OpenCPI FPGA Top Level (FTop)

Shepard Siegel, Atomic Rules LLC (Shepard.Siegel@atomicrules.com)

Revision	Date	By	Notes
0.00	2010-01-21	ssiegel	Creation
0.01	2010-01-24	ssiegel	Additional info added as requested by Jim Kulp

This document describes the OpenCPI FPGA Top Level (FTop), its purpose, structure, and how it may be modified to cope with arbitrary pairings of board- and device-specific ASIC or FPGA top-level pinouts to underlying OpenCPI containers of infrastructure and application.

Regardless of what RTL is used; it is common practice to have an enclosing top-level wrapper named "fpgaTop" surrounding "FTop". The utility of such a wrapper includes:

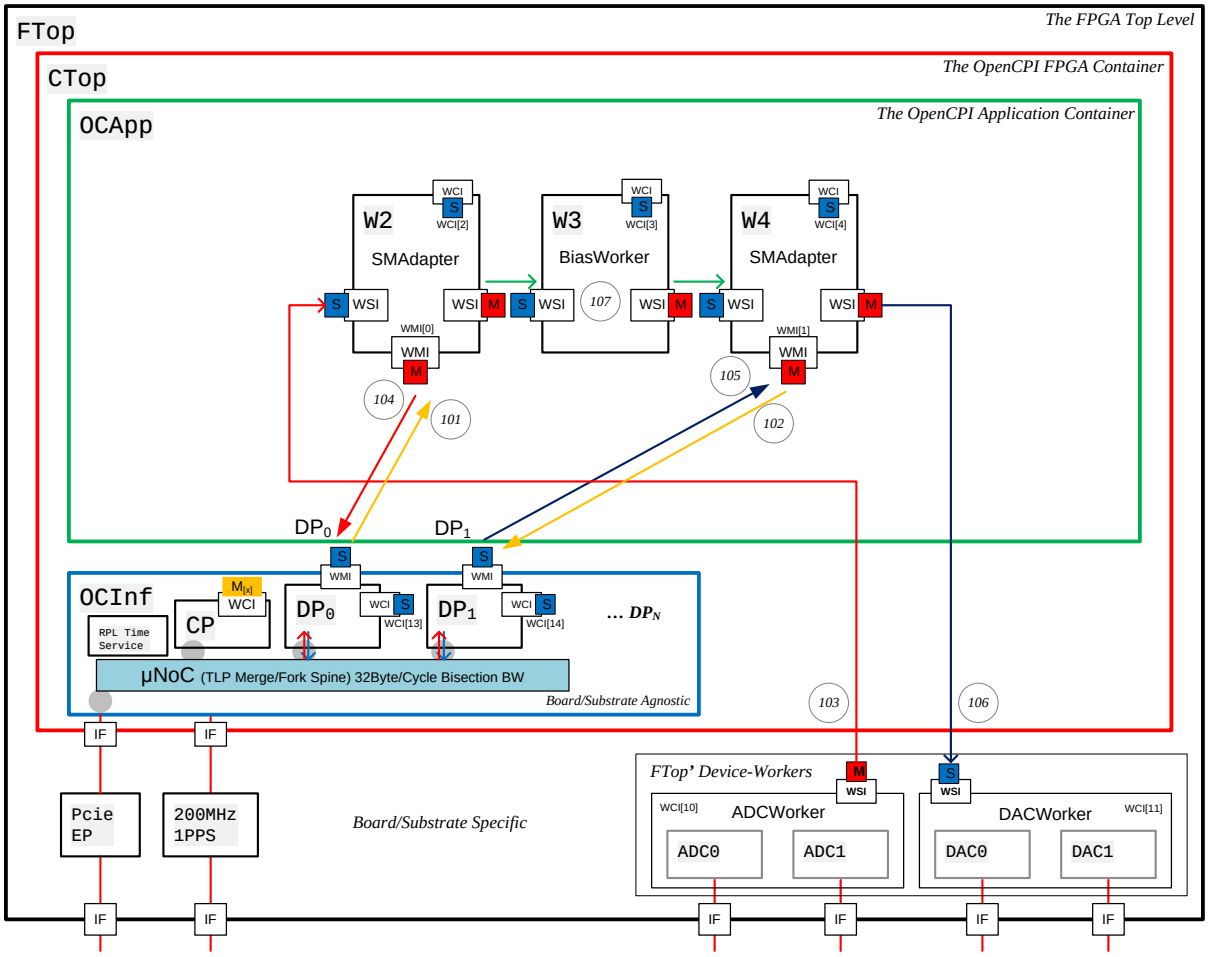
- Rooting all leaf IPs below the singleton "ftop" instance (of mkFTOP in example below)
- Providing an insertion point for ECOs and signal name adapation (if needed)

