SMASH:
THE SERENITY MANAGEMENT SHELL

ANDY ROSE, IMPERIAL COLLEGE LONDON

WHY?

- The usual problem in writing generic, reusable control software is the complex interdependence of different components

EXAMPLE

- To configure to an optical module you may first need to talk via I2C to a port-expander to read the module-present signal to check that the optical module is present; to then configure the port-expander via I2C to asset the optical module’s module-select line; to talk I2C to the optical module itself; and to reconfigure the port-expander via I2C to deassert the optical module’s module-select line.

BUT…

- The simple statement of “talk via I2C” may, itself, mask myriad complexities since this may be two independent I2C buses for the port-expander and the optical module; if our software is running on a device without embedded I2C controllers, we shall require an additional transport-layer to talk to a device which converts PCIe, USB or Ethernet to I2C.

AND…

- If the I2C port-expander becomes deprecated
- Or we chose to drive the module-present and -select line from the GPIO of an FPGA or SOC
- Or your pesky hardware designer inserts an I2C bus-multiplexer on the I2C bus which itself requires its own layer of configuration
- Then there is typically nothing for it but to write completely new software, copying-and-pasting your old code, and hacking it around.

AND…

- If the I2C port-expander becomes deprecated
- Or we chose to drive the module-present and -select line from the GPIO of an FPGA or SOC
- Or your pesky hardware designer inserts an I2C bus-multiplexer on the I2C bus which itself requires its own layer of configuration
- Then there is typically nothing for it but to write completely new software, copying-and-pasting your old code, and hacking it around.

NEW BOARD REVISION = NEW SOFTWARE

IN THE PAST

- Major structural changes to a board design and entirely new board designs happened on a sufficiently infrequent basis as to be manageable, albeit inefficient, way of working.

SERENITY'S PROBLEM

- Its has a paradigm of structural reconfigurability
- It uses pluggable modules:
  - Different types or counts of FPGAs
  - With or without a QSFP
  - Different numbers, directions and speeds of serial links
    - Passive-electrical, or uni- or bi-directional, 16G or 28G firefly optics.

SERENITY’S PROBLEM

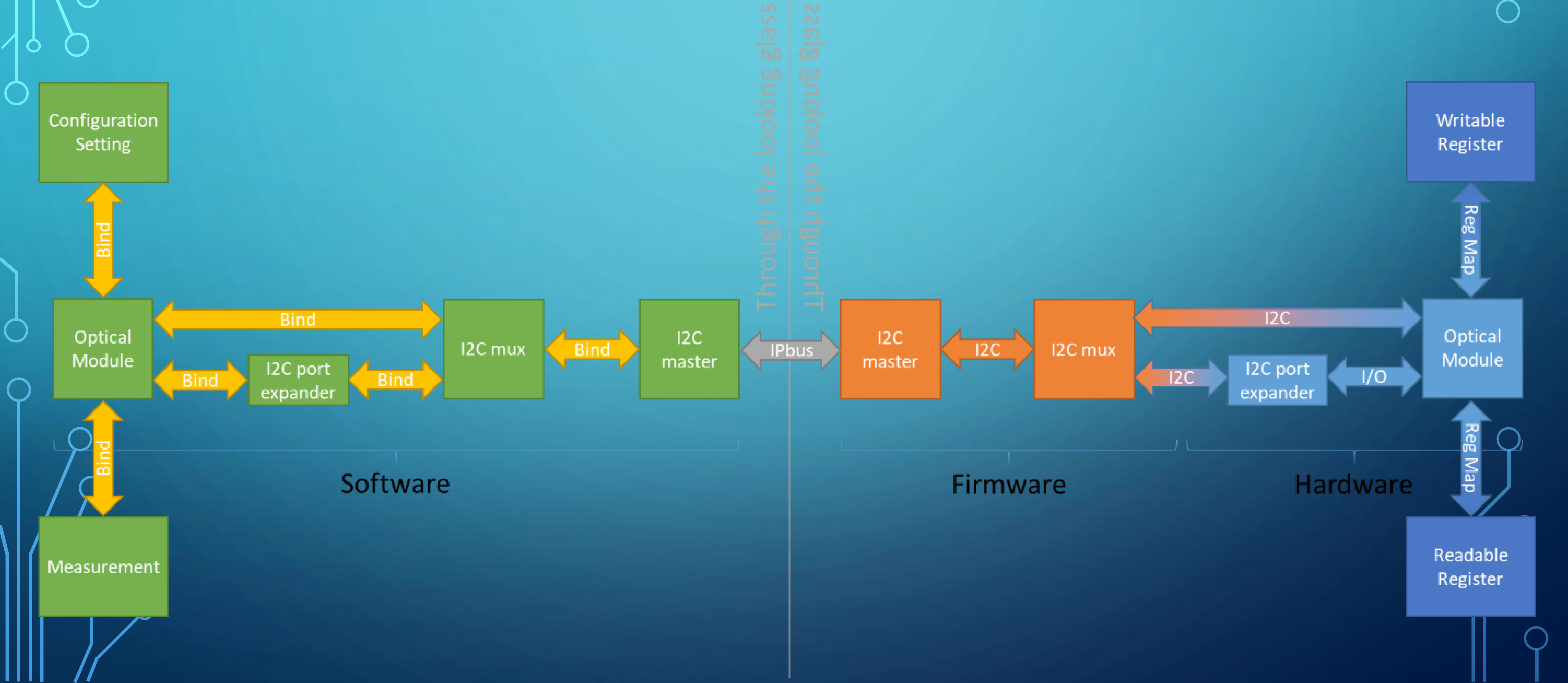
- Its has a paradigm of structural reconfigurability
- It uses pluggable modules:
  - Different types or counts of FPGAs
  - With or without a QSFP
  - Different numbers, directions and speeds of serial links
    - Passive-electrical, or uni- or bi-directional, 16G or 28G firefly optics.
- And with proposals for alternative form-factors and for different control modules (such as SOCs) this problem would only get worse.

THAT IS “WHY SMASH”!

HOW SMASH?

- SMASH builds, in software, a mirror image of the hardware and firmware

HOW SMASH?


Configuration Setting
Bind
Optical Module
Bind
I2C mux
Bind
I2C port expander
Bind
Bind
I2C master
Bind
Measurement
Software
Through the looking glass
Through the looking glass
IPbus
I2C master
I2C
I2C mux
I2C
Optical Module
I2C
I2C port expander
I/O
Writable Register
Reg Map
Reg Map
Readable Register
Firmware
Hardware

HOW SMASH?

- Each type of hardware or firmware has a separate class representation, which is referred to as an “element”.
- Each element is a self-contained entity and has no dependence on any other element.

HOW SMASH?

- Like physical components, which interact with each other by exposing interfaces, be that GPIO, I2C, SPI, JTAG, USB, GbE, PCIe or some other, so do SMASH elements.
- SMASH elements do this through bound function calls, which in SMASH parlance are called “ports”.

HOW SMASH?

- Any element which provides a service, such as an I2C master or a GPIO-driver, creates and exposes a port.
- Any element which uses a service, such as an I2C slave or a component configured by a GPIO, uses a copy of a bound function (not caring where it came from) as a tool to perform the required operation
  - It has and needs no knowledge of the mechanism by which it is implemented, maintaining the conceptual independence of the service provider and the service consumer

HOW SMASH?

- Ports are native C function calls
- Because of the diversity of hardware, firmware and software elements, all exposed interfaces (other than the “ports”) use strings as the interface mechanism
  - Which has the additional advantage of making SMASH highly suitable for scripting and interactive use.

IS IT JUST FOR SERENITY?

- No, it was developed for Serenity, but has already been used on
  - Two different revisions of Serenity
  - Serenity in different form-factors (i.e. with a pizzabox adapter mezzanine)
  - Prototype Serenity Board Tester
  - Serenity 1.2 Daughter-Card Tester
  - Bodged-together systems using an old PC, an old passive board, and some long lengths of wire

IS IT JUST FOR SERENITY?

