FrontPanel SDK Examples

Examples Home

The FrontPanel SDK solves a wide variety of PC (Software) to FPGA (Hardware) communication scenarios. We’ve compiled this set of common scenarios as part tutorial, part FAQ to help you understand how FrontPanel can be put to use in your design.

The examples are arranged in general categories:

- **Infrastructure** – Enumerating, opening, and configuring devices.
- **Communication** – Synchronous and asynchronous data communication.
- **API Functions** – Additional API support functionality.

Device Support

The FrontPanel SDK supports our USB 2.0, USB 3.0 and PCI Express devices. While software and HDL compatibility among these device families is very strong, there are some subtle differences. For example, the USB 2.0 Wires, Triggers, and Pipes are 16-bit wide at the HDL interface. The USB 3.0 Wires, Triggers, and Pipes are 32-bit wide.

Aside from these minor differences, you should find a lot of familiarity with the examples here even if they aren’t exact to your device.

https://www.opalkelly.com/examples/home/
## Enumerating Devices

Prior to opening a device, you can use an instance of `okCFrontPanel` to list available devices. It’s important to note that a device is only available in one thread or process at a time and therefore will not be visible in this list if another thread or process has already opened it.

We’ll use `GetDeviceCount()` to query available devices, then use `GetDeviceListModel()` and `GetDeviceListSerial()` to list the model numbers and serial numbers for each attached device, respectively.

```python
1. device = ok.okCFrontPanel()
2. deviceCount = device.GetDeviceCount()
3. for i in range(deviceCount):
   4.    print 'Device[{0}] Model: {1}'.format(i, device.GetDeviceListModel(i))
   5.    print 'Device[{0}] Serial: {1}'.format(i, device.GetDeviceListSerial(i))
```

## Open the first device available

Opening the first available device requires only an instance of `okCFrontPanel` and is done by calling `OpenBySerial` without a specified serial number. In C++ `OpenBySerial` can take NULL as an input, but C#, Java, and Python require an empty string.

```python
1. dev = ok.okCFrontPanel()
2. dev.OpenBySerial('')
```

## Configure the FPGA

To configure the FPGA, use a bitfile generated by the Xilinx or Altera tools. Xilinx tools output a bitfile with the extension “.bit” that may be used directly. Altera tools must be setup to output a raw bitfile with the extension “.rbf”. Checking for errors with the `ConfigureFPGA` method can help flag important errors that may be difficult to identify later in the application. Note that `ConfigureFPGA` requires a bitfile to be specified, as it will not automatically detect an available bitfile.

**Important Note:** When configuring using a specific bitfile, make sure that the bitfile is in the working directory of the application, which may or may not be the same directory as the source file or executable.

```python
1. dev = ok.okCFrontPanel()
2. dev.OpenBySerial('')
3. error = dev.ConfigureFPGA('video_capture.bit')
4. # It’s a good idea to check for errors here!!
```
Test the FrontPanel Interface

After configuring the FPGA with your desired bitfile, it’s a good idea to test that FrontPanel is enabled in your FPGA configuration using `okCFrontPanel::IsFrontPanelEnabled()`. Testing FrontPanel after configuration is a good way to check for a variety of issues. For example,

- It assures that the bitfile was found and transferred to the device.
- It assures that the bitfile has proper host interface pin mappings.
  