An example Verilog FTop wrapper is shown in the listing below:

```
 1 // fpgaTop_v5.v - ssiegel 2009-03-17
 2
 3 module fpgaTop(
 4   input  wire        sys0_clkp,      // sys0 Clock +
 5   input  wire        sys0_clkn,      // sys0 Clock -
 6   input  wire        pci0_clkp,      // PCIe Clock +
 7   input  wire        pci0_clkn,      // PCIe Clock -
 8   input  wire        pci0_reset_n,   // PCIe Reset
 9   output wire [7:0]  pci_exp_txp,    // PCIe lanes...
10   output wire [7:0]  pci_exp_txn,
11   input  wire [7:0]  pci_exp_rxp,
12   input  wire [7:0]  pci_exp_rxn,
13   output wire [2:0]  led,             // LEDs ml555
14   input  wire        ppsExtIn,        // PPS in
15   output wire        ppsOut           // PPS out
16 );
17
18 // Instance and connect mkFTop...
19  mkFTop ftop(
20   .sys0_clkp         (sys0_clkp),
21   .sys0_clkn         (sys0_clkn),
22   .pci0_clkp         (pci0_clkp),
23   .pci0_clkn         (pci0_clkn),
24   .pci0_reset_n      (pci0_reset_n),
25   .pcie_rxp_i        (pci_exp_rxp),
26   .pcie_rxn_i        (pci_exp_rxn),
27   .pcie_txp          (pci_exp_txp),
28   .pcie_txn          (pci_exp_txn),
29   .led               (led),
30   .ppsExtIn_x        (ppsExtIn),
31   .ppsOut            (ppsOut)
32 );
33
34 endmodule
```

Often, but not always, there is 1:1 correspondence between the signatures of "fpgaTop" and "ftop"

Figure 1 - An OpenCPI RPL Assembly (showing logical hierarchy and some interconnect)



As shown in an example logical hierarchy above, "FTop" contains all the modules and sub-modules of an OpenCPI RPL assembly. It is the root node for all the instances it contains. The purpose of FTop is to contain all of the circuit-board and FPGA or ASIC substrate ***specific*** IPs, so that the enclosed instance of CTop, the OpenCPI container has less heterogeneous adaptation to perform, and can therefore be more reusable and more easily verified.

Qualitatively, FTop instanced logic and IP can be considered one of two flavors (although they co-exist logically in the hierarchy together):

> **FTop** : Containing "core" functions such as a PCIe fabric endpoint and reference oscillator. (As shown on the lower left in Figure 1 above.)
>
> **FTop'** : Containing "device-workers" which adapt specialized IO functions to well-known interfaces. For example, an ADCWorker for a particular ADC, which provides a OCP::WIP::WSI interface. (As shown on the lower right of Figure 1 above.)

FTop (Core) Considerations

FTop core IPs interact almost exclusively with the CTop contained OCInf. Where OCInf then provides well-defined OCP::WIP interfaces to the application OCApp. Unless doing OpenCPI RPL Infrastructure development, creating new OCInf capabilities or implementations, or adapting an existing CTop to a new board or ASIC/FPGA substrate, most users may seldom need to change the FTop core components.

