# OpenCPI FPGA Reference Platform, PCIe Memory Address Map

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

| Revision | Date | By | Notes |
|---|---|---|---|
| 0.01 | 2009-03-27 | ssiegel | Draft |
| 0.02 | 2009-04-01 | ssiegel | Worker Status Register Added |
| 0.03 | 2009-04-03 | ssiegel | Worker Attention Register Added |
| 0.04 | 2009-04-22 | ssiegel | WCI "Last Access" Added, Details Added, GCD Worker Added |
| 0.05 | 2009-04-24 | ssiegel | Minor Refinements |
| 0.06 | 2009-06-03 | ssiegel | Remove Mixed Registers |
| 0.07 | 2009-06-24 | ssiegel | Add Data-Plane Registers |
| 0.08 | 2009-06-30 | ssiegel | Data Plane Evolution, Parametric Runtime Buffers, Size, Bases |
| 0.09 | 2009-07-02 | ssiegel | Data Plane Functional Change |
| 0.10 | 2009-07-06 | ssiegel | Data Plane DMA Registers |
| 0.11 | 2009-07-10 | ssiegel | Inserted BiasWorker into example application container |
| 0.12 | 2009-08-14 | ssiegel | Rework Control-Plane memory map to 16 MB BAR0 |
| 0.13 | 2009-08-21 | ssiegel | Refine Data Plane DMA Registers |
| 0.14 | 2009-09-08 | ssiegel | Allow Data Plane programmable Producer/Consumer Roles |
| 0.15 | 2009-10-13 | ssiegel | Change Data Plane Control Options |
| 0.16 | 2009-10-30 | ssiegel | OCCP changes to enhance function of PPS.NS clock |
| 0.17 | 2009-11-24 | ssiegel | Begin to add RPL Time Services |
| 0.18 | 2009-12-10 | ssiegel | Add Data-Plane buffer size at DPCP+0x4C |
| 0.19 | 2009-12-11 | ssiegel | Add fabFlowSize, fabFlowBase, flowDiagCount |
| 0.20 | 2010-01-06 | ssiegel | Add Data-Plane info structures, FTop' workers |
| 0.21 | 2010-01-25 | ssiegel | Introduce SMAdapter |
| 0.22 | 2010-02-04 | ssiegel | More Time Service Details, Diagram |
| 0.23 | 2010-02-18 | ssiegel | Added Time Set and Delta Measure |
| 0.24 | 2010-03-03 | ssiegel | Refine Time, ADC Capture |
| 0.25 | 2010-06-01 | ssiegel | WIP Interface Debug Registers Added |

This document describes the PCIe memory address map of the OpenCPI Reference Platform.

# PCIe Block Plus Endpoint (BPEP) Configuration v1.13 (ISE11.4)

The table below lists the changes from default for the BPEP not related to the BARs.

**Table 1 – Virtex5 PCIe Gen1 Parameters**

| Parameter | Value | Value |
|---|---|---|
| lane_width [1] | x8, x4, x1 | x8, x4, x1 |
| interface_freq | 125 MHz | 250 MHz |
| device_id | 16'h_4243 | 16'h_4243 |
| revision_id | 02 | 03 |

**Table 2 – Virtex6 PCIe Gen2 Parameters**

| Parameter | Value |
|---|---|
| lane_width | x4 |
| interface_freq | 250 MHz |
| device_id | 16'h_4243 |
| revision_id | 02 |

**Table 3 –Spartan6 PCIe Gen2 Parameters**

| Parameter | Value |
|---|---|
| lane_width | X1 |
| interface_freq | 250 MHz |
| device_id | 16'h_4243 |
| revision_id | 02 |

[1] lane width set per board target.

BAR0 and BAR1 are set to the following parameters; the other BARs are disabled.

| BAR0 | BAR1 |
|---|---|
| bar0_64bit=false | bar1_64bit=false |
| bar0_enabled=true | bar1_enabled=true |
| bar0_prefetchable=false | bar1_prefetchable=false |
| bar0_scale=Megabytes | bar1_scale=Kilobytes |
| bar0_size=16 | bar1_size=64 |
| bar0_type=Memory | bar1_type=Memory |
| bar0_value=FF00_0000 | bar1_value=FFFF_0000 |

# BAR0 – 16 MB Memory for Administration and Workers

BAR0 is a 16MB space that is used for accessing a 1 MB Administrative area, as well as fifteen identical 1MB Worker Configuration Property areas. The table below shows how the top four Byte-address bits[23:20] of the 24b Byte address are used to select one of the sixteen 1MB regions.

