FOBOS 2.0

Release 2.0

Abubakr Abdulgadir, William Diehl and Jens-Peter Kaps

Nov 04, 2019

CONTENTS

The Flexible Opensource workBench fOr Side-channel analysis FOBOS is a platform to perform side-channel analysis (SCA). FOBOS uses commercially avilable boards when possible to reduce the cost of building a working SCA setup. Using FOBOS, power traces can be collected and attacks like Correlation Power Analysis (CPA) can be mounted. Also, scripts to perform leakage assesment are included.

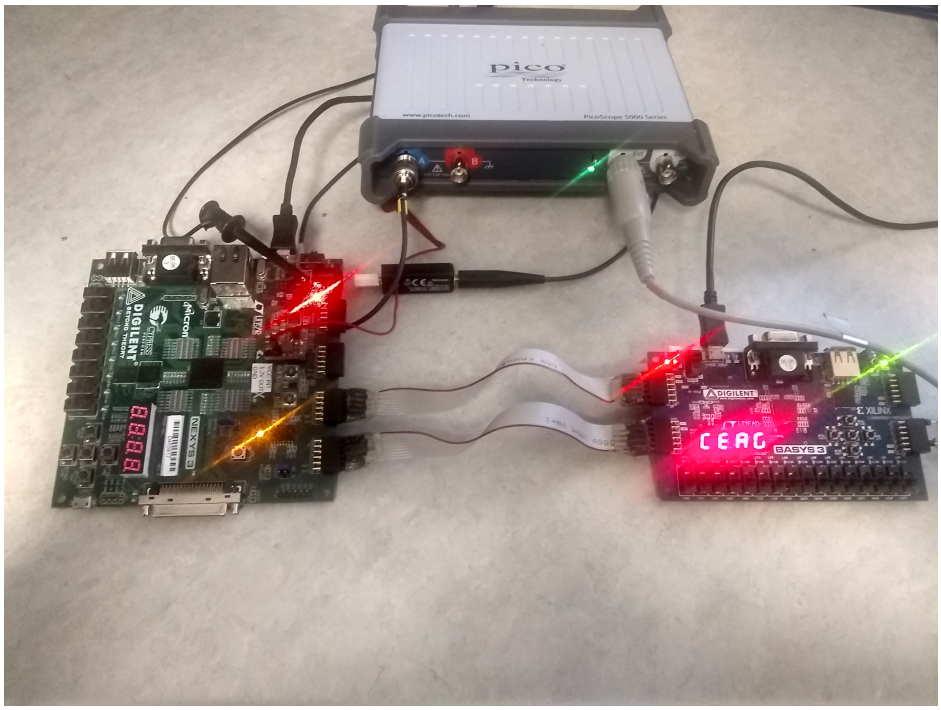

Fig. 1: Typical FOBOS2 Setup

CHAPTER

ONE

INTRODUCTION

The Flexible Opensource workBench fOr Sidechannel analysis (FOBOS) is a platform that can be used for side channel analysis. You can perform trace collection (Data Acquisition) and attacks (Data Analysis). The system is suitable for educational and research purposes.

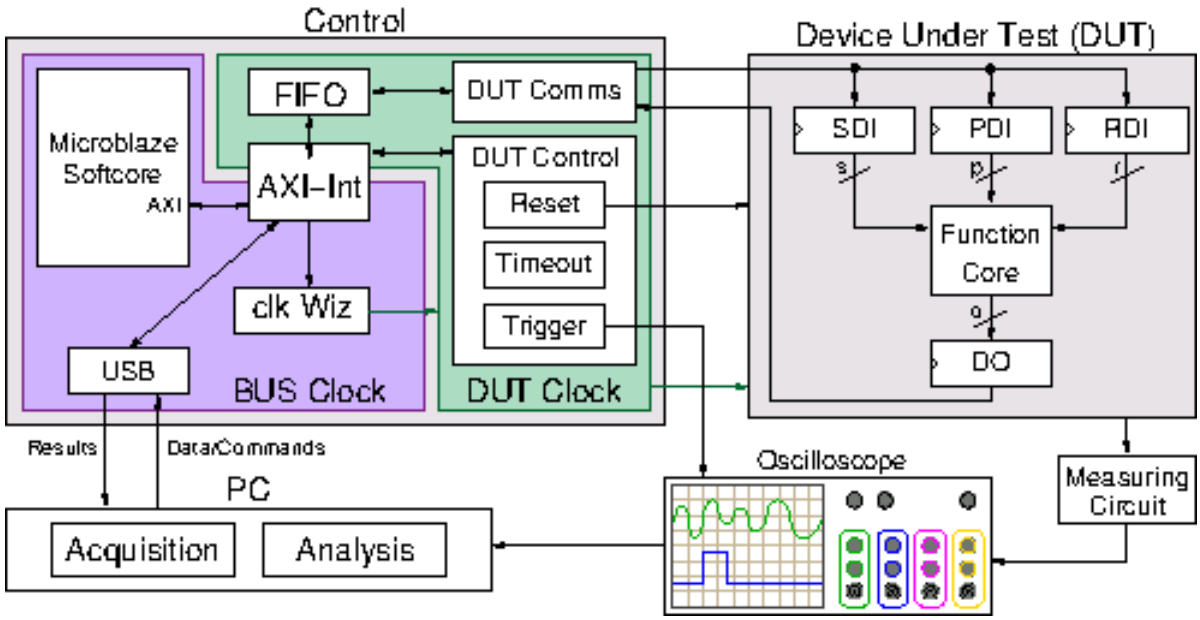

Fig. 1: FOBOS Block Diagram

1.1 Feature Overview

- Separate control and (Design Under Test) boards allowing addition of new DUTs easily.
- Uses commercial over-the-shelf boards when possible to reduce cost.
- Digilent Basys3 and Nexys A7 control boards.
- DUT support includes Digilent Nexys3 board (Xilinx Spartan6) and compatible with NewAE CW305 (Xilinx Artix7 boards).
- Adjustable DUT clock.
- Adjustable trigger signal.
- Software to perform Correlation Power Analysis (CPA).

- Leakage assessment using t-test and chi-squared test.

CHAPTER

TWO

SETUP

Below, we describe how to setup FOBOS 2.0 hardware and software and test that everything is working. This page only describes setting up the control board and software for acquisition and anlysis. For DUT board please refer to *DUT Board Setup*.

2.1 Requirements

1. Digilent Basys3 board (control board).
2. A PC with Linux installed.
3. Python2.7 installed.
4. Xilinx Vivado 2017.2.
5. Picoscope 5000 series.

2.2 Software Setup

Note: The following installation procedure is tested on Linux Ubuntu 16.04.

1. Download FOBOS from the FOBOS home page.
2. Extract the archive into the directory of your choice

```
$ tar xvfz fobos-v2.0.tgz
```

3. Use the following commands to install pip and few necessary Python packages:

```
$ sudo apt-get install python-pip
$ tar xvfz fobos-v2.0.tgz
$ pip isnatll -r requirements.txt
```

2.3 Control board Setup

Follow these steps to compile the control software, generate the bitstream and program the the control board.

1. Build the control board Vivado project.

```
$ cd fobos/capture/ctrl/basys3ctrl/vivado
$ make project
```

2. A Vivado project will be created at fobos/capture/ctrl/basys3ctrl/vivado/basys3ctrl. Open it using Vivado.

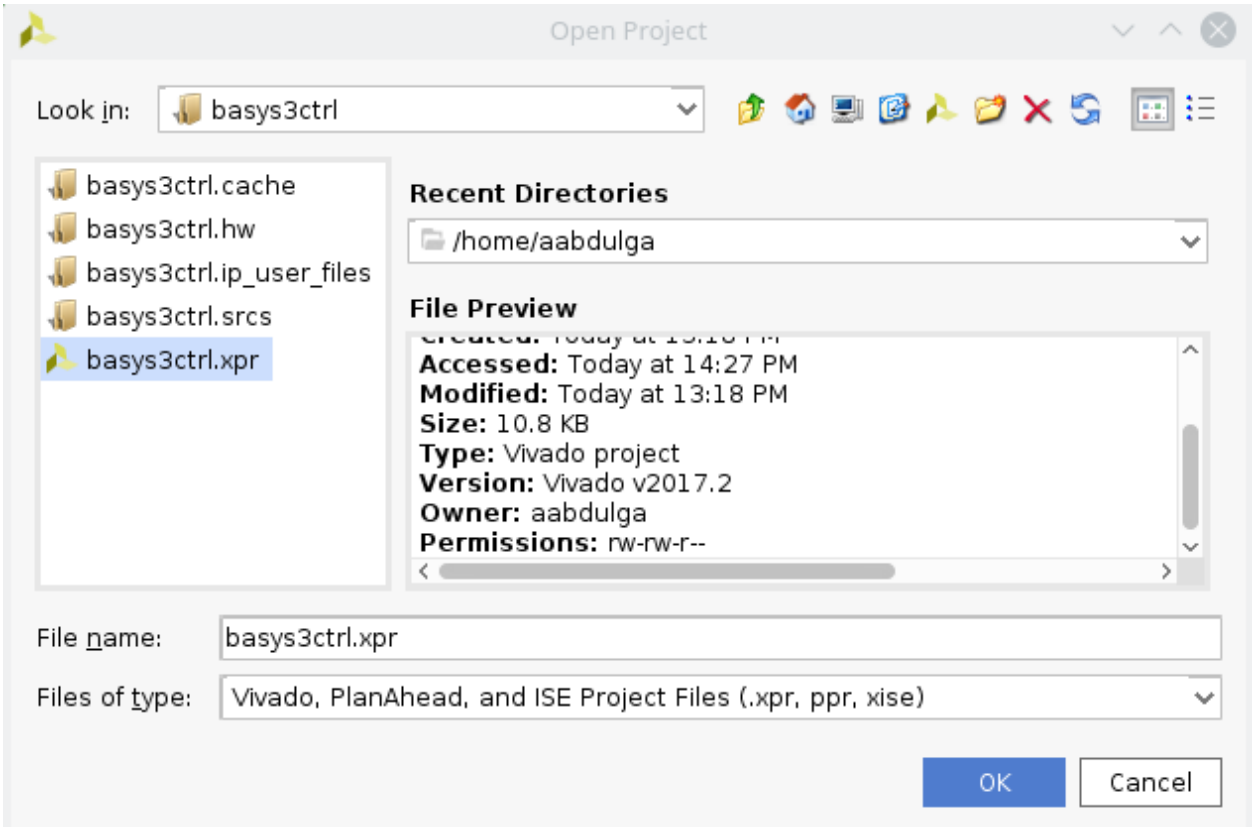

Fig. 1: Open Vivado project

3. In Vivado's Flow Navigator window, click 'Generate Bitstream'.
4. After bitstream is generated, export the hardware. Click File > Export > Export Hardware ... make sure to select 'Include bitstream'.
5. Launch the Xilinx SDK (File > Launch SDK).

6. In the SDK, create a new empty project(File> New application project). Set the project name to *ctrl* and select the hardware platform, click Next and make sure you select 'Empty project'.

7. Link all the .c and .h files in the fobos/capture/ctrl/basys3ctrl/sdk/src/ to the project (right-click on ctrl/src folder -> Import -> General-> file system -> browse to folder). make sure to check "Advanced-> Create links in the workspace" and "Create virtual folders" .
8. Program the control borad FPGA. Connect the Basys3 board to your PC via USB. In the Xilinx Tools menu, select Program FPGA -> program.
9. Run the control software. Make sure to select the *ctrl* project created in step 6 then go to the Run menu and select 'Run'.
10. You should see the word CERG in the seven-segment display of the Basys3 board.

