2022
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
EUROPE
MUNICH, GERMANY
DECEMBER 6 - 7, 2022

Unified firmware debug throughout SoC development lifecycle

D. Ciaglia [1], *T. Winkler* [1], J. Kundrata [2]

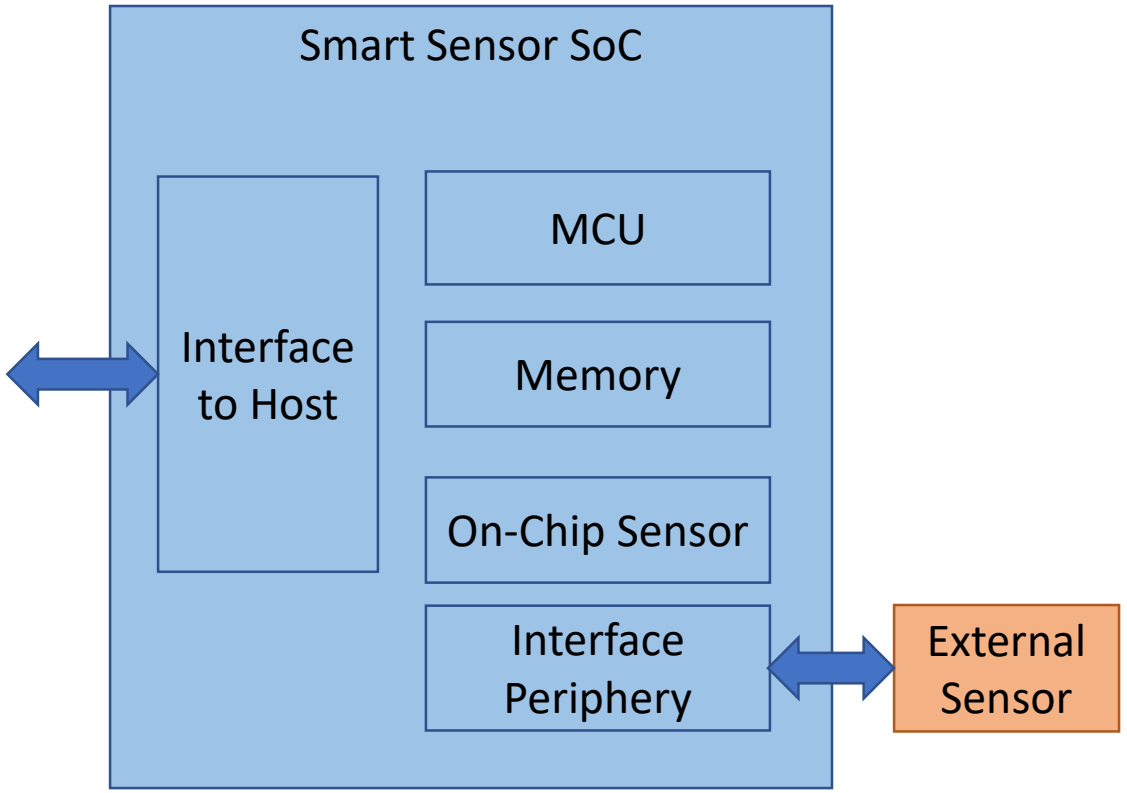
[1] ams-OSRAM International GmbH
[2] University of Zagreb

accellera SYSTEMS INITIATIVE

Agenda

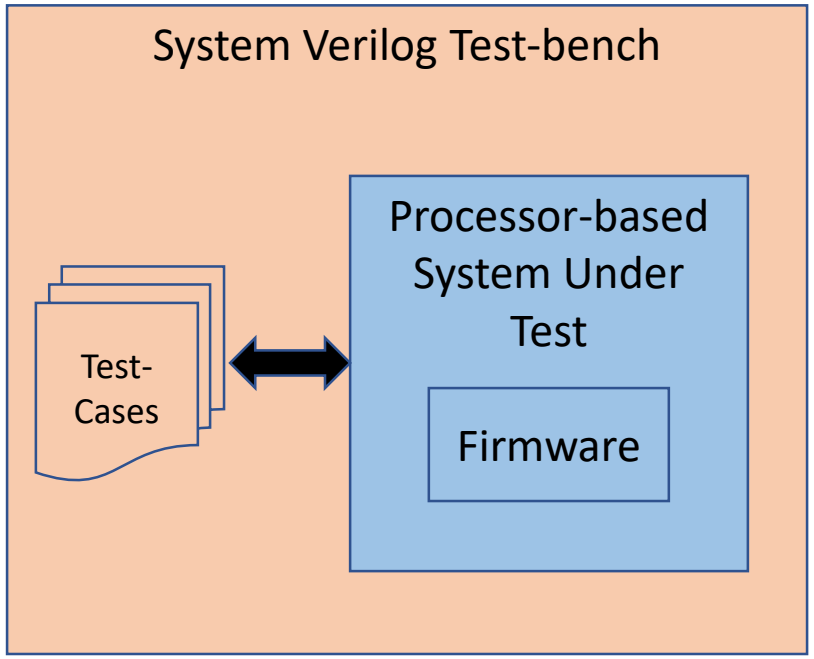
- Introduction
- Background
- Proposed Approach
- Implementation of the DPI-based SoC Simulator
- Conclusions

