Tutorial 4: Interfacing the Sakura-G board with Python

The Sakura-G board[1] allows to analyze the leakage of a hardware design. It consists of two FPGAs (Spartan-6 XC6SLX9 and XC6SLX75) and low-noise power amplifier. In one of the FPGAs we load the target implementation and from the other FPGA (i.e. the control FPGA) we receive inputs from outside such as a plaintext or an encryption key that are then sent to the main FPGA for performing a certain operation e.g. a block cipher encryption. During this process, power traces can be extracted by connecting the the output of the low-noise power amplifier to an oscilloscope.

In this tutorial we will see how to interface the Sakura-G board with Python. The goal is to have more flexibility so we can load our own designs and interface the Sakura-G board with an oscilloscope and retrieve power traces. By default, the Sakura-G board comes with an implementation of the AES-128 block cipher that we will use in this tutorial to demonstrate how to interact with the board. The first part of this tutorial is partially based on the *SAKURA-G Quick Start Guide*, available at `http://satoh.cs.uec.ac.jp/SAKURA/doc/SAKURA-G_Quik_Start_Guide_Ver1.0_English.pdf`.

For this tutorial we will need:

- A Sakura-G board
- A USB cable (A to B)
- Windows 7
- Xilinx Platform Cable USB

And the following software:

- Microsoft .Net Framework 4.0[2]
- Xilinx ISE 14.7 (Ask me for a license at `A.delaPiedra@cs.ru.nl` and include: MAC address, hostname, operating system (32 or 64 bits) in the e-mail). During the installation process also install the cable drivers. We will need them to use the Platform Cable[3].
- FTDI drivers D2XX[4].
- FT PROG[5].

First we we will configure the Sakura FPGA with the default AES-128 implementation and install the required drivers. Then, we will set up the development environment and finally we will see how to obtain ciphertexts using the AES implementation of the Sakura-G board.

Programming the Sakura board

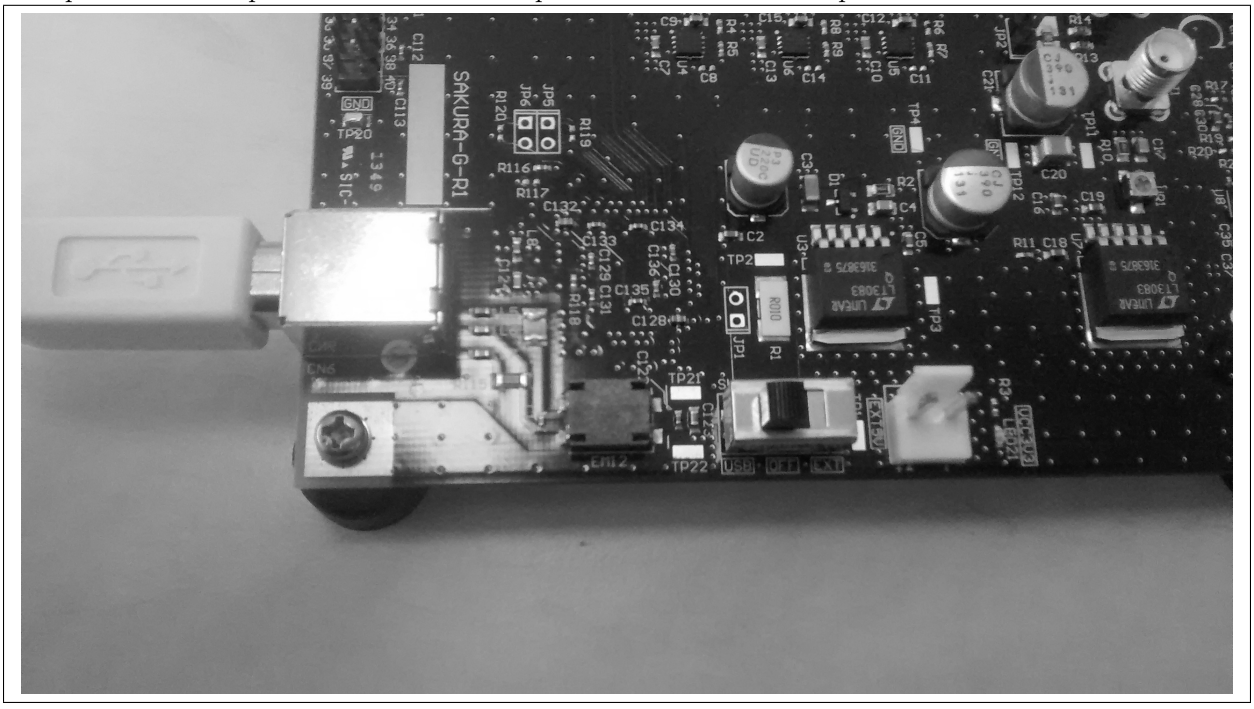
Go the website of the Sakura-G board[6], download and uncompress the following files:

- SAKURA-G Quick Start Guide Source and Binary Codes
- SAKURA-G Verilog-HDL Top Models and ucf Files

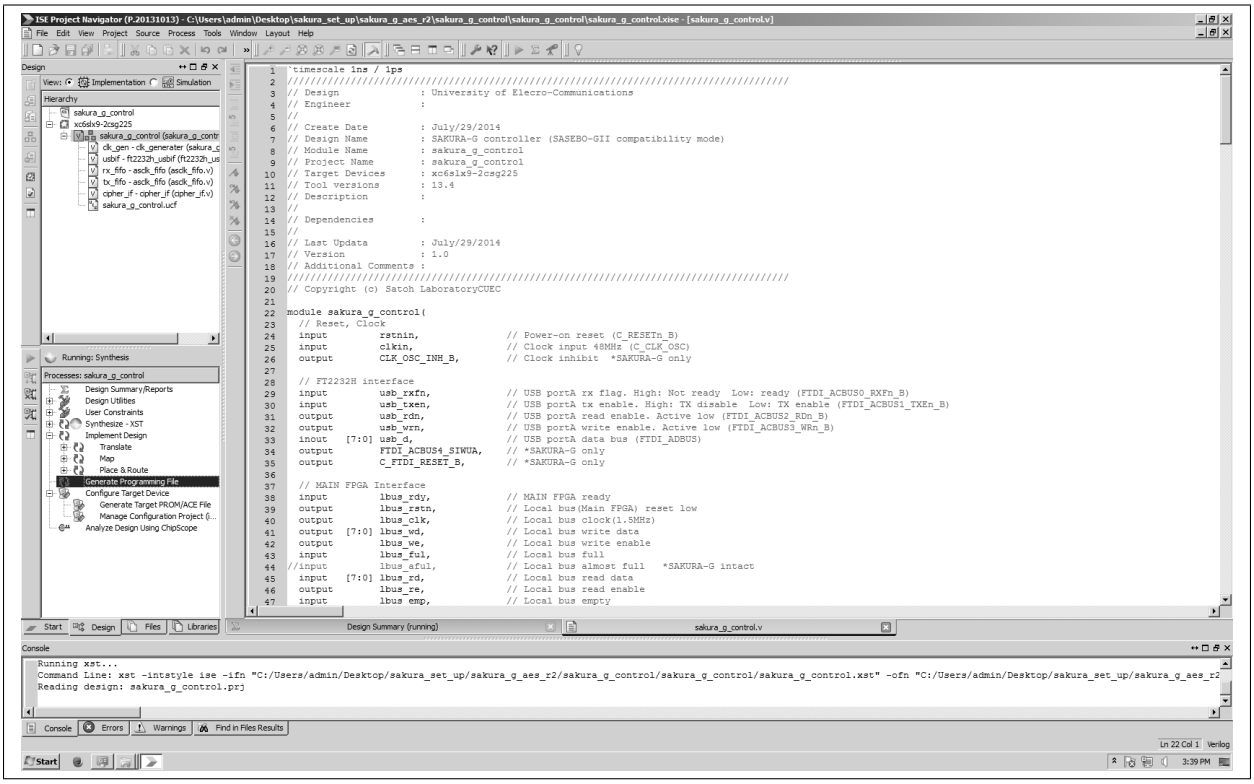
[1] `http://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-G.html`
[2] `https://www.microsoft.com/nl-nl/download/details.aspx?id=17851`
[3] `http://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/design-tools.html`
[4] `http://www.ftdichip.com/Drivers/D2XX.htm`
[5] `http://www.ftdichip.com/Support/Utilities.htm#FT_PROG`
[6] `http://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-G.html`