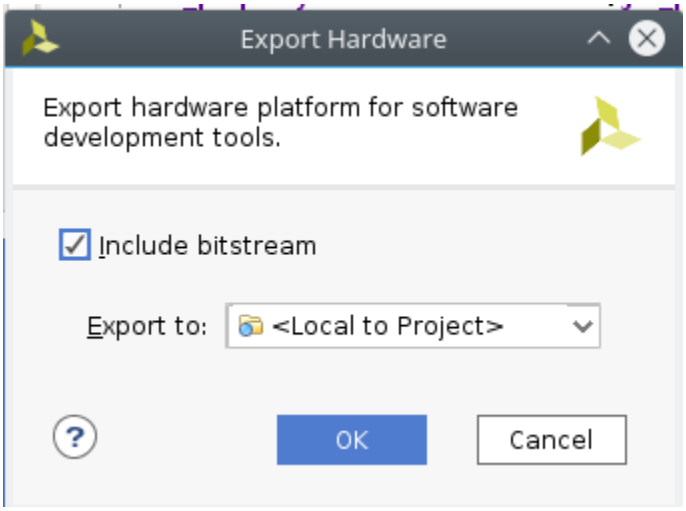

Fig. 2: Export Hardware

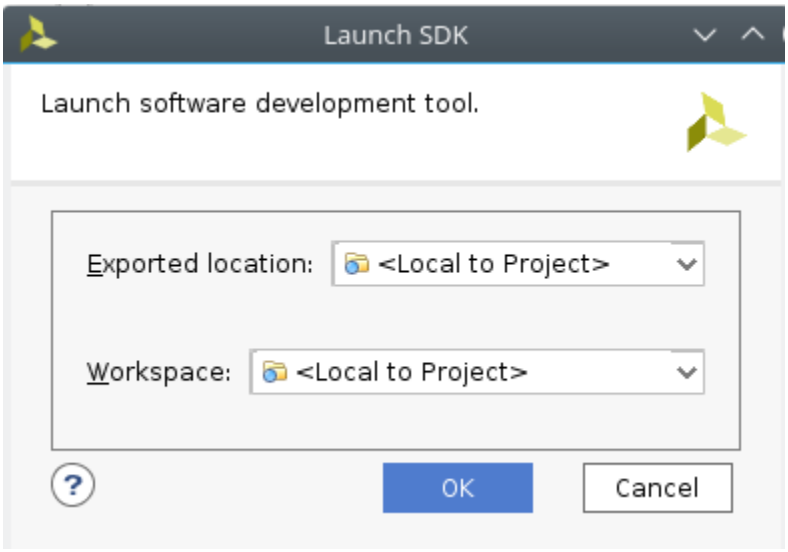

Fig. 3: Launch SDK

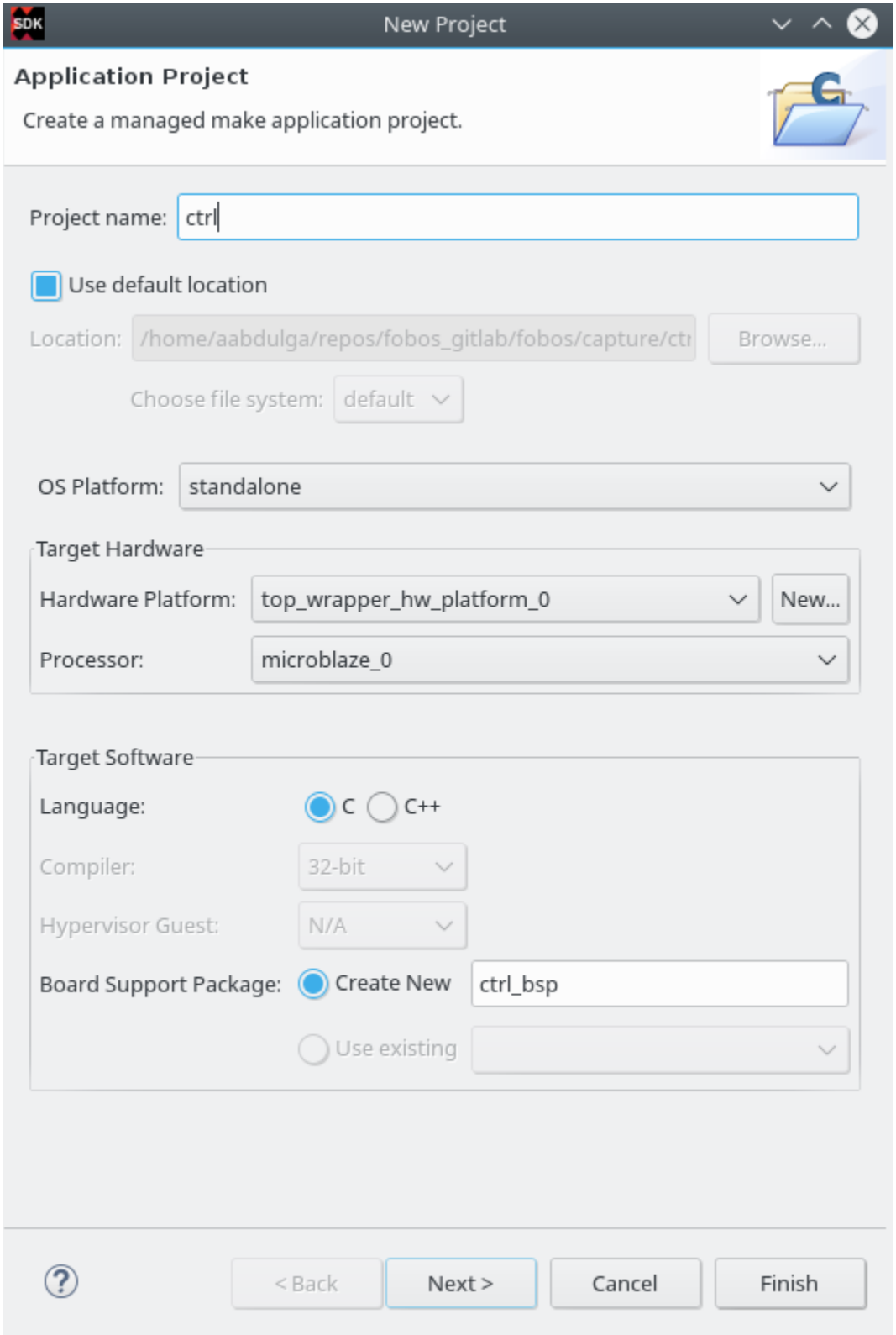

Fig. 4: Create Project

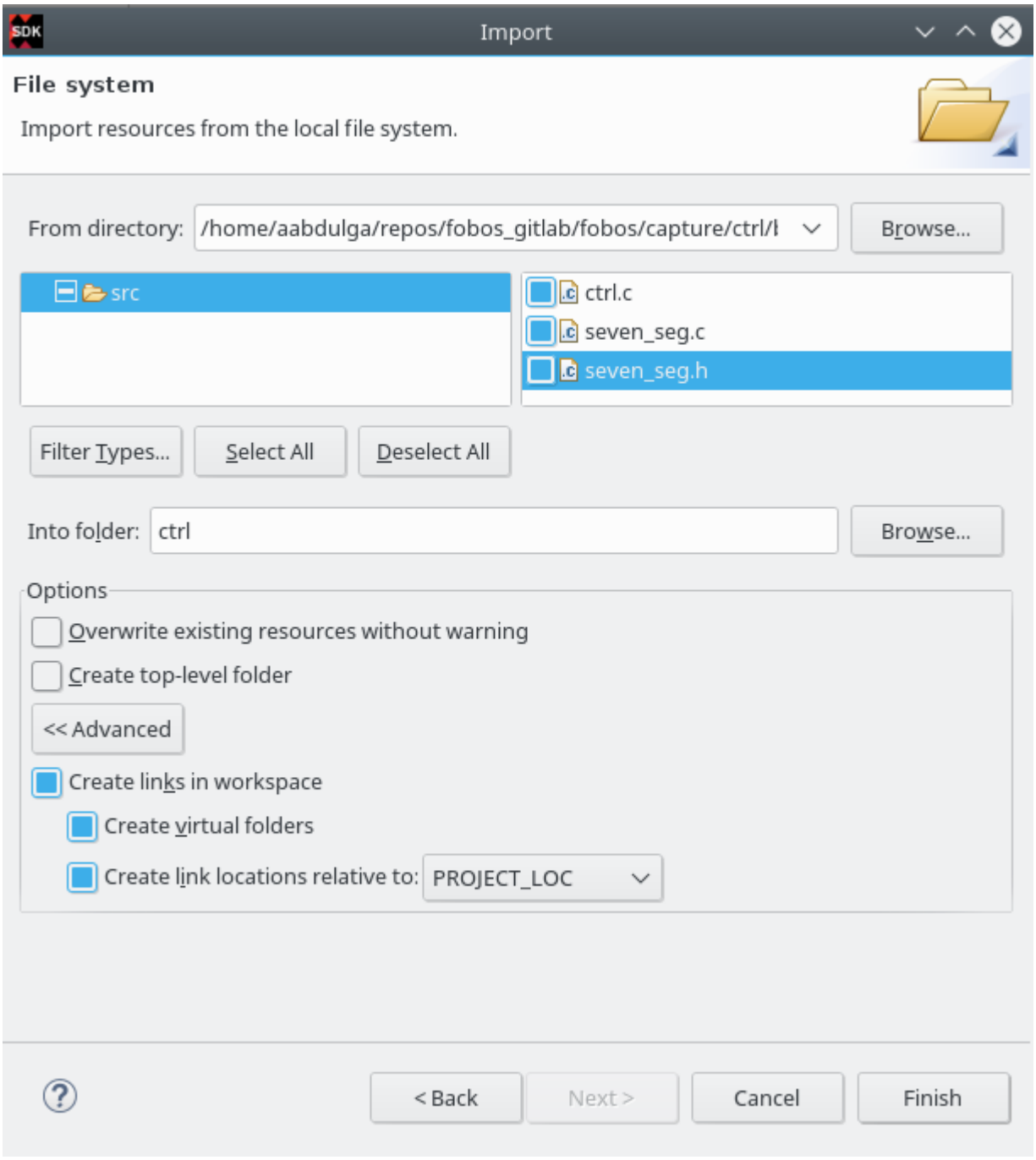

Fig. 5: Launch SDK

2.4 Testing the control board

To make sure the control board is working, you can run the *dymmyCaptureBasic.py* script in the fobos/software directory. This script send data to the board wich echos data back.

```
$ cd path-to-fobos/software
$ python dummyCaptureBasic.py
Sending configuration...
f0030006000900000001
Status= 00000000
f0030006000000000007
Status= 00000000
Sending data..
f001001e00c0000761996dc996d4ac00c100070f7821507a22a00081000700800001
OK.    Status= 00000000
61 99 6d c9 96 d4 ac
f001001e00c00007fd8771fe717de400c100073e1fe5b4aa357c0081000700800001
OK.    Status= 00000000
fd 87 71 fe 71 7d e4
f001001e00c0000782051f5484702200c10007980d05d4ea25bc0081000700800001
OK.    Status= 00000000
82 05 1f 54 84 70 22
f001001e00c0000767881b702afe5200c10007b08a5e036de72b0081000700800001
OK.    Status= 00000000
67 88 1b 70 2a fe 52
f001001e00c0000726a1d601ccdf7a00c1000773539e52672d5d0081000700800001
OK.    Status= 00000000
26 a1 d6 01 cc df 7a
```

If you see this output, your control board is now ready!

CHAPTER

THREE

TEST VECTOR GENERATION

The user must prepare test vectors before running data acquisition. User defined scripts or scripts provided with FOBOS can be used. The data acquisition scripts will send the test vectors one at a time and collect traces from the oscilloscope.

