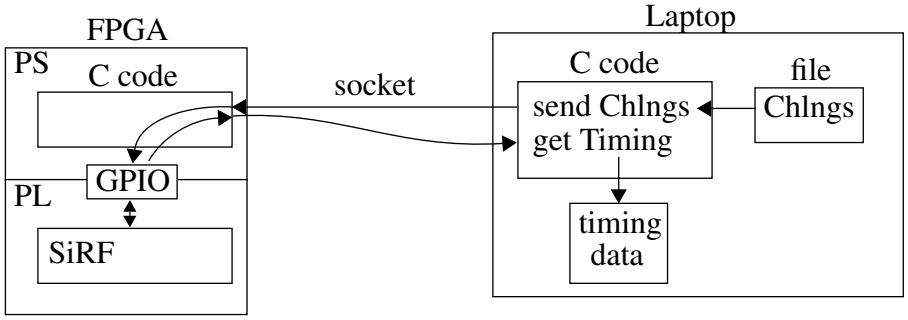
LAB Assignment #2 for ECE 525

Description: Run SiRF Provisioning Process

This lab is the first one in a sequence, where each subsequent lab will build on previous labs and culminate in the project for this course. The goal of these labs and projects is to have you implement an actual PUF-based authentication scheme that will authenticate your FPGA with a trusted authority (a server). I have provided a set of 4 hardware implementations of the SiRF PUF, as FPGA bitstreams. Each implementation is identical in design but is placed in a different location on the PL side of the FPGA (See HELP PUFs II(A) through II(J) screencasts). Therefore, each of these bitstream implementations represents a different device despite the fact that they are located on the same FPGA device. I will refer to the 4 implementations as *instances* in the following.

The first step of the process is to run provisioning, which is what you will do in this laboratory. For SiRF, provisioning collects digitized timing data for a set of paths and stores the timing data in a separate file for each instance. The timing data can then be combined into a database, and used for subsequent (in-field) authentications (PUF-based authentication will be covered in detail in the screencasts including the SiRF protocol that you will use here).

I have also provided the C code that you will use to collect provisioning data for the 4 instances. The provisioning process involves running one C program on the server, called *verifier_provision*, and a second C program on the Zybo/Cora board, called *device_provision.elf*. The two programs use a network connection to communicate with each other. The verifier sends challenges to the device and stores timing data sent by the device to a data file. The *device_provision.elf* program coordinates with a PL-side implementation of the SiRF PUF algorithm. The VHDL code implementation of SiRF PUF programmed into the PL-side carries out a TDC timing operation to measure path delays of paths defined within a combinational logic function, which is also embedded on the PL-side. The combinational logic function is an engineered structural netlist (I'll provide a paper that describes it), which serves as the source of entropy for the SiRF PUF. The digitized path delays are transferred through a GPIO interface between the PL and PS-sides on the FPGA. The *device_provision.elf* running on the PS-side collects this data and transmits it across the network to the verifier, which stores the data in a file.



The provisioning process is completely automated. You need to run it once for each of 4 instances of the SIRF PUF (instructions are given below). I will supply a compiled version of *device_provision.elf* (and a Makefile to compile it on your laptop) but you are encouraged to get familiar with the Xilinx SDK/Vitis tool, networking and other components of the experimental

testbed. The sequence of steps you need to follow are given below. A separate screencast, titled *SDKInstruction_SiRFProvision*, gives details on using the Xilinx SDK tool. The remaining laboratories and project will build on what you do in this lab so I encourage you to use this lab to become familiar with SDK, linux and the networking environment.

1) Create a directory called lab2 and cd into it.

```
mkdir lab2
cd lab2
mkdir ../ANALYSIS
mkdir ../ANALYSIS/data
mkdir ../ANALYSIS/data/2_15_2023
mkdir PROTOCOL
mkdir BITSTREAMS
mkdir Vivado
```

NOTE: You should NOT change the date show above -- make the directory name as shown, 2_15_2023. It is hardcoded in *verifier_provision.c*. Search 'data' and you'll find the reference in the source code. You can change it in *verifier_provision.c* if you like, but don't forget to recompile.