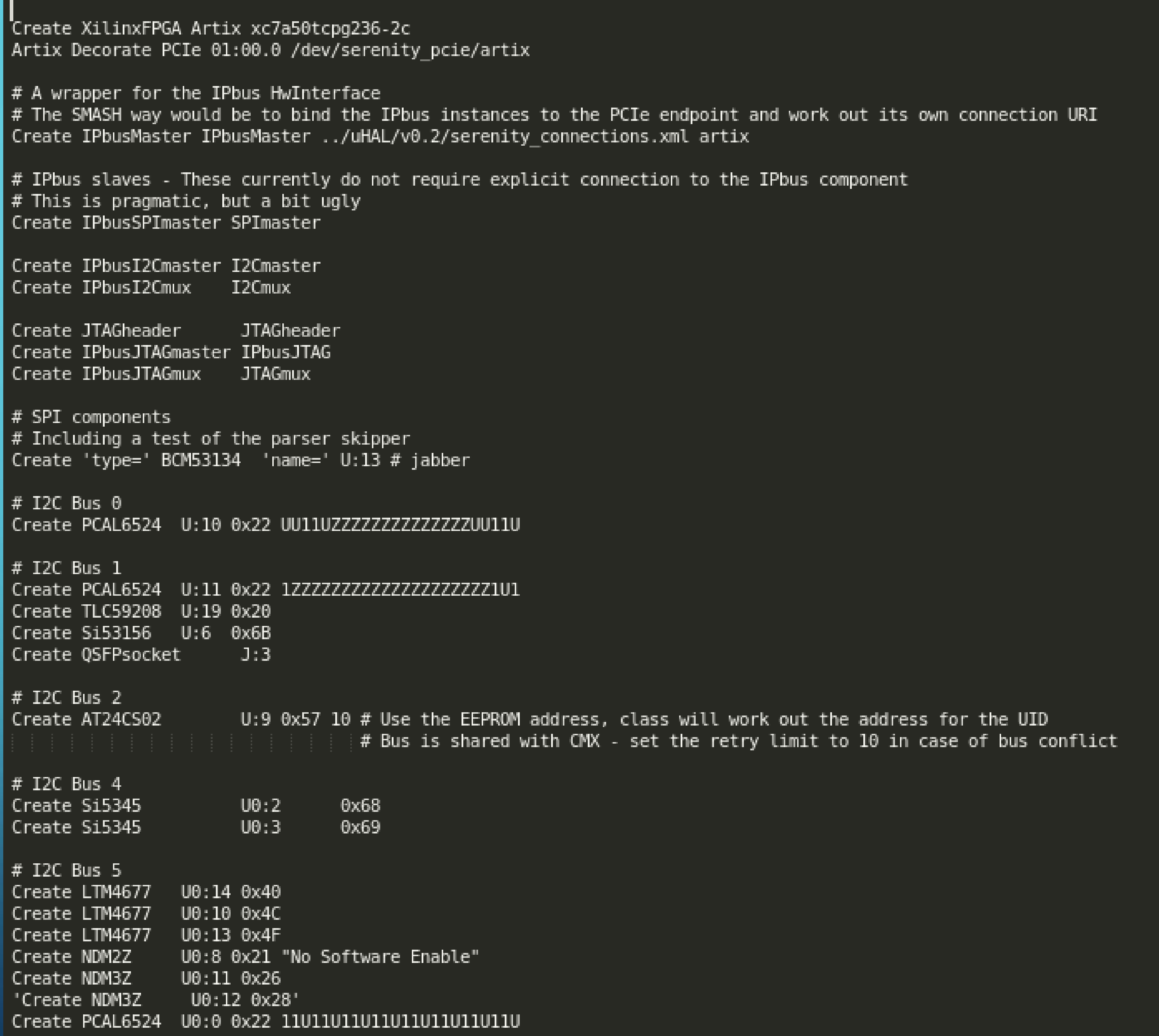
- No, it was developed for Serenity, but has already been used on
  - Two different revisions of Serenity
  - Prototype Serenity Board Tester
  - Serenity 1.2 Daughter-Card Tester
  - Bodged-together systems using an old PC, an old passive board, and some long lengths of wire
- My definition of generic truly means generic

DETAILS

- Currently run as an executable
- Plan is to make it a kernel process – a daemon – that user-code interacts with

DETAILS

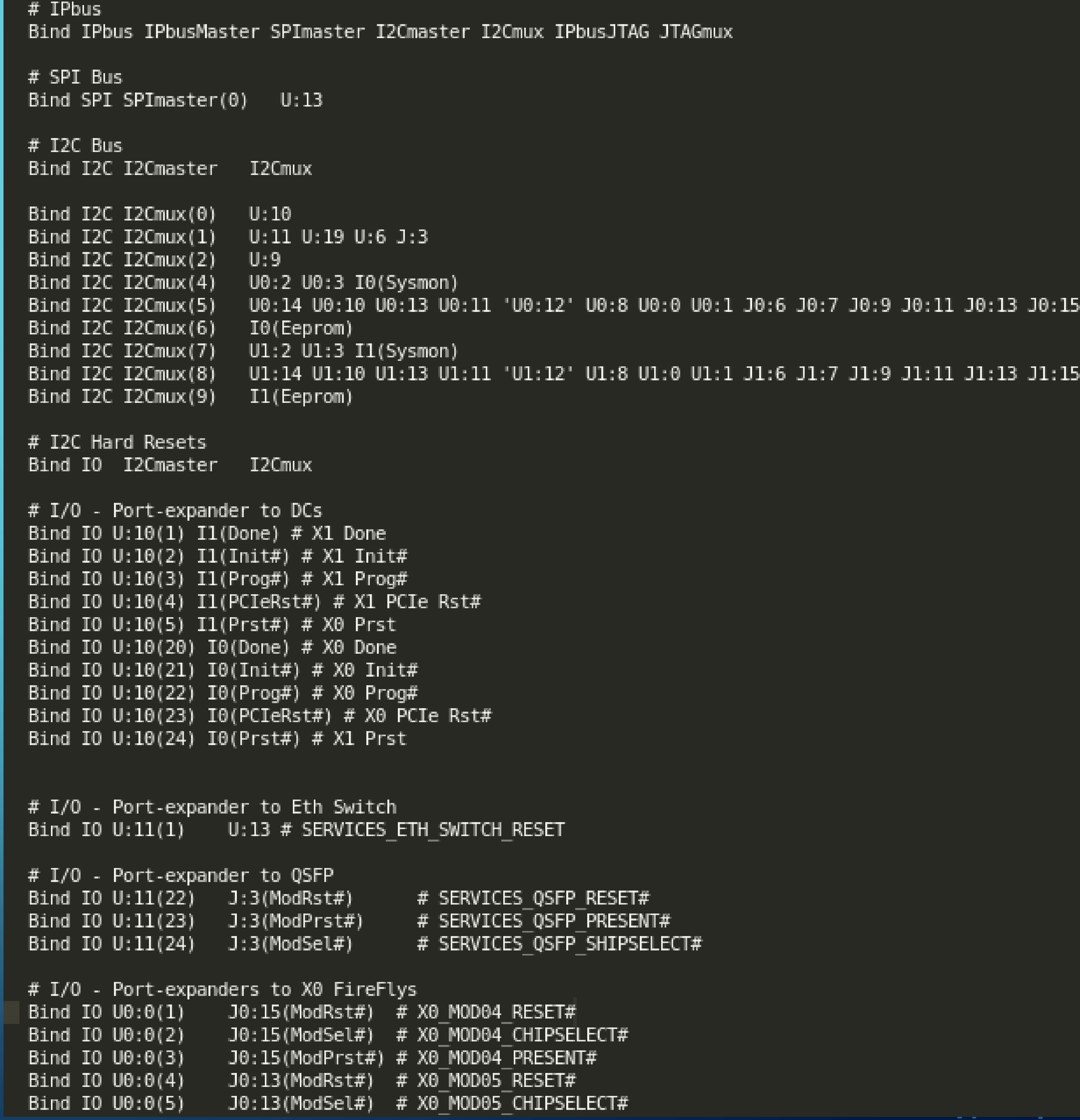
- A board is described by a plain text configuration file
- Same scripting interface as the interactive terminal



```
Create XilinxFPGA Artix xc7a50tcpg236-2c
Artix Decorate PCIe 01:00.0 /dev/serenity_pcie/artix

# A wrapper for the IPbus HwInterface
# The SMASH way would be to bind the IPbus instances to the PCIe endpoint and work out its own connection URI
Create IPbusMaster IPbusMaster ../uHAL/v0.2/serenity_connections.xml artix

# IPbus slaves - These currently do not require explicit connection to the IPbus component
# This is pragmatic, but a bit ugly
Create IPbusSPImaster SPImaster

Create IPbusI2Cmaster I2Cmaster
Create IPbusI2Cmux    I2Cmux

Create JTAGheader      JTAGheader
Create IPbusJTAGmaster IPbusJTAG
Create IPbusJTAGmux    JTAGmux

# SPI components
# Including a test of the parser skipper
Create 'type=' BCM53134  'name=' U:13 # jabber

# I2C Bus 0
Create PCAL6524  U:10 0x22 UU11UZZZZZZZZZZZZZZUU11U

# I2C Bus 1
Create PCAL6524  U:11 0x22 1ZZZZZZZZZZZZZZZZZZZZ1U1
Create TLC59208  U:19 0x20
Create Si53156   U:6  0x6B
Create QSFPsocket      J:3

# I2C Bus 2
Create AT24CS02        U:9 0x57 10 # Use the EEPROM address, class will work out the address for the UID
                                   # Bus is shared with CMX - set the retry limit to 10 in case of bus conflict

# I2C Bus 4
Create Si5345          U0:2      0x68
Create Si5345          U0:3      0x69

# I2C Bus 5
Create LTM4677   U0:14 0x40
Create LTM4677   U0:10 0x4C
Create LTM4677   U0:13 0x4F
Create NDM2Z     U0:8 0x21 "No Software Enable"
Create NDM3Z     U0:11 0x26
'Create NDM3Z     U0:12 0x28'
Create PCAL6524  U0:0 0x22 11U11U11U11U11U11U11U11U
```

