

hwdbg

Debugging Hardware Like Software

Mohammad Sina Karvandi*, Soroush Meghdadizanjani **, Saleh Khalaj Monfared ***,
Erik van der Kouwe *, Asia Slowinska *

* Vrije Universiteit Amsterdam (VUsec), The Netherlands

** Stony Brook University (SBU), United States

*** Worcester Polytechnic Institute (WPI), United States



**“To err is human,
to debug is divine.”**





Hardware Debugger

Debugging Hardware like Software



Key (must-have) Software Debugger Features



Memory

Reading or Writing Memory

Registers



Inspecting or Modifying
Registers Values



Breakpoint

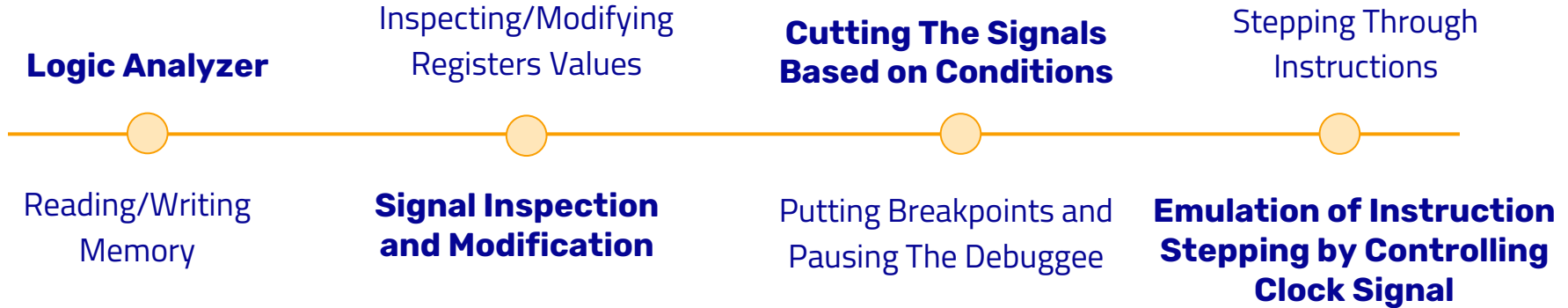
Putting Breakpoints and
Pausing The Debuggee