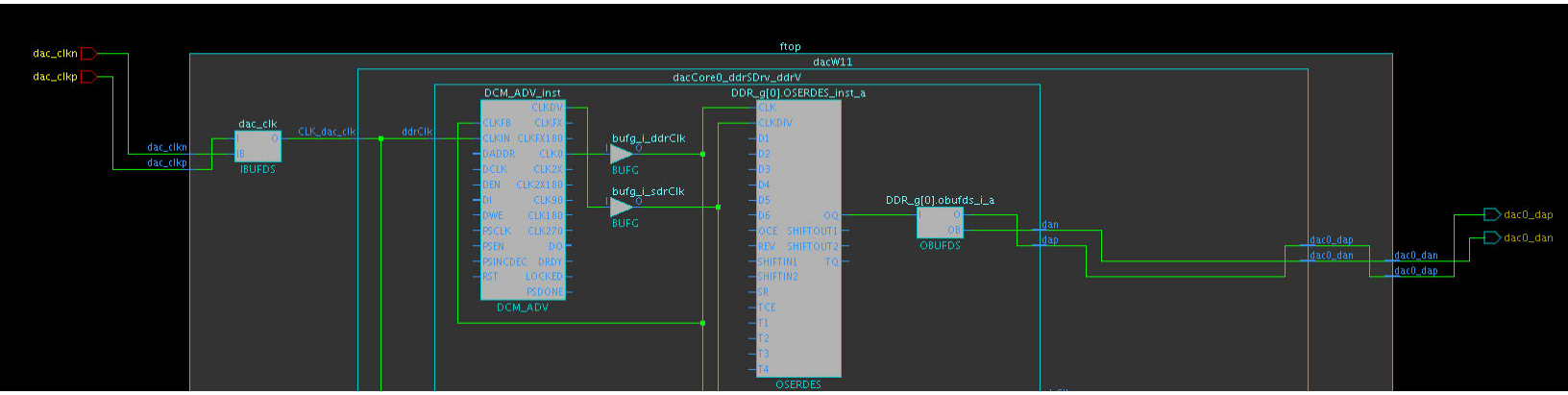
FTop core functions are allowed the leeway of interacting with CTop and the underlying OCInf by way of specific interfaces other than OCP::WIP. For example, the transaction-layer PCIe packets for all OpenCPI devices are normalized to a vendor-agnostic interface at the FTop/CTop/OCInf bounds that can remain the same over most implementations. FTop core instances the PCIe endpoint and performs the normalization to that vendor-agnostic interface.

One significant rationale for placing a board- and substrate- specific IP such as a PCIe endpoint in FTop core is the abstraction value it provides to CTop and below. By placing such a core in FTop, and providing to CTop, for example, the abstracted PCIe transaction-level packets, the physical and link layers do not need to be simulated when simulating CTop. Simulating at the PCIe transaction-level (TL) significantly improves the speed of functional simulation; while at the same time allowing CTop to “travel” (be made portable and reusable) among a diverse collection of board and device specific FTops.

FTop' (Prime) Considerations

FTop' provides a means for board- and device- specific IPs to be crafted that adapt any feasible interface to the velvet handcuffs of the OCP::WIP specifications. An IP that adapts a particular device in this way is called a "device worker". By placing the device worker at the FTop' level, it is easier for the implementer to use whatever implementation languages and methodologies are best suited for their task. On the device worker chip-facing side, they are free to specialize as much as they want to reduce area and increase performance. This includes instancing device-specific elements as needed (e.g. IOBs, BUFIOs, IODELAYs, SERDES, PLL, etc) within the device worker. On the device worker application-facing side, standard OCP::WIP interfaces are used to penetrate CTop and interact with OCApp. This separation allows a CTop and contained OCApp to be agnostic to a particular chip or board; while the FTop' device-workers provide the needed specialization.

Figure 2 - Part of Device Worker dacW11



This approach, especially for IO, helps recognize and factor out potentially unintended coupling between an IO producer/consumer and a specific device. Furthermore, it keeps the scope of all interfaces to OCApp within the OCP::WIP definition. This is the intended methodology, which provides ***explicit visibility*** of all IO to an OpenCPI RPL application with well-defined interfaces.