DETAILS

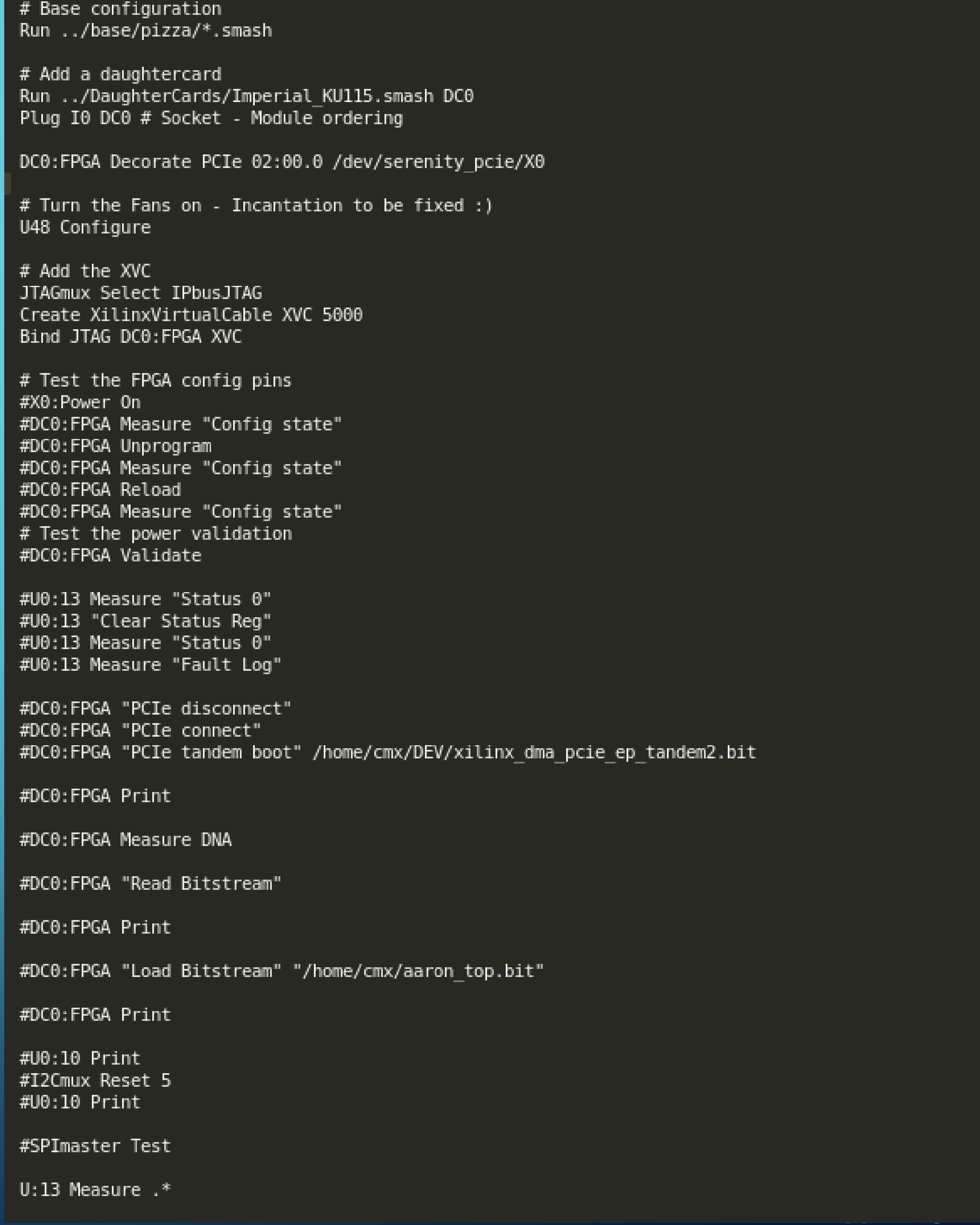
- The binding of component (joining the wires) is also done in via the same plain text interface



```
# IPbus
Bind IPbus IPbusMaster SPImaster I2Cmaster I2Cmux IPbusJTAG JTAGmux

# SPI Bus
Bind SPI SPImaster(0)   U:13

# I2C Bus
Bind I2C I2Cmaster   I2Cmux

Bind I2C I2Cmux(0)   U:10
Bind I2C I2Cmux(1)   U:11 U:19 U:6 J:3
Bind I2C I2Cmux(2)   U:9
Bind I2C I2Cmux(4)   U0:2 U0:3 I0(Sysmon)
Bind I2C I2Cmux(5)   U0:14 U0:10 U0:13 U0:11 'U0:12' U0:8 U0:0 U0:1 J0:6 J0:7 J0:9 J0:11 J0:13 J0:15
Bind I2C I2Cmux(6)   I0(Eeprom)
Bind I2C I2Cmux(7)   U1:2 U1:3 I1(Sysmon)
Bind I2C I2Cmux(8)   U1:14 U1:10 U1:13 U1:11 'U1:12' U1:8 U1:0 U1:1 J1:6 J1:7 J1:9 J1:11 J1:13 J1:15
Bind I2C I2Cmux(9)   I1(Eeprom)

# I2C Hard Resets
Bind IO  I2Cmaster   I2Cmux

# I/O - Port-expander to DCs
Bind IO U:10(1) I1(Done) # X1 Done
Bind IO U:10(2) I1(Init#) # X1 Init#
Bind IO U:10(3) I1(Prog#) # X1 Prog#
Bind IO U:10(4) I1(PCIeRst#) # X1 PCIe Rst#
Bind IO U:10(5) I1(Prst#) # X0 Prst
Bind IO U:10(20) I0(Done) # X0 Done
Bind IO U:10(21) I0(Init#) # X0 Init#
Bind IO U:10(22) I0(Prog#) # X0 Prog#
Bind IO U:10(23) I0(PCIeRst#) # X0 PCIe Rst#
Bind IO U:10(24) I0(Prst#) # X1 Prst


# I/O - Port-expander to Eth Switch
Bind IO U:11(1)    U:13 # SERVICES_ETH_SWITCH_RESET

# I/O - Port-expander to QSFP
Bind IO U:11(22)    J:3(ModRst#)      # SERVICES_QSFP_RESET#
Bind IO U:11(23)    J:3(ModPrst#)     # SERVICES_QSFP_PRESENT#
Bind IO U:11(24)    J:3(ModSel#)      # SERVICES_QSFP_SHIPSELECT#

# I/O - Port-expanders to X0 FireFlys
Bind IO U0:0(1)    J0:15(ModRst#)  # X0_MOD04_RESET#
Bind IO U0:0(2)    J0:15(ModSel#)  # X0_MOD04_CHIPSELECT#
Bind IO U0:0(3)    J0:15(ModPrst#) # X0_MOD04_PRESENT#
Bind IO U0:0(4)    J0:13(ModRst#)  # X0_MOD05_RESET#
Bind IO U0:0(5)    J0:13(ModSel#)  # X0_MOD05_CHIPSELECT#
```

DETAILS

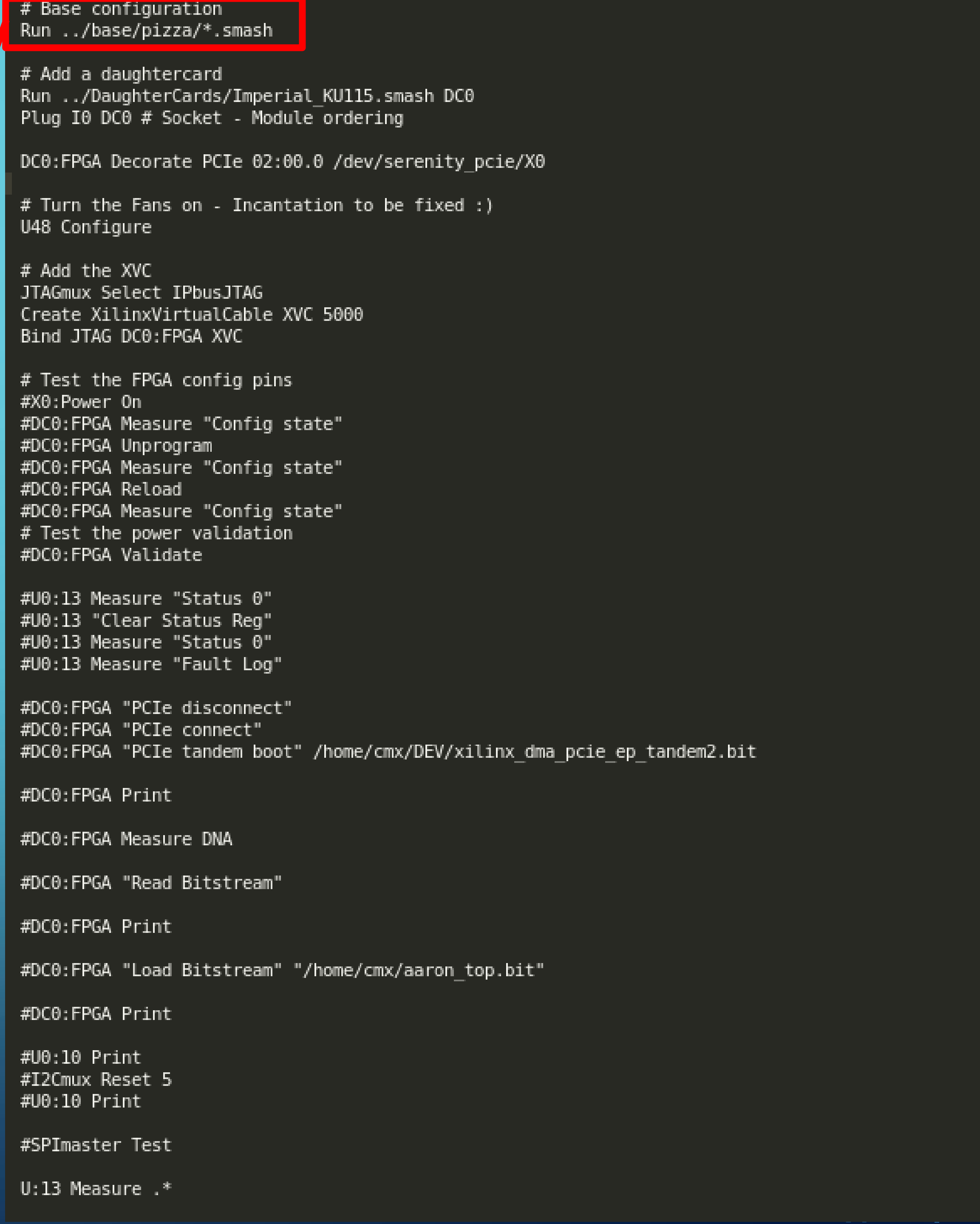
- And you can configure the board via scripts using the same scripting interface
- Or you can do it via an interactive terminal



```
# Base configuration
Run ../base/pizza/*.smash

# Add a daughtercard
Run ../DaughterCards/Imperial_KU115.smash DC0
Plug I0 DC0 # Socket - Module ordering

DC0:FPGA Decorate PCIe 02:00.0 /dev/serenity_pcie/X0

# Turn the Fans on - Incantation to be fixed :)
U48 Configure

# Add the XVC
JTAGmux Select IPbusJTAG
Create XilinxVirtualCable XVC 5000
Bind JTAG DC0:FPGA XVC

# Test the FPGA config pins
#X0:Power On
#DC0:FPGA Measure "Config state"
#DC0:FPGA Unprogram
#DC0:FPGA Measure "Config state"
#DC0:FPGA Reload
#DC0:FPGA Measure "Config state"
# Test the power validation
#DC0:FPGA Validate

#U0:13 Measure "Status 0"
#U0:13 "Clear Status Reg"
#U0:13 Measure "Status 0"
#U0:13 Measure "Fault Log"

#DC0:FPGA "PCIe disconnect"
#DC0:FPGA "PCIe connect"
#DC0:FPGA "PCIe tandem boot" /home/cmx/DEV/xilinx_dma_pcie_ep_tandem2.bit

#DC0:FPGA Print

#DC0:FPGA Measure DNA

#DC0:FPGA "Read Bitstream"

#DC0:FPGA Print

#DC0:FPGA "Load Bitstream" "/home/cmx/aaron_top.bit"

#DC0:FPGA Print

#U0:10 Print
#I2Cmux Reset 5
#U0:10 Print

#SPImaster Test

U:13 Measure .*
```