- SAKURA-G Checker

The Sakura board consists of two FPGAs. The first FPGA is the *control* FPGA, that communicates with a PC via the FTDI protocol and the other one, *main*, it is used for loading the design we want to test and analyze.
First we will program the control FPGA (Spartan-6 XC6SLX9). We will connect the Sakura board to the USB port of our computer and will move the power switch to the *USB* position.

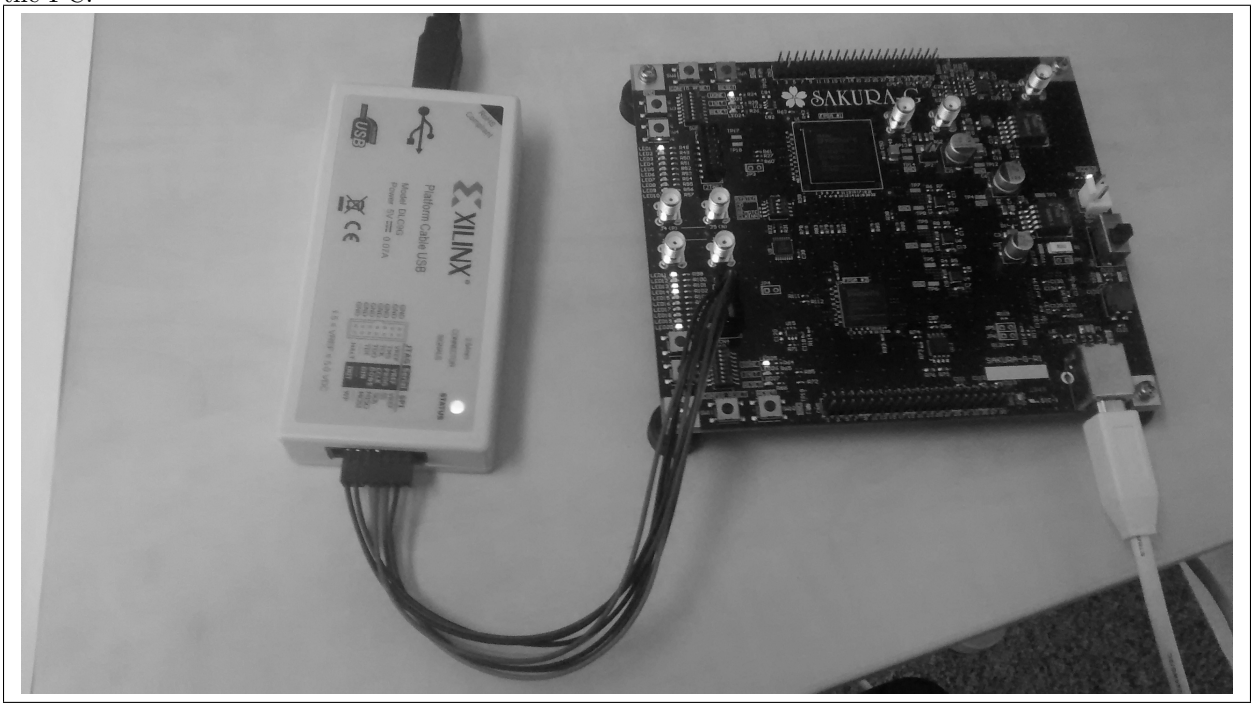


Open Xilinx ISE and go to to *sakura_g_aes_r2, sakura_g_control, sakura_g_control, sakura_g_control.xise* and open that project. Ignore the warning about not finding the *sakura_g_control_tb.v* file. Double click on the *Generate Programming File* process.