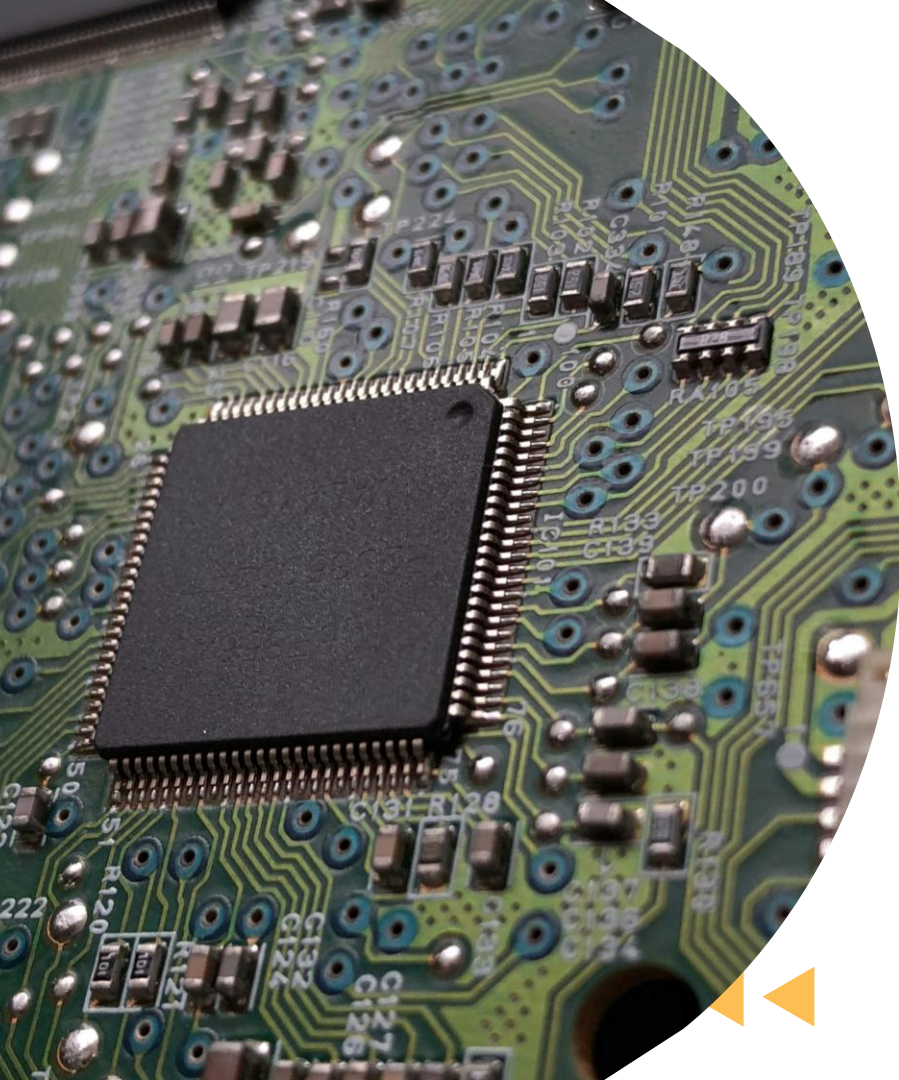
Stepping



Stepping Through Instructions

Mapping Software Concepts to Hardware





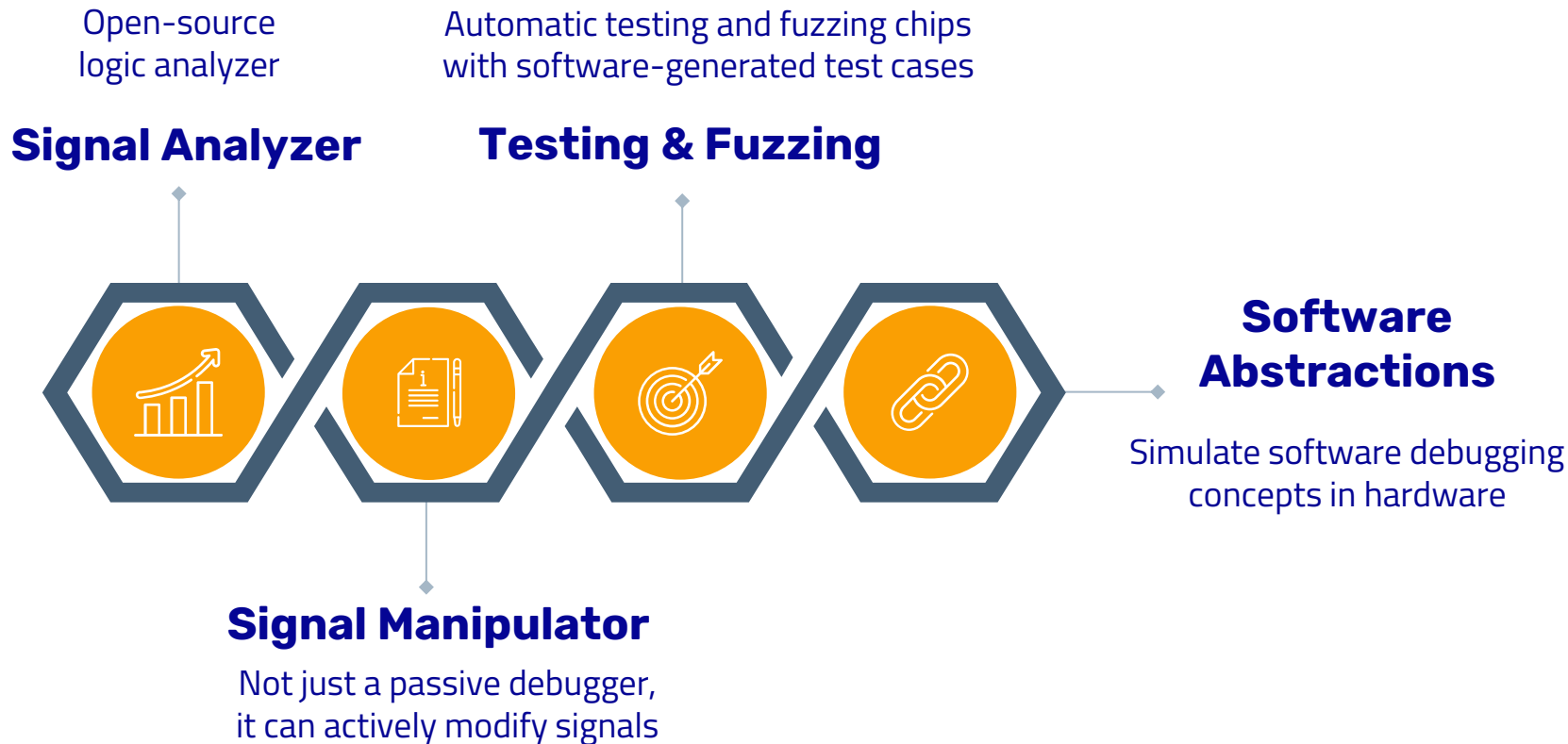
Key Intuition

- Allow security researchers and hardware engineers to run **custom actions**
- Run in smallest (shortest) event in digital circuit: **rising** and **falling** edge of the **clock cycle**



hwdbg is a debugger chip generator based on chisel language (scala)
& LLVM circt

Key Features



Why do we need hwdbg for security?

hardware side channels

**change clock
domain**

fault injection

fuzzing

cut signal

remove signal at
random stages

hwdbg

HyperDbg's chip-level debugger

using PWM with
different duty cycles

**reverse
engineering**



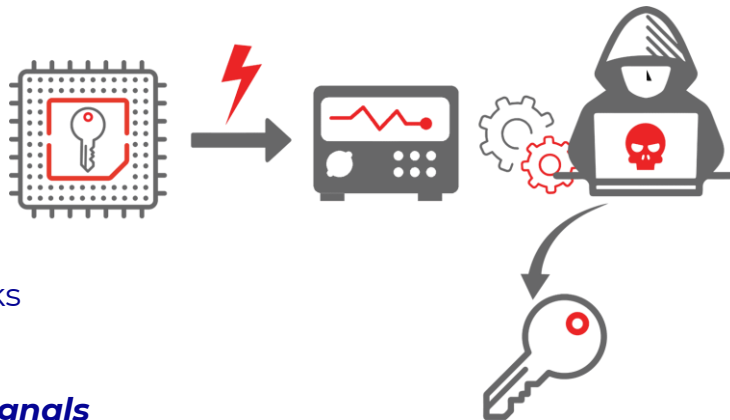
What are hardware side channels?

Side-channel attacks in chips and IP cores

- Exploit indirect information leakage (e.g., **power consumption, electromagnetic emissions**)
- Used to extract sensitive data

Detection using sensors

- Ring Oscillators (RO) are used for monitoring
- Detect variations in hardware characteristics