DETAILS

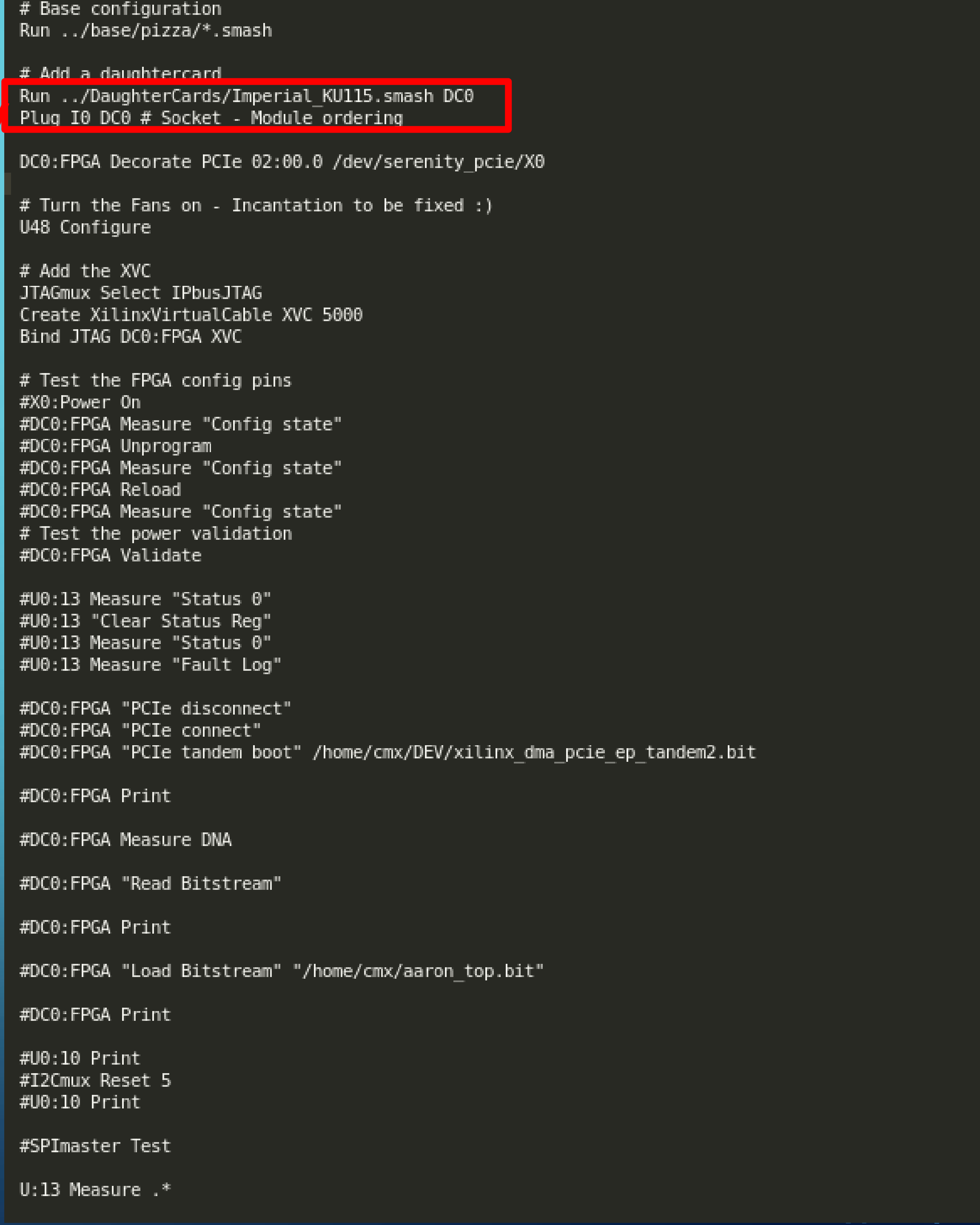
Call the scripts which describe the board



```
# Base configuration
Run ../base/pizza/*.smash

# Add a daughtercard
Run ../DaughterCards/Imperial_KU115.smash DC0
Plug I0 DC0 # Socket - Module ordering

DC0:FPGA Decorate PCIe 02:00.0 /dev/serenity_pcie/X0

# Turn the Fans on - Incantation to be fixed :)
U48 Configure

# Add the XVC
JTAGmux Select IPbusJTAG
Create XilinxVirtualCable XVC 5000
Bind JTAG DC0:FPGA XVC

# Test the FPGA config pins
#X0:Power On
#DC0:FPGA Measure "Config state"
#DC0:FPGA Unprogram
#DC0:FPGA Measure "Config state"
#DC0:FPGA Reload
#DC0:FPGA Measure "Config state"
# Test the power validation
#DC0:FPGA Validate

#U0:13 Measure "Status 0"
#U0:13 "Clear Status Reg"
#U0:13 Measure "Status 0"
#U0:13 Measure "Fault Log"

#DC0:FPGA "PCIe disconnect"
#DC0:FPGA "PCIe connect"
#DC0:FPGA "PCIe tandem boot" /home/cmx/DEV/xilinx_dma_pcie_ep_tandem2.bit

#DC0:FPGA Print

#DC0:FPGA Measure DNA

#DC0:FPGA "Read Bitstream"

#DC0:FPGA Print

#DC0:FPGA "Load Bitstream" "/home/cmx/aaron_top.bit"

#DC0:FPGA Print

#U0:10 Print
#I2Cmux Reset 5
#U0:10 Print

#SPImaster Test

U:13 Measure .*
```