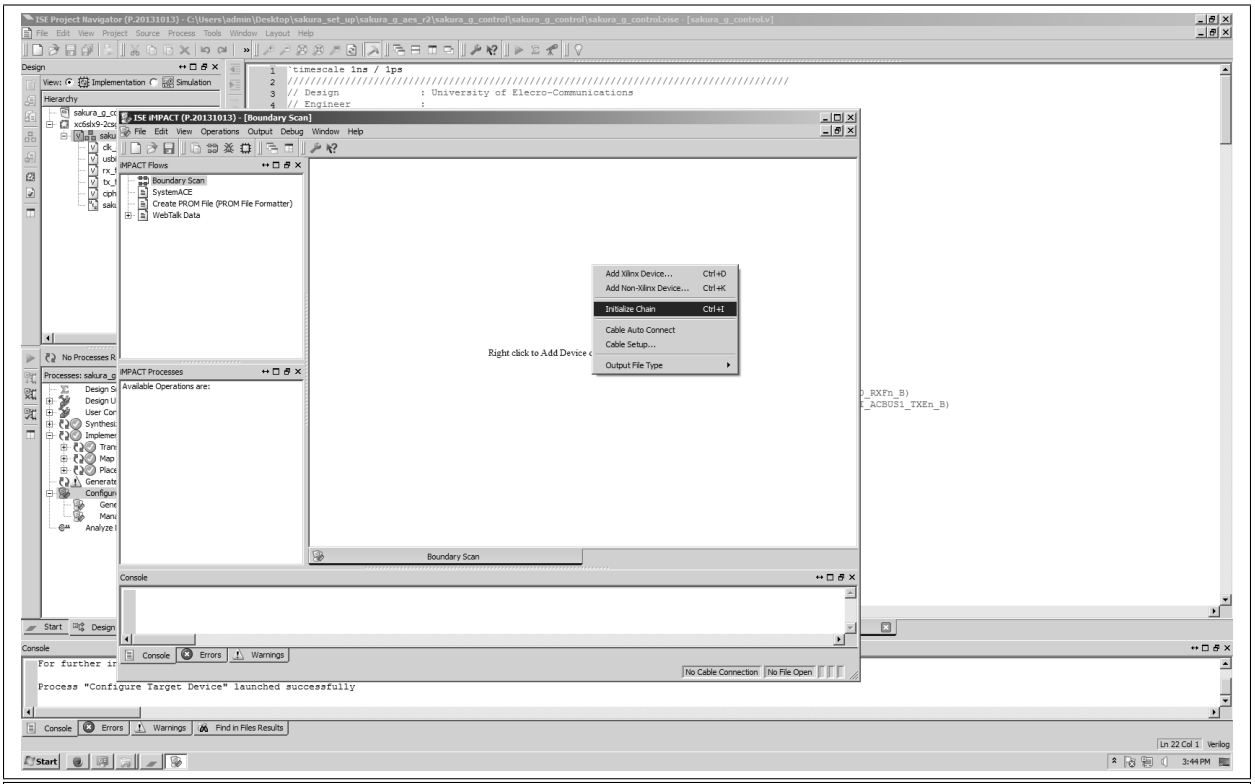


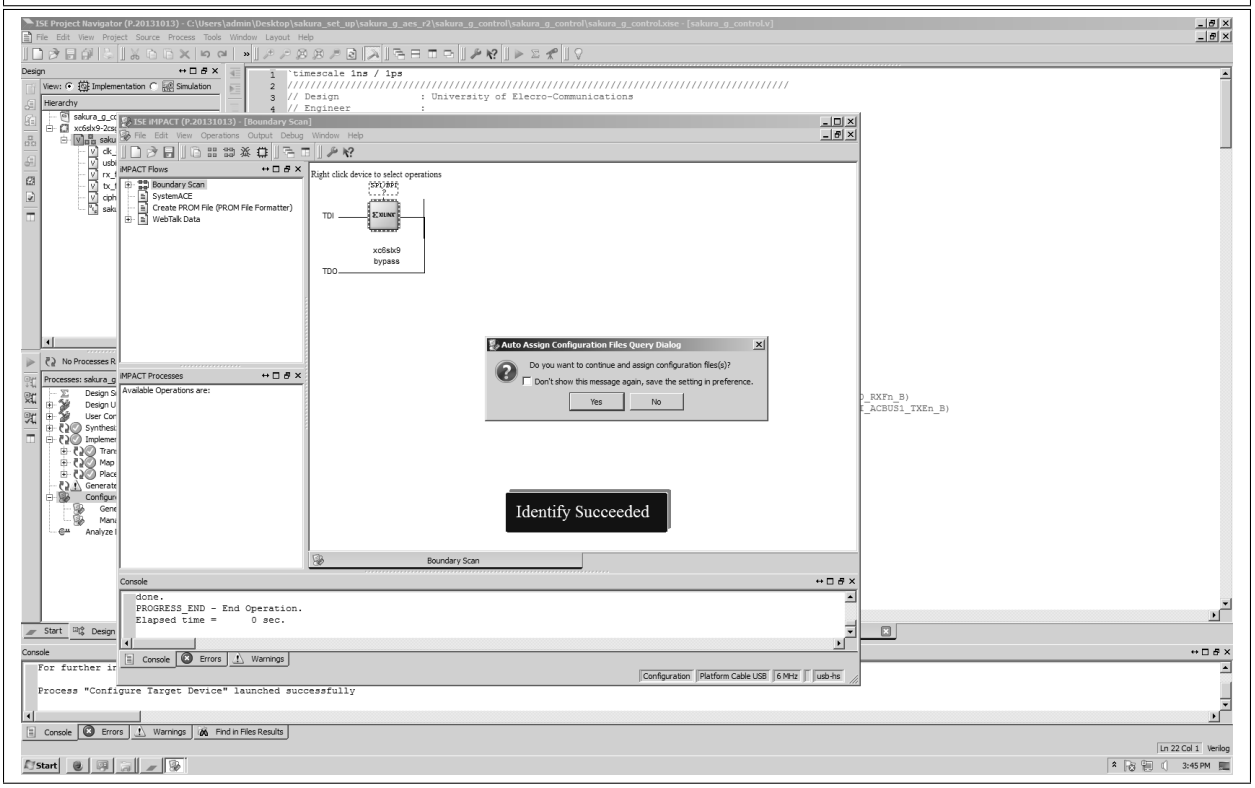

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Design              : University of Elecro-Communications
// Engineer            :
//
// Create Date         : July/29/2014
// Design Name         : SAKURA-G controller (SASEBO-GII compatibility mode)
// Module Name         : sakura_g_control
// Project Name        : sakura_g_control
// Target Devices      : xc6slx9-2csg225
// Tool versions       : 13.4
// Description         :
//
// Dependencies        :
//
// Last Updata         : July/29/2014
// Version             : 1.0
// Additional Comments :
//////////////////////////////////////////////////////////////////////////////////
// Copyright (c) Satoh LaboratoryCUEC

module sakura_g_control(
  // Reset, Clock
  input         rstnin,             // Power-on reset (C_RESETn_B)
  input         clkin,              // Clock input 48MHz (C_CLK_OSC)
  output        CLK_OSC_INH_B,      // Clock inhibit  *SAKURA-G only

  // FT2232H interface
  input         usb_rxfn,           // USB portA rx flag. High: Not ready  Low: ready (FTDI_ACBUS0_RXFn_B)
  input         usb_txen,           // USB portA tx enable. High: TX disable  Low: TX enable (FTDI_ACBUS1_TXEn_B)
  output        usb_rdn,            // USB portA read enable. Active low (FTDI_ACBUS2_RDn_B)
  output        usb_wrn,            // USB portA write enable. Active low (FTDI_ACBUS3_WRn_B)
  inout   [7:0] usb_d,              // USB portA data bus (FTDI_ADBUS)
  output        FTDI_ACBUS4_SIWUA,  // *SAKURA-G only
  output        C_FTDI_RESET_B,     // *SAKURA-G only