The 1MB Administrative ("Admin") region contains a 64KB area that can affect the entire FPGA; along with fifteen identical 64KB worker control regions, which provide Control (and Status) of individual workers. Each of the 15 Worker Configuration Property regions provides a full, contiguous 1MB memory map to each worker's configuration property space.

| BAR0 Offset | Size | Region | In oc1001 | In ocXXXX |
|---|---|---|---|---|
| +0x00_0000 | 64KB | Administrative ("Admin") | admin | admin |
| +0x01_0000 | 64KB | Control, Worker 0 | Gcd | |
| +0x02_0000 | 64KB | Control, Worker 1 | Gcd | |
| +0x03_0000 | 64KB | Control, Worker 2 | FCAdapter | |
| +0x04_0000 | 64KB | Control, Worker 3 | Bias | |
| +0x05_0000 | 64KB | Control, Worker 4 | FPAdapter | |
| +0x06_0000 | 64KB | Control, Worker 5 | | |
| +0x07_0000 | 64KB | Control, Worker 6 | | |
| +0x08_0000 | 64KB | Control, Worker 7 | | |
| +0x09_0000 | 64KB | Control, Worker 8 | | |
| +0x0A_0000 | 64KB | Control, Worker 9 | | |
| +0x0B_0000 | 64KB | Control, Worker 10 | ADC | |
| +0x0C_0000 | 64KB | Control, Worker 11 | DAC | |
| +0x0D_0000 | 64KB | Control, Worker 12 | FtpUtil | |
| +0x0E_0000 | 64KB | Control, Worker 13 | DP0 | |
| +0x0F_0000 | 64KB | Control, Worker 14 | DP1 | |
| +0x10_0000 | 1MB | Configuration, Worker 0 | Gcd | |
| +0x20_0000 | 1MB | Configuration, Worker 1 | Gcd | |
| +0x30_0000 | 1MB | Configuration, Worker 2 | FCAdapter | |
| +0x40_0000 | 1MB | Configuration, Worker 3 | Bias | |
| +0x50_0000 | 1MB | Configuration, Worker 4 | FPAdapter | |
| +0x60_0000 | 1MB | Configuration, Worker 5 | | |
| +0x70_0000 | 1MB | Configuration, Worker 6 | | |
| +0x80_0000 | 1MB | Configuration, Worker 7 | | |
| +0x90_0000 | 1MB | Configuration, Worker 8 | | |
| +0xA0_0000 | 1MB | Configuration, Worker 9 | | |
| +0xB0_0000 | 1MB | Configuration, Worker 10 | ADC | |
| +0xC0_0000 | 1MB | Configuration, Worker 11 | DAC | |
| +0xD0_0000 | 1MB | Configuration, Worker 12 | FtpUtil | |
| +0xE0_0000 | 1MB | Configuration, Worker 13 | DP0 | |
| +0xF0_0000 | 1MB | Configuration, Worker 14 | DP1 | |

In addition to 32b DWORD access, the hardware respects multi-DWORD transactions (e.g. 64b 2-DW, or DMA *N*-DW) to this BAR.

Not all Workers need be populated in the application container.

# BAR0 – Administrative "Admin" Region Registers

This section describes the administrative region registers, the 64KB space at the base of BAR0.

| Admin Region Offset | Name | Access | Description |
|---|---|---|---|
| +0x0000 | ByteSwap(0x4F70656E) | RO | 0x4F70656E = ASCII "Open" |
| +0x0004 | ByteSwap(0x43504900) | RO | 0x43504900 = ASCII "CPI" [1] |
| +0x0008 | cpRevision | RO | Control Plane Revision |
| +0x000C | cpBirthday | RO | Control Plane Birthday |
| +0x0010 | cpConfig | RO | Control Plane Configuration |
| +0x0014 | pciDevice | RO | PCIe Device ID |
| +0x0018 | wrkAttn | RO | Worker Attention Register |
| +0x001C | cpStatus | RO | Control Plane Status |
| +0x0020 | scratch20 | Read/Write | GP Scratch Register 0x20 |
| +0x0024 | scratch24 | Read/Write | GP Scratch Register 0x24 |
| +0x0028 | cpControl | RW | Control Plane Control |
| +0x002C | | RO | Reserved |
| +0x0030 | rplTimeStatus | RO | Time Status Register |
| +0x0034 | rplTimeControl | RW | Time Control Register |
| +0x0038 | rplTimeMS | [2] RW (stage) | RPL Time Integer Seconds |
| +0x003C | rplTimeLS | [2] RW (commit) | RPL Time Fractional Seconds |
| +0x0040 | rplTimeCompareMS | [2] RW (stage) | Time Compare Argument MS |
| +0x0044 | rplTimeCompareLS | [2] RW (commit) | Time Compare Argument LS |
| +0x0048 | rplTimeRefPerPps | RO | Number of Ref Clocks Per PPS |
| +0x4C~78 | | | Reserved |
| +0x007C | numDPMemRegions | Read-Only | Number of DP Mem Regions |
| +0x0080-+0x00BC | dpMemRegion 0-15 | Read-Only | DP MemRegion 0-15 Info |