DETAILS

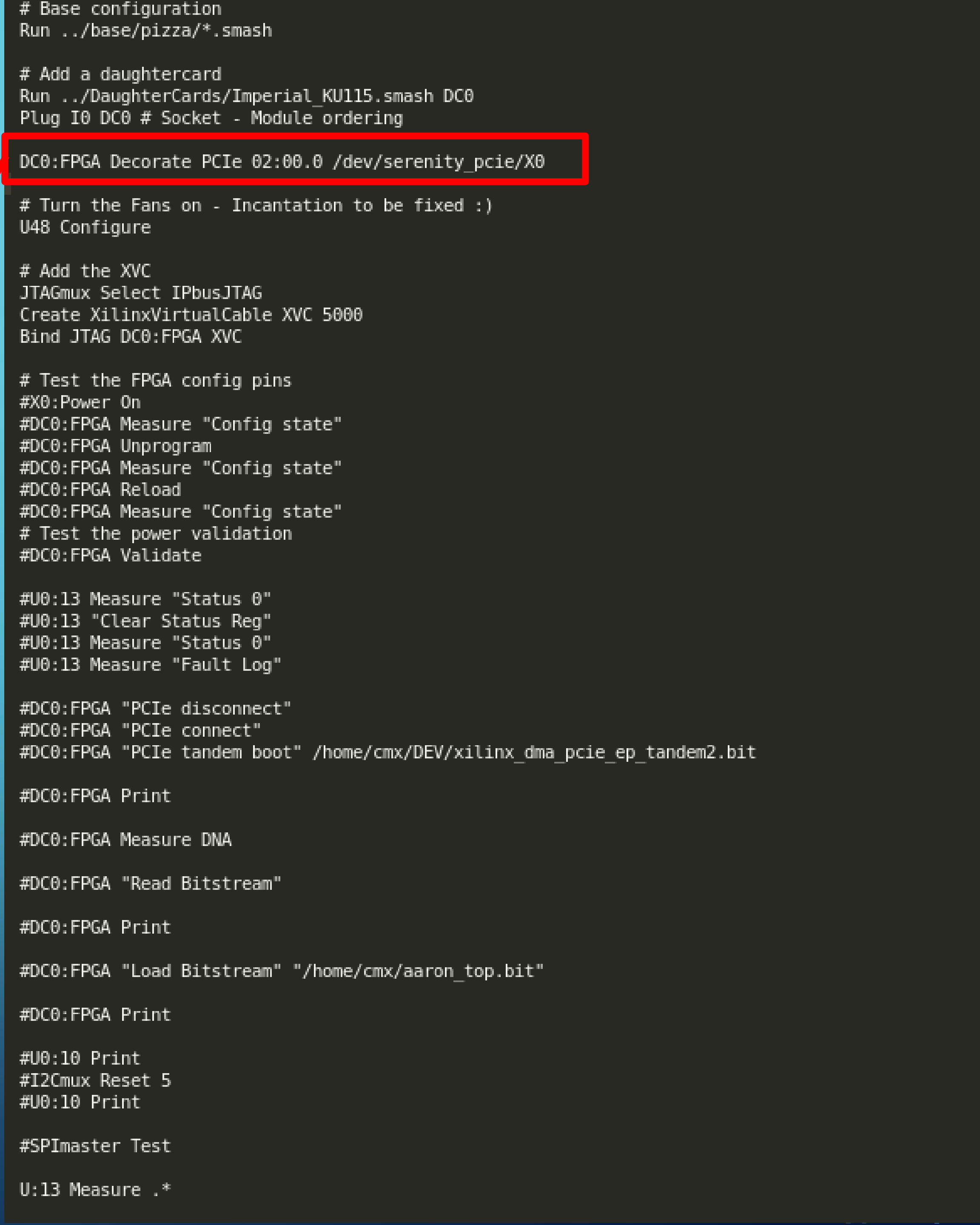
Create a daughter-card and plug it onto the board



```
# Base configuration
Run ../base/pizza/*.smash

# Add a daughtercard
Run ../DaughterCards/Imperial_KU115.smash DC0
Plug I0 DC0 # Socket - Module ordering

DC0:FPGA Decorate PCIe 02:00.0 /dev/serenity_pcie/X0

# Turn the Fans on - Incantation to be fixed :)
U48 Configure

# Add the XVC
JTAGmux Select IPbusJTAG
Create XilinxVirtualCable XVC 5000
Bind JTAG DC0:FPGA XVC

# Test the FPGA config pins
#X0:Power On
#DC0:FPGA Measure "Config state"
#DC0:FPGA Unprogram
#DC0:FPGA Measure "Config state"
#DC0:FPGA Reload
#DC0:FPGA Measure "Config state"
# Test the power validation
#DC0:FPGA Validate

#U0:13 Measure "Status 0"
#U0:13 "Clear Status Reg"
#U0:13 Measure "Status 0"
#U0:13 Measure "Fault Log"

#DC0:FPGA "PCIe disconnect"
#DC0:FPGA "PCIe connect"
#DC0:FPGA "PCIe tandem boot" /home/cmx/DEV/xilinx_dma_pcie_ep_tandem2.bit

#DC0:FPGA Print

#DC0:FPGA Measure DNA

#DC0:FPGA "Read Bitstream"

#DC0:FPGA Print

#DC0:FPGA "Load Bitstream" "/home/cmx/aaron_top.bit"

#DC0:FPGA Print

#U0:10 Print
#I2Cmux Reset 5
#U0:10 Print

#SPImaster Test

U:13 Measure .*
```

DETAILS

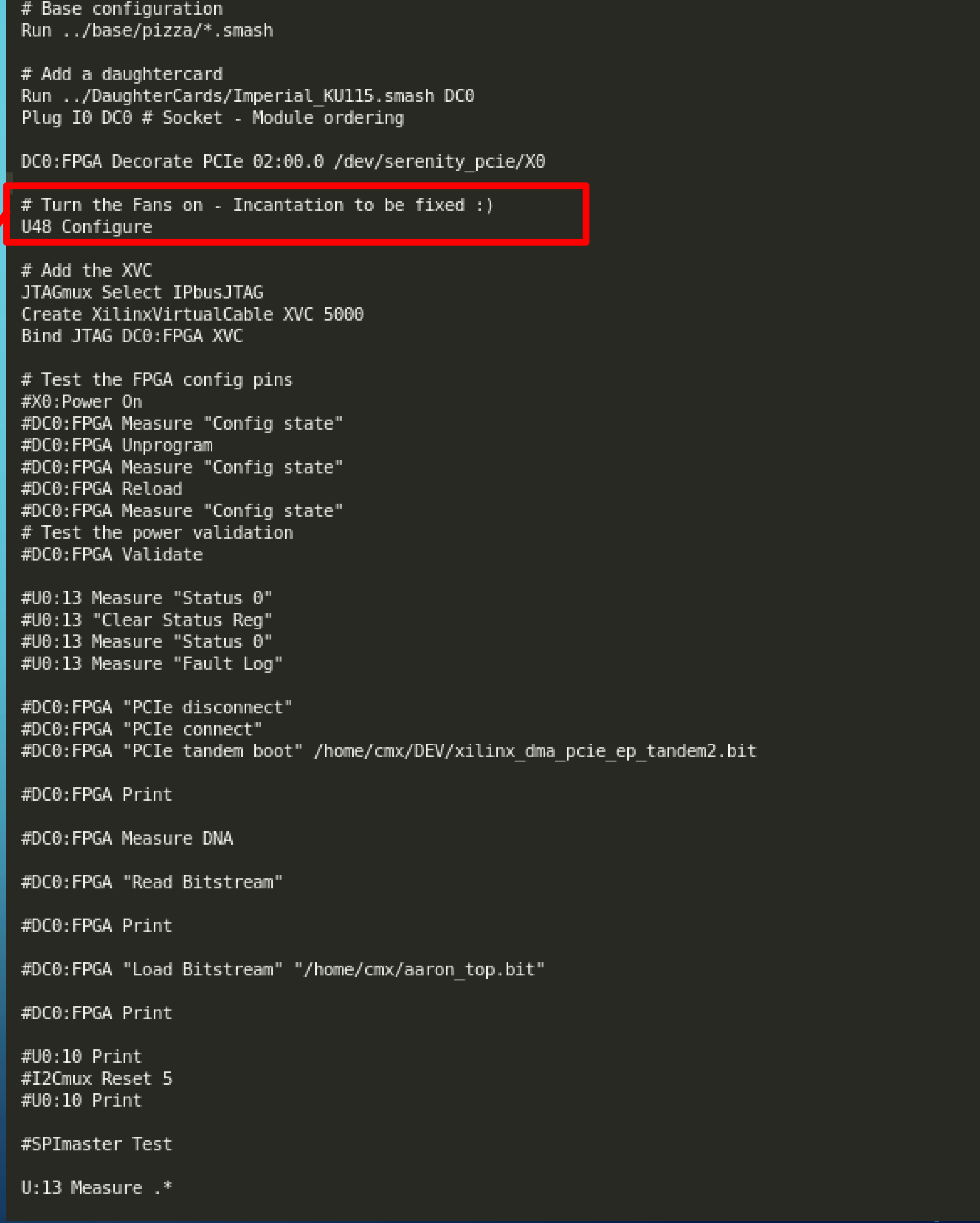
Add a PCIe interface to the FPGA on the Daughtercard



```
# Base configuration
Run ../base/pizza/*.smash

# Add a daughtercard
Run ../DaughterCards/Imperial_KU115.smash DC0
Plug I0 DC0 # Socket - Module ordering

DC0:FPGA Decorate PCIe 02:00.0 /dev/serenity_pcie/X0

# Turn the Fans on - Incantation to be fixed :)
U48 Configure

# Add the XVC
JTAGmux Select IPbusJTAG
Create XilinxVirtualCable XVC 5000
Bind JTAG DC0:FPGA XVC

# Test the FPGA config pins
#X0:Power On
#DC0:FPGA Measure "Config state"
#DC0:FPGA Unprogram
#DC0:FPGA Measure "Config state"
#DC0:FPGA Reload
#DC0:FPGA Measure "Config state"
# Test the power validation
#DC0:FPGA Validate

#U0:13 Measure "Status 0"
#U0:13 "Clear Status Reg"
#U0:13 Measure "Status 0"
#U0:13 Measure "Fault Log"

#DC0:FPGA "PCIe disconnect"
#DC0:FPGA "PCIe connect"
#DC0:FPGA "PCIe tandem boot" /home/cmx/DEV/xilinx_dma_pcie_ep_tandem2.bit

#DC0:FPGA Print

#DC0:FPGA Measure DNA

#DC0:FPGA "Read Bitstream"

#DC0:FPGA Print

#DC0:FPGA "Load Bitstream" "/home/cmx/aaron_top.bit"

#DC0:FPGA Print

#U0:10 Print
#I2Cmux Reset 5
#U0:10 Print

#SPImaster Test

U:13 Measure .*
```