  // MAIN FPGA Interface
  input         lbus_rdy,           // MAIN FPGA ready
  output        lbus_rstn,          // Local bus(Main FPGA) reset low
  output        lbus_clk,           // Local bus clock(1.5MHz)
  output  [7:0] lbus_wd,            // Local bus write data
  output        lbus_we,            // Local bus write enable
  input         lbus_ful,           // Local bus full
//input         lbus_aful,          // Local bus almost full   *SAKURA-G intact
  input   [7:0] lbus_rd,            // Local bus read data
  output        lbus_re,            // Local bus read enable
  input         lbus_emp,           // Local bus empty
```

Connect the programmer to the control FPGA JTAG port (XC6SLX9). And connect the programmer to the PC.

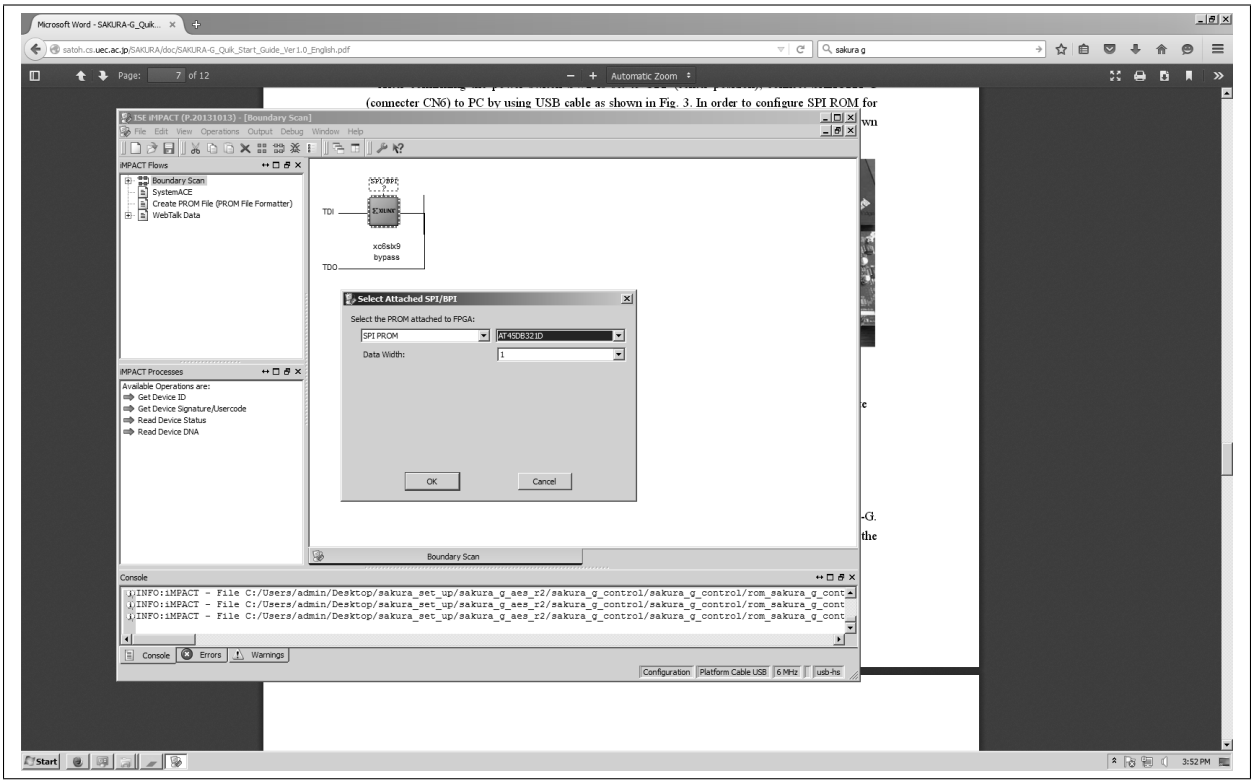


Double-click on *Configure Target Device*, press *OK* and let Impact starts. Select *Boundary Scan, Initialize chain*. You should see the XC6SLX9 FPGA now. Say *No* to assign a file.

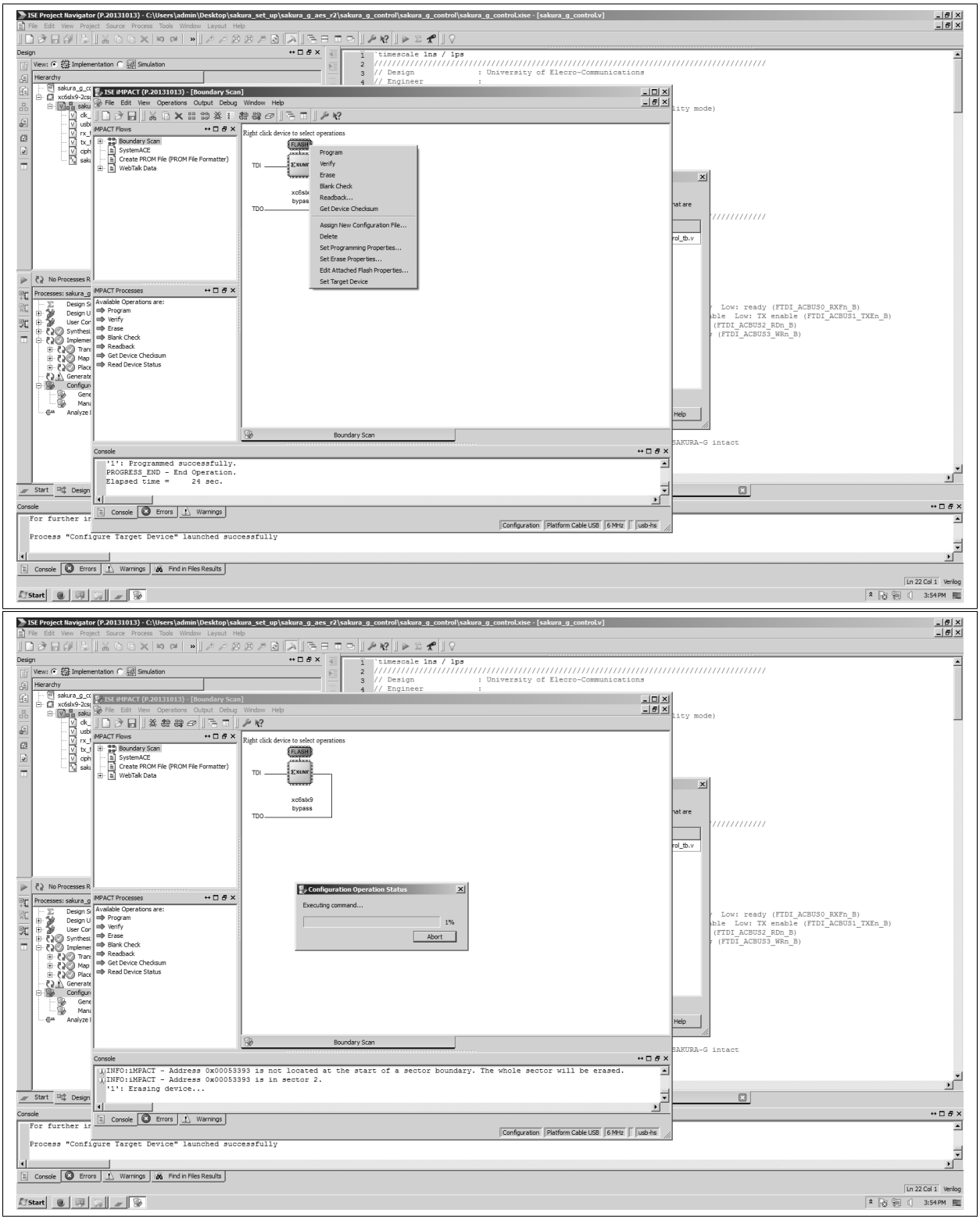




Right-click on the FPGA and click on *add SPI/BPI Flash*. Select *sakura_g_control, sakura_g_control, rom_sakura_g_control.mcs*. Select the SPI PROM device, which is an AT45DB321D.

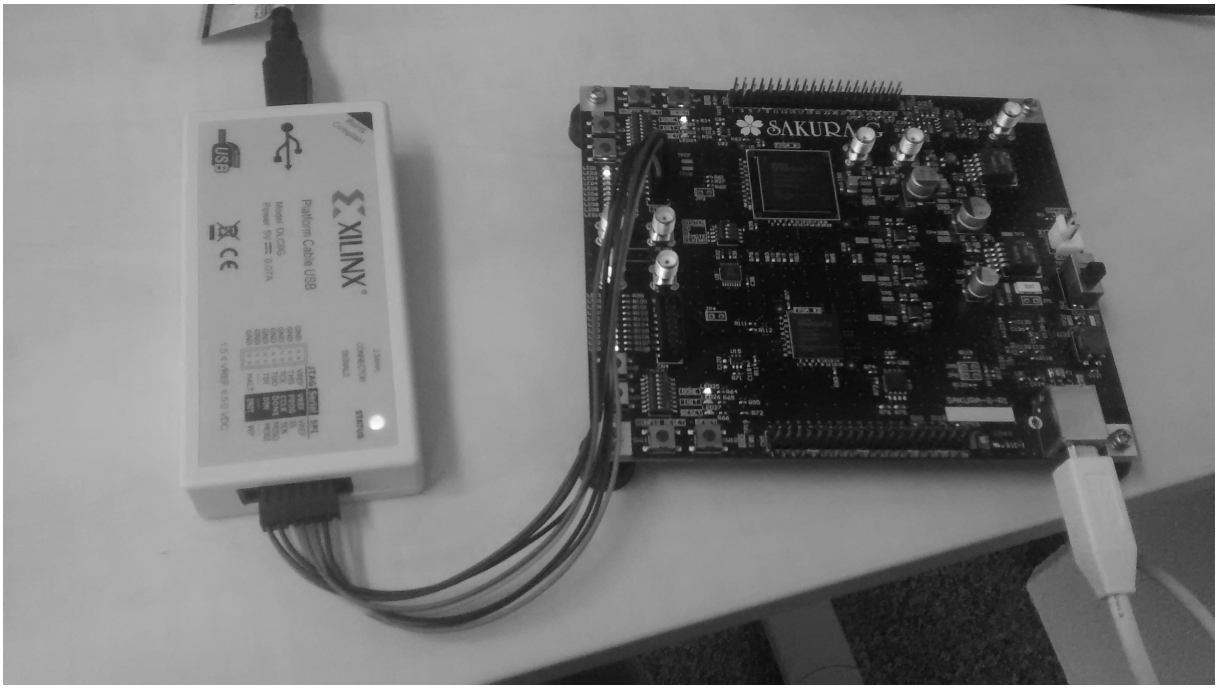


Then, select FLASH, right click on *Program*. After programming in a few seconds you should see the LEDS 1-20 and 9-10 moving.



Now, we will repeat the process with the project of the main FPGA of the Sakura board (XC6SLX75). The SPI PROM in this case is an AT45DB321D. Open the project via Xilinx ISE at *sakura_g_aes_128, sakura_g_aes_128, sakura_g_aes_128, sakura_g_aes128.project.xise*. Double-click on *Generate Programming file* and then double-click *Configure target device*. Now you will have to move the JTAG programmer to the

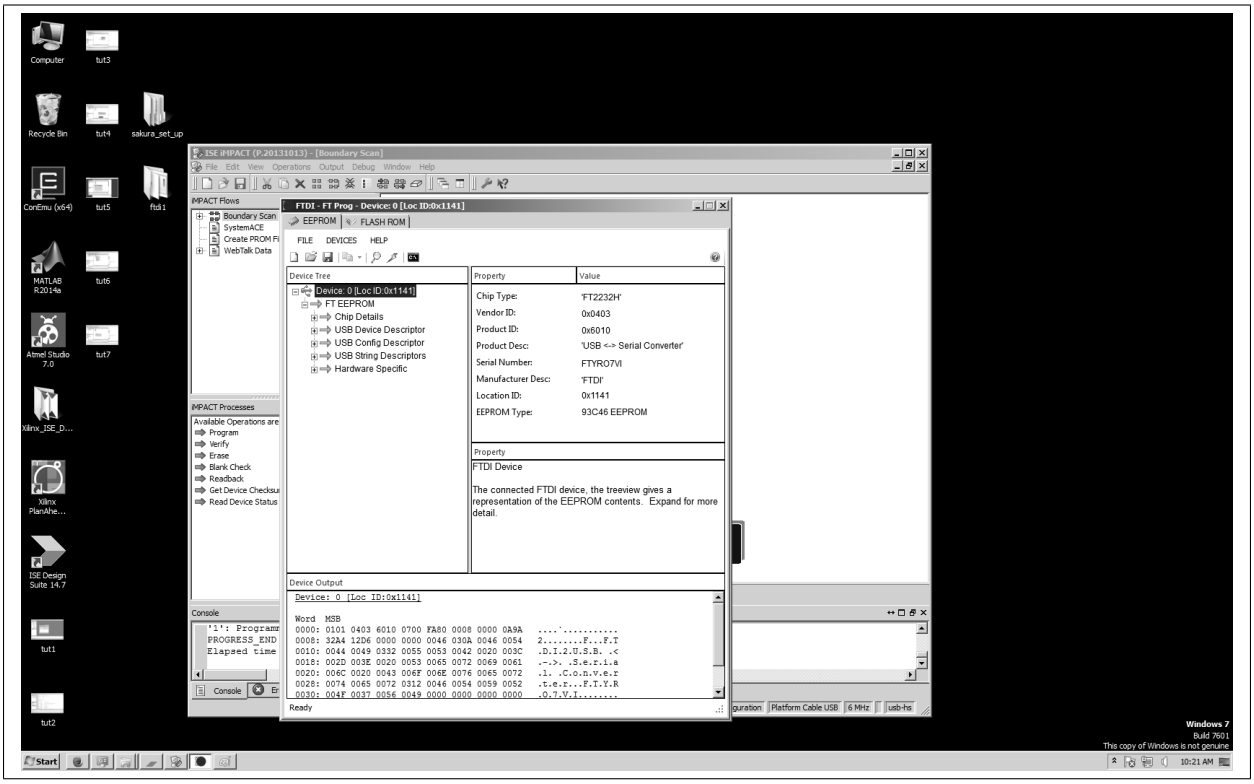
main FPGA.

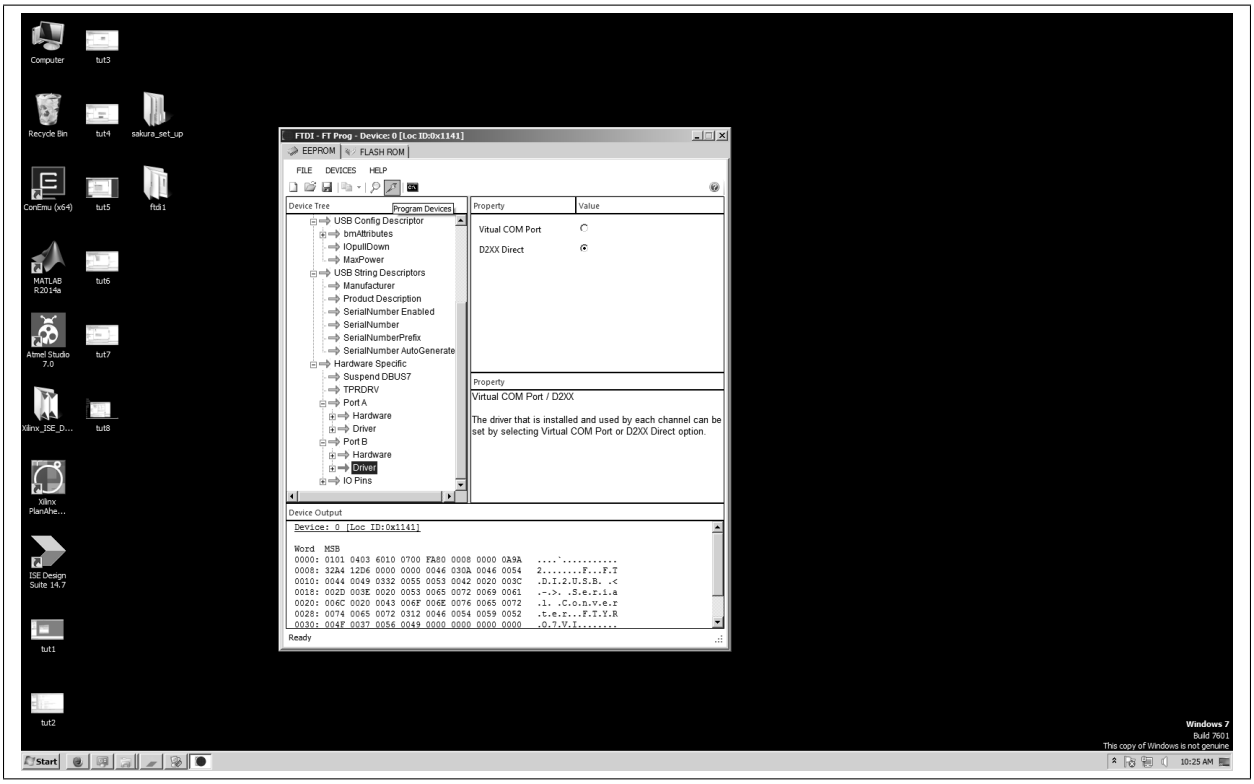


Program the SPI following the prior procedure but this time select the SPI PROM model as AT45DB321D.