Smart Sensors



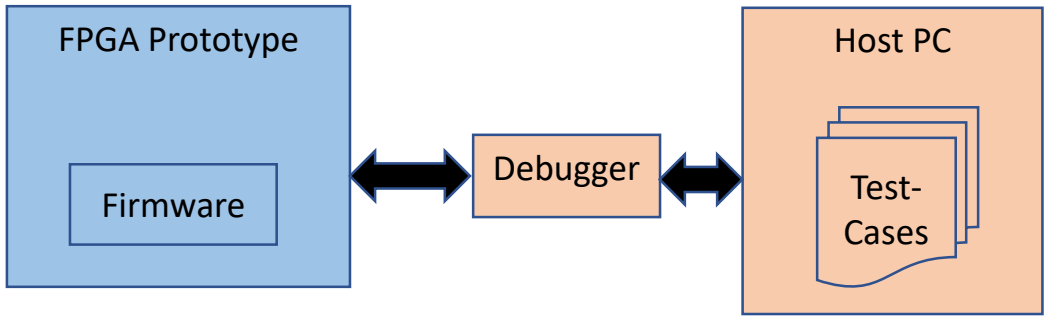
- Consumer and automotive market demand
- Build-up intelligence
- Provide a smart sensor solution is key

Pre-Silicon Verification



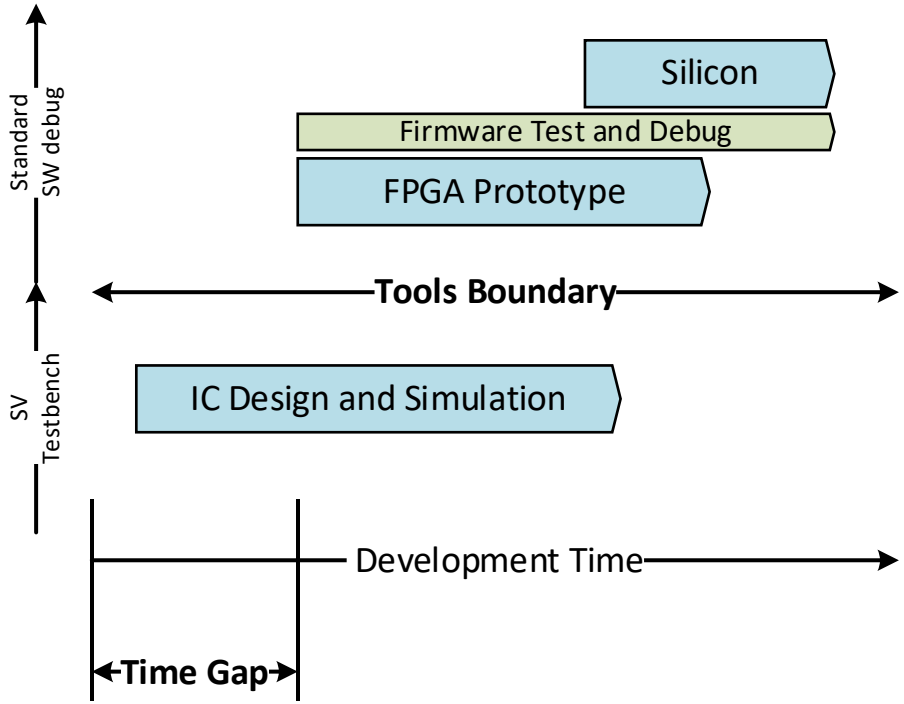
- Create a test-bench close to the real system
- Leverage complete system internal observability
- Ensure bug-free design (hardware and firmware)

Pre-Silicon Firmware Debugging



- Build FPGA Prototype
- Implement SW test-cases
- Use the standard debug and test tools