DETAILS

Configure a component

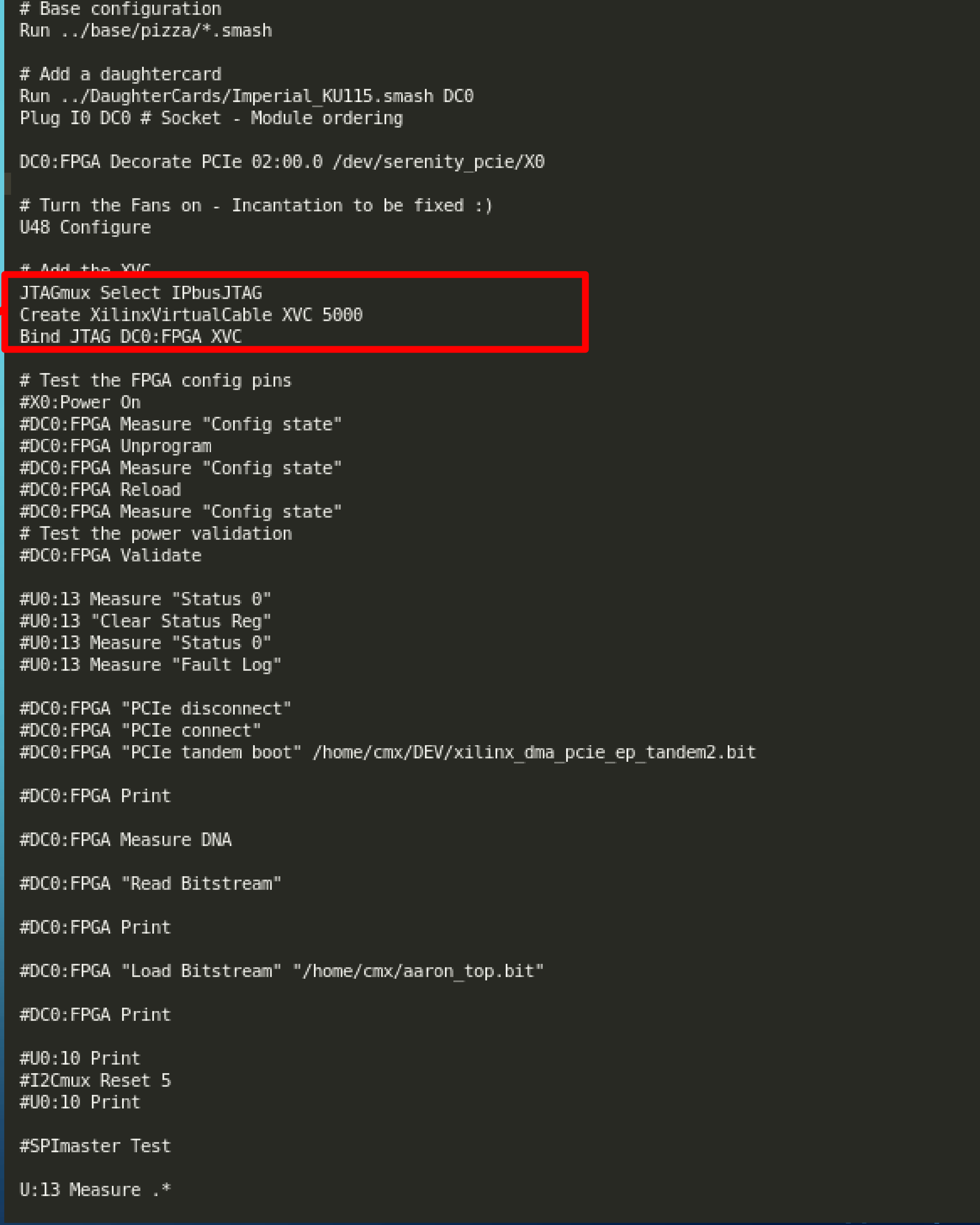


```
# Base configuration
Run ../base/pizza/*.smash

# Add a daughtercard
Run ../DaughterCards/Imperial_KU115.smash DC0
Plug I0 DC0 # Socket - Module ordering

DC0:FPGA Decorate PCIe 02:00.0 /dev/serenity_pcie/X0

# Turn the Fans on - Incantation to be fixed :)
U48 Configure

# Add the XVC
JTAGmux Select IPbusJTAG
Create XilinxVirtualCable XVC 5000
Bind JTAG DC0:FPGA XVC

# Test the FPGA config pins
#X0:Power On
#DC0:FPGA Measure "Config state"
#DC0:FPGA Unprogram
#DC0:FPGA Measure "Config state"
#DC0:FPGA Reload
#DC0:FPGA Measure "Config state"
# Test the power validation
#DC0:FPGA Validate

#U0:13 Measure "Status 0"
#U0:13 "Clear Status Reg"
#U0:13 Measure "Status 0"
#U0:13 Measure "Fault Log"

#DC0:FPGA "PCIe disconnect"
#DC0:FPGA "PCIe connect"
#DC0:FPGA "PCIe tandem boot" /home/cmx/DEV/xilinx_dma_pcie_ep_tandem2.bit

#DC0:FPGA Print

#DC0:FPGA Measure DNA

#DC0:FPGA "Read Bitstream"

#DC0:FPGA Print

#DC0:FPGA "Load Bitstream" "/home/cmx/aaron_top.bit"

#DC0:FPGA Print

#U0:10 Print
#I2Cmux Reset 5
#U0:10 Print

#SPImaster Test

U:13 Measure .*
```

DETAILS

Configure the JTAG chain

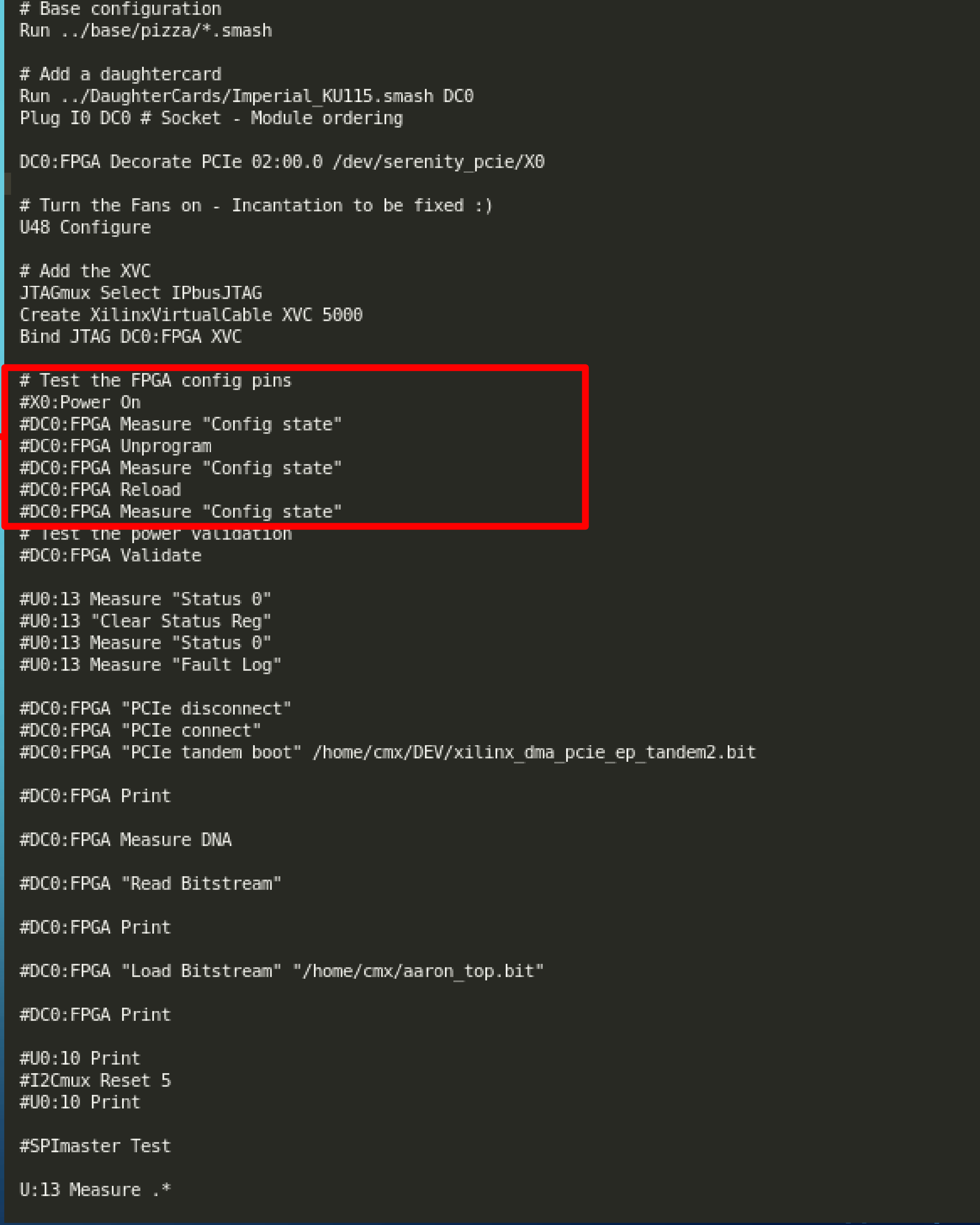
Bind an XVC – a “software component” – to the FPGA



```
# Base configuration
Run ../base/pizza/*.smash

# Add a daughtercard
Run ../DaughterCards/Imperial_KU115.smash DC0
Plug I0 DC0 # Socket - Module ordering

DC0:FPGA Decorate PCIe 02:00.0 /dev/serenity_pcie/X0

# Turn the Fans on - Incantation to be fixed :)
U48 Configure

# Add the XVC
JTAGmux Select IPbusJTAG
Create XilinxVirtualCable XVC 5000
Bind JTAG DC0:FPGA XVC

# Test the FPGA config pins
#X0:Power On
#DC0:FPGA Measure "Config state"
#DC0:FPGA Unprogram
#DC0:FPGA Measure "Config state"
#DC0:FPGA Reload
#DC0:FPGA Measure "Config state"
# Test the power validation
#DC0:FPGA Validate

#U0:13 Measure "Status 0"
#U0:13 "Clear Status Reg"
#U0:13 Measure "Status 0"
#U0:13 Measure "Fault Log"

#DC0:FPGA "PCIe disconnect"
#DC0:FPGA "PCIe connect"
#DC0:FPGA "PCIe tandem boot" /home/cmx/DEV/xilinx_dma_pcie_ep_tandem2.bit

#DC0:FPGA Print

#DC0:FPGA Measure DNA

#DC0:FPGA "Read Bitstream"

#DC0:FPGA Print

#DC0:FPGA "Load Bitstream" "/home/cmx/aaron_top.bit"

#DC0:FPGA Print

#U0:10 Print
#I2Cmux Reset 5
#U0:10 Print

#SPImaster Test

U:13 Measure .*
```