FTDI chip configuration of the Sakura board

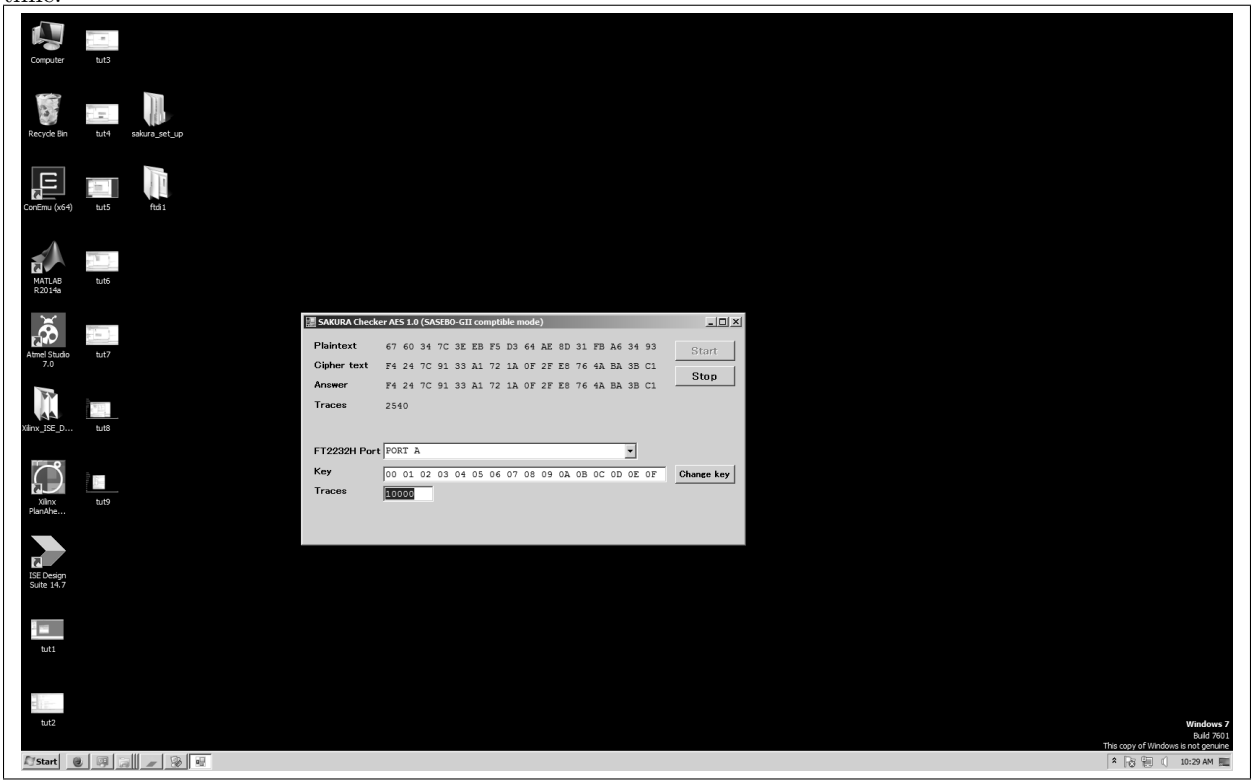
Connect the USB port of the Sakura board to the PC. Open FT Prog, go to *devices*, *scan* and *parse*. So you will see the actual FTDI device of the Sakura.
Unroll hardware specifications and ensure that in both Ports (A,B,) Hardware mode 245FIFO has been selected. For both Ports, select Driver and ensure that D2XX Direct has been also selected. Otherwise, select the four options and click *Program Devices*.





In order to test that the FPGA has been correctly programmed we will test the Checker program that we downloaded before from the website of Sakura-G. Open *sakura_g_aes_r2, SAKURA_checker_AES, Release, SAKURA_Checker_AES*. Select Port A (that is, the control FPGA) and press *Start*. You should be able to see the coming encryptions and compare it with the actual value being encrypted in software at the same

time.



We will replace this program by a script in Python that we could easily modify later.

Controlling the FTDI port via Python

If you are using your own computer, you will need to install Python 2.7 for Windows 32 bits (due to the libraries we will use)[7].
Then, we will install the module ftd2xx in order to communicate with the FTDI port[8], uncompress it and do `python setup.py install` from command line.
Then, install the win32con module[9]. Select *Build 219* and download *pywin32-219.win32-py27.exe*. Install and select the path to your Python 2.7 installation. We will use this module later for communicating with the oscilloscope.
We will be using the HDF5 binary data format for storing the power traces later. Download the h5py python module for Windows at `www.h5py.org`: *h5py-2.5.0.win32-py.2.7.exe*. Finally, install pycrypto[10].
The idea of having a script for controlling the execution of the target device is to have more flexibility about how to send and receive data from the Sakura-G board, extra checkings for validation and for the extraction of power traces. In this section we will prepare a script that performs the following operations:

1. Send to the Sakura a random plaintext and an encryption key.
2. Both encrypt the plaintext in hardware and in software.
3. Recover the ciphertext from hardware.
4. Check if both ciphertexts matches.

[7] `https://www.python.org/downloads/windows/`
[8] `http://github.com/snmishra/ftd2xx/archive/master.zip`
[9] `http://sourceforge.net/projects/pywin32/files/pywin32`
[10] `http://www.voidspace.org.uk/python/modules.shtml#pycrypto`

By default, the control FPGA implements a set of registers for managing the AES implementation of the main FPGA. These registers are 16 bit each:

- 0x0002 Control register
- 0x000c Mode (encryption (0x00) or decryption (0x01)
- 0x0100-0x010e Encryption key
- 0x0140-0x014e Plain text

The control register can be written and read according to the operation being done:

- 0x0002 Init key schedule
- 0x0004 Reset
- 0x0000 Done
- 0x0001 Start

Ideally, you would rely on these registers for controlling your own target on the main FPGA if the 128 bits input of the 0x0140-0x014e registers is large enough together with the control register.
Once both the plaintext and the encryption key have been loaded, we write 0x0001 in the control register (0x0002) and wait till it is changed to 0x0000.
For establishing a connection to the Sakura and writing/reading their registers we rely on the class below, part of the ChipWhisperer project[11].

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
#
# Copyright (c) 2013-2014, NewAE Technology Inc
# All rights reserved.
#
#    Class SASEBOGII is part of chipwhisperer.
#
#    chipwhisperer is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    chipwhisperer is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU Lesser General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with chipwhisperer.  If not, see <http://www.gnu.org/licenses/>.
#=================================================

class SASEBOGII():
    def connect(self, serialNo):
```

