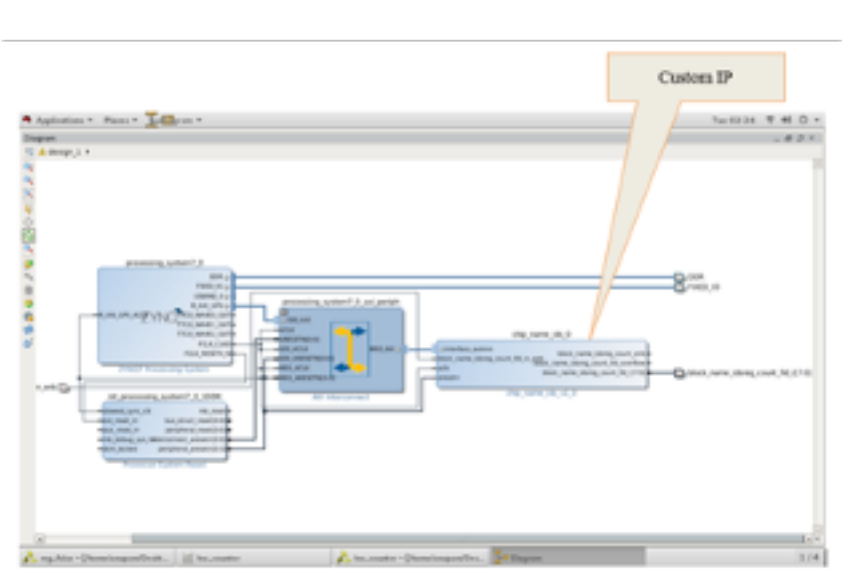
Trigger Buffer Register:

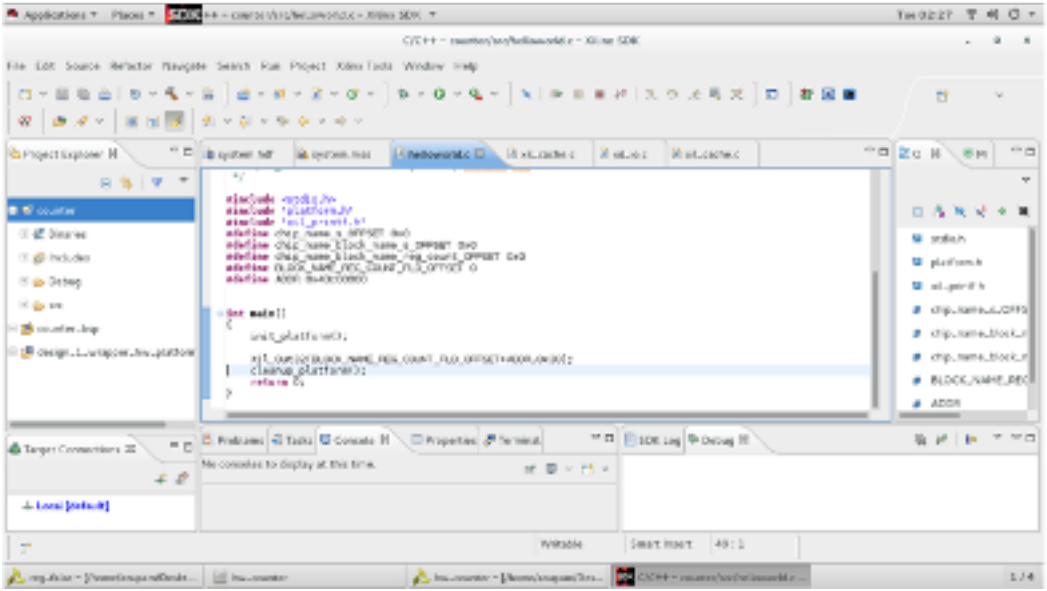
- **Vivado HW Design:**
  - TRIGGER BUFFER REGISTER IP with Zynq processor.
- **SDK Sequences:**
  - C-Program validating TRIGGER BUFFER Register.





- **Counters:**
  - **Vivado Design:**
    - COUNTER IP with Zynq processor.
  - **Software Embedded C Code:**
    - C-Program validating COUNTER.