Cryptographic hardware interfaces typically use multiple data types as input to cryptographic cores. For example, some algorithms might need plaintext/ciphertext, cryptographic keys, and random data. We provide a simple wrapper to split data provided by the control board to separate streams. This wrapper is directly compatible with CAESAR Hardware API interface and is expected to be directly compatible with a future Hardware API for Lightweight Cryptography (LWC API). We developed a simple, yet versatile protocol to enable the wrapper to split the data types. The wrapper receives data from the control board and distributes it into three FIFOs:

1. The Public Data Input (PDI) FIFO (i.e. plaintext)
2. The Secret Data Input (SDI) FIFO (i.e. key)
3. The Random Data Input (RDI) FIFO which stores random data which can be used for protected implementations that use masking schemes.

Once the wrapper prepares the data for the function core, it starts the core which consumes the data in the input FIFOs and produces output. The wrapper accumulates the output into a fourth FIFO called the Data Out (DO) FIFO until the expected number of bytes are stored. Then, the wrapper returns the data to the control board which forwards it back to the PC.

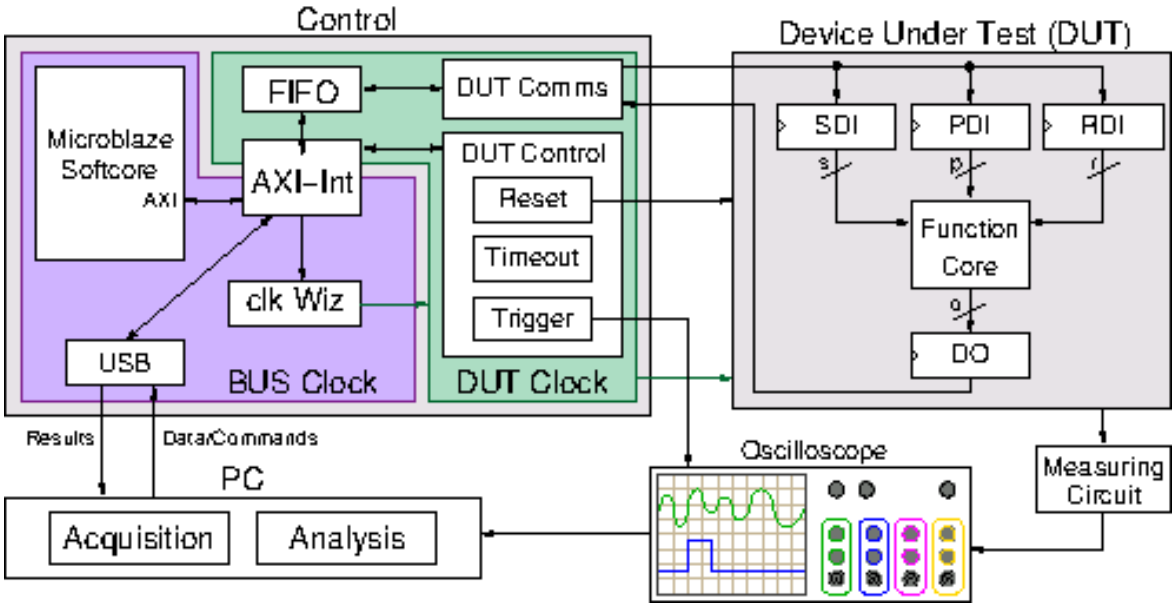

Fig. 1: FOBOS2 block diagram

The following is a brief description for the test vector format

3.1 Supported commands

- 00C0 # pdi fifo (length in bytes to follow)
- 00C1 # sdi fifo (length in bytes to follow)
- 00C2 # rdi fifo (length in bytes to follow)
- 0081 # store expected output size (expected output size in bytes to follow)
- 0080 # select command register (command to follow)

3.1.1 FOBOS Protocol Example

Here is an example of a signle test vector(split into multiple lines):

```
00C0 # pdi fifo (length in bytes to follow)
0008 # 8 bytes
FFFF # 8 bytes of pdi
FFFF
FFFF
FFFF
00C1 # sdi fifo (length in bytes to follow)
000A # 10 bytes
0000 # 10 bytes of sdi
0000
0000
0000
0000
0081 # store expected output size
0008 # 8 bytes of output expected
0080 # select command register
0001 # "start signal"
```

3.2 Using the blockCipherTVGen.py script

The blockCipherTVGen.py can be used to generate test vectors to be used by block ciphers. The script is located at fobos/software/tvgen/ There are two steps to use it:

1. Set user defined parameters.
2. Run the script. It will generate the test vector file and plaintext file (not required for acquisition).

3.2.1 Example: Generating AES-128 test vectors

AES-128 takes 16 bytes (128 bits) plaintext, 16 key and returns 16 byte ciphertext. Here is an example to generate 4 test vectors with 16 byte blocks, key and ciphertext. Key is fixed in this case.

Settings:

```
############user defined settings
TRACE_NUM = 1000                            # Number of traces
PDI_LENGTH = 16                             # In byets
SDI_LENGTH = 16                             # In bytes
EXPECTED_OUT = 16                           # Expected output in bytes
DIN_FILE = 'dinFile.txt'                    # Desitination file name
PLAINTEXT_FILE = 'plaintext.txt'            # Desitination file name
FIXED_KEY = 'yes'                           # Fixed key = yes | no
KEY = '123456789009876ABCDFE12456789ABF'    # Fixed key
```

To run the script, use the following command:

```
$ python blockCipherTVGen.py
```

Here is how the generated dinFile.txt looks like.:

```
$ cat dinFile.txt
00C000103AD5305EBD0C99C7482263E2D7ECEAED00C1001012345...0081001000800001
00C000105C09504D713BF9B5925601E671EA257800C1001012345...0081001000800001
00C00010A6D6DE2548E4CCF446ECA8E620E4E55500C1001012345...0081001000800001
00C00010E0792CDE9AFDA7EAC33A8D0EADE524CB00C1001012345...0081001000800001
00C000104A09A00A4C4268F0B6F4FCE4F514A6BB00C1001012345...0081001000800001
```

This file can now be used in FOBOS as a test vector file.

A plaintext.txt file is also generated, it includes only the PDI portion dinFile.txt:

```
$ cat plaintext.txt
3A D5 30 5E BD 0C 99 C7 48 22 63 E2 D7 EC EA ED
5C 09 50 4D 71 3B F9 B5 92 56 01 E6 71 EA 25 78
A6 D6 DE 25 48 E4 CC F4 46 EC A8 E6 20 E4 E5 55
E0 79 2C DE 9A FD A7 EA C3 3A 8D 0E AD E5 24 CB
4A 09 A0 0A 4C 42 68 F0 B6 F4 FC E4 F5 14 A6 BB
```

This file can be used later by the analysis module.

CHAPTER

FOUR

DUT BOARD SETUP

A DUT (Design Under Test) board must be connected to the control board to run SCA. Here, we show how to connect and (modify board if needed) so that it can be used in FOBOS.

4.1 Digilent Nexy3 (Spartan 6 FPGA)

4.1.1 Connection to Basys3 control board

The PMOD ports on both boards are used to transfer data. The ports should be connected as follows

- Basys3 JA -> Nexys3 JC
- Basys3 JXADC -> Nexys3 JD

The connector must connect each pin to its corresponding pin in the other board. The GND of the two ports must be connected, However, the Vcc SHOULD NOT be connected.

The pin mapping in the Nexys3 DUT is as follows:

```
#JC
#+----------+----------+----------+----------+
#|          |  din3    |  do_ready|  rst     |
#|          |          |          |          |
#+----------+----------+----------+----------+
#|  din0    |  din2    |  din1    |  do_valid|
#|          |          |          |          |
#+----------+----------+----------+----------+

#JD
#+----------+----------+----------+----------+
#|  dout1   |  dout3   |  di_ready|  clk     |
#|          |          |          |          |
#+----------+----------+----------+----------+
#|  dout0   |  dout2   |  di_valid|          |
#|          |          |          |          |
#+----------+----------+----------+----------+
```

The pin mapping in the Basys3 control board is as follows:

```
#JA
#+----------+----------+----------+----------+
#|          |  din3    |  do_ready|  rst |
#|  G2      |  J2      |  L2      |  J1      |
```

(continues on next page)

(continued from previous page)

```
#+----------+----------+----------+----------+
#|  din0    |  din2    |  din1    |  do_valid|
#|  G3      |  H2      |  K2      |  H1      |
#+----------+----------+----------+----------+

#JXADC
#+----------+----------+----------+----------+
#|  dout1   |  dout3   |  di_ready|  dut_clk     |
#|  N2      |  M2      |  L3      |  J3      |
#+----------+----------+----------+----------+
#|  dout0   |  dout2   |  di_valid|          |
#|  N1      |  M1      |  M3      |  K3      |
#+----------+----------+----------+----------+
```

4.1.2 Board modification

- To make perfoming SCA attacks easier, all capacitors on the FPGA core voltage rail have been removed.
- The board is modified to be able to access the FPGA core voltage rail.
- To measere power, an inductive power porbe may be used (e.g. Tektronix CT-1).

4.2 NewAE CW305 (Artix7 FPGA)

The PMOD ports JA and JXADC in the Basys3 control board should be connected to 40-pin port JP3. The pin mapping in the NewAE CW305 is as follows:

```
######JP3
#+----------+----------+----------+----------+
#|          |  din3    |  do_ready|  rst     |
#|  NC      |  D16     |  E16     |  F12     |
#+----------+----------+----------+----------+
#|  din0    |  din2    |  din1    |  do_valid|
#|  D15     |  E15     |  E13     |  F15     |
#+----------+----------+----------+----------+

#+----------+----------+----------+----------+
#|  dout1   |  dout3   |  di_ready|  clk     |
#|  B12     |  A13     |  B15     |  C11     |
#+----------+----------+----------+----------+
#|  dout0   |  dout2   |  di_valid|          |
#|  A12     |  A14     |  A15     |  C12     |
#+----------+----------+----------+----------+
```

To measure the power, you can connect your oscilloscope to test poin X4. For more information, refer to the manufacturer web-site.

CHAPTER

FIVE

DUT ALGORITHM DEVELOPMENT

This document describes how to interface the DUT wrapper and the Function Core (victim). The Function Core (a.k.a victim), is the algorithm to be tested. The DUT wrapper is hardware that is instantaited on the same FPGA as the function core and used to communication to the control board. The function core is user provided. However, the DUT wrapper is included with FOBOS. The DUT Wrapper handles communication to the control board and includes FIFOs to store input and output data.

5.1 Data flow description:

Test vectors are sent form PC one at a time to the control board which stores them breifly. The control board starts sending the test vector to the DUT board through the interface described below. The DUT wrapper then puts data in the correct FIFOs (PDI, SDI and RDI). Once the DUT wrapper receives the start command from the controller, it de-assersts the function core reset signal and the function core will run and consume the data in the FIFOs. The output of the function core is stored in the DO fifo. Once the DO FIFO accumulates EXPECTED_OUTPUT bytes, the DUT wrapper will send this data to the control board which forwards it to the PC.

5.2 The DUT Wrapper <–> Function core interface

The protocol follows a simple AXI stream protocol. The 'valid' signals indicates data from source are valid and 'ready' signals indicates destination is ready to use data. When both 'valid' and 'ready' signals are set to logic 1, data is transfered. All the data signals shown in the listing below, are connected to the FIFOs PDI, SDI, RDI and DO.

The function core (victim) is instantiated as follows in the FOBOS_DUT.vhd file.

```
victim: entity work.victim(behav)
    -- Choices for W and SW are independently any multiple of 4, defined in generics␣
↪above
    generic map  (
        G_W           => W, -- ! pdi and do width (mulltiple of 4)
        G_SW          => SW -- ! sdi width (multiple of 4)
    )
    port map(
        clk => clk,
        rst => start,
        -- data signals
        pdi_data  => pdi_data,
        pdi_valid => pdi_valid,
        pdi_ready => pdi_ready,
        sdi_data => sdi_data,
        sdi_valid => sdi_valid,
```

(continues on next page)

(continued from previous page)

```
        sdi_ready => sdi_ready,
        do_data => result_data,
        do_ready => result_ready,
        do_valid => result_valid

    --  ! if rdi_interface for side-channel protected versions is required,
    --  ! uncomment the rdi interface
    --  ,rdi_data => rdi_data,
    --  rdi_ready => rdi_ready,
    --  rdi_valid => rdi_valid
);
```

The generic W is the PDI and DO width in bits. The generic SW is the SDI width.