- Identify potential side-channel leakage or attacks



Hardware side channels often use debugging signals

Debugging Sensors



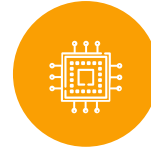
Ring Oscillator (RO)

Measures variations in signal propagation delay to detect changes in environmental conditions or circuit behavior



IDELAYE2/3

Provides adjustable signal delay capabilities for timing adjustments and alignment in high-speed circuits



Time-to-Digital Converter (TDC)

Converts time interval between two events into digital value for precise timing measurements

Debugging Sensors (cont.)



Voltage Sensor

Monitors and measures voltage levels within circuit to ensure stable and correct operation



Frequency & Clock Sensors

Monitors frequency and stability of clock signals to ensure they remain within expected range and performance criteria



Temperature Sensor

Measures temperature within chip to monitor thermal conditions and prevent overheating

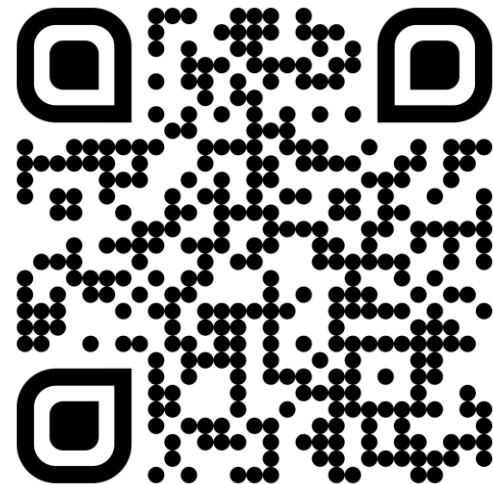


hwdbg is available at:

<https://url.hyperdbg.org/hwdbg-repo>

Documentation:

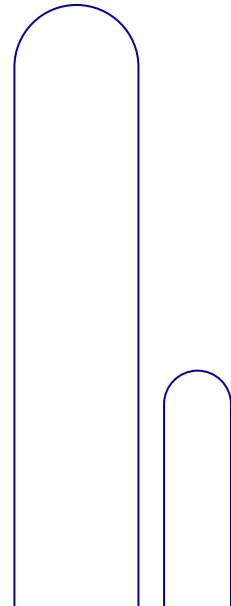
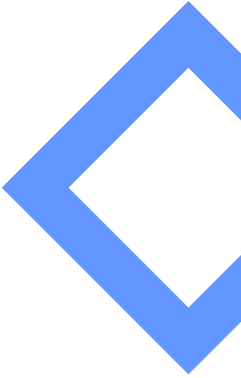
<https://hwdbg.hyperdbg.org/docs>





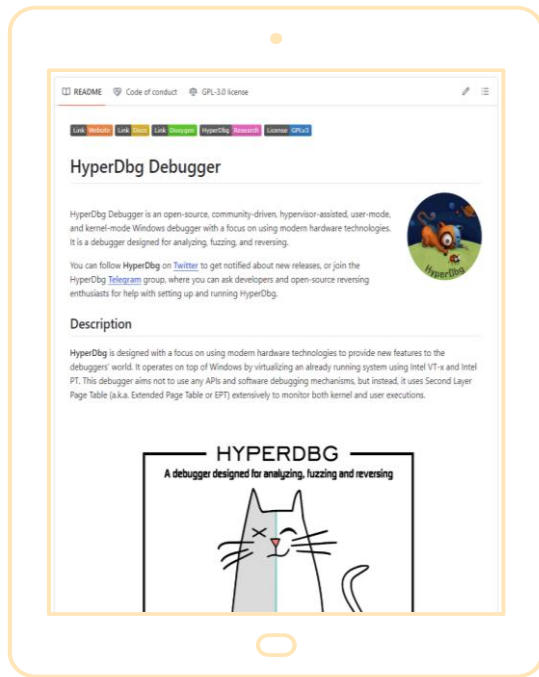
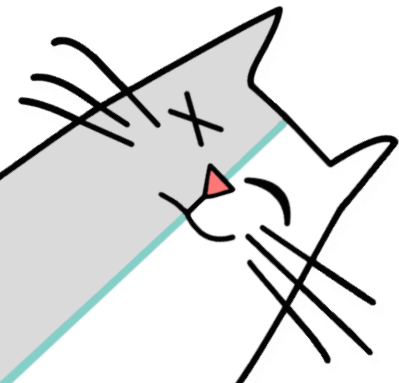
Backup!

Backup slides...



What is HyperDbg?

HyperDbg is an open-source platform that brings debugging infrastructures like hypervisor-assisted, user-mode, and kernel-mode debuggers mainly with a focus on using modern hardware technologies for **analyzing**, **fuzzing**, and **reversing**.



Key Contributions

hwdbg comes with these contributions.

Introduction of a Unified Debugging Framework

Open-source and Customizable Tool

Integration with FPGAs

Robust Debugging Experience

Enhancement of Reverse Engineering and Chip Fuzzing Capabilities

Comprehensive Scripting Support

01

Software

The software side is implemented in C/C++ for sending debugging commands and parsing scripts.

02

Hardware

The hardware side will be written in chisel programming language as well as Verilog and SystemVerilog.