An alternative for application IP in OCApp to reach IO exists, but since it does not provide explicit visibility, it is not the preferred method: It is possible for workers in OCApp to structurally instance IO resources (IOBs, SERDES, etc) at a non-FTop' level (e.g. `/ftop/ctop/ocapp/myWorker/IOB_inst`). Then, using an absolute path to the IOB, "surface" the IO with the ASIC/FPGA vendor's design tools, if they support this capability. There are many reasons why this is discouraged, including but not limited to a module's signature not containing a full list of signals which comprise its IO ports, the resulting OCApp now having a ***hidden and implicit specialization*** towards a specific IO. These reasons are in part why FTop' exists as a first-class method for implementing device workers.

New Board or New ASIC/FPGA Substrate Check-List

The layered, component-based design of OpenCPI helps to reduce the effort of introducing a new board (possibly with new devices) or a new ASIC/FPGA (possibly with new features). This section contains a check-list of tasks that may require attention and should be considered in such a case. Depending on the situation, it may be harder (or easier) to "start-from-scratch" vs. making a change to an existing "nearly-correct" artifact.

1. Insure that the fpgaTop top level RTL wrapper accurately reflect the names of all the ASIC/FPGA device pins as they relate to this board.
    a. Consider rtl/fpgaTop_xx.v as a template
    b. Try to segregate signals destined for FTop core from FTop' with comments
    c. Ensure fpgaTop_xxx is aligned with the constraints file (e.g. UCF)
2. Insure that the top pad facing interface of FTop matches the fpgaTop wrapper's instance
3. For the FTop core elements, ensure the adaptation to CTop is sufficient for elements such as
    a. PCIe endpoint, Clock, and Reset
    b. Sys0 Clock
    c. PPS In/Out
    d. LEDs and Switches
4. For the FTop' elements, ensure the adaptation to CTop is sufficient for the device-workers
    a. The device-worker's view "up and out" to the IOs
    b. The device-worker's OCApp-facing view
5. As best as possible, segregate the FTop core from FTop' elements in the body of the FTop source code. (See example at end of this document)
6. Modify the CTop and OCApp instances to route device worker signals between FTop' and OCApp

FTop Overview

The following points summarize:

- FTop contains board and substrate specific IPs
    - By factoring board- and substrate- specific adaptation into FTop, the underlying CTop is largely agnostic to board and substrate changes
- FTop has direct access to all ASIC/FPGA device pins
- FTop instances CTop and provides all interfaces it requires