It is highly recommended that the DUT is tested using the capture/dut/fpga_dut/fobos_dut_tb.vhd test bench and ensure that the output is valid. This testbench needs one test vector to be stored in the file dinFile.txt and generates doutFile.txt output file.

5.3 Dummy DUT Example

You can find an example dummy DUT in fobos/capture/dut/example_cores/dummy1. This dummy core is used to test FOBOS DUT. It simply echos back configurable number of words of the PDI sent in the test vector. The dummy core in the listing below, echos seven 8-bit words of the PDI from the test vector received from the DUT wrapper.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
entity dummy is
    Generic(
        N        : integer := 8;
        NUMWORDS : integer := 7
    );
    port(clk       : in  STD_LOGIC;
         rst       : in  STD_LOGIC;
         pdi_data  : in  STD_LOGIC_VECTOR(N - 1 downto 0);
         pdi_valid : in  STD_LOGIC;
         pdi_ready : out STD_LOGIC;
         sdi_data  : in  STD_LOGIC_VECTOR(N - 1 downto 0);
         sdi_valid : in  STD_LOGIC;
         sdi_ready : out STD_LOGIC;
         do_data   : out STD_LOGIC_VECTOR(N - 1 downto 0);
         do_valid  : out STD_LOGIC;
         do_ready  : in  STD_LOGIC
        );
end dummy;

architecture behav of dummy is
    type state is (IDLE, RUN);
    signal current_state             : state;
    signal next_state                : state;
    signal cnt_clr, cnt_en, cnt_done : std_logic;
    signal cnt, next_cnt             : std_logic_vector(15 downto 0);

begin
```

(continues on next page)

(continued from previous page)

```
    ctrl : process(clk)
    begin
        if (rising_edge(clk)) then
            if (rst = '1') then
                current_state <= IDLE;
            else
                current_state <= next_state;
            end if;

        end if;

    end process;

    comb : process(current_state, pdi_valid, sdi_valid, do_ready, cnt_done)
    begin
        -- defaults
        pdi_ready <= '0';
        sdi_ready <= '0';
        do_valid  <= '0';
        cnt_clr   <= '0';
        cnt_en    <= '0';

        case current_state is
            when IDLE =>
                cnt_clr <= '1';
                if pdi_valid = '1' and sdi_valid = '1' and do_ready = '1' then
                    next_state <= RUN;
                else
                    next_state <= IDLE;
                end if;

            when RUN =>
                if cnt_done = '1' then
                    next_state <= IDLE;
                else
                    if pdi_valid = '1' and sdi_valid = '1' and do_ready = '1' then
                        pdi_ready <= '1';
                        sdi_ready <= '1';
                        do_valid  <= '1';
                        cnt_en    <= '1';
                    end if;
                    next_state <= RUN;
                end if;

            when others =>
                next_state <= IDLE;

        end case;

    end process;
    --do_data <= pdi_data xor sdi_data;
    do_data <= pdi_data;

    count : process(clk)
    begin
        if (rising_edge(clk)) then
```

(continues on next page)

(continued from previous page)

```
            cnt <= next_cnt;
        end if;
    end process;
    next_cnt <= (others => '0') when cnt_clr = '1'
                else cnt + 1 when cnt_en = '1'
                else cnt;

    cnt_done <= '1' when (cnt = NUMWORDS) else '0';

end behav;
```

5.4 Generating the dummy DUT bitstream

This procedure describes how to genrate the bitstream for the dummy DUT. You don't need to perform this procedure to run the dummy example since the bitstream is already generated. However, this procedure aims to show how to instantiate a function core in FOBOS DUT wrapper.

1. Create a project in Vivado (or ISE) and add all vhdl files from fobos/capture/dut/fpga_dut (except half_duplex_du.vhd) and fobos/capture/dut/example_cores/dummy1.

2. Note that in FOBOS_DUT.vhd, the dummy dut is instantaited as follows:

```
victim: entity work.dummy(behav)

-- Choices for W and SW are independently any multiple of 4, defined in generics above

    generic map  (
        N           => 8,
        NUMWORDS         => 7
    )

port map(
    clk => clk,
    rst => start,  --! The FOBOS_DUT start signal meets requirements
                   --!for synchronous resets used in
                   --! CAESAR HW Development Package AEAD

-- data signals

    pdi_data  => pdi_data,
    pdi_valid => pdi_valid,
    pdi_ready => pdi_ready,

sdi_data => sdi_data,
    sdi_valid => sdi_valid,
    sdi_ready => sdi_ready,

    do_data => result_data,
    do_ready => result_ready,
    do_valid => result_valid

----! if rdi_interface for side-channel protected versions is required,
----! uncomment the rdi interface
--  ,rdi_data => rdi_data,
```

(continues on next page)

(continued from previous page)

```
--  rdi_ready => rdi_ready,
--  rdi_valid => rdi_valid

);
```

3. Note that the W and SW (PDI and SDI width) generics in FOBOS_DUT.vhd are set to 8.
4. Add the constain file CW305.xdc for NewAE CW305 (or Nexys3.ucf for Nexys3 DUT) from fobos/capture/dut/fpga_dut.
5. Generate the bitstream.
6. Find your bitstream file FOBOS_DUT.bit in the Vivado/ISE project folders.

5.5 Running the dummu DUT example (on Nexys3 DUT)

1. Make sure your hardware is setup properly and the DUT is connected to the control board.
2. Run the dummyDUTCapture.py script as follows:

```
$ cd path-to-fobos/software
$ python dummyDUTCapture.py
```

This script is preconfigured to use the fobos/workspace/DummyProject as a project folder. The folder includes a pre-generated bitstream file that FOBOS will use to program the Nexys3 DUT. This requires digilent Adept tool 'djtgcfg' to be installed and callable from the Linux shell. The project folder also includes a pre-generated test vector file 'dinFile.txt'.

CHAPTER

SIX

DATA ACQUISITION - PROCESSING DATA

After test vectors have been generated, user can run data acquisition (capture). The PC will send one test vector at a time to the control board, which sends it to DUT. The control board will trigger the oscilloscope to capture the trace. The process will be repeated until all traces are collected.

6.1 FOBOS control-DUT protocol

The control board receives test vectors from the PC one at a time. Then, it sends the vector to the DUT which uses the header information in the vector to put the data (plaintext, key etc.) into the correct FIFOs. The DUT wrapper then allows the victim algorithm to run by setting the victim reset to zero. The victim then drains the FIFOs (sdi and pdi FIFOs) and stores the output in the DO FIFO. Once the DO FIFO accumulates the expected amount of data, the DUT wrapper sends data to the controller which sends it to the PC.

The following diagram shows the components of FOBOS.

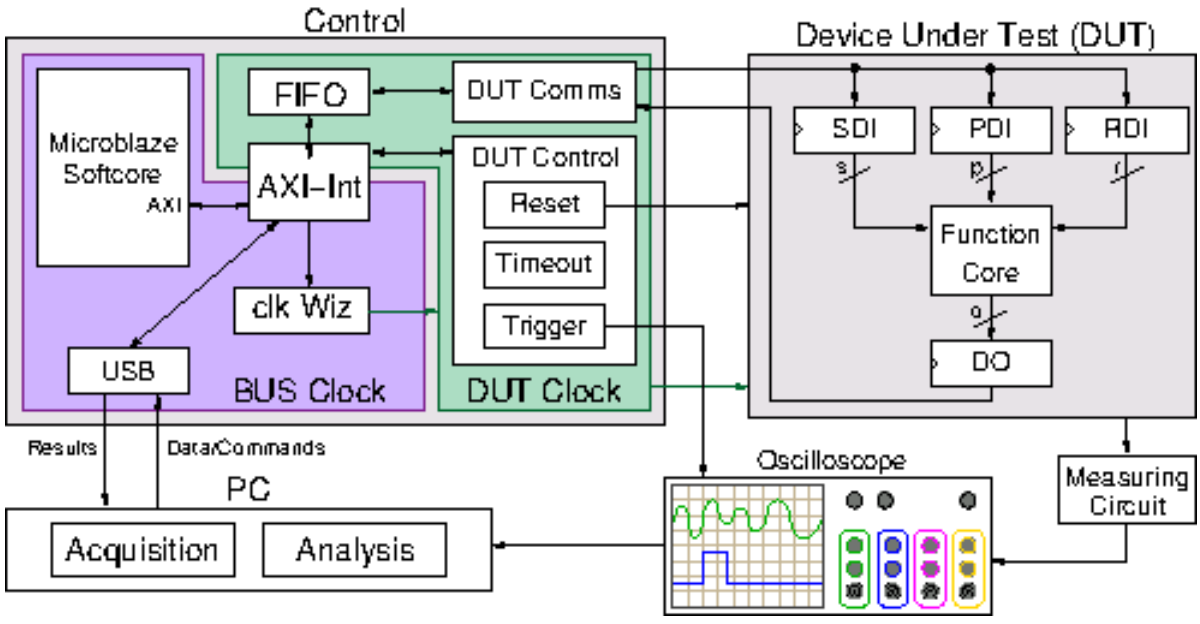

Fig. 1: FOBOS2 Block Diagram

6.2 A Basic Example

Below, we show an example of using the controller internal test feature. When enabled, the controller process the data and echo back the PDI without sending the data to the DUT.

```
import fobos
# Constants#################################################################
SERIAL_PORT = '/dev/ttyUSB1'
TRACE_NUM = 5
OUT_LEN = 7
# Instantiate FOBOS objects#################################################
print("Sending configuration...")
ctrl = fobos.Basys3Ctrl(SERIAL_PORT)
ctrl.enableTestMode()
ctrl.setOutLen(OUT_LEN)
# Run DUT operations
testVectors = ['00c0000761996dc996d4ac00c100070f7821507a22a00081000700800001',
               '00c00007fd8771fe717de400c100073e1fe5b4aa357c0081000700800001',
               '00c0000782051f5484702200c10007980d05d4ea25bc0081000700800001',
               '00c0000767881b702afe5200c10007b08a5e036de72b0081000700800001',
               '00c0000726a1d601ccdf7a00c1000773539e52672d5d0081000700800001']

print 'Sending data..'
traceNum = 0
while traceNum < TRACE_NUM:
    data = testVectors[traceNum]
    status, result = ctrl.processData(data, OUT_LEN)
    if status != fobos.OK:
        print "TIMEOUT"
    print(result)
    traceNum += 1
```

If you run the code above, the control board will echo the PDI as shown below.

```
$ cd path-to-fobos/software
$ python dummyCaptureBasic.py
Sending configuration...
f0030006000900000001
Status= 00000000
f0030006000000000007
Status= 00000000
Sending data..
f001001e00c0000761996dc996d4ac00c100070f7821507a22a00081000700800001
OK.    Status= 00000000
61 99 6d c9 96 d4 ac
f001001e00c00007fd8771fe717de400c100073e1fe5b4aa357c0081000700800001
OK.    Status= 00000000
fd 87 71 fe 71 7d e4
f001001e00c0000782051f5484702200c10007980d05d4ea25bc0081000700800001
OK.    Status= 00000000
82 05 1f 54 84 70 22
f001001e00c0000767881b702afe5200c10007b08a5e036de72b0081000700800001
OK.    Status= 00000000
67 88 1b 70 2a fe 52
f001001e00c0000726a1d601ccdf7a00c1000773539e52672d5d0081000700800001
OK.    Status= 00000000
26 a1 d6 01 cc df 7a
```

6.3 An Extended Example

The following example shows how to test the controller using the internal test feature. However, this time we show more features like setting the DUT clock and timeout etc.