1. Note that the ASCII characters "CPI" are followed by a NULL, not a Space.
2. Locations 0x38, 0x40, and 0x48 are 8B stores where the LS DWORD must first be staged, and the MS DWORD performs the commit. On a little-endian (e.g. x86) platform, this will happen automatically for a 64b store over PCIe, where the first DWORD of the 8B write goes to the lower address; and the second DWORD of the 8B write goes to the higher address.

## cpRevision (Admin Base +0x0008)

This register indicates control plane revision number.


## cpBirthday (Admin Base +0x000C)

This register indicates when the bitstream was built. (32b POSIX time)


## cpConfig (Admin Base +0x0010)

This register indicates in bits [14:0] which of 15 possible WCI interfaces have been connected in the application. A logic '1' in a particular bitfield means that a worker is connected to that interface; a logic '0' means that there is no WCI interface present, and thus no worker. The Hamming Weight, the number of logic '1's in this register, is the number of workers connected. Bit 0 corresponds to Worker 0.


## pciDevice (Admin Base +0x0014)

This register indicates PCI device ID in bits [15:0]


## wrkAttn (Admin Base +0x0018)

This register indicates in bits [14:0] which, if any, worker has a non-zero sticky bit status. There is a one-to-one correspondence between worker number and bitfield. If a particular bit is set, further details may be found by reading that worker's Worker Status Register. This register provides the utility of a single read which can indicate if any worker has a sticky bit set. Bit 0 corresponds to Worker 0.


## cpStatus (Admin Base +0x001C)

Control plane admin register. Bits [3:0] hold the number of "rogue" TLP packets received since reset.


## scratch20/24 (Admin Base +0x0020/24)

These registers may be used for DWORD R/W tests with no side effect.

## cpControl(Admin Base +0x0028)

The bitfields in this register control the functions described in the table below:

| Bits | Name | Access | Description |
|---|---|---|---|
| 31:24 | | Reserved | 0 |
| 23:16 | | Reserved | 0 |
| 15:8 | | Reserved | 0 |
| 7:0 | | Reserved | 0 |

### rplTimeStatus (Admin Base +0x0030)

The bitfields in this register provide the status described in the table below:

| Bits | Name | Access | Description |
|---|---|---|---|
| 31 | ppsLostSticky | RO | 1 = An active PPS fell outside the $\pm 0.1\%$ (1000 PPM) window |
| 30 | gpsInSticky | RO | 1 = At least one valid GPS time has updated rplTime |
| 29 | ppsInSticky | RO | 1 = At least one valid PPS input has aligned rplTime |
| 28 | timeSetSticky | RO | 1 = At least one valid CP set has been issued to rplTime |
| 27 | ppsOK | RO | 1 = PPS fell within the valid window in the last second |
| 26 | ppsLost | RO | 1 = PPS went missing from the valid window in the last second |
| 25:8 | | RO | 0 |
| 7:0 | rollingPPSIn | RO | 8b rolling count of (external) PPS Events |

### rplTimeControl (Admin Base +0x0034)

The bitfields in this register control the functions described in the table below.

| Bits | Name | Access | Description |
|---|---|---|---|
| 31 | clearStickyBits | WO | Writing a 1 to this location clears time sticky bits |
| 30:5 | | Reserved | |
| 4 | disableServo | RW | 0 = Use CP, PPS, or GPS to discipline local XO timebase<br>1 = Disable servo, local XO runs w/o compensation |
| 3 | disableGPS | RW | 0 = Acquire GPS Time from (external) GPS<br>1 = Disable (external) GPS dialog attempts |
| 2 | disablePPSIn | RW | 0 = Observe and Synchronize to (external) PPS Input<br>1 = Disable observation of (external) PPS Input |
| 1:0 | drivePPSOut | RW | 2'h0 = Time Server Integer Seconds (1 Hz)<br>2'h1 = PPS Input Loop-through (PPS Hz)<br>2'h2 = Local  XO Reference / 2  (100 MHz)<br>2'h3 = Mute (output disabled) |