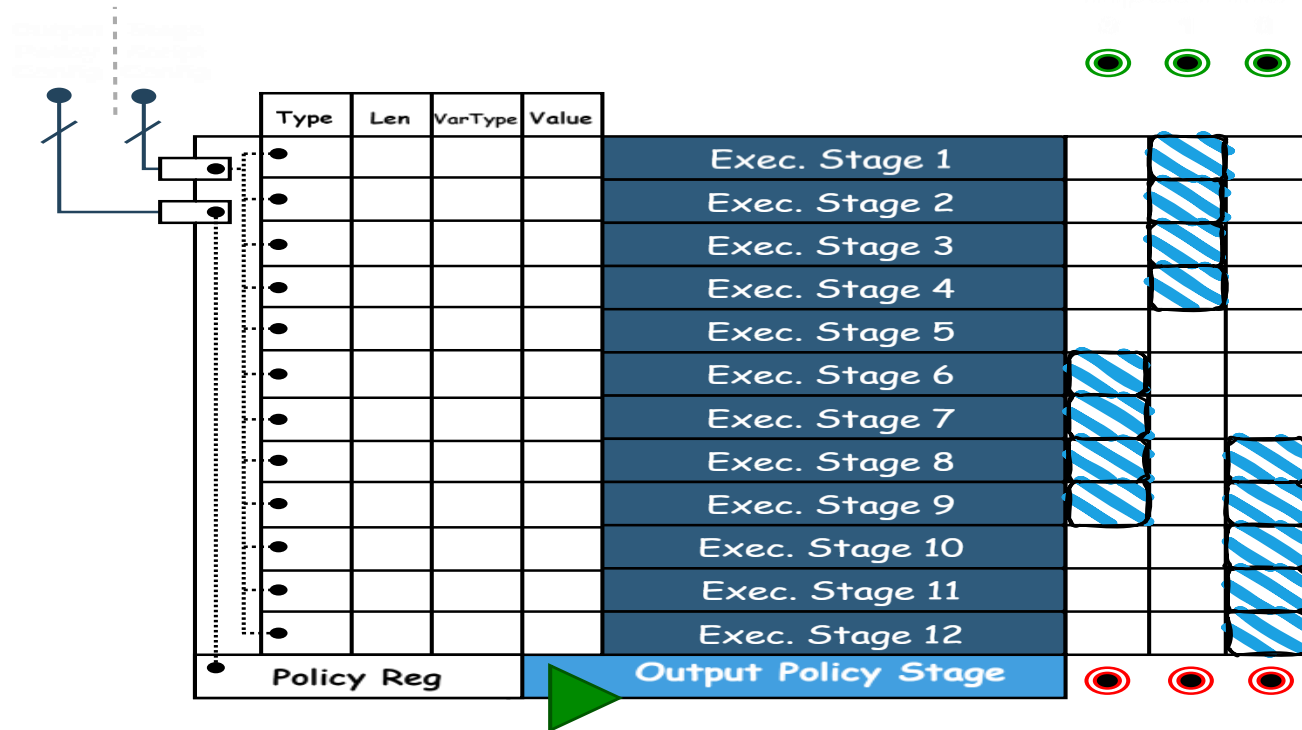
03

Platform

The testing platform is AMD Xilinx FPGAs.

Script Execution Stages

This picture demonstrates the execution of different IR scripts in each stage.



Debugging Modes

Passive debugging

In this debugging mode, **hwdbg** acts as an introspection tool, and monitors different signals to the target debugging module.

Active debugging

In this debugging mode, **hwdbg** is able to both monitor and manipulate signals (Input/Output). These signal modifications are configured by the user's custom scripts.

Communication Modules

Sending module is responsible for preparing mandatory headers and write to BRAM as well as interrupting PS.

–Sending Module

Once share PS <> PL line is high, receiving verifies BRAM data headers and notifies interpreter.

–Receiving Module

Because only one BRAM port is shared with PL, synchronization is needed to avoid data corruption in case of simultaneous writes.

–Send/Receive Synchronization Module

Once a valid packet is received, the interpreting module configures the chips and sends the response to the debugger.

–Interpreting Module

Physical Unclonable Function (PUF)



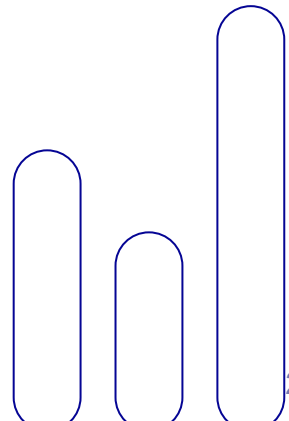
What is PUF?

A PUF is a **physical entity** utilized in microchips and other electronic devices to generate **unique, device-specific cryptographic keys** based on the uncontrollable physical variations that occur during the manufacturing process.



What is special about PUFs?

These **unique differences**, such as variations in silicon pathways, ensure that each PUF is unique and make it theoretically **impossible** to **duplicate** or **clone**.



BRAM Simulator

Here is the BRAM simulator (PS to PL area).