Example FTop implementation in BSV

```
39 (* synthesize, no_default_clock, clock_prefix="", reset_prefix="pci0_reset_n" *)
40 module mkFTop#(Clock sys0_clkp, Clock sys0_clkn,
41                Clock pci0_clkp, Clock pci0_clkn,
42                Clock dac_clkp,  Clock dac_clkn,
43                Clock adc_clkp,  Clock adc_clkn,
44                Clock adc0_clkp, Clock adc0_clkn,
45                Clock adc1_clkp, Clock adc1_clkn)(FTopIfc);
46   Clock            sys0_clk  <- mkClockIBUFDS(sys0_clkp, sys0_clkn);
47   Clock            pci0_clk  <- mkClockIBUFDS(pci0_clkp, pci0_clkn);
48   Reset            pci0_rst  <- mkResetIBUF;
49   PCIExpress#(8)   pci0      <- mkPCIExpressEndpoint(?,clocked_by pci0_clk, reset_by pci0_rst);
50   Clock            trn_clk   =  pci0.trn.clk;
51   Reset            trn_rst_n <- mkAsyncReset(1, pci0.trn.reset_n, trn_clk);
52   Bool             pciLinkUp =  pci0.trn.link_up;
53   MakeResetIfc     pciLinkUpResetGen <-mkReset(1,True,trn_clk, clocked_by trn_clk, reset_by trn_rst_n);
54   rule plr (!pciLinkUp); pciLinkUpResetGen.assertReset; endrule
55   Reset            pciLinkReset = pciLinkUpResetGen.new_rst;
56
57   PciId            pciDevice =  PciId { bus  : pci0.cfg.bus_number,
58                                         dev  : pci0.cfg.device_number,
59                                         func : pci0.cfg.function_number};
60
61   InterruptControl pcie_irq       <- mkInterruptController(trn_clk, trn_rst_n,
62                                       clocked_by trn_clk, reset_by trn_rst_n);
63
64   FIFO#(TLPData#(8))     fP2I  <- mkSizedFIFO(4,    clocked_by trn_clk, reset_by trn_rst_n);
65   FIFO#(TLPData#(8))     fI2P  <- mkSizedFIFO(4,    clocked_by trn_clk, reset_by trn_rst_n);
66   CTopIfc                ctop  <- mkCTop(pciDevice, clocked_by trn_clk, reset_by trn_rst_n);
67
68   mkConnection(pci0.trn_rx, toPut(fP2I));
69   mkConnection(toGet(fI2P), pci0.trn_tx);
70   mkConnection(toGet(fP2I), ctop.server.request,    clocked_by trn_clk, reset_by trn_rst_n);
71   mkConnection(ctop.server.response, toPut(fI2P),   clocked_by trn_clk, reset_by trn_rst_n);
72
73   mkConnection(pci0.cfg_irq, pcie_irq.pcie_irq);
74   mkTieOff(pci0.cfg);
75   mkTieOff(pci0.cfg_err);
76
77   ReadOnly#(Bit#(2)) infLed    <- mkNullCrossingWire(noClock, ctop.led);
78   ReadOnly#(Bool)    ppsOutdrv <- mkNullCrossingWire(noClock, ctop.cpNow.ppsDrive);
79
80   SyncBitIfc#(Bit#(1)) ppsExtInSync <- mkSyncBit(trn_clk,trn_rst_n,trn_clk);
81   Reg#(Bool)           ppsDFF       <- mkReg(False, clocked_by trn_clk, reset_by trn_rst_n);
82   rule ppsDFFCapture; ppsDFF <= unpack(ppsExtInSync.read); endrule
83   Bool ppsEdge    = (unpack(ppsExtInSync.read) != ppsDFF);
84   Bool ppsRising  = (ppsEdge && !ppsDFF);
85   Bool ppsFalling = (ppsEdge &&  ppsDFF);
86   rule ppsRises (ppsRising); ctop.ppsExtIn; endrule
87
88   // ADC Clocks...
89   Clock            adc_clk   <- mkClockIBUFDS(adc_clkp,  adc_clkn);
90   Clock            adc0_clk  <- mkClockIBUFDS(adc0_clkp, adc0_clkn);
91   Clock            adc1_clk  <- mkClockIBUFDS(adc1_clkp, adc1_clkn);
92   Reset            adc_rst   <- mkAsyncReset(3, pciLinkReset, adc_clk);
93   // DAC Clocks...
94   Clock            dac_clk   <- mkClockIBUFDS(dac_clkp, dac_clkn);
95   Reset            dac_rst   <- mkAsyncReset(3, pciLinkReset, dac_clk);
96
97   Vector#(Nwci_ftop,Wci_m#(20)) vWci = ctop.wci_m;
98   ADCWorkerIfc     adcW10     <-  mkADCWorker(sys0_clk, adc_clk,
99   DACWorkerIfc     dacW11     <-  mkDACWorker(sys0_clk, dac_clk,
100  GCDWorkerIfc     gcdW12     <-  mkGCDWorker(12,
101  mkConnection(vWci[0], adcW10.wci_s);
102  mkConnection(vWci[1], dacW11.wci_s);
103  mkConnection(vWci[2], gcdW12.wci_s);
104  mkConnection(adcW10.wsi_m, ctop.wsi_s_adc);
105  rule connect_now;
106    adcW10.cpNow(ctop.cpNow);
107    dacW11.cpNow(ctop.cpNow);
108  endrule
109
110  interface pcie     = pci0.pcie;
111  method    led      = ~{infLed, pack(ppsOutdrv)}; //leds are on when active-low
112  method Action ppsExtIn (Bit#(1) x) = ppsExtInSync.send(x);
113  method    ppsOut   = ppsOutdrv;
114  interface trnClk   = trn_clk;
115  interface adx      = adcW10.adx;
116  interface adc0     = adcW10.adc0;
117  interface adc1     = adcW10.adc1;
118  interface dac0     = dacW11.dac0;
119 endmodule: mkFTop
```

FTop Core

FTop'