[11] https://www.assembla.com/spaces/chipwhisperer/wiki

```
       # connect to port
        try:

            self.sasebo = ft.openEx(serialNo)

            #['FTYJVALDA', 'FTYJVALDB']

        except ft.ftd2xx.DeviceError, e:
            self.sasebo = None
            return False

        self.sasebo.setTimeouts(1000, 1000)
        return True

    def disconnect(self):
        return

    def flush(self):
        num = self.sasebo.getQueueStatus()
        if num > 0:
            self.sasebo.read(num)

    def write(self, address, MSB, LSB):
        msg = bytearray(5)

        msg[0] = 0x01;
        msg[1] = (address >> 8) & 0xFF; #MSB
        msg[2] = address & 0xFF; #LSB
        msg[3] = MSB;
        msg[4] = LSB;

        strmsg = str(msg);

        #msg = bytearray(strmsg)
        #print "Write: %x %x %x %x %x"%(msg[0],msg[1],msg[2],msg[3],msg[4])

        self.sasebo.write(strmsg)

    def read(self, address):
        self.flush()
        msg = bytearray(3)
        msg[0] = 0x00;
        msg[1] = (address >> 8) & 0xFF; #MSB
        msg[2] = address & 0xFF; #LSB
        self.sasebo.write(str(msg))
        #print "Write: %x %x %x"%(msg[0],msg[1],msg[2]),
        msg = self.sasebo.read(2)
        msg = bytearray(msg)
```

```
        #print " Read: %x %x"%(msg[0],msg[1])

        #Order = MSB, LSB
        return msg

    def read128(self, address):
        self.flush()
        msg = bytearray(3*8)
        for i in range(0, 8):
            msg[i*3] = 0x00;
            msg[i*3+1] = (address >> 8) & 0xFF;
            msg[i*3+2] = (address & 0xFF) + (i*2);
        self.sasebo.write(str(msg))
        msg = self.sasebo.read(16)
        return bytearray(msg)

    def close(self):
        self.sasebo.close()

    def init(self):
        #Select AES
        self.write(0x0004, 0x00, 0x01)
        self.write(0x0006, 0x00, 0x00)

        #Reset AES module
        self.write(0x0002, 0x00, 0x04)
        self.write(0x0002, 0x00, 0x00)

        #Select AES output
        self.write(0x0008, 0x00, 0x01)
        self.write(0x000A, 0x00, 0x00)

    def setModeEncrypt(self):
        self.write(0x000C, 0x00, 0x00)

    def setModeDecrypt(self):
        self.write(0x000C, 0x00, 0x01)

    def loadEncryptionKey(self, key):
        """Encryption key is bytearray"""

        if key:
            self.write(0x0100, key[0], key[1])
            self.write(0x0102, key[2], key[3])
            self.write(0x0104, key[4], key[5])
            self.write(0x0106, key[6], key[7])
            self.write(0x0108, key[8], key[9])
            self.write(0x010A, key[10], key[11])
            self.write(0x010C, key[12], key[13])
```

```
            self.write(0x010E, key[14], key[15])

        # Generate key schedule
        self.write(0x0002, 0x00, 0x02)

        # Wait for done
        while self.isDone() == False:
            continue

    def loadInput(self, inputtext):
        self.write(0x0140, inputtext[0], inputtext[1])
        self.write(0x0142, inputtext[2], inputtext[3])
        self.write(0x0144, inputtext[4], inputtext[5])
        self.write(0x0146, inputtext[6], inputtext[7])
        self.write(0x0148, inputtext[8], inputtext[9])
        self.write(0x014A, inputtext[10], inputtext[11])
        self.write(0x014C, inputtext[12], inputtext[13])
        self.write(0x014E, inputtext[14], inputtext[15])

    def isDone(self):
        result = self.read(0x0002)

        if result[0] == 0x00 and result[1] == 0x00:
            return True
        else:
            return False

    def readOutput(self):
        return self.read128(0x0180)

    def setMode(self, mode):
        if mode == "encryption":
            self.write(0x000C, 0x00, 0x00)
        elif mode == "decryption":
            self.write(0x000C, 0x00, 0x01)
        else:
            print "Wrong mode!!!!"

    def go(self):
        self.write(0x0002, 0x00, 0x01)
```