When you run *verifier_provision* on your laptop, the data files fetched over the network from the ZYBO/CORA board will be stored in ../ANALYSIS/data/2_15_2023/.

2) Download the following files from lab2/PROTOCOL on the course website to your PROTOCOL directory.

utility.h
utility.c
common.h
common.c
commonDB.h
commonDB.c
verifier_common.c
verifier_common.h
verifier_provision.c
device_common.c
device_common.h
device_provision.c

SR_RFM_V4_MMCM_Random_Rise_1000Vs_Fall_1000Vs_NumSeeds_10_vecs.txt
README.txt
Makefile_VP
Makefile

device_provision.elf

3) Download the following files from lab2/DEVICE_FILES/CORA/*.bit or lab2/DEVICE_FILES/ZYBO/*.bin to the BITSTREAMS directory on your laptop

4) Install sqlite3 on your laptop from:

https://www.sqlite.org/index.html

You can fetch the sources from https://www.sqlite.org/download.html and compile them (click 'C source code as an amalgamation, version 3.40.1') and compile.

For ubuntu, you can use 'apt install libsqlite3-dev', which gives you pre-compiled version (no compilation needed).

5) Compile *device_provision.elf* (This is optional right now since I've given you *device_provision.elf* but you will need to be able to compile C programs for your CORA and ZYBO boards in the near future).

```
cd PROTOCOL
vi Makefile
```

Change the path at the top of the file to your SDK installation:

SDK_PATH = /home/Xilinx/SDK/2018.2

6) Boot your board, connect the twisted pair cable between your board and laptop and configure the network (see supplemental instructions if needed).

ssh to your board, e.g., `ssh root@192.168.1.10, type root, root,...`

7) ZYBO ONLY:

In the ZYBO window, make a directory in /lib called firmware

```
mkdir /lib/firmware
```

In a host (laptop) window, copy the BITSTREAMS/*.bin to the ZYBO board

```
scp BITSTREAMS/*.bin root@192.168.1.10:/lib/firmware
```

In the ZYBO window, program the board

```
echo SR_RFM_V4_TDC_Macro_P1.bit.bin > /sys/class/fpga_manager/fpga0/firmware
```

You will do this once for each of the four bitstreams during provisioning (described below).

8) CORA ONLY:

In a host (laptop) window, cd into the Vivado directory

```
cd Vivado
vivado&
```

In vivado, click 'Open Hardware Manager'

NOTE: If running under linux, you will need to give yourself permission to access the JTAG port for the CORA board, type the following

```
cd Vivado
```

Look for the 'Future Device Technologies' line, it will have 'bus xxx Device yyy'

Become superuser

```
chmod a+rwx /dev/bus/usb/xxx
chmod a+rwx /dev/bus/usb/xxx/yyy
```

This will allow vivado, running in user mode (not superuser mode) to access the JTAG port.

Click 'Open target'

If successful, your board will be listed and you'll have an option to program the board called 'Program device'

Click 'Program device' and select BITSTREAMS/SR_RFM_V4_TDC_Macro_P1.bit

Click okay

You will do this once for each of the four bitstreams during provisioning (described below).

9) Copy *device_provision.elf* to your ZYBO or CORA board /home/root directory

```
cd PROTOCOL
scp device_provision.elf root@192.168.1.10:
```

10) You are now ready to carry out the provisioning process.

In a host (laptop) window, do the following. Change the IP address in the following, 192.168.1.20, to the IP of your host if necessary. Do NOT CHANGE 'xxx.xxx.x.x' OR ANY OTHER ARGUMENT -- use this as is. Note, the last argument, **0**, will change to 1, 2 and 3 as you do the other bitstreams.

```
cd PROTOCOL
./verifier_provision 127.0.0.1 192.168.1.20 8888 xxx.xxx.x.x 0 0 0 0 4 0
```

You should see

```
Waiting for incoming connections from First device for Provisioning at
address 192.168.1.20
```

In the ZYBO or CORA window, type the following

```
mkdir output
./device_provision.elf    CXX    192.168.1.20    8888    0    >    output/
CXX_SR_RFM_V4_TDC_P1_25C_1.00V_NCs_2000_E_PUFNums_output.txt
```

Change 'XX' to the board number that I have assigned to you IN BOTH PLACES in this command.