DETAILS

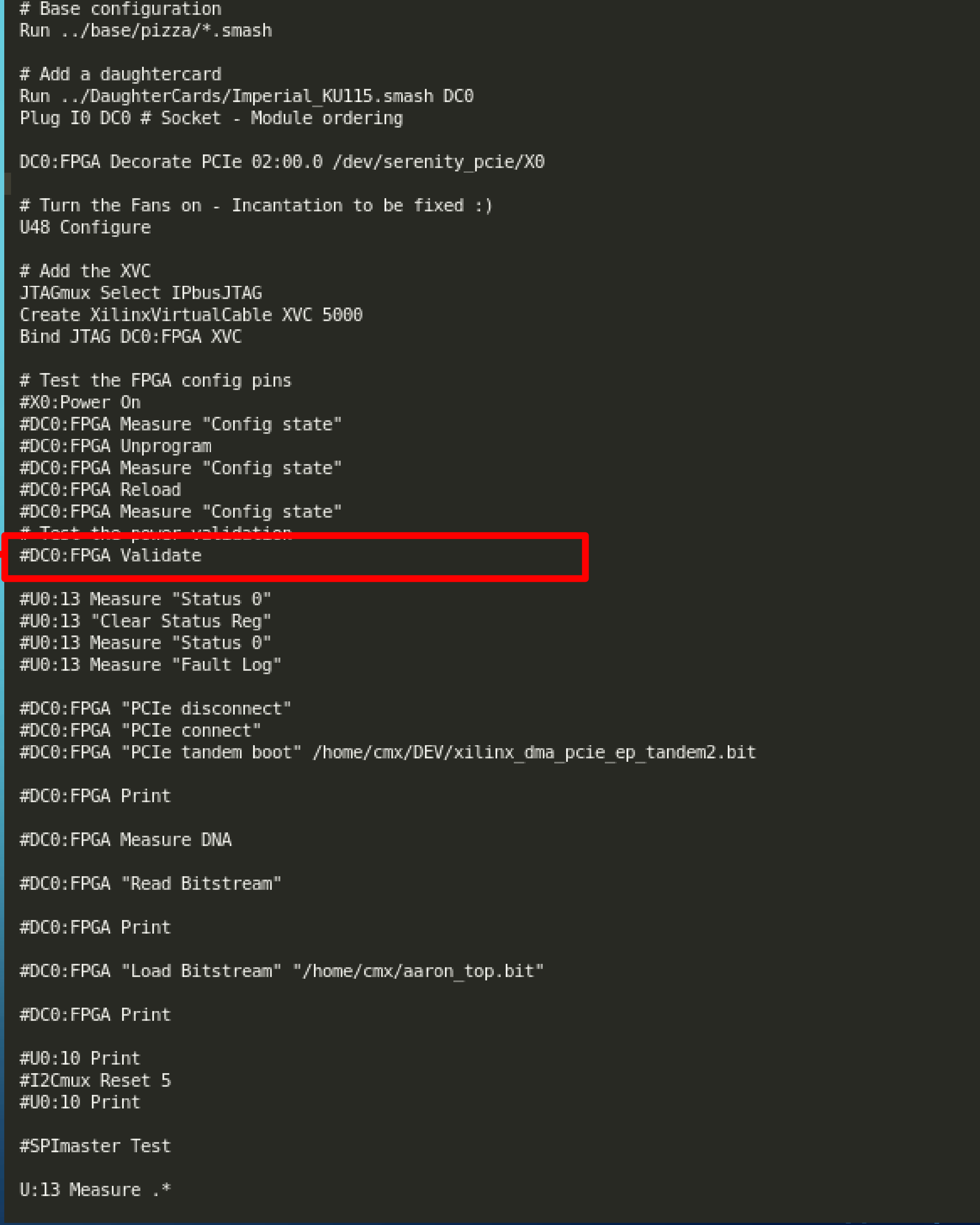
“Measure” values from the component



```
# Base configuration
Run ../base/pizza/*.smash

# Add a daughtercard
Run ../DaughterCards/Imperial_KU115.smash DC0
Plug I0 DC0 # Socket - Module ordering

DC0:FPGA Decorate PCIe 02:00.0 /dev/serenity_pcie/X0

# Turn the Fans on - Incantation to be fixed :)
U48 Configure

# Add the XVC
JTAGmux Select IPbusJTAG
Create XilinxVirtualCable XVC 5000
Bind JTAG DC0:FPGA XVC

# Test the FPGA config pins
#X0:Power On
#DC0:FPGA Measure "Config state"
#DC0:FPGA Unprogram
#DC0:FPGA Measure "Config state"
#DC0:FPGA Reload
#DC0:FPGA Measure "Config state"
# Test the power validation
#DC0:FPGA Validate

#U0:13 Measure "Status 0"
#U0:13 "Clear Status Reg"
#U0:13 Measure "Status 0"
#U0:13 Measure "Fault Log"

#DC0:FPGA "PCIe disconnect"
#DC0:FPGA "PCIe connect"
#DC0:FPGA "PCIe tandem boot" /home/cmx/DEV/xilinx_dma_pcie_ep_tandem2.bit

#DC0:FPGA Print

#DC0:FPGA Measure DNA

#DC0:FPGA "Read Bitstream"

#DC0:FPGA Print

#DC0:FPGA "Load Bitstream" "/home/cmx/aaron_top.bit"

#DC0:FPGA Print

#U0:10 Print
#I2Cmux Reset 5
#U0:10 Print

#SPImaster Test

U:13 Measure .*
```

DETAILS

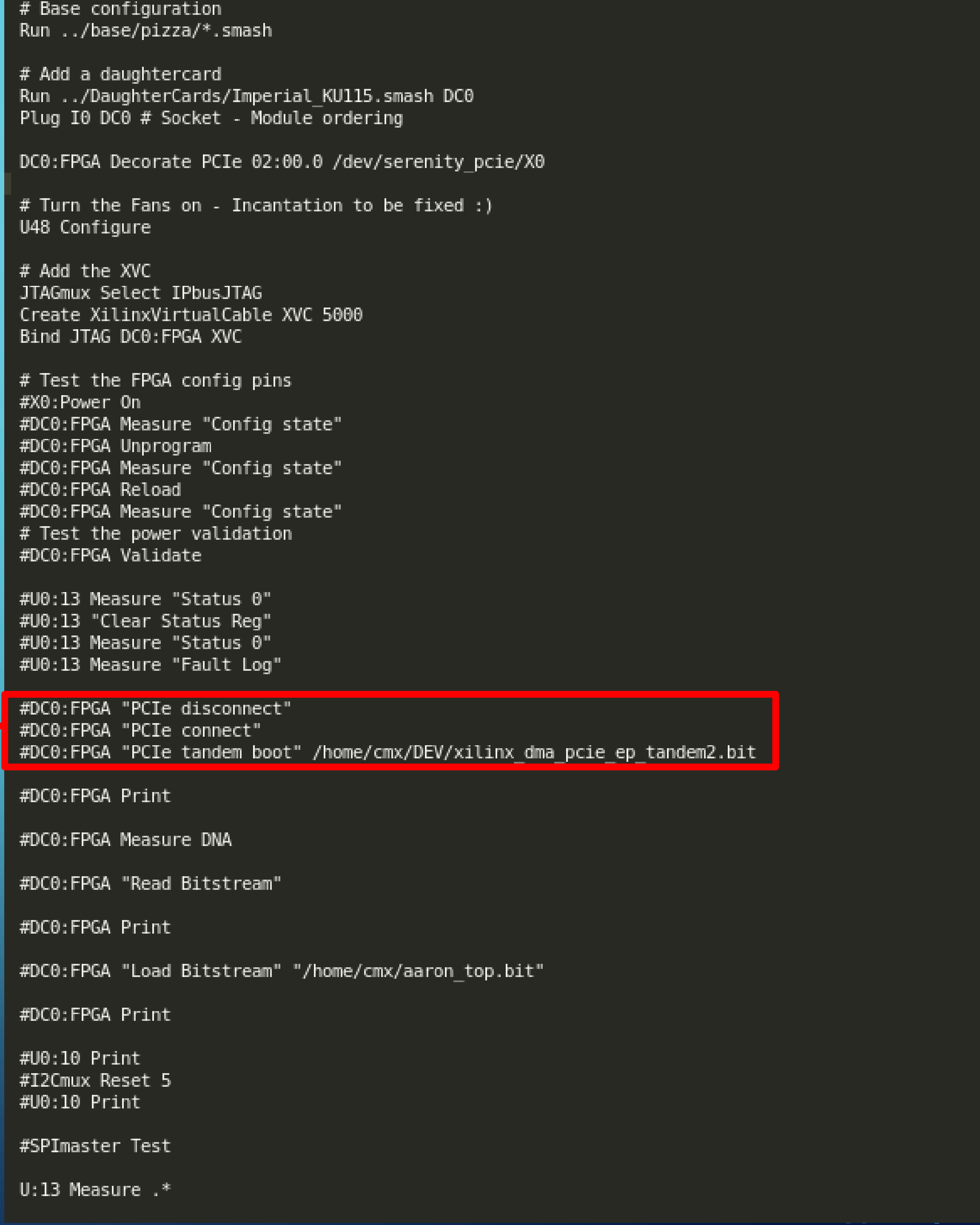
Get the software to check it's hardware counterpart (whatever that may mean)



```
# Base configuration
Run ../base/pizza/*.smash

# Add a daughtercard
Run ../DaughterCards/Imperial_KU115.smash DC0
Plug I0 DC0 # Socket - Module ordering

DC0:FPGA Decorate PCIe 02:00.0 /dev/serenity_pcie/X0

# Turn the Fans on - Incantation to be fixed :)
U48 Configure

# Add the XVC
JTAGmux Select IPbusJTAG
Create XilinxVirtualCable XVC 5000
Bind JTAG DC0:FPGA XVC

# Test the FPGA config pins
#X0:Power On
#DC0:FPGA Measure "Config state"
#DC0:FPGA Unprogram
#DC0:FPGA Measure "Config state"
#DC0:FPGA Reload
#DC0:FPGA Measure "Config state"
# Test the power validation
#DC0:FPGA Validate

#U0:13 Measure "Status 0"
#U0:13 "Clear Status Reg"
#U0:13 Measure "Status 0"
#U0:13 Measure "Fault Log"

#DC0:FPGA "PCIe disconnect"
#DC0:FPGA "PCIe connect"
#DC0:FPGA "PCIe tandem boot" /home/cmx/DEV/xilinx_dma_pcie_ep_tandem2.bit

#DC0:FPGA Print

#DC0:FPGA Measure DNA

#DC0:FPGA "Read Bitstream"

#DC0:FPGA Print

#DC0:FPGA "Load Bitstream" "/home/cmx/aaron_top.bit"

#DC0:FPGA Print

#U0:10 Print
#I2Cmux Reset 5
#U0:10 Print

#SPImaster Test

U:13 Measure .*
```