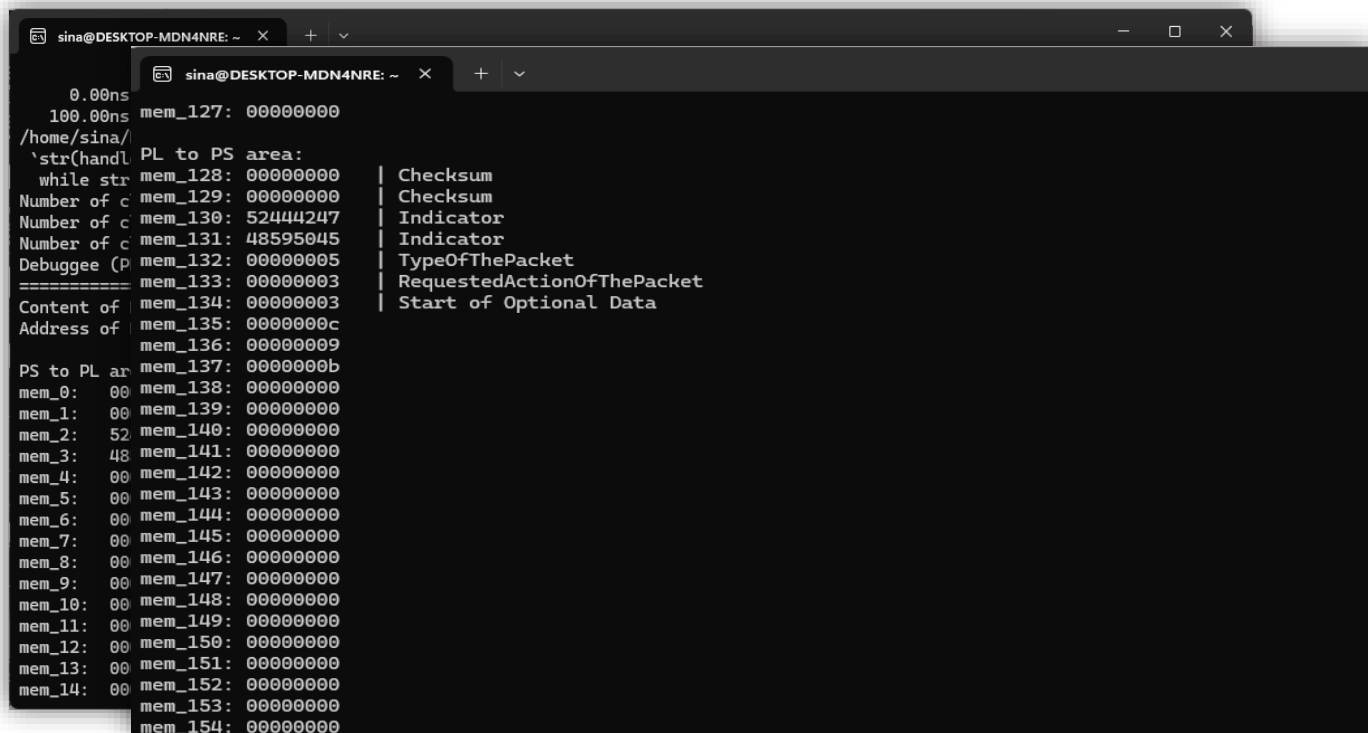
```
sina@DESKTOP-MDN4NRE: ~ X + v

Test hwdbg module (with pre-defined BRAM)
0.00ns INFO cocotb.DebuggerModuleTestingBRAM Initialize and reset module
100.00ns INFO cocotb.DebuggerModuleTestingBRAM Enabling an interrupting chip to receive commands from BRAM
/home/sina/HyperDbg/hwdbg/sim/hwdbg/DebuggerModuleTestingBRAM/test_DebuggerModuleTestingBRAM.py:354: DeprecationWarning:
'str(handle)' casts have been deprecated. Use 'str(handle.value)' instead.
while str(dut.io_psOutInterrupt) != "1":
Number of clock cycles spent in debuggee (PL): 0
Number of clock cycles spent in debuggee (PL): 10
Number of clock cycles spent in debuggee (PL): 20
Debuggee (PL) interrupted Debugger (PS)
=====
Content of BRAM after emulation:
Address of PL to PS communication: mem_128

PS to PL area:
mem_0: 00000000 | Checksum
mem_1: 00000000 | Checksum
mem_2: 52444247 | Indicator
mem_3: 48595045 | Indicator
mem_4: 00000004 | TypeOfThePacket
mem_5: 00000002 | RequestedActionOfThePacket
mem_6: 00000000 | Start of Optional Data
mem_7: 00000000
mem_8: 00000000
mem_9: 00000000
mem_10: 00000000
mem_11: 00000000
mem_12: 00000000
mem_13: 00000000
mem_14: 00000000
```

BRAM Simulator (cont.)