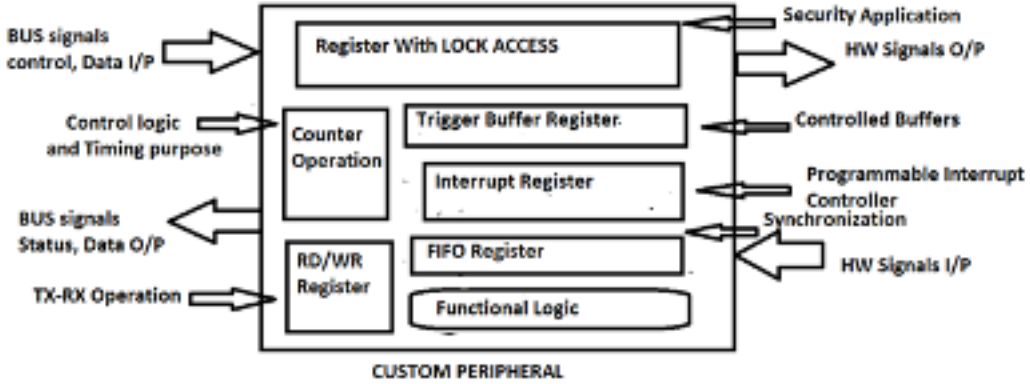


- **Additional Registers:**
  - Validation of Interrupt, FIFO, and Read/Write Registers.
  - Utilizes SystemRDL with UDP for design and Vivado for verification.

Applications of Validated Registers

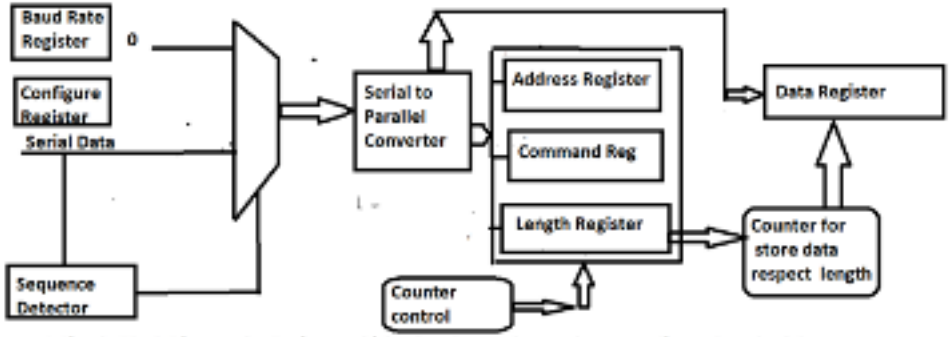
Example 1: Reuse of Validated Registers

- **Process:**
  - Validated registers are directly connected to functional logic.
  - Registers are reusable across different custom IPs.
  - Only functional blocks within the IP need testing.
  - Ensures minimal re-verification and faster time to market.
- **Advantage:** Streamlined design and verification/validation process.



Example 2: Receiver Block Diagram

- **Description:**
  - **Packet Structure:** Start-address-command-data-stop sequences.
  - **Functionality:**
    - Sequence detector identifies start conditions.
    - Data passed through multiplexer.
    - Counter control mechanism stores data in 8-bit segments.
    - Counter down-counts data length.
    - Programmer sets baud rate and configures registers.
  - **Component Reuse:** Verified registers and counters are reused, eliminating need for further verification.



A Simple Block Diagram for Reciever which takes data and store data according to length of data

Efficiency and Effectiveness of Co-Design and Co-Verification

Efficiency Improvement:

- **Cycle Time Reduction:** Up to 30%.
- **Advantages:**
  - Faster compared to traditional methodologies.
  - More cost-effective and compact.

Enhanced Productivity:

- Simultaneous result generation during testing.
- Improved reliability in both verification and validation environments.

Transformative Impact on Embedded Systems

Generalized Approach:

- Applies to any custom peripheral IP.

Benefits:

- Addresses embedded system challenges effectively.
- Integrates hardware and software co-design and co-verification.

Industry Impact:

- Sets a new standard for efficiency and effectiveness in development and deployment.
- Provides a transformative impact on the industry.

Questions