DETAILS

PCIe management



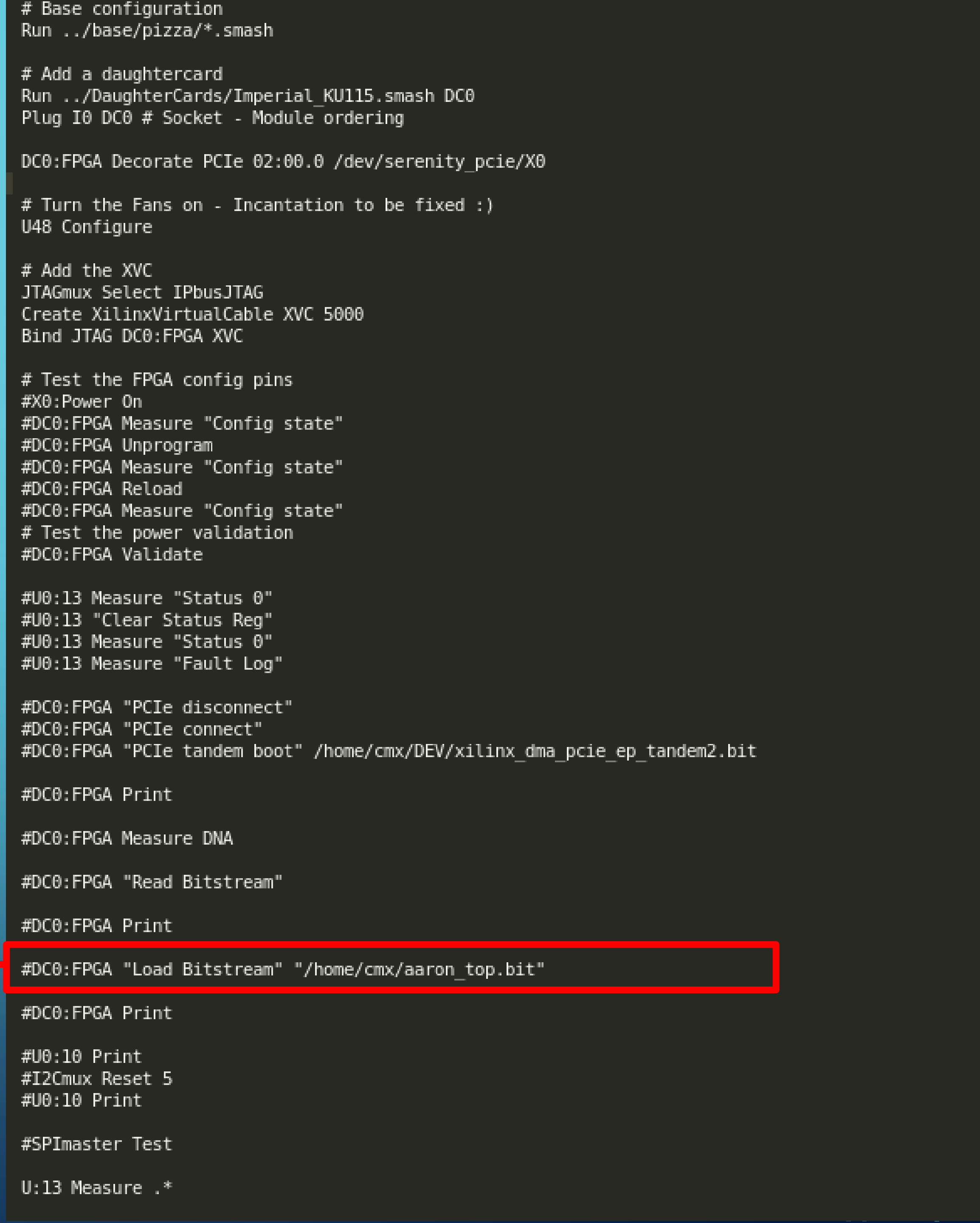
```
# Base configuration
Run ../base/pizza/*.smash

# Add a daughtercard
Run ../DaughterCards/Imperial_KU115.smash DC0
Plug I0 DC0 # Socket - Module ordering

DC0:FPGA Decorate PCIe 02:00.0 /dev/serenity_pcie/X0

# Turn the Fans on - Incantation to be fixed :)
U48 Configure

# Add the XVC
JTAGmux Select IPbusJTAG
Create XilinxVirtualCable XVC 5000
Bind JTAG DC0:FPGA XVC

# Test the FPGA config pins
#X0:Power On
#DC0:FPGA Measure "Config state"
#DC0:FPGA Unprogram
#DC0:FPGA Measure "Config state"
#DC0:FPGA Reload
#DC0:FPGA Measure "Config state"
# Test the power validation
#DC0:FPGA Validate

#U0:13 Measure "Status 0"
#U0:13 "Clear Status Reg"
#U0:13 Measure "Status 0"
#U0:13 Measure "Fault Log"

#DC0:FPGA "PCIe disconnect"
#DC0:FPGA "PCIe connect"
#DC0:FPGA "PCIe tandem boot" /home/cmx/DEV/xilinx_dma_pcie_ep_tandem2.bit

#DC0:FPGA Print

#DC0:FPGA Measure DNA

#DC0:FPGA "Read Bitstream"

#DC0:FPGA Print

#DC0:FPGA "Load Bitstream" "/home/cmx/aaron_top.bit"

#DC0:FPGA Print

#U0:10 Print
#I2Cmux Reset 5
#U0:10 Print

#SPImaster Test

U:13 Measure .*
```

DETAILS

Load a bitstream directly from file

Note – this involves converting the bitfile to a JTAG stream in software then sending the JTAG data over IPbus over PCIe

The user sees none of this



```
# Base configuration
Run ../base/pizza/*.smash

# Add a daughtercard
Run ../DaughterCards/Imperial_KU115.smash DC0
Plug I0 DC0 # Socket - Module ordering

DC0:FPGA Decorate PCIe 02:00.0 /dev/serenity_pcie/X0

# Turn the Fans on - Incantation to be fixed :)
U48 Configure

# Add the XVC
JTAGmux Select IPbusJTAG
Create XilinxVirtualCable XVC 5000
Bind JTAG DC0:FPGA XVC

# Test the FPGA config pins
#X0:Power On
#DC0:FPGA Measure "Config state"
#DC0:FPGA Unprogram
#DC0:FPGA Measure "Config state"
#DC0:FPGA Reload
#DC0:FPGA Measure "Config state"
# Test the power validation
#DC0:FPGA Validate

#U0:13 Measure "Status 0"
#U0:13 "Clear Status Reg"
#U0:13 Measure "Status 0"
#U0:13 Measure "Fault Log"

#DC0:FPGA "PCIe disconnect"
#DC0:FPGA "PCIe connect"
#DC0:FPGA "PCIe tandem boot" /home/cmx/DEV/xilinx_dma_pcie_ep_tandem2.bit

#DC0:FPGA Print

#DC0:FPGA Measure DNA

#DC0:FPGA "Read Bitstream"

#DC0:FPGA Print

#DC0:FPGA "Load Bitstream" "/home/cmx/aaron_top.bit"

#DC0:FPGA Print

#U0:10 Print
#I2Cmux Reset 5
#U0:10 Print

#SPImaster Test

U:13 Measure .*
```