  ```
  dev = ok.okCFrontPanel()
  dev.OpenBySerial(""")
  error = dev.ConfigureFPGA("example.bit")
  # It's a good idea to check for errors here!!
  ``

- `IsFrontPanelEnabled` returns true if FrontPanel is detected.
  
  ```
  if true == dev.IsFrontPanelEnabled():
    print "FrontPanel host interface enabled."
  else:
    sys.stderr.write("FrontPanel host interface not detected.")
  ```
HDL Framework (USB 3.0)

This section contains template definitions of the top-level module for FrontPanel-enabled USB 3.0 devices in Verilog and VHDL. It is applicable to the examples for Wires, Triggers, Pipes, and Registers. This boilerplate is familiar if you’ve looked through our samples. To keep the examples brief, we won’t duplicate this code for each one.

1. module Framework(
2.   input wire [4:0] okUH,
3.   output wire [2:0] okHU,
4.   input wire [31:0] okUHU,
5.   inout wire okAA,
6.   input wire sys_clkn,
7.   input wire sys_clkp,
8.   input wire reset
9.   // Your signals here
10. );
11.
12. // Clock
13. wire sys_clk;
14.
15. IBUFGDS osc_clk(
16.   .O(sys_clk),
17.   .I(sys_clkp),
18.   .IB(sys_clkn)
19. );
20.
21. // FP wires
22. wire okClk;
23. wire [112:0] okHE;
24. wire [64:0] okEH;
25. wire [5*65-1:0] okEHx;
26. // Adjust size of okEHx to fit the number of outgoing endpoints in your design (n*65-1:0)
27.
28. // Your HDL here
29.
30. okHost hostIF (  
31.   .OKUH(okUH),
32.   .OKHU(okHU),
33.   .OKUHU(okUHU),
34.   .OKClk(okClk),
35.   .OKAA(okAA),
36.   .OKHE(okHE),
37.   .OKEH(okEH)
38. );
39.
40. // Adjust N to fit the number of outgoing endpoints in your design (.N(n))
41. okWireOR # (.N(1)) wireOR (okEH, okEHx);
42.
43. // Your FrontPanel module instantiations here
44.
45. endmodule
Simple Pin I/O

The SetWireInValue and GetWireOutValue methods in the okCFrontPanel class allow you to set and get individual pin values across the entire defined Wire address space (0x00-0x1F for WireIn, 0x20-0x3F for WireOut). You can set the WireIn values by passing in the address of the WireIn and its new value. You can read individual WireOut values by passing the address of the WireOut and a bit mask to extract only the relevant bits. After setting a wire in value and before reading a wire out value, remember to update the WireIns and WireOuts, respectively. Otherwise, the values will not be accurate.

Note: This section contains both software and HDL portions. The software and HDL must work in tandem if FrontPanel is to be used on the PC end to perform tasks on the FPGA. The HDL in this section is designed to be set within the FrontPanel Framework HDL, available on the HDL Framework page for USB 2.0 and USB 3.0. It is assumed that the FPGA is configured with the bitfile generated by the HDL before the hardware is run. For specific information about each of these methods or modules, consult the FrontPanel User’s Manual, the FrontPanel API guide, and the samples and README files provided with the FrontPanel download.

Python Code:

```python
1. dev = ok.okCFrontPanel()
2. dev.OpenBySerial("")
3. dev.ConfigureFPGA("example.bit");
4. dev.SetWireInValue(0x03, 0x0A)
5. dev.UpdateWireIns()
6. # Send value 0x0A to Wire endpoint with address 0x03
7. dev.SetWireInValue(0x03, 0x0A)
8. dev.UpdateWireIns()
9. # Retrieve value on Wire endpoint with address 0x20
10. dev.GetWireOutValue(0x20)
A = dev.GetWireOutValue(0x20)
```

Verilog Code:

```verilog
1. // FrontPanel module instantiation
2. okWireIn inA(
3. .ok1(ok1),
4. .ep_addr(8'h03),
5. .ep_dataout (dataA)
6. );
7. okWireOut outA(
8. .ok1(ok1),
9. .ok2(ok2[17:0*17 +: 17]),
10. .ep_addr (8'h20),
11. .ep_datain (dataA)
12. );
```
Logic Registers Across a Clock Boundary (USB 2.0)

The `SetWireInValue` and `GetWireOutValue` methods in the `okCFrontPanel` class allow you to set and get up to 32-bit bus values across the entire defined Wire address space (0x00-0x1F for WireIn, 0x20-0x3F for WireOut). You can set the WireIn values by passing in the address of the WireIn and its new value. You can read WireOut values by passing the address of the WireOut and a bit mask to extract only the relevant bits. After setting a wire in value and before reading a wire out value, remember to update the WireIns and WireOuts, respectively. Otherwise, the values will not be accurate.

These techniques are also available on USB 3.0 devices, but those devices include the `okRegisterBridge` module along with methods and structs for reading registers. If you are using Wires on a USB 3.0 device, note that instead of 16-bit bus values, you can set and get up to 32-bit bus values.

Wires (both in and out) are synchronized to the host interface clock which may not be the same clock used for your internal logic. Therefore, when reading values via wires, it’s a good idea to consider the implications of this mismatch. Below is an example of a recommended synchronizer.

Note: This section contains both software and HDL portions. The software and HDL must work in tandem if FrontPanel is to be used on the PC end to perform tasks on the FPGA. The HDL in this section is designed to be set within the FrontPanel Framework HDL, available on the HDL Framework page for USB 2.0 and USB 3.0. It is assumed that the FPGA is configured with the bitfile generated by the HDL before the hardware is run. For specific information about each of these methods or modules, consult the FrontPanel User’s Manual, the FrontPanel API guide, and the samples and README files provided with the FrontPanel download.

Python:

```python
1. dev = ok.okCFrontPanel()
2. 3. # Send 0x0A to wire at endpoint 0x03
4. dev.OpenBySerial(""")
5. dev.SetWireInValue(0x03, 0x0A)
6. dev.UpdateWireIns();
7. 8. # Read value from WireOut endpoint at 0x20
9. dev.UpdateWireOuts()
10. A = dev.GetWireOutValue(0x20)
```

Verilog code:

```verilog
1. // Circuit behavior
2. always @ (posedge clk1) begin
3. counter <= counter + count_by;
4. end
5.
```
6. reg [31:0] counter_dff1, counter_dff2;
7. always @(posedge ti_clk) begin
8.   counter_dff1 <= counter; // Potentially metastable capture.
9.   counter_dff2 <= counter_dff1; // Much Less likely metastable.
10. end
11. // FrontPanel endpoint instantiation
12. okWireIn inA(
13.   .ok1(ok1),
14.   .ep_addr(8'h03),
15.   .ep_dataout(dataA)
16. );
17. okWireOut outA(
18.   .ok1(ok1),
19.   .ok2(ok2[0*17 +: 17]),
20.   .ep_addr(8'h20),
21.   .ep_datain(dataA)
22. );
Transferring Data

The following example uses a simple FIFO to hold a piped-in value until it is read by the pipe out. Note that the FIFO in this example is two bytes wide, which is the transfer size for a USB 2.0 device. In a USB 3.0 device, a pipe transfer consists of four bytes and should be handled with a four byte-wide FIFO.

In this example, the length of data is being ignored. It is assumed here that the HDL extracts data from the FIFO as available and has no expectation on transfer length.

Note: This section contains both software and HDL portions. The software and HDL must work in tandem if FrontPanel is to be used on the PC end to perform tasks on the FPGA. The HDL in this section is designed to be set within the FrontPanel Framework HDL, available on the HDL Framework page for USB 2.0 and USB 3.0. For specific information about the FrontPanel methods or modules, consult the FrontPanel User’s Manual, the FrontPanel API guide, and the samples and README files provided with the FrontPanel download.

Python Code:

```python
1. dev = ok.okCFrontPanel()
2. # data buffer in Python (mutable type bytearray) must be initialized upon declaration
3. dataout = bytearray('abcdefghijklmnopqrstuvwxyz')
4. datain = bytearray('00000000000000000000000000')
5. dev.OpenBySerial(""")
6. error = dev.ConfigureFPGA("example.bit")
7. # It's a good idea to check for errors here!!
8. dev.SetWireInValue(0x10, 0xff, 0x01);
9. dev.UpdateWireIns();
10. dev.SetWireInValue(0x10, 0x00, 0x01);
11. dev.UpdateWireIns();
12. # Send buffer to PipeIn endpoint with address 0x80
13. data = dev.WriteToPipeIn(0x80, dataout)
14. # Read to buffer from PipeOut endpoint with address 0xA0
15. data = dev.ReadFromPipeOut(0xA0, datain)
```

Verilog Code:

```verilog
1. // Circuit wires
2. wire fifowrite;
3. wire fiforead;
4. wire [15:0] datain;
5. wire [15:0] dataout;
6. wire reset;
7. wire [15:0] wireout;
8. //Circuit behavior
9. assign reset = wireout[0];
10. // Xilinx Core IP Generated FIFO
11. FIFO_16bit fifo(
12. .din(datain),
```
15. .dout(dataout),
16. .wr_en(fifowrite),
17. .rd_en(fiforead),
18. .clk(ti_clk),
19. .rst(reset)
20. );
21.
22. // FrontPanel module instantiations
23. okWireIn wire10(
24. .ok1(ok1),
25. .ep_addr(8'h10),
26. .ep_dataout(wireout)
27. );
28.
29. okPipeIn pipe80(
30. ok1(ok1),
31. .ok2(ok2[0*17 +: 17]),
32. .ep_addr(8'h80),
33. .ep_write(fifowrite),
34. .ep_dataout(datain)
35. );
36.
37. okPipeOut pipeA0(
38. ok1(ok1),
39. .ok2(ok2[1*17 +: 17]),
40. .ep_addr(8'hA0),
41. .ep_read(fiforead),
42. .ep_datain(dataout)
43. );