Limitations of Standard Approach



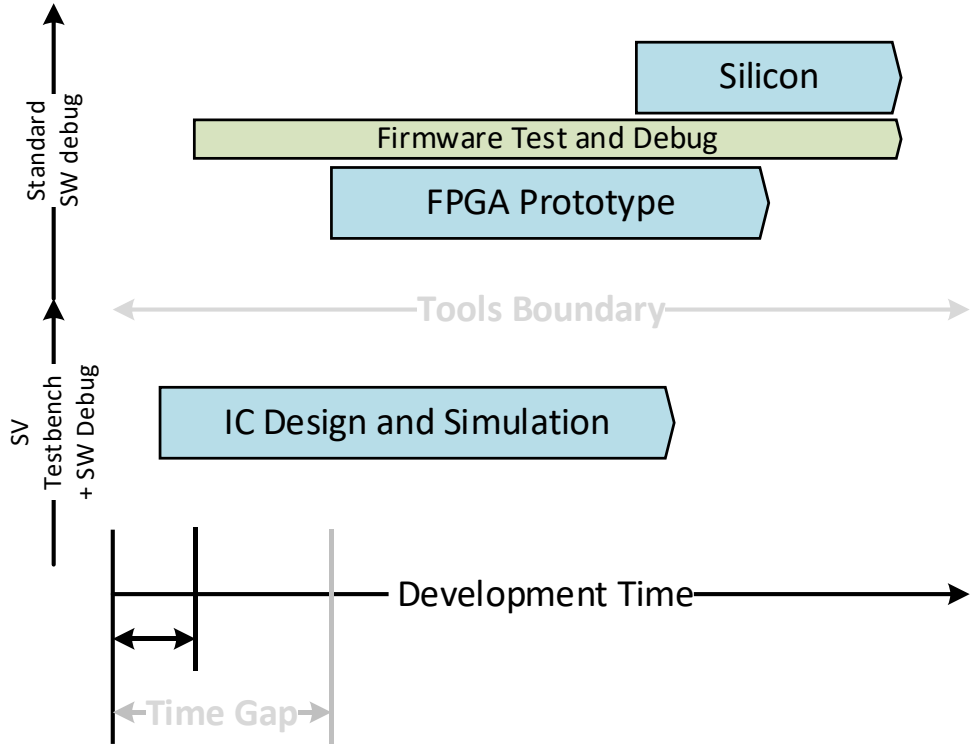
- System-Verilog tests cannot be re-used for firmware debugging
- Firmware debugging is bound to FPGA prototype availability

Co-Verification Techniques

Technique	Speed	Timing	Software	Debug	Cost
Instruction set simulation	Medium	No	C Model	Algorithm	Low
HW/SW co-verification	Slow	Yes	Real	HW/SW	High
Rapid Prototyping	Fast	Yes	Real	Low	Medium
Emulation	Very fast	Yes	Real	Low	Very high

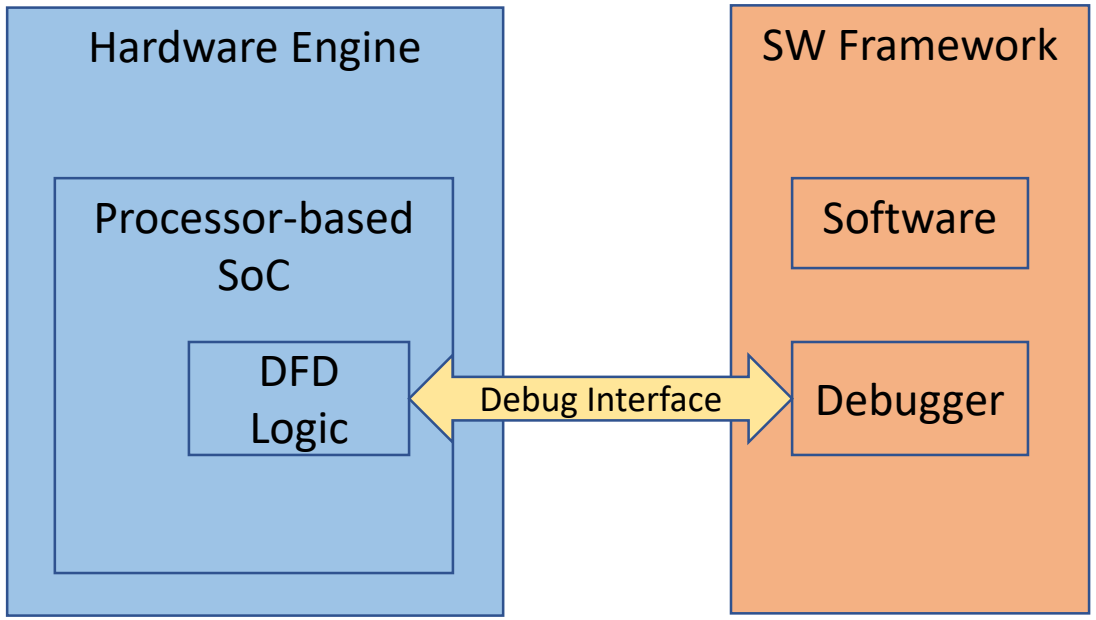
- Use a technique during design phase
- Ensure high observability
- Maximize tests re-use