```
import os
import shutil
import fobos
# Constants#####################################################
WORKSPACE = "../workspace/fobosworkspace"
PROJECT_NAME = "dummyProject"
DIN_FILE_NAME = "dinFile.txt"
CIPHER_FILE = "ciphertext.txt"
TRACE_FILE = "powerTraces.npy"
DUT_BIT_FILE = "FOBOS_DUT.bit"
SERIAL_PORT = '/dev/ttyUSB1'
TRACE_NUM = 5
DUT_CLk = 1
OUT_LEN = 7
TIMEOUT = 5
TRIG_WAIT = 1
TRIG_LENGTH = 1
TRIG_MODE_NORM = 0
TRIG_MODE_FULL = 1
TIME_TO_RST = 0
# Instantiate FOBOS objects#####################################################
ctrl = fobos.Basys3Ctrl(SERIAL_PORT)
ctrl.setDUTClk(DUT_CLk)
ctrl.enableTestMode()
ctrl.setTimeToReset(TIME_TO_RST)
ctrl.setOutLen(OUT_LEN)
ctrl.setTimeout(TIMEOUT)
ctrl.setTriggerWait(TRIG_WAIT)
ctrl.setTriggerLen(TRIG_LENGTH)
ctrl.setTriggerMode(TRIG_MODE_FULL)

# Configure project directories#####################################################
pm = fobos.ProjectManager()
pm.setWorkSpaceDir(WORKSPACE)
pm.setProjName(PROJECT_NAME)
projDir = pm.getProjDir()

tvFileName = os.path.join(projDir, DIN_FILE_NAME)
tvFile = open(tvFileName, "r")
captureDir = pm.getCaptureDir()
cipherFileName = os.path.join(captureDir, CIPHER_FILE)
cipherFile = open(cipherFileName, "w")
shutil.copy(tvFileName, captureDir)
# Get traces#####################################################
print 'Sending data..'
traceNum = 0
while traceNum < TRACE_NUM:
    data = tvFile.readline()
    status, result = ctrl.processData(data, OUT_LEN)
    if status != fobos:
        print "TIMEOUT"
    print(result)
```

(continues on next page)

(continued from previous page)

```
53      cipherFile.write(result + "\n")
54      traceNum += 1
55
56  tvFile.close()
57  cipherFile.close()
```

This script connects to the Basys3 controller, configure the controller and then prepares the input and output files. Once this setup is done, the script reads the input test vector file and sends one line at a time to the controller to process it. The controller sends the data to the DUT (in this example we use the internal dummy DUT).

Here is a line by line description of the code. lines 1-3 imports packages the we will need in the code. The fobos package is the package we need to have access to the classes we need to communicate with the controller and other useful tools.

Lines 22-30 configures the controller. we set the following parameters

- The DUT clock frequency is set to 1 MHz. This is the clock generated by the control board and used to clock the DUT.
- Test mode is enabled. In this mode, the controller uses its internal dummy DUT to process data. This DUT simply echoes PDI.
- Time-to-reset is set to 0. This disables this feature. If set to a number greater than zero, the DUT will be reset after that number of clock cycles.
- The controller timeout is set to 5 seconds. The control board will wait for 5 seconds for the DUT to respond before giving up.
- The trigger mode is set to Normal mode, trigger wait cycles to 1 and trigger length to 1. For more information about he trigger module, please refer to [ref trigger]

Lines 33-43 configures the workspace. The ProjectManager class is used to setup a simple directory structure to store input and output files. Each run of this scripts creates a new directory to store the output files. This script expects all data to be in WORKSPACE/PROJECT_NAME directory (created by the user). The user should also put the test vector file 'dinFile.txt' that directory. All results will be saved in the WORKSPACE/PROJECT_NAME/Capture. A new directory will be created for each new run. Also, different files are specified relative to the project directory.

Lines 46-56 reads the test vector file one line at a time and sends it to the control board which processes it and sends back the result.

CHAPTER

SEVEN

DATA ACQUISITION - COLLECTING TRACES

This section explains how to use PicoScope to collect traces. This is best explained by example.

7.1 PicoScope Setup

This example code uses the Picoscope 5000 series (5244D) oscilloscope. To setup the oscilloscope please refer to the manufacturer website. We expect that modifying the code to work with other series to be easy since the manufacturer provides an SDK.

7.2 Power Measurement (Nexy3 DUT)

The oscilloscope must be able to measure a voltage propotional to the power consumed by the device. You can use a current probe (e.g Tektronix CT-1). We modified the DUT board so that we have a jumper on the power line (core FPGA voltage). We then used a bench power supply to power the FPGA (1.2V). Note that the power wire should go through the current probe. The power probe senses the current variations in the power wire and coverts that to volatge for the oscilloscope.

An alternative method is to insert a shunt resistor in the power line and then measure the volatge drop across the resistor.

7.3 Trace Collection Example

```
import numpy as np
import fobos
# Constants#################################################################
SERIAL_PORT = '/dev/ttyUSB1'
TRACE_NUM = 5
OUT_LEN = 7
# Instantiate FOBOS objects#################################################
print("Sending configuration...")
ctrl = fobos.Basys3Ctrl(SERIAL_PORT)
ctrl.enableTestMode()
ctrl.setOutLen(OUT_LEN)
# open file to save traces.
traceFile = open('powerTraces.npy', 'w')
# configure oscilloscope
scope = fobos.picoscope.Picoscope(sampleResolution = 8, # number of bits for each␣
↪sample.
```

(continues on next page)

(continued from previous page)

```
16                      postTriggerSamples = 1000 # samples in one trace.
17                      )
18  scope.setChannel(channelName = 'CHANNEL_A', rangemv = '100mV')
19  scope.setSamplingInterval(samplingIntervalns = 2) # T=2 ns, Sampling rate= 500MHz
20  scope.setTrigger(channelName ='EXTERNAL', direction = 'RISING_EDGE',
21                   thresholdmv = 200)
22  scope.setDataBuffers()
23
24  # Run DUT operations
25  testVectors = ['00c0000761996dc996d4ac00c100070f7821507a22a00081000700800001',
26              '00c00007fd8771fe717de400c100073e1fe5b4aa357c0081000700800001',
27              '00c0000782051f5484702200c10007980d05d4ea25bc0081000700800001',
28              '00c0000767881b702afe5200c10007b08a5e036de72b0081000700800001',
29              '00c0000726a1d601ccdf7a00c1000773539e52672d5d0081000700800001']
30
31  print 'Sending data..'
32  traceNum = 0
33  while traceNum < TRACE_NUM:
34      scope.arm() # arm scope. Now, it expects a trigger at any time.
35      data = testVectors[traceNum]
36      status, result = ctrl.processData(data, OUT_LEN)
37      if status != fobos.OK:
38          print "TIMEOUT"
39      print(result)
40      trace = scope.readTrace() # get trace from oscilloscope buffers.
41      np.save(traceFile, trace) # save the trace in the trace file.
42      traceNum += 1
```

This is very similar to the capture data example but with few more lines added to capture traces from the oscilloscope.

Lines 15-22 configure the oscilloscope settings. We first intantiate a Picoscope object (line 15). Lines 15 sets sampleResolution to 8 bit and number of samples to be collected to 1000 samples per trace. Line 18 enables channel A and sets its range to 100 mV. Line 19 sets the sampling interval (T) to 2 nano seconds. This implies that ampling rate is 1/T = 500 M Sample/s. Line 20 configers the EXTERNAL (Ext) channel to be used for trigger on risinge edge with a threshold of 200 mV. Trigger channel must be connected to the control board trigger output. Line 22 allocates memory buffers to store oscilloscope data.

In the main loop, we use the arm() (line 34) function to tell the oscilloscope to be ready to get the trigger and store the data. Once the crypto operation is over, we can collect the trace from the memory as a numpy array using the readTrace() method (line 40). The trace is saved into a file in line 41. For more information about the PicoScope class see *Picoscope*.

CHAPTER

EIGHT

CONTROL BOARD FEATURE

8.1 Trigger Settings

The controller can send a trigger to the Oscilloscope once the DUT starts processing the data (ie. di_ready = 0). Or it can be configured to trigger any number of clock cycles after this event occurs.

- TRIGGER_WAIT_CYCLES : The number of clock cycles after which the trigger is asserted (after di_ready goes to zero).
- TRIGGER_LENGTH_CYCLES : The time the trigger signal is asserted.
- **TRIGGER_TYPE** [possible values: TRG_NORM | TRG_FULL | TRG_NORM_CLK | TRG_FULL_CLK]
  - TRG_NORM : normal trigger mode. in this mode the TRIGGER_WAIT_CYCLES and TRIGGER_LENGTH_CYCLES are applied.
  - TRG_FULL : Full trigger mode. While DUT is running (between di_ready = 0 and do_valid = 1) the trigger is asserted.
  - TRG_NORM_CLK : same as TRG_NORM but the trigger signal is anded with the clock.
  - TRG_FULL_CLK : same as TRG_FULL but the trigger signal is anded with the clock.
- **CUT_MODE** [Controls how the trace retrieved from the scope will be processed. Possible values: FULL | TRIG_HIGH]
  - FULL : The trace is cut starting at the rising edge of the trigger to the end of the screen.
  - TRIG_HIGH : the trace is cut from the rising edge to the falling edge of the trigger ie. the trace where the trigger is high will be saved.

8.2 Setting DUT Clock

The control board can provide a clock for the DUT ranging from 400 KHz to 100 MHz. The default value is 5 MHz. To set it, use the following method:

ctrl.setDUTClk(clkValue).

8.3 DUT Reset Feature

In some cases, the control board may need to reset the DUT because the interesting part of the victim algorithm has already executed. This is specifically valuable for ciphers that take a long time to complete. In this case, the cipher runs for a configurable number of clock cycles and then is reset without waiting for it to complete. This helps reduce

acquisition time. The number of cycles is counted after di_ready goes to 0. Note: When you use this feature, no output is returned from the DUT. To set it use the following command and set TIME_TO_RESET to any number other than zero. This number is set to zero by default which disables this feature.

ctrl.setTimeToReset(TIME_TO_RST)

8.4 Timeout Setting

In some cases, due to communication error or DUT non-responsiveness the control board sends a timeout error message to the control PC when a configurable time has elapsed. The default value is 5 seconds which is enough for almost all cases. Once timeout is reached, the control board resets the DUT, clears any pending DUT data transfers and return the a timeout status to the capture software. To set the timeout value, use the method:

ctrl.setTimeout(TIMEOUT)

8.5 Port mapping

Below, we show how the pins on the Basys3 PMOD ports are assinged.