### rplTime MS, LS (Admin Base +0x0038, 3C)

This 8B location provides a method for the control plane (CP) fabric to set and observe the FPGA master chip clock. Time is represented in unsigned fixed-point 32.32 format where the radix point is immediately to the right of integer seconds. Writes set the time; reads provide the time.

### rplTimeCompare MS, LS (Admin Base +0x0040, 44)

This 8B location is a diagnostic instrument that measures the difference between the time written and the value of the time server's clock at this instant this register is written. There is no side-effect to the use of this instrument. The 8B signed 32.32 delta value is read from this location at any time after the measurement was made.

### rplTimeRefPerPPS (Admin Base +0x0048)

This 4B RO location is a diagnostic instrument, literally a frequency counter, that measures the number of uncompensated local XO cycles that occur between successive (external) PPS events. It is updated after each PPS event. If the PPS signal is considered "exact"; then this count may be used to observe the second-to-second behavior of the local 200 MHz XO.  Given a +/- 50 PPM XO, this count may be 200e6 +/- 10e3

# Dataplane Memory Region Provisioning

The value conveyed in numDPMemRegions in conjunction with the subsequent data structure in dpMemRegion[] advertises the provisioned capabilities of this bitstream's dataplane memory regions.

## numDPMemRegions (Admin Base +0x007C)

The bitfields in this register control the functions described in the table below:

| Bits | Name | Access | Description |
|------|------|--------|-------------|
| 31:4 | | Reserved | 0 |
| 3:0 | numDPMemRegions | RO | The number of dataplane memory regions |

## dpMemRegion[x] Structure (Admin Base +0x0080, 84, …)

Each 32b DW starting at AdminBase+0x80 describes a dataplane memory region. Only numDPMemRegions are populated. For example, if numDPMemRegions==2, then only the region structure members at 0x80 (for member 0) and 0x84 (for member 1) will be valid.

| Bits | Name | Description |
|------|------|-------------|
| 31:28 | PCIe BAR | The PCIe BAR is this memory region belongs to |
| 27:14 | Offset, 4KB pages | Offset into this is PCIe BAR, in 4KB pages |
| 13:0 | Size, 4KB pages | Size of this memory region, in 4KB pages |

The PCIe BAR says in which BAR (never 0), this memory region is decoded.

The Offset, specifies the offset from the base of the BAR, in 4KB pages.

The Size, specifies the size of the memory region, in 4KB pages.

# BAR0 – Worker Control Operation Space

Every worker has a fixed set of **Control Operations**. The worker control aspect is a common model which allows all workers to be managed without having to customize the management software for each individual worker. (Do not confuse with **Configuration Properties**, described next, which allow components to be specialized at runtime.)

This section describes the worker control region registers, the 64KB space allocated to each worker, where the base address is 0x0$W_A$_0000.

Where the hex nibble $W_A=(W_N + 1)$ is one of the 15 worker ordinal numbers, $W_N$ = {0,1,…14}.

For example, the first worker, worker W0, has its control region based at BAR0 offset 0x01_0000. The second worker, worker W1 , at offset 0x02_0000, and so on.

| Worker Control Offset | Access | Description |
|---|---|---|
| +0x00 | Read-Only | Control-Op: Initialize |
| +0x04 | Read-Only | Control-Op: Start |
| +0x08 | Read-Only | Control-Op: Stop |
| +0x0C | Read-Only | Control-Op: Release |
| +0x10 | Read-Only | Control-Op: Test |
| +0x14 | Read-Only | Control-Op: BeforeQuery |
| +0x18 | Read-Only | Control-Op: AfterConfig |
| +0x1C | Read-Only | Control-Op: Rsvd7 |
| +0x20 | Read-Only | Worker Status Register |
| +0x24 | Read/Write | Worker Control Register |
| +0x28 | Read-Only | lastConfigAddress |
| +0x2C | Write-Only | Worker Sticky Clear |
| +0x30 | Reserved | - |
| +0x34 | Reserved | - |
| +0x38 | Reserved | - |
| +0x3C | Reserved | - |

There are four possible responses to Control-Op Reads, they are

| Value | Description |
|---|---|
| 0xC0DE_4201 | OK |
| 0xC0DE_4202 | ERROR |
| 0xC0DE_4203 | TIMEOUT |
| 0xC0DE_4204 | WORKER IS RESET |

# Worker Status Register (Worker Control Offset +0x20)

Each worker has its own worker status register located 32B before the end of each worker's control space. The worker status register is located at an offset of 0xFFE0. If *N* is a hex nibble specifying the **worker ordinal plus one**, then that worker's status register is located at BAR0 address of 0x0*N*_FFE0.