HW/SW Co-verification



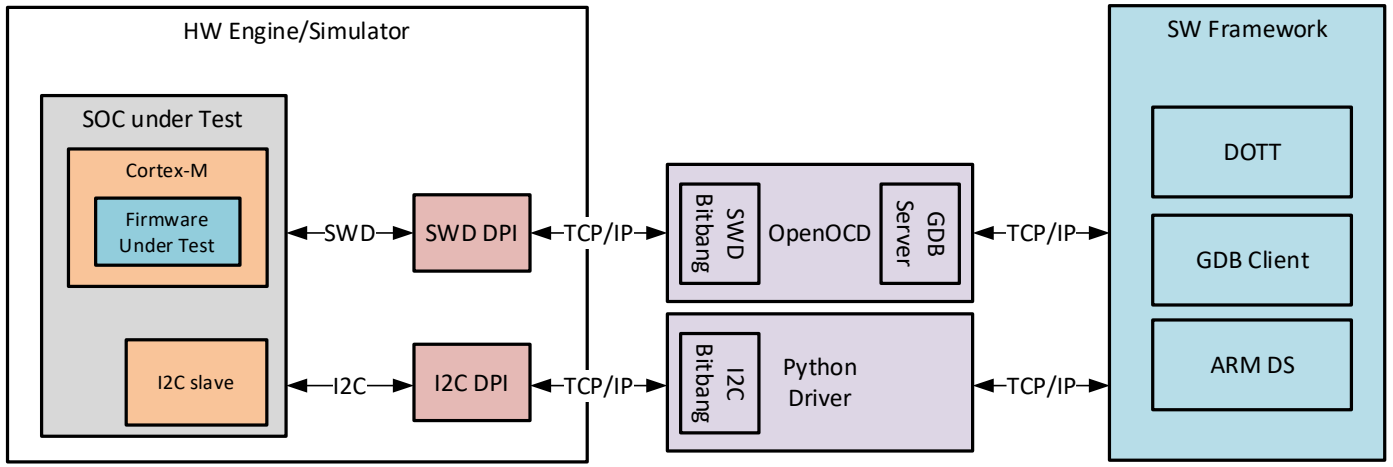
- Key enabler for “shift left”
- Maximize firmware tests re-use
- Enables on-chip debugging

Co-verification Environment



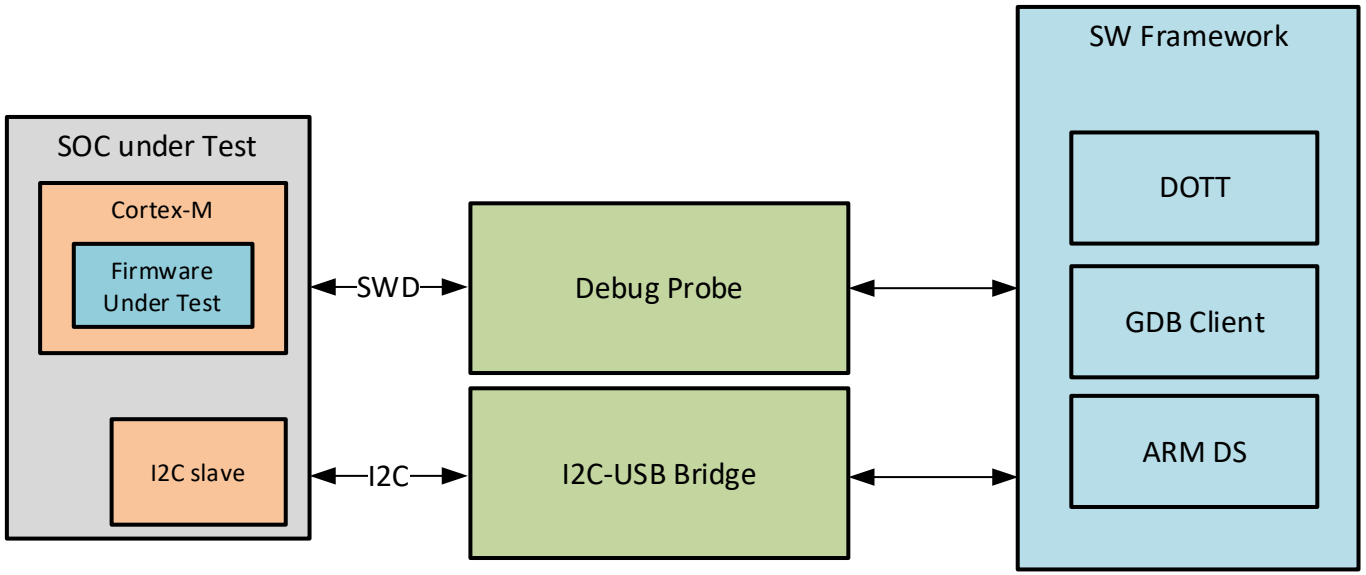
- Direct SW interaction with HW
- Full observability guaranteed by HW Engine
- Design for Debug Logic for standard interface
- “Soft” debug interface

Proposed Approach – Case Study



- Direct programming interface for HW/SW interaction
- Open source debugger
- Standard SW framework based on GDB
- Python-based DOTT for tests development

FPGA or Post-Si Debug Environment



- Swap the DPI with real debug probe
- Re-use of standard SW debug tools
- Re-use of test-cases