```
#JA
#+----------+----------+----------+----------+
#|          |  din3    |  do_ready|  rst     |
#|  G2      |  J2      |  L2      |  J1      |
#+----------+----------+----------+----------+
#|  din0    |  din2    |  din1    |  do_valid|
#|  G3      |  H2      |  K2      |  H1      |
#+----------+----------+----------+----------+

#JXADC
#+----------+----------+----------+----------+
#|  dout1   |  dout3   |  di_ready|  dut_clk |
#|  N2      |  M2      |  L3      |  J3      |
#+----------+----------+----------+----------+
#|  dout0   |  dout2   |  di_valid|          |
#|  N1      |  M1      |  M3      |  K3      |
#+----------+----------+----------+----------+

#JC
#+----------+----------+----------+------------+
#|          |          |          | trigger_out|
#|          |          |          |            |
#+----------+----------+----------+------------+
#|          |          |          |            |
#|          |          |          |            |
#+----------+----------+----------+------------+
```

CHAPTER

NINE

RUNNING CORRELATION POWER ANALYSIS

One of the most used variants of Deferential Power Analysis (DPA) is Correlation Power Analysis (CPA). In this document, we show the theory behind CPA before showing concrete examples. This discussion follows the notation and concepts discussed in the book "Power Analysis Attacks - Revealing the secrets of Smart Cards" by Mangard, Oswald and Popp.

9.1 How CPA Works?

Correlation Power Analysis uses an intermediate value that is a function of part of the key and known data. The power consumption of the devices when the intermediate value is processed is estimated for each key guess. A statistical method is then used to find out which key was most likely used by correlating the hypothetical power and the real power consumption. below we discuss this process in detail.

9.1.1 Step 1: Choose the intermediate value

We choose an intermediate variable that is processed in the algorithm. The intermediate value is calculated as *f(d, k)* Where *d* is a known non-constant value that can be derived from known data (e.g. plain text) and *k* is small part of the key.

9.1.2 Step 2: Power measurement

Measure the power consumption of the crypto device while it encrypts/decrypts D data blocks. We need to know the value *d* the corresponds to each data block. These values can be written as a vector $\mathbf{d} = [d_1, d_2, \ldots, d_D]$. The power consumption signal for a single encryption/decryption operation is called a trace. A trace is vector that records instantaneous power consumption for the time of interest (the time when the intermediate value is being processed). The trace generated while encrypting/decrypting data block *i* consists of *T* samples and can be viewed as a vector $\mathbf{t}_i = [t_{i,1}, t_{i,2}, \ldots, t_{i,T}]$. The traces are stacked in a matrix **T** with dimensions *D x T* where each row i is a trace generated while encrypting/decrypting block *i*.

9.1.3 Step 3: Calculate hypothetical intermediate

Next, we need to guess the key part that goes into the calculation of the intermediate value. We list all possible sub keys as a vector $\mathbf{k} = [k_1, k_2, \ldots, k_K]$. Then, we calculate the intermediate value *f(d, k)* for all values in the vectors **d** and **k**. The result is stored in a *D x K* matrix **V**. Where

$\mathbf{V}_{i,j} = f(d_i, k_j)$

and *i = 1, 2, ... D and j = 1, 2 ... K*

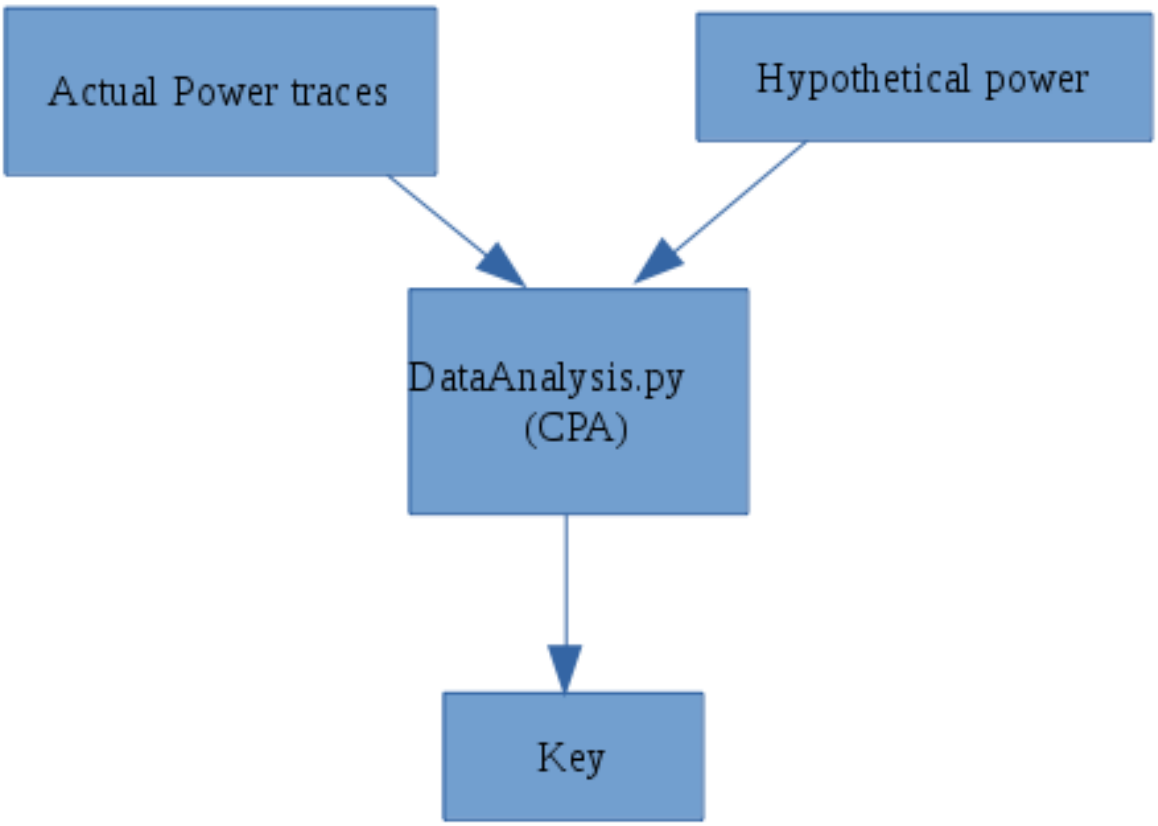

Fig. 1: CPA Flow

Note that each column in **V** is the intermediate value calculated for all data values **d** for one key guess.

9.1.4 Step 4: Calculate Hypothetical Power

Now we have calculated the intermediate values matrix **V**, we estimate the power consumption when each value in **V** is processed in the device. Two power models are widely used:

1. The Hamming Weight model (HW). this model counts the ones in the value e.g HW(1100 0000) = 2
2. The Hamming distance model (HD). This model counts the the number of bits that flips when a the value of a variable (e.g. register) changes. That is

HD(x, y) = HW(x *xor* y)

e.g HD(0000 0011, 0000 0101) = HW(0000 0011 *xor* 0000 0101) = HW(0000 0110) = 2

The result is a *D x K* matrix called **H**. These are the same dimensions as the matrix **V**.

9.1.5 Step 5: Correlate the hypothetical power and the real power traces

We correlate H and T to find the key. The question is: which column in **V** was most likely processed in the device? We use a correlation algorithm that takes two vectors as input and returns a real number that measures how 'similar' the vectors are. Each column *i* in **H** is compared with column *j* in the trace matrix **T**. The result is stored in the correlation matrix **R** which is a *K x T* matrix. Note that column *i* in matrix **H** is the hypothetical power if $\mathbf{k}_i$ is used in the device and column *j* in **T** is the real power measurement at sample *j* for all data blocks.

The element $\mathbf{R}_{i,j}$ measures how similar the column *i* in **H** to column *j* in **T**. The index of the highest element in the matrix **R** *ck*, *ct* are the in index of the key and the sample in time of the supposedly correct key since it indicates that the corresponding columns in **H** and **T** are similar so it is likely that the guessed key was indeed used in the device.

CHAPTER

TEN

AES CPA EXAMPLE

This section describes a Correlation Power Analysis (CPA) attack of an implementation of the Advanced Encryption Standard (AES) using FOBOS. AES is an symmetric-key cipher used ex- tensively in security sensitive applications world wide. AES applies four different transformations, SubBytes, ShiftRows, MixColumns, and AddRoundKey, per round and iterates through several such rounds depending upon the key size. An intermediate key called "round key" is generated and used per round which is derived from the original key through a reversible key scheduling function. We have implemented a basic iterative architecture of AES with 128-bit key length and 128-bit wide datapath requiring 11 clock cycles for one encryption. Key scheduling is done on-the-fly and the SubBytes function is realized through look-up-tables. The block diagram for this design is shown in the figure below:

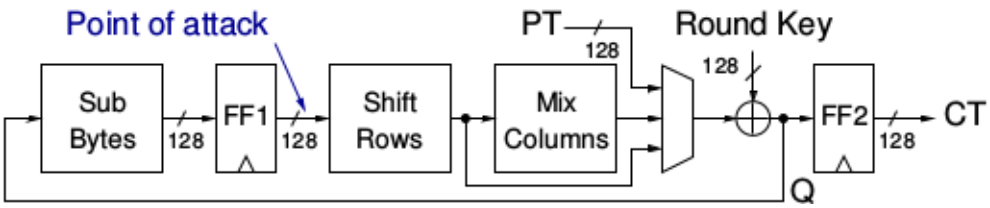

Fig. 1: AES-128 implementation

We attack our AES design during the first round at the output of the register FF1 (see the figure above). The equation for calculating the Hamming Distance (HD) is shown in the eqation below. We use Pearson's Correlation to correlate the instantaneous power consumption with the HD model.

H(i,j) = HD(SBOX(CT(i)), SBOX(kguess(j) xor PT(i+1)))

For trace collection, we used the Basys3 control board and connected a NewAE CW305 DUT (Artix7 FPGA). The DUT clock was set to 1 MHz and the oscilloscope sampling rate was set to 62 M Sample/sec. We then collected 500K traces. The formula above was used to generate the hypothetical power matrix.

Below is the script used for the CPA analysis. This script is at fobos/software/cpaExample.py and the example files are stored in fobos/workspace/aes_artix7_picoscope :

```
import os
import numpy as np
import fobos.traceset as traceset
import fobos.cpa as cpa
import fobos.postprocess as postprocess
import fobos.projmgr as projmgr
import fobos.powermodels.AESFirstLast as powermodel

def main():
    # Configure project directories#################################
    WORKSPACE = "../workspace/aes_artix7_picoscope/capture"
    PROJECT_NAME = "aes_artix_pico_62MSps"
```

(continues on next page)

(continued from previous page)