The worker status register contains both status bits with diagnostic information of the last access and sticky bits which capture events and require a specific action top clear. The worker status register collection of sticky bits are initially set to zero at power up reset and set and held if a specific exception event occurs. The sticky bit remains set until the register is cleared.

- Clearing the sfCap bit: Write a '1' to bit location 9 (0x0000_0200) at the worker sticky clear address.
- Clearing the other sticky bits: Write a '1' to bit location 8 (0x0000_0100) at worker sticky clear address.

The sFlag Captured bit signals that the worker asserted sFlag for one or more control clock cycles. The nine other sticky bits serve to capture the cause(s) of error, should the worker not respond normally to a sequence of one or more requests. The sticky bits capture what kind of exception (timeout, fail, err) as well as what was the pending request when the exception occurred (cfgWt, cfgRd, ctlOp). Since these nine bits are sticky, they accumulate all exceptions that may have occurred since the status was reset.

| Bit | Name | Access | Description |
|---|---|---|---|
| 31-28 | | Reserved | 0 |
| 27 | lastOpWrite | RO | 1=last op was write |
| 26-24 | lastControlOp | RO | Last control Op |
| 23-20 | lastConfigBE | RO | Last BE used |
| 19 | | RO | lastOpWrite is Valid |
| 18 | | RO | lastControlOp is Valid |
| 17 | | RO | lastConfigBE is Valid |
| 16 | | RO | lastConfigAddr is Valid |
| 15-10 | | Reserved | 0 |
| 9 | sfCap | RO (Sticky) | sFlag Captured |
| 8 | reqTO.cfgWt | RO (Sticky) | Time-Out *of a* Config-Write |
| 7 | reqTO.cfgRd | RO (Sticky) | Time-Out *of a* Config-Read |
| 6 | reqTO.ctlOp | RO (Sticky) | Time-Out *of a* Control-OP |
| 5 | reqFAIL.cfgWt | RO (Sticky) | Resp-Fail *of a* Config-Write |
| 4 | reqFAIL.cfgRd | RO (Sticky) | Resp-Fail *of a* Config-Read |
| 3 | reqFAIL.ctlOp | RO (Sticky) | Resp-Fail *of a* Control-OP |
| 2 | reqERR.cfgWt | RO (Sticky) | Resp-Err *of a* Config-Write |
| 1 | reqERR.cfgRd | RO (Sticky) | Resp-Err *of a* Config-Read |
| 0 | reqERR.ctlOp | RO (Sticky) | Resp-Err *of a* Control-OP |

# Worker Control Register (Worker Control Offset +0x24)

Each worker has its own worker control register located 28B before the end of each worker's control space. The worker status register is located at an offset of 0xFFE4. If *N* is a hex nibble specifying the *worker ordinal plus one*, then that worker's status register is located at BAR0 address of 0x0*N*_FFE4.

The worker reset_n bit [bit 31] drives the reset_n signal to each worker. When clear, the worker is reset. At power on reset, this bit is clear for all workers. This bit must be set to disable reset before interacting with the worker.

The worker timeout bitfield [4:0] sets the number of 8 nS cycles to wait after a command before a timeout condition is declared. This field is the log2 of that threshold. A setting of 4 (the power on default) results in $2^4$ = 16 cycle timeout (128 nS). The maximum setting of 31 corresponds to about 17 seconds. ($2^{31}$ x 8 nS/cy). The default timeout setting of 16 cycles (128 nS) may be inadequate for workers requiring more time than that to service their requests. Workers will return the "0xC0DE_4203" TIMEOUT indication if access timer expires before the worker completes the control operation, or acknowledges the configuration property access. In such cases, it is recommended that default timeout for that worker be increased, only as much as needed. Higher timeout values add to the upper bound on the maximum latency that can be expected for a response from the system.