Simulation environment

- **SV testbench** --> SoC under Test + DPIs
- **SoC under Test** --> ARM Cortex-M core with SWD and I2C interfaces
- **SWD DPI** --> SWD debugger probe controlled via TCP port using a bitbang protocol and OpenOCD tool
- **I2C DPI** --> I2C master core controlled via TCP port using a bitbang protocol and a Python driver module

Implementation of the SWD-DPI

- Based on OpenTitan implementation of JTAG DPI
- JTAG --> all pins are unidirectional
- SWD --> the data pin (SWDIO) is bidirectional

```
1  module swddpi #(
2    parameter string Name = "swd0", // name of the SWD interface (display only)
3    parameter int ListenPort = 44853 // TCP port to listen on
4  )(
5    input  logic clk_i,
6    input  logic rst_ni,
7
8    output logic swd_swclk,
9    inout  logic swd_swdio,
10   output logic swd_nrst_n
11 );
12 ////////////////////////////////////////////////////////////////////////////////
13   // bidirectional behaviour of SWDIO signal
14   reg swd_swdoen, swd_swdi, swd_swdo;
15
16   assign swd_swdio = swd_swdoen ? swd_swdo : 1'bZ;
17   assign swd_swdi  = swd_swdio;
18 ////////////////////////////////////////////////////////////////////////////////
19   import "DPI-C"
20   function void swddpi_tick(input chandle ctx, output bit swclk, output bit swdo,
21                             output bit swdoen, input bit swdi, output bit nrst_n);
```

swddpi.sv

Implementation of the SWD-DPI

```
1  struct swddpi_ctx {
2    // Server context
3    struct tcp_server_ctx *sock;
4    // Signals
5    uint8_t swclk;
6    uint8_t swdo;
7    uint8_t swdoen;
8    uint8_t swdi;
9    uint8_t nrst_n;
10 };
11 ////////////////////////////////////////////////////////////////////////////////
12   // parse received command byte
13   if (cmd >= 'd' && cmd <= 'g' || cmd == 'O' || cmd == 'o') {
14     // swd write
15     char cmd_bit = cmd - 'd';
16     ctx->swclk = (cmd_bit >> 1) & 0x1;
17     ctx->swdo  = (cmd_bit >> 0) & 0x1;
18     if (cmd == 'O')
19       ctx->swdoen = 1;
20     if (cmd == 'o')
21       ctx->swdoen = 0;
22   }
```

swddpi.c

Command	Description
O	Set the SWDIO line to output mode
o	Set the SWDIO line to input mode
c	Read the SWDIO line
d	Set SWCLK = 0 and SWDIO = 0
e	Set SWCLK = 0 and SWDIO = 1
f	Set SWCLK = 1 and SWDIO = 0
g	Set SWCLK = 1 and SWDIO = 1

Configuration of the OpenOCD tool

- OpenOCD compiled using the SWD remote bitbang protocol patch
  - Changes #3908, #4205 & **#6044**
- Configuration file:

```
1 adapter driver remote_bitbang
2 remote_bitbang_host localhost
3 remote_bitbang_port 44853
4 transport select swd
5
6 swd newdap cortex_m0p_sim cpu -dp-id 0x0BC11477
7 dap create cortex_m0p_sim.dap -chain-position cortex_m0p_sim.cpu
8 target create cortex_m0p_sim cortex_m -endian little -dap cortex_m0p_sim.dap
```

Implementation of the I2C-DPI

- Based on OpenTitan implementation of JTAG DPI
- JTAG --> all pins are unidirectional
- I2C --> the data pin (SDA) is bidirectional

```
1  module i2cdpi #(
2    parameter string Name = "i2c0", // name of the i2c interface (display only)
3    parameter int ListenPort = 44855 // TCP port to listen on
4  )(
5    input  logic clk_i,
6    input  logic rst_ni,
7
8    output logic i2c_scl,
9    inout  logic i2c_sda
10 );
11 ////////////////////////////////////////////////////////////////////////////////
12   // bidirectional behaviour of the sda line
13   reg i2c_sdaoen, i2c_sdai, i2c_sdao;
14
15   assign i2c_sda = i2c_sdaoen ? i2c_sdao : 1'bZ;
16   assign i2c_sdai  = i2c_sda;
17 ////////////////////////////////////////////////////////////////////////////////
18   import "DPI-C"
19   function void i2cdpi_tick(input chandle ctx, output bit scl, output bit sdao,
20                             output bit sdaoen, input bit sdai);
```

i2cdpi.sv

Implementation of the I2C-DPI

```c
struct i2cdpi_ctx {
  // Server context
  struct tcp_server_ctx *sock;
  // Signals
  uint8_t scl;
  uint8_t sdao;
  uint8_t sdaoen;
  uint8_t sdai;
};
////////////////////////////////////////////////////////////////////////////////
  // parse received command byte
  if (cmd >= 'd' && cmd <= 'g' || cmd == 'O' || cmd == 'o') {
    // i2c write
    char cmd_bit = cmd - 'd';
    ctx->scl = (cmd_bit >> 1) & 0x1;
    ctx->sdao  = (cmd_bit >> 0) & 0x1;
    if (cmd == 'O')
      ctx->sdaoen = 1;
    if (cmd == 'o')
      ctx->sdaoen = 0;
```