```
13      pm = projmgr.ProjectManager()
14      pm.setWorkSpaceDir(WORKSPACE)
15      pm.setProjName(PROJECT_NAME)
16      projDir = pm.getProjDir()
17      analysisDir = pm.getAnalysisDir()
18      ######
19      TRACES_FILE = os.path.join(projDir, 'powerTraces.npy')
20      PLAIN_FILE = os.path.join(projDir, 'plaintext.txt')
21      CIPHER_FILE = os.path.join(projDir, 'ciphertext.txt')
22      HYPO_FILE = os.path.join(projDir, "hypotheticalPower.npy")
23
24      CROP_START = 218
25      CROP_END = 282
26
27      NUM_TRACES = 220000
28      MTD_STRIDE = 10000
29      traceSet = traceset.TraceSet(traceNum=NUM_TRACES,
30                                   fileName=TRACES_FILE,
31                                   cropStart=CROP_START,
32                                   cropEnd=CROP_END)
33
34      measuredPower = traceSet.traces
35      hypotheticalPower = powermodel.getHypotheticalPower(PLAIN_FILE,
36                                                          CIPHER_FILE,
37                                                          NUM_TRACES)
38      # hypotheticalPower = np.load(HYPO_FILE)
39      cpaAttacker = cpa.CPA()
40      C = cpaAttacker.doCPA(measuredPower=measuredPower,
41                            hypotheticalPower=hypotheticalPower,
42                            numTraces=NUM_TRACES,
43                            analysisDir=analysisDir,
44                            MTDStride=MTD_STRIDE
45                            )
46
47
48  if __name__ == '__main__':
49      main()
```

In line 7 we import the power model module that will calcualte the hypothetical power matrix. This module was used in line 35. After we set the directories and specify input files, we set CROP_START and CROP_END which selects the first round of the encryption in the power trace. We set the number of traces to be used in the attack using NUM_TRACES variable and set the step used in plotting the MTD graph using the variable MTD stride.

Once all setting are complete, we call the doCPA() method in line 40. This method takes the measuredPower matrix, the hypotheticalPower matrix, the number of traces, the directory used to store results and the MTD stride.

Below, we show the correlation graph and the MTD graph for the first byte of the key.

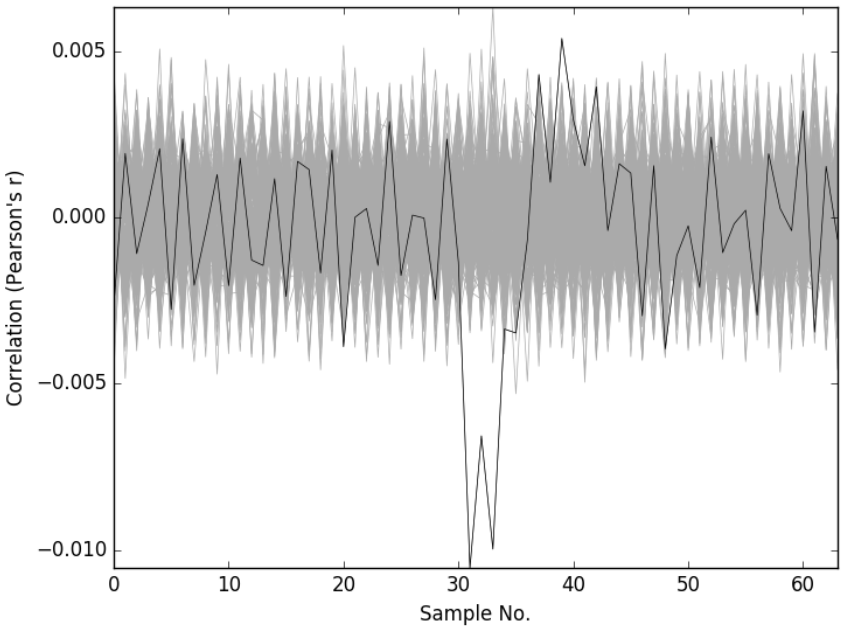

Fig. 2: CPA Correlation for key byte-0

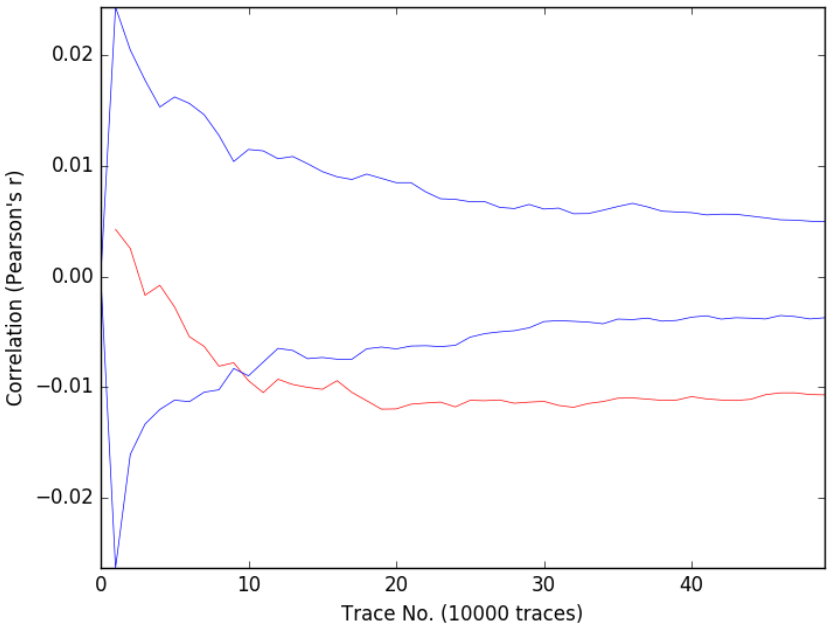

Fig. 3: CPA MTD graph for key byte-0

CHAPTER

ELEVEN

T-TEST LEAKAGE ASSESSMENT

11.1 T-test Flow

Welch's T-test is used as a tool for leakage assessment. This guide describes using FOBOS to perform a fixed-vs-random t-test.

To perform a t-test you need to generate test-vectors that interleaves fixed and random traces. FOBOS data acquisition is run to get the traces, then the traces are splitted into two sets and the statistical Welch's t-test is performed.

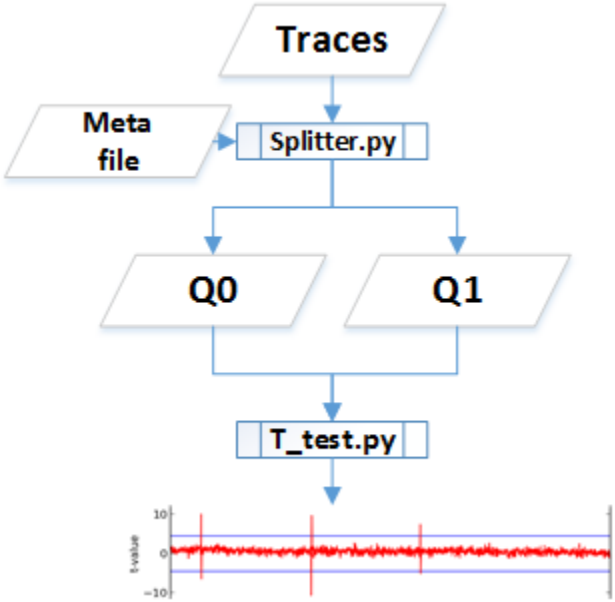

Fig. 1: T test flow

11.2 Test vector generation

The user generates the test-vector file and a meta file that specifies which test vectors are random and which are fixed. Fixed-vs-random t-test uses interleaved fixed and random test vectors. We can select a fixed test vector D and create a set of test vectors that interleaves D and a randomly selected test vector. The interleaving is random. For example the following test vector can be used to perform a t-test on an algorithm implemented on FOBOS DUT.

```
00C00160700000002000000D600000CB0B1B2B3B4B ...
00C00160700000002000000D600000C0EEB8882501 ...
00C00160700000002000000D600000CF402296C454 ...
00C00160700000002000000D600000CB0B1B2B3B4B ...
00C00160700000002000000D600000CE919E2FC539 ...
00C00160700000002000000D600000CB0B1B2B3B4B ...
00C00160700000002000000D600000CB0B1B2B3B4B ...
.
.
```

The corresponding fvrchoicefile specifies which traces are random '1' and which are fixed '0'. For example the fvrchoicefile that corresponds to the test-vectors above is

```
0110100 ...
```

See *Test Vector Generation* for more information.

11.3 Peforming a t-test

Once traces have been collected, scripts are used to perform the t-test.

1- Copy the *cleanTraces.py*, *splitter.py* and *t-test2.py* scipts from fobos/software/tools/t-test to the locatation where you have the power traces you collected.

2- Make sure you have your trace file named *powerTraces.npy* and your fvr choice file named *fvrchoicefile.txt*. These files need to be at the same directory as the scripts copied in step 1.

3- Convert the power traces format using the read_traces.py. Blow, you can find the parameters used with this script.

```
$ python read_traces.py -h
###### readtraces.py script
usage: read_traces.py [-h] source_file destination_file num_of_traces

positional arguments:
source_file       Traces from FOBOS dataAcquisition
destination_file  .npy file that store traces as MxN Nupmy array.
num_of_traces     .npy file that store traces as MxN Nupmy array.

optional arguments:
-h, --help        show this help message and exit
```

For exmple, the following command takes the powerTraces.npy file and coverts it to cleanTraces.npy and it uses only the first 2000 traces.

```
python read_traces.py powerTraces.npy cleanTraces.npy 2000
```

4- Next, we split the cleanTraces.npy file into two trace files

- *traces0.npy* for the traces that came from the fixed test vector
- *traces1.npy* for traces that came from using the random test vectors.

To do this, splitter.py needs the fvrchoicefile.txt file. It also needs to know how many traces to consider. These are configurable parameters in the script. Here is all the parameters you can edit in splitter.py:

```
######Parameters
MAX_TRACE=2000
traces0File = 'traces0.npy'
traces1File = 'traces1.npy'
fvrFile = 'fvrchoicefile.txt'
cleanTraceFile = 'cleanTraces.npy'
##########################
```

Once you configure the parameters, you can run splitter using:

```
python splitter.py
```

The reuslt will be 2 files called traces0.npy and traces1.npy.

5- The two files are now ready to be fed into the *t-test2.py* script to perform the t-test. Here are the parameters that this script needs:

```
$ python t-test2.py -h
usage: t-test2.py [-h] trace_file0 trace_file1 plot_file

positional arguments:
trace_file0  .npy file that store traces as MxN Nupmy array
trace_file1  .npy file that store traces as MxN Nupmy array
plot_file    File name to store the output figure

optional arguments:
-h, --help   show this help message and exit
```

Before running the script, also edit the plotting parameters in the script. These parameters specify the limits on the x and y axis.

```
# Parameters ########
start_ylim = -40
end_ylim = 40
start_xlim = 0
end_xlim = 1000
#####################
```

This limits the t-values to -40-40 and displays samples from 0 to 1000 on the x-axis.

Once you configure all the parameters, you can run the t-test using the following command.

```
python t-test2.py traces0.npy traces1.npy result_t_test.png
```

The output will be a file called result_t_test.png that shows the t-values on the y-axis and samples in the x-axis. Here is sample file that shows a failing test and a successful one:

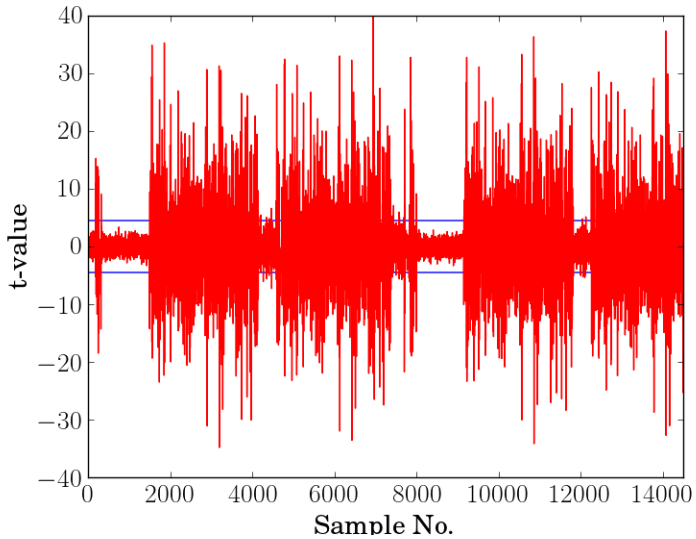

Fig. 2: T test on an unprotected AES-GCM implementation.

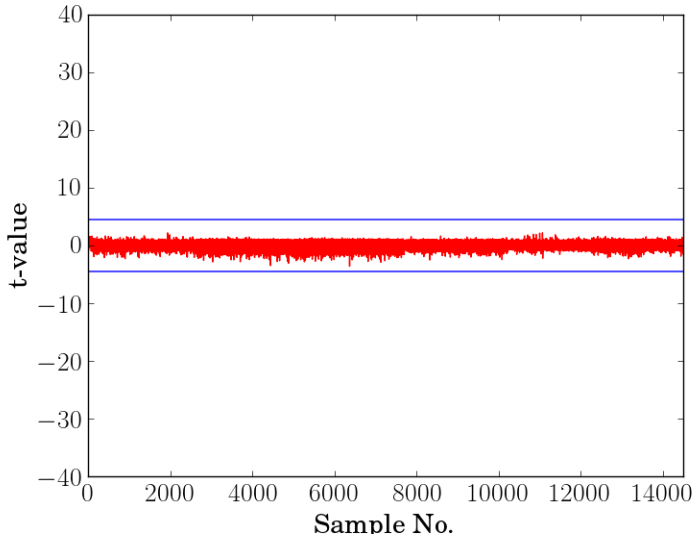

Fig. 3: T test on a protected AES-GCM implementation.

CHAPTER

TWELVE

CHI-SQUARED TEST LEAKAGE ASSESSMENT

The chi-squared test can be used with the t-test to assess leakage. The chi-squared script that is provided uses 'random' and 'fixed' classes so the inputs to the script are the same as the ones used for t-test2.py. For more information, please refer to the T-test page. The script is named x-test3.py and is located in fobos/software/tools/x-test. To run the test, use the following command.