The power-on reset value of this register is 0x0000_0004. This means the worker is reset.

To take a worker out of reset, retaining the default 16 cycle timeout, write 0x8000_0004 to this register.

| Bit | Name | Access | Description |
|-----|------|--------|-------------|
| 31 | wrkReset_n | RW | Worker Reset_n |
| 30-5 | | RW | Spare |
| 4-0 | wrkTimeOut | RW | $Log_2$ control cycles for Timeout |

# BAR0 – Worker Configuration Properties Space

Worker **Configuration Properties** are named storage locations of a worker that may be read or written. Their values indicate or control aspects of the worker's operation. Reading and writing configuration property values may or may not have side effects on the operation of the worker. Configuration properties with side effects can be used for custom worker control. Each worker may have its own, possibly unique, set of configuration properties. They may include hardware resource such registers, memory, and state.

This section describes the worker configuration properties, the contiguous 1 MB space allocated to each worker, where the base address is 0x$W_A$0_0000. Where the hex nibble $W_A=(W_N + 1)$ is one of the 15 worker ordinal numbers, $W_N$ = {0,1,…14}.

For example, the first worker, worker W0, has its configuration properties based at BAR0 offset 0x10_0000. The second worker, worker W1 , at offset 0x20_0000, and so on.

Each worker has a contiguous 1MB address space which maps directly to $2^{20}$ Bytes of configuration properties.

# About the RPL Time Service (RTS)

RTS maintains a local time clock on an RPL device. It keeps time though a phase-accumulator whose increment is closely related to the frequency of a local crystal oscillator (XO). This XO is typically a timebase from a local, stable "brick"; distinct from other clocks that may have be recovered from a transport (such as a PCIe or Ethernet PHY), whose absolute stability is often less constrained.

The time on the RTS may be set (or observed) by writing (or reading) a configuration property. Select implementations of the RTS employ logic to assist in the IEEE1588 Precision Time Protocol (PTP).

 The accuracy of the stable, uncompensated XO (typically +/- 50 PPM)[1] can be dramatically improved by connecting a GPS device with a 1 PPS output. Select implementations of RTS logic may automatically detect the presence or absence of the 1 PPS signal and, when available, digitally compensate for the local timebase frequency/phase error. The 1 PPS signal allows the stable, but not necessarily precise, local XO to deliver the long-term precision a GPS reference can provide.

# How the RTS is used with Workers that desire Time
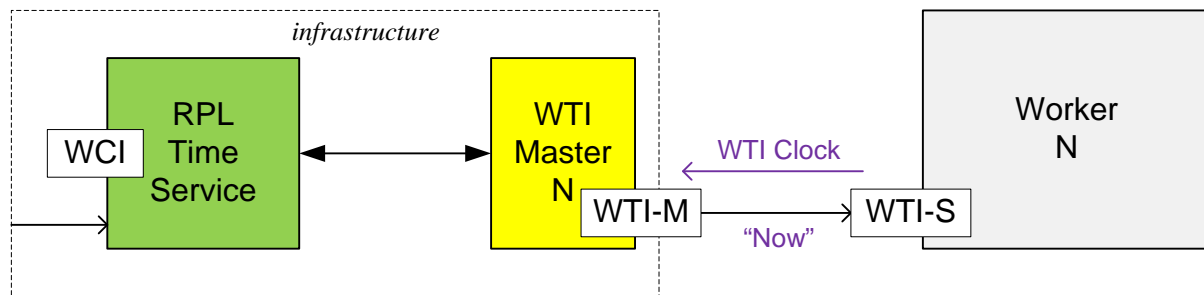


Any worker that desires time can use the Worker Time Interface (WTI) defined in the WIP specification. The worker implementer chooses in which clock domain they want to observe time, and how many bits to the left and right of the "1 second" radix point they wish to observe. The infrastructure then provides each worker with pure-binary, continuous, monotonic, GPS time, in the clock domain and format specified.

The RTS clock supports multiple workers each asking for time in a different clock domain by instancing as many WTI Masters as are required. Baseline implementations of WTI masters are area-efficient synchronizers that bring the RPL time into the WTI clock domain. Higher-accuracy implementations shadow the RPL time with a phase-accumulator in the WTI clock domain.

---

[1] Local XO frequency tolerance is board dependent. Contemporary platforms (e.g. Xilinx ML605) use a 200 MHz SAW XO with +/- 50PPM (Epson EG-2121CA)