Here is the BRAM simulator (PL to PS area).



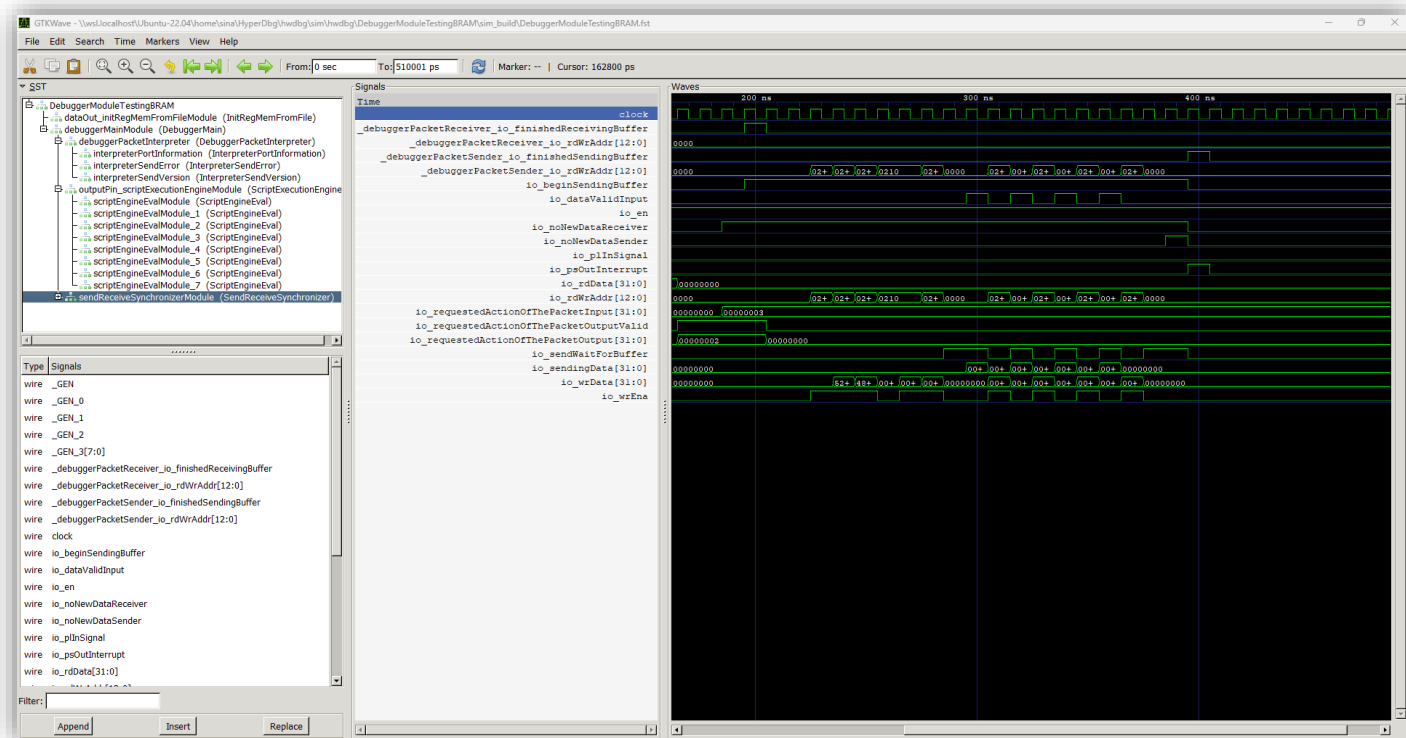
```

sina@DESKTOP-MDN4NRE: ~
0.00ns
100.00ns mem_127: 00000000
/home/sina/
`str(handl
while str
Number of c mem_128: 00000000 | Checksum
Number of c mem_129: 00000000 | Checksum
Number of c mem_130: 52444247 | Indicator
Number of c mem_131: 48595045 | Indicator
Debuggee (P mem_132: 00000005 | TypeOfThePacket
===== mem_133: 00000003 | RequestedActionOfThePacket
Content of | mem_134: 00000003 | Start of Optional Data
Address of | mem_135: 0000000c
mem_136: 00000009
PS to PL ar mem_137: 0000000b
mem_0: 00 mem_138: 00000000
mem_1: 00 mem_139: 00000000
mem_2: 52 mem_140: 00000000
mem_3: 48 mem_141: 00000000
mem_4: 00 mem_142: 00000000
mem_5: 00 mem_143: 00000000
mem_6: 00 mem_144: 00000000
mem_7: 00 mem_145: 00000000
mem_8: 00 mem_146: 00000000
mem_9: 00 mem_147: 00000000
mem_10: 00 mem_148: 00000000
mem_11: 00 mem_149: 00000000
mem_12: 00 mem_150: 00000000
mem_13: 00 mem_151: 00000000
mem_14: 00 mem_152: 00000000
mem_153: 00000000
mem_154: 00000000