```
python x-test3.py traces0.npy traces1.npy result_x_test.png
```

Where all the input files are similar to the t-test inputs.

Below, we show a chi-sqaured test that show a leakage and one that does not. In these two graphs we choose a threshold of 10^-5 and reason that the implementation is leaking if p-values are below the threshold.

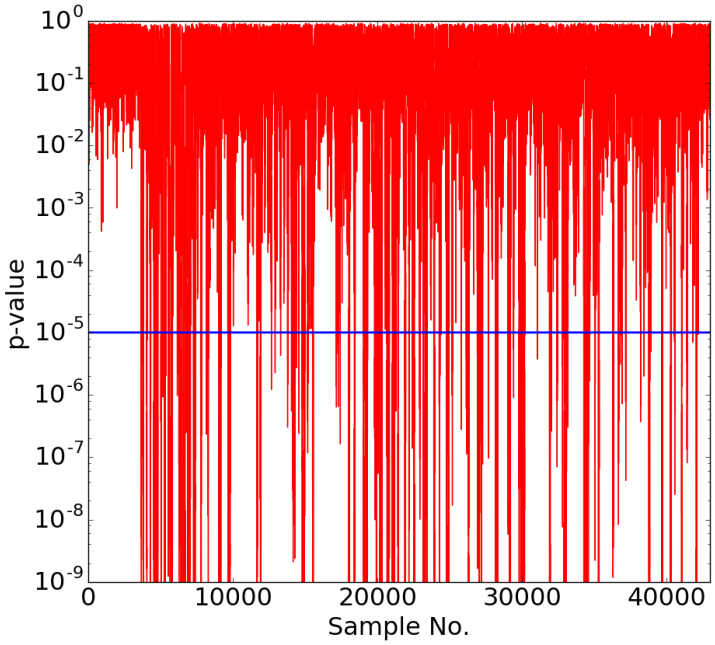

Fig. 1: Chi-squared test on Unprotected Ascon

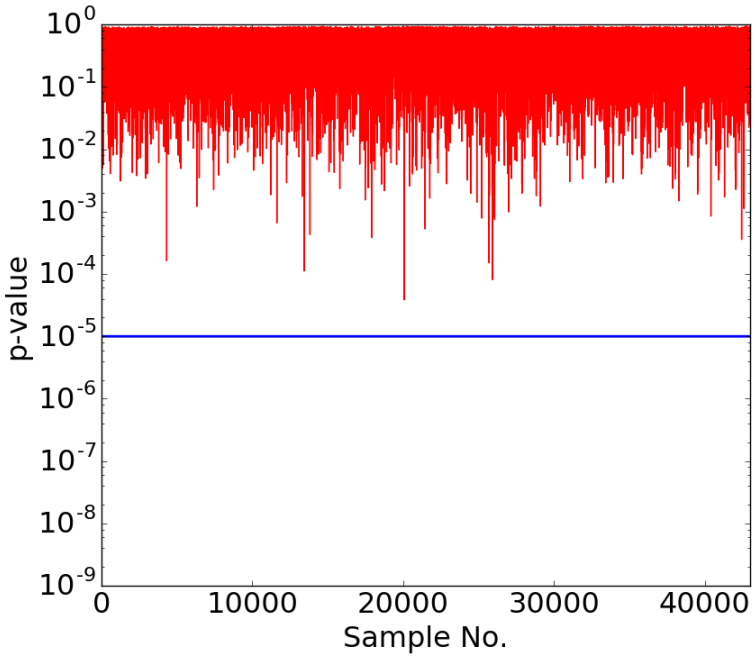

Fig. 2: Chi-squared test on Protected Ascon

CHAPTER

THIRTEEN

API REFERENCE

Here we provide documentation for important classes and methods.

13.1 Basys3Ctrl Class (controller)

class fobos.**Basys3Ctrl**(*self*, *port*, *baudRate=115200*, *dummy=False*)

Class to interface with Basys3 controller.

Parameters

- **port** (*str*) – The serial port where the Basys3 board is connected(e.g /dev/ttyUSB1).
- **baudRate** (*int*) – Baud rate. Default is 115200.
- **dummy** (*bool*) – When set to true, no communication with Basys3 is done. This is to test the software only. Default is False.

processData(*self*, *data*, *outLen*)

Sends data to FOBOS hardware for processing, e.g. encryption

Parameters **data** (*str*) – The data to be processed. This is a hexadecimal string.

Rtype (str, str)

Returns (Status ("00000000" OK | "20000000" TIMOUT) , Result from DUT (i.e. ciphertext))

setOutLen(*self*, *outLen*)

set DUT Expected Output Length (result length) in bytes.

Parameters **outLen** (*int*) – The number of output bytes to be returned by the DUT (e.g 16 for AES-128).

Rtype int

Returns Status.

setTriggerWait(*self*, *trigWait*)

Set number of trigger wait cycles

Parameters **int** (*trigWait*) – The number of DUT cycles after di_ready goes to 0 to issue the trigger.

Return type int

Returns Status

setTriggerMode(*self*, *trigType*)

Set trigger mode.

param int trigType. Possible values (TRG_NORM = 0 | TRG_FULL = 1 | TRG_NORM_CLK = 2 | TRG_FULL_CLK = 3)

rtype int

return Status

setTimeToReset (*self*, *dutCycles*)
set number of clock cycles after di_ready goes to 0 to reset the DUT.
Parameters **dutCycles** (*int*) – The number of DUT cycles to reset the DUT.
Return type int
Returns Status.

setTimeout (*self*, *seconds*)
set number of seconds to stop waiting for DUT result.

Parameters **seconds** (*in*) – Time in seconds. Range 1-40 seconds.

Return type int

Returns Status

enableTestMode (*self*)
Enable test mode. When this mode is enabled, the controller sends test-vectors to its internal dummy DUT.

Return type int

Returns Status

disableTestMode (*self*)
Disable test mode. In this mode the ctrl board uses the real DUT. This is the default mode.

Return type int

Returns Status

forceReset (*self*)
Reset DUT and clear current communication with DUT.

Return type int

Returns Status

releaseReset (*self*)
Release DUT reset signal

Return type int

Returns Status

setDUTClk (*self*, *clkFreqMhz*)
Set DUT clock frequency generated by the control board. Range is between 0.4 MHz - 100 MHz.

Parameters **clkFreqMhz** (*float*) – The DUT clock frequency in Mhz.

Return type int

Returns Status

13.2 Scope (Picoscope Class)

class Picoscope (*self*, *sampleResolution=8*, *preTriggerSamples=0*, *postTriggerSamples=1000*)
A class to interface with the Picoscope oscilloscope.

__init__ (*self*, *sampleResolution=8*, *preTriggerSamples=0*, *postTriggerSamples=1000*)
Open oscilloscope

Parameters

- **sampleResolution** (*int*) – The number of bits used to represent each sample. Possible values 8, 12, 14, 15 and 16.
- **preTriggerSamples** (*int*) – The number of samples before the trigger to return to the user.
- **postTriggerSamples** (*int*) – The number of samples after the trigger to return to the user.

setChannel (*self*, *channelName='CHANNEL_A'*, *coupling='DC'*, *rangemv='1V'*)
Configure voltage range and coupling for a channel.

Parameters

- **channelName** (*str*) – The name of the oscilloscope channel (CHANNEL_A | CHANNELB).
- **coupling** (*str*) – Select DC/AC coupling (DC | AC)
- **rangemv** (*str*) – The voltage range for the selected channel (10mV|20mV|50mV|100mV|200mV|500mV|1V|2V|5V|10V|20V)

setSamplingInterval (*self*, *samplingIntervalns*)
Sets the sampling interval (time between samples T) in nano seconds. Sampling rate = 1/T.

Parameters **samplingInterval** (*int*) – Sampling interval (time between samples T) in nano seconds. Sampling rate = 1/T.

setTrigger (*self*, *channelName= 'CHANNEL_A'*, *thresholdmv = 500*, *direction = 'RISING_EDGE'*, *autoTriggerDelay = 2000*)
Configure trigger channel.

Parameters

- **channelName** (*str*) – The channel to use as a trigger channel (CHANNEL_A | CHANNELB | EXTERNAL).
- **thresholdmv** (*int*) – The minimum level of voltage to assert a trigger.
- **direction** (*str*) – The trigger signal direction (RISING_EDGE| FALLING_EDGE).
- **autoTriggerDelay** (*int*) – Time in milliseconds after which auto trigger occurs (i.e start storing data even if no valid trigger received).

setDataBuffers ()
Allocates memory buffers to store oscilloscope data.

13.3 ProjectManager Class

class ProjectManager
A class to facilitate organizing and saving results files/data files in automatically created directories. For example, when new capture is performed a directory named attempt-[number] will be created in the capture directory.

setWorkSpaceDir (*self*, *workSpaceDir*)
Sets the directory where all data will be saved.

Parameters **workSpaceDir** (*str*) – Workspace directory path.

setProjName()
Sets the project name.

Parameters **projectName** (*str*) – Project name. A directory will be created in the workspace with the project name.

getProjDir()
Gets the project directory path.

Returns Full project directory path.

getCaptureDir()
Creates and returns the path of a new directory created in *worksapce/projectName/capture/* the folder will be named *attempt-(number)* for uniqueness.

13.4 Nexy3DUT Class

class Nexys3DUT
A class to program Digilent Nexy3 DUT. This class requires that Digilent Adept command line tools (djtgcfg command) is installed.

setBitFile(*bitFile*)
Sets the bit file name used to program the DUT FPGA.

Parameters **bitFile** (*str*) – Full path of the .bit file to program the FPGA.

program()
Programs the FPGA using the bit file specified using he setBitFile method.

13.5 CPA Class

class CPA
A class to perform Correlation Power Analysis.

doCPA(*self*, *measuredPower*, *hypotheticalPower*, *numTraces*, *analysisDir*, *MTDStride*)
Perform CPA.

Parameters

- **measuredPower** (*numpy_array*) – The power measure using the oscilloscope.
- **hypotheticalPower** (*numpy_array*) – The hypotheticalPower estimated using the power model.
- **numTraces** (*int*) – Number of traces in to be analyzed.
- **analysisDir** (*str*) – Full path of the directory to store analysis results.
- **MTDStride** (*int*) – The number of traces to add in each step when plotting the MTD graph.

13.6 DataGenerator Class

class DataGenerator
A class to generate FOBOS-ready test vectors.

randTVFile(*self*, *pdiLen*, *sdiLen*, *rdiLen*, *outLen*, *fileName*, *numTVs*)

Generate a FOBOS-ready test-vector file.

Parameters

- **pdiLen** (*int*) – The lenght of PDI data (e.g plaintext) in bytes.
- **sdiLen** (*int*) – The lenght of SDI data (e.g secret key) in bytes.
- **rdiLen** (*int*) – The lenght of RDI data (random data) in bytes.
- **outLen** (*int*) – The lenght of expected output (e.g ciphertext) in bytes.
- **fileName** (*str*) – The name of the file to store the testvectors.

randTestVector(*self*, *pdiLen*, *sdiLen*, *rdiLen*, *outLen*)

Generate a single FOBOS-ready test-vector.

Parameters

- **pdiLen** (*int*) – The lenght of PDI data (e.g plaintext) in bytes.
- **sdiLen** (*int*) – The lenght of SDI data (e.g secret key) in bytes.
- **rdiLen** (*int*) – The lenght of RDI data (random data) in bytes.
- **outLen** (*int*) – The lenght of expected output (e.g ciphertext) in bytes.

CHAPTER

FOURTEEN

LICENSE

Apache License Version 2.0, January 2004 http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

   "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the

purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service

marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

> To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

> http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

CHAPTER

FIFTEEN

INDICES AND TABLES

- genindex
- search

INDEX

Symbols

C

D

E

F

G

N

P

R

S