In order to test the functionality of the control FPGA we will write a random plaintext and encryption key on the Sakura and we will encrypt. Then we will check if the AES encryption is correct by encrypting the same plaintext in software.
We will do the connection to the control FPGA, with a serial number ending by 'A':

```
# List FTDI devices

 ftdiDevices = ft.listDevices()
```

```
    print "Devices:", ftdiDevices

    # Select the first FPGA ("A", the controller one)

    sakura = SASEBOGII()

    print "Connection to Sakura G, FPGA:", ftdiDevices[0], "Result:", sakura.connect(ftdiDevices[0])
```

Now you can test the code below in your computer with the Sakura connected to the USB port:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
#
# Copyright (c) 2013-2014, NewAE Technology Inc
# Copyright (c) 2015, Radboud University Nijmegen, Antonio de la Piedra
# All rights reserved.
#
#    Class SASEBOGII is part of chipwhisperer.
#
#    chipwhisperer is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    chipwhisperer is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU Lesser General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with chipwhisperer.  If not, see <http://www.gnu.org/licenses/>.
#=================================================

import sys
import os
import threading
import time
import logging
import math
import ftd2xx as ft
import random
import h5py

from time import gmtime, strftime
from binascii import hexlify, b2a_hex
from Crypto.Cipher import AES

class SASEBOGII():
```

```
    def connect(self, serialNo):
       # connect to port
        try:

            self.sasebo = ft.openEx(serialNo)

        except ft.ftd2xx.DeviceError, e:
            self.sasebo = None
            return False

        self.sasebo.setTimeouts(1000, 1000)
        return True

    def disconnect(self):
        return

    def flush(self):
        num = self.sasebo.getQueueStatus()
        if num > 0:
            self.sasebo.read(num)

    def write(self, address, MSB, LSB):
        msg = bytearray(5)

        msg[0] = 0x01;
        msg[1] = (address >> 8) & 0xFF; #MSB
        msg[2] = address & 0xFF; #LSB
        msg[3] = MSB;
        msg[4] = LSB;

        strmsg = str(msg);
        self.sasebo.write(strmsg)

    def read(self, address):
        self.flush()
        msg = bytearray(3)
        msg[0] = 0x00;
        msg[1] = (address >> 8) & 0xFF; #MSB
        msg[2] = address & 0xFF; #LSB
        self.sasebo.write(str(msg))
        msg = self.sasebo.read(2)
        msg = bytearray(msg)

        return msg

    def read128(self, address):
        self.flush()
        msg = bytearray(3*8)
        for i in range(0, 8):
```

```
            msg[i*3] = 0x00;
            msg[i*3+1] = (address >> 8) & 0xFF;
            msg[i*3+2] = (address & 0xFF) + (i*2);
        self.sasebo.write(str(msg))
        msg = self.sasebo.read(16)
        return bytearray(msg)

    def close(self):
        self.sasebo.close()

    def init(self):
        #Select AES
        self.write(0x0004, 0x00, 0x01)
        self.write(0x0006, 0x00, 0x00)

        #Reset AES module
        self.write(0x0002, 0x00, 0x04)
        self.write(0x0002, 0x00, 0x00)

        #Select AES output
        self.write(0x0008, 0x00, 0x01)
        self.write(0x000A, 0x00, 0x00)

    def setModeEncrypt(self):
        self.write(0x000C, 0x00, 0x00)

    def setModeDecrypt(self):
        self.write(0x000C, 0x00, 0x01)

    def loadEncryptionKey(self, key):
        """Encryption key is bytearray"""

        if key:
            self.write(0x0100, key[0], key[1])
            self.write(0x0102, key[2], key[3])
            self.write(0x0104, key[4], key[5])
            self.write(0x0106, key[6], key[7])
            self.write(0x0108, key[8], key[9])
            self.write(0x010A, key[10], key[11])
            self.write(0x010C, key[12], key[13])
            self.write(0x010E, key[14], key[15])

        # Generate key schedule
        self.write(0x0002, 0x00, 0x02)

        # Wait for done
        while self.isDone() == False:
            continue
```

```
    def loadInput(self, inputtext):
        self.write(0x0140, inputtext[0], inputtext[1])
        self.write(0x0142, inputtext[2], inputtext[3])
        self.write(0x0144, inputtext[4], inputtext[5])
        self.write(0x0146, inputtext[6], inputtext[7])
        self.write(0x0148, inputtext[8], inputtext[9])
        self.write(0x014A, inputtext[10], inputtext[11])
        self.write(0x014C, inputtext[12], inputtext[13])
        self.write(0x014E, inputtext[14], inputtext[15])

    def isDone(self):
        result = self.read(0x0002)

        if result[0] == 0x00 and result[1] == 0x00:
            return True
        else:
            return False

    def readOutput(self):
        return self.read128(0x0180)

    def setMode(self, mode):
        if mode == "encryption":
            self.write(0x000C, 0x00, 0x00)
        elif mode == "decryption":
            self.write(0x000C, 0x00, 0x01)
        else:
            print "Wrong mode!!!!"

    def go(self):
        self.write(0x0002, 0x00, 0x01)

if __name__ == "__main__":

    print "[*] Sakura SETUP"

    # List FTDI devices

    ftdiDevices = ft.listDevices()

    print "Devices:", ftdiDevices

    # Select the first FPGA ("A", the control FPGA)

    sakura = SASEBOGII()

    print "Connection to Sakura G, FPGA:", ftdiDevices[0], "Result:", sakura.connect(ftdiDevices[0])

    sakura.init()
```

```
sakura.setModeEncrypt()

# Generate a random encryption key (AES-128)

rand_key = b2a_hex(os.urandom(16))
tmp_key = rand_key.decode("hex")
rand_key_list = map(ord, tmp_key)

print "[*] key:", rand_key
sakura.loadEncryptionKey(rand_key_list)

rand_pt  = b2a_hex(os.urandom(16))
tmp_pt = rand_pt.decode("hex")
rand_pt_list  = map(ord, tmp_pt)

print "[*] plaintext:", rand_pt

    # Encrypt the plaintext in software

aesEncrypt = AES.new(rand_key.decode("hex"), AES.MODE_ECB)
aesCiphertext = aesEncrypt.encrypt(rand_pt.decode("hex"))

ct_sw = aesCiphertext.encode("hex")
print "[*] expected ciphertext", ct_sw

# Encrypt the plaintext on hardware (Sakura)

sakura.loadInput(rand_pt_list)
sakura.go()

while sakura.isDone() == False:
    continue

ct_hw = hexlify(sakura.readOutput())

print "[*] Received ciphertext (Sakura)", ct_hw
print "[!] Ciphertexts match:", ct_sw == ct_hw

sakura.disconnect()
```

Exercises

1. When acquiring power traces we will need to generate N measurements with a certain key and random plaintexts. In order to prepare this part of the script, modify the script above for generating N random plaintexts and for each computation, obtain the ciphertext and check that the obtained ciphertext equals to the ciphertext generated in software. Finally, since we will use the plaintexts and ciphertexts together with the power trace, we need to save in a file the pair plaintext/ciphertext. During a loop for generating N plaintexts, also save sequentially in a file the pair plaintext/ciphertext.

2. Since you will be analyzing the leakage of your design, implement your algorithm in Python and interface it with the script above.