If successful, you will need to about a minute and the process will complete. Check the ../ANALYSIS/data/2_15_2023 directory for a data file that ends in .txt

If it fails, several things could have gone wrong

a) If your ZYBO or CORA board is locked up (no response at the terminal) then you forgot to program the board. You need to power cycle the board and start again with 1) programming the board after it boots, ssh'ing to your board to setup the network, and then running the *device_provision.elf* command again.

b) No route to host. This usually happens if you have not setup the network properly, e.g., given the ZYBO/CORA board a unique IP, e.g., 192.168.1.10 and your laptop a unique IP, e.g., 192.168.1.20. Or, if running under linux, you may need to disable IP tables, e.g., as superuser, run /etc/rc.d/init.d/iptables stop

11) Generate the data files for the other three programming bitstreams, e.g., for CORA:

// Program device with bitstream 'P2' using Vivado

In the host (laptop) window (NOTE: The last parameter is '1', which will change the filename used to store the file in ../ANALYSIS/data/2_15_2022.

```
./verifier_provision 127.0.0.1 192.168.1.20 8888 xxx.xxx.x.x 0 0 0 0 4 1
```

In the CORA window

```
./device_provision.elf    CXX    192.168.1.20    8888    0    >    output/
CXX_SR_RFM_V4_TDC_P2_25C_1.00V_NCs_2000_E_PUFNums_output.txt
```

Replace 'XX' in BOTH places with your chip number.

For ZYBO:

In the ZYBO window

```
echo SR_RFM_V4_TDC_Macro_P2.bit.bin > /sys/class/fpga_manager/fpga0/firmware
```

In the host (laptop) window

```
./verifier_provision 127.0.0.1 192.168.1.20 8888 xxx.xxx.x.x 0 0 0 0 4 1
```

In the ZYBO window

```
./device_provision.elf CXX 192.168.1.20 8888 0 > output/CXX_SR_RFM_V4_TDC_P2_25C_1.00V_NCs_2000_E_PUFNums_output.txt
```

12) Repeat previous step for P3 and P4

13) I have created a github site and added all of you as collaborators. You will use this site, initially, to push up your provisioning data. The provisioning data will be kept in a subdirectory called "ProvisionData"

14) To accomplish the task of pushing up your data, do the following

Complements of Brian Dubber:
Here are the extra steps that I used to get push/pull access to the class repo. This is especially relevant to people who are creating a new github account. User will need either an SSH key or an HTTPS access key. I personally think that the HTTPS route is a little easier.

```
cd to where you created lab2
mkdir HOST2023
cd HOST2023
git clone https://github.com/jfplusq/HOST2023.git
```

Your directory structure should have 'lab2' and 'HOST2023' at the same level.

15) In order to PUSH, you will need to set up an access key. Brian recommends following the instructions here:

```
https://stackoverflow.com/questions/68775869/message-support-for-password-authentication-was-removed-please-use-a-personal
```

* Links to an external site.
* Make sure to use the username and email associated with your github account
* Make sure your email address is verified

Once you have the access key setup, you will use this when prompted for a password during a push INSTEAD of your actual github password. Apparently password verification is no longer supported by github.

16) Add your data to the repository, after you have done the 'clone' command above:

```
cd lab2/ANALYSIS/data/2_15_2023/
cp -p * ../../../../HOST2023/ProvisionData/
```

17) Push up your data to the repository:

```
cd ../../../../HOST2023/
git add *
git commit -m "<Your name> Provisioning Data, <date>"
git push
```

18) You will need to periodically download data from the repository. Make sure you are in HOST2023 when you do this

```
git pull https://github.com/jfplusq/HOST2023.git
```

IMPORTANT NOTE: DO NOT DO ANY OF YOUR LABS in the HOST2023 git repository directory. Always work in lab2, lab3, etc. If you need to push or pull to the repository, copy your data into the appropriate directory in HOST2023 and do the commands above.

Assuming you do this (NEVER DO THE LABS IN THE HOST2023 respository), if you ever need to recreate the repository, you can delete HOST2023 and start over with the commands above.