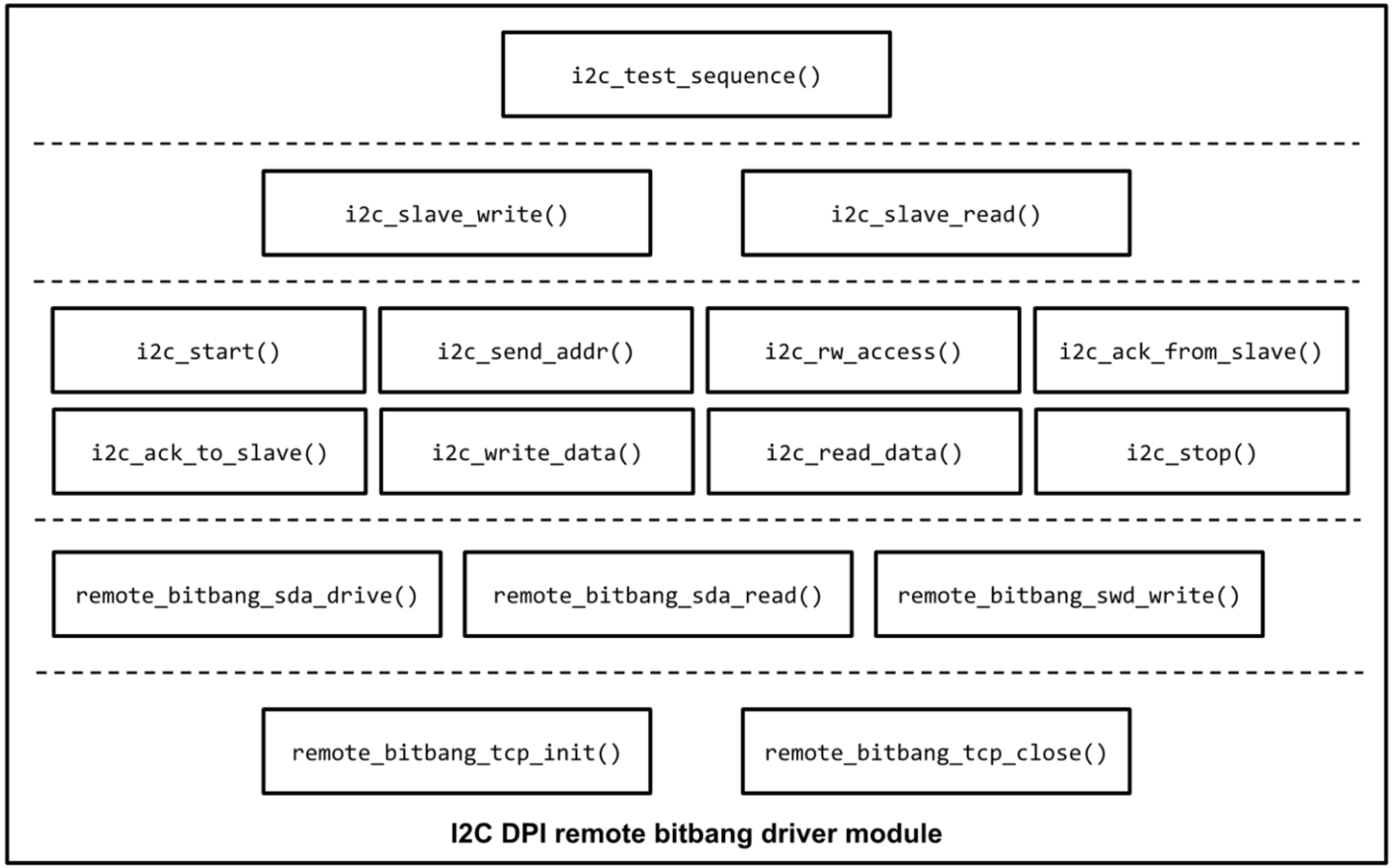
i2cdpi.c

Command	Description
O	Set the SDA line to output mode
o	Set the SDA line to input mode
c	Read the SDA line
d	Set SCL = 0 and SDA = 0
e	Set SCL = 0 and SDA = 1
f	Set SCL = 1 and SDA = 0
g	Set SCL = 1 and SDA = 1