```

BRAM Simulator Waves

Here is the BRAM simulator wave results for **hwdbg**.



Anomaly Definition

Anomalies In Expected Functionalities

These occur when the device fails to compute the correct output due to specific inputs, detectable through emulation.

Anomalies Based On Sensors Indications

Sensors can indicate anomalies through abnormal state changes, signaling potential internal issues in the chip.

Supported Operators

Precedence	Operators
Parentheses	()
Unary Operators	-, +, ~, &, *
Arithmetic Operators	*, /, %
Arithmetic Operators	+, -
Shift Operators	>>, <<
Comparison Operators	>=, <=, >, ==, !=
Bitwise AND Operator	&
Bitwise XOR Operator	^
Logical AND Operator	&&
Logical OR Operator	

dslang vs TSM

	hwdbg (dslang)	Xilinx ILA (TSM)
Can act as a logic analyzer	Yes	Yes
Speed comparison	Slower	Faster
Can trigger an event (e.g., a breakpoint)	Yes	Yes
Has debugging sensors	Yes	No
Can modify signals on the fly (configurable)	Yes	No
Can interpret complex computations	Yes	No
Supports on-the-fly configuration	Yes	No
Notify the debugger without triggering event	Yes	No
Supports software stepping	Yes	No
Simplicity of state machine scripting	Simpler, software-like scripting	More complex
Can perform stepping fault-injection and signal manipulation	Yes	No
Can perform active debugging (producing fuzzing signals)	Yes	No

Example Script 1

```
? {  
  
    if (@hw_port5 == 55) {  
        printf("@hw_port5 is equal to 0x55\n");  
    }  
    elseif(@hw_port5 == 66) {  
        printf("@hw_port5 is equal to 0x66\n");  
    }  
    elseif(@hw_port5 == 77) {  
        printf("@hw_port5 is equal to 0x77\n");  
    } else {  
        printf("@hw_port5 is not equal to 0x55, 0x66,  
            0x77. It is equal to %llx\n", @hw_port5);  
    }  
}
```

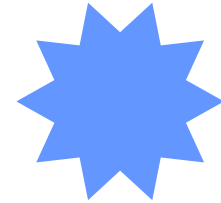
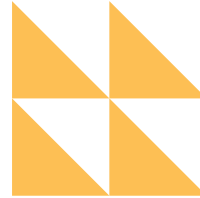
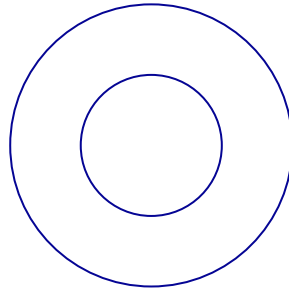
Example Script 2

```
? {
    int my_func1(int var1) {
        result = var1 + 1;
        printf("my_func1 %d\n", result);
        return result;
    }
    int my_func2(int var1) {
        result = var1 * 2;
        printf("my_func2 %d\n", result);
        return result;
    }

    int my_func3(int var1) {
        result = my_func1(var1) + my_func2(var1);
        printf("my_func3 %d\n", result);
        return result;
    }
    printf("%d\n", my_func3(2));
}
```

Software Stepping in Hardware

- Once we can control the clock signal, we are able to see and modify I/O ports at each cycle.
- The next rising-edge clock will be inserted once the introspection/modification is done.
- Not possible in chips with internal clock sources.





Thanks!



Do you have any questions?

m.s.karvandi@vu.nl

vusec.net

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**

Please keep this slide for attribution