The I2C DPI remote bitbang driver

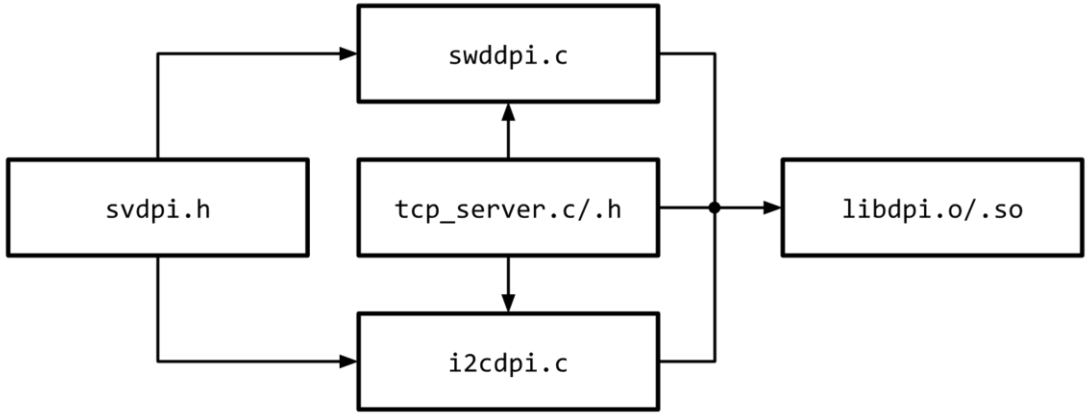
- I2C test sequence
- I2C writes and reads
- Atomic I2C steps

- Bitbang protocol
- TCP connection handling



Compilation of the DPI library

- File organization



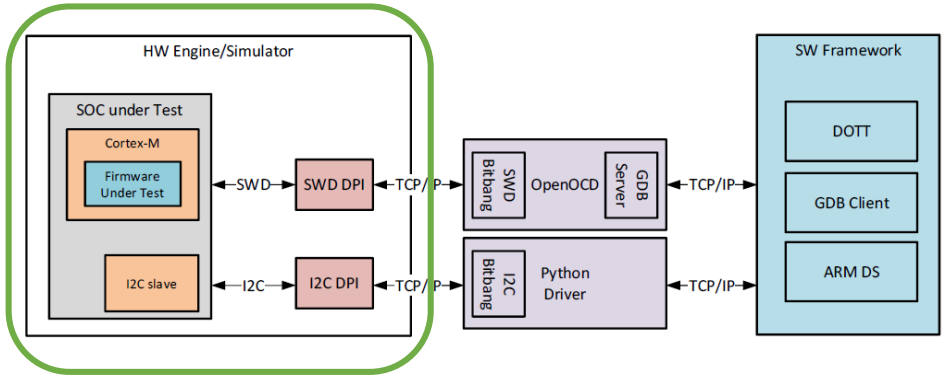
- Compilation script

```
1 #!/bin/sh
2 g++ -c -fPIC tcp_server.c -o libdpi.o -include swddpi.c -include i2cdpi.c
3 g++ -shared -Wl,-soname,libdpi.so -o libdpi.so libdpi.o
```

Testing results

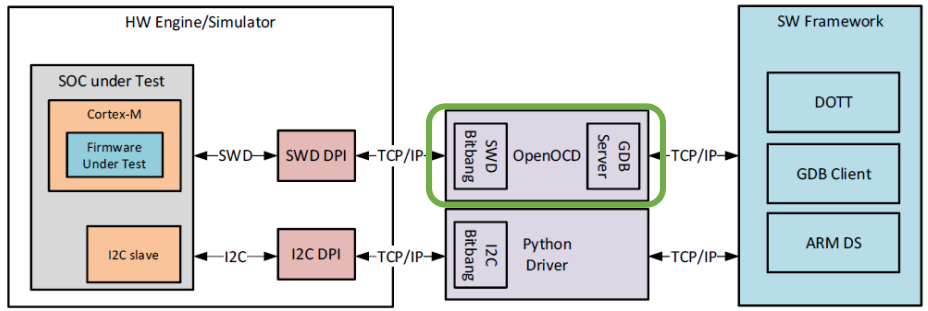
Xcelium **Simulation** (SoC, SWD & I2C DPI)	**I2C Testing** **Sequence** (I2C remote bitbang via TCP)
Telnet **Client** (SWD via OpenOCD)	**OpenOCD** **Server** (SWD remote bitbang via TCP)

Testing results - Xcelium Simulation



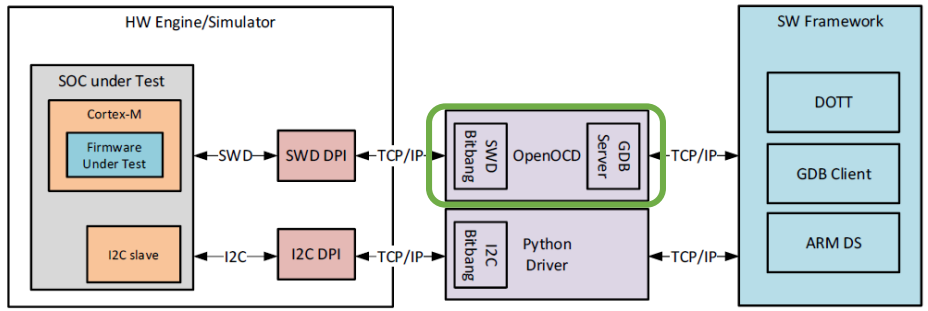

```
xcelium> run

swd: Virtual SWD interface swd0 is listening
on port 44853. Use OpenOCD and the following
configuration to connect:
  interface remote_bitbang
  remote_bitbang_host localhost
  remote_bitbang_port 44853

I2C: Virtual I2C interface i2c0 is listening
on port 44855.

i2c0: Accepted client connection
I2C DPI: Remote disconnected.
swd0: Accepted client connection
```

Testing results - OpenOCD Server (I)



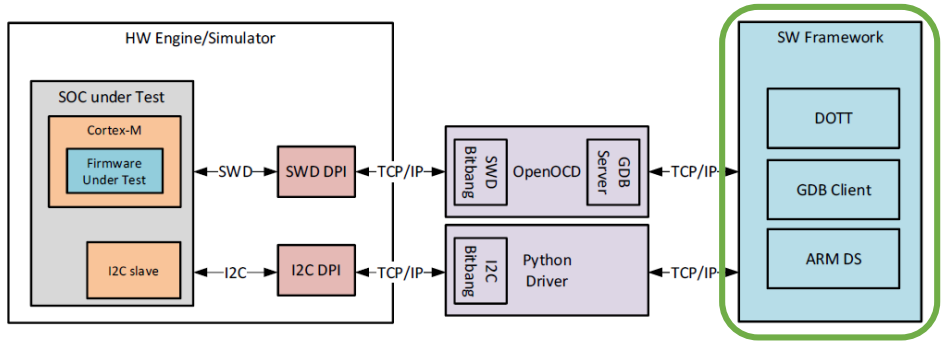

```
[user@linux openocd]$ ./openocd
Open On-Chip Debugger 0.11.0-rc2+dev-00003-
gb5563b75d-dirty (2022-09-02-17:21)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.h
tml
Info : Listening on port 6666 for tcl
connections
Info : Listening on port 4444 for telnet
connections
Info : Initializing remote_bitbang driver
Info : Connecting to localhost:44853
Info : remote_bitbang driver initialized
Info : This adapter doesn't support
configurable speed
```

Testing results - OpenOCD Server (II)



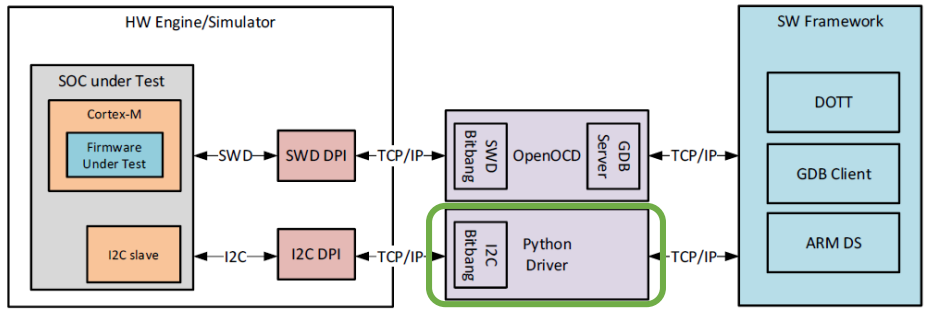

```
Info : SWD DPIDR 0x0bc11477
Info : cortex_m0p_sim: hardware has 4 breakpoints, 2 watchpoints
Info : starting gdb server for cortex_m0p_sim on 3333
Info : Listening on port 3333 for gdb connections
Info : accepting 'telnet' connection on tcp/4444
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x20000180 msp: 0x20000508
```

Testing results - Telnet Client




```
[user@linux openocd]$ telnet localhost 4444
Connected to localhost.
Escape character is '^]'.
Open On-Chip Debugger
> halt
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x20000180 msp: 0x20000508
> reg
===== arm v7m registers
(0) r0 (/32): 0xe000e100
. . . . . .
(13) sp (/32): 0x20000508
(14) lr (/32): 0x200000e9
(15) pc (/32): 0x20000180
(16) xPSR (/32): 0x01000000
(17) msp (/32): 0x20000508
```

Testing results - I2C Testing Sequence




```
[user@linux python]$ ./run_i2c_dpi_remote_test.py
========== I2C DPI Remote Bitbang Test =========
Write access ACKnowledged.
I2C write @ 0x12 = 0xba.
Write access ACKnowledged.
I2C write @ 0x12 = 0xdc.
Read access ACKnowledged.
I2C read @ 0x12 = 0x47.
Read access ACKnowledged.
I2C read @ 0x12 = 0x23.
```

Conclusions

- Introduce a methodology for pre-FPGA debugging
- Design the architecture for HW/SW co-verification
- Maximize re-use of debug tools and test-cases
- Leverage software development expertise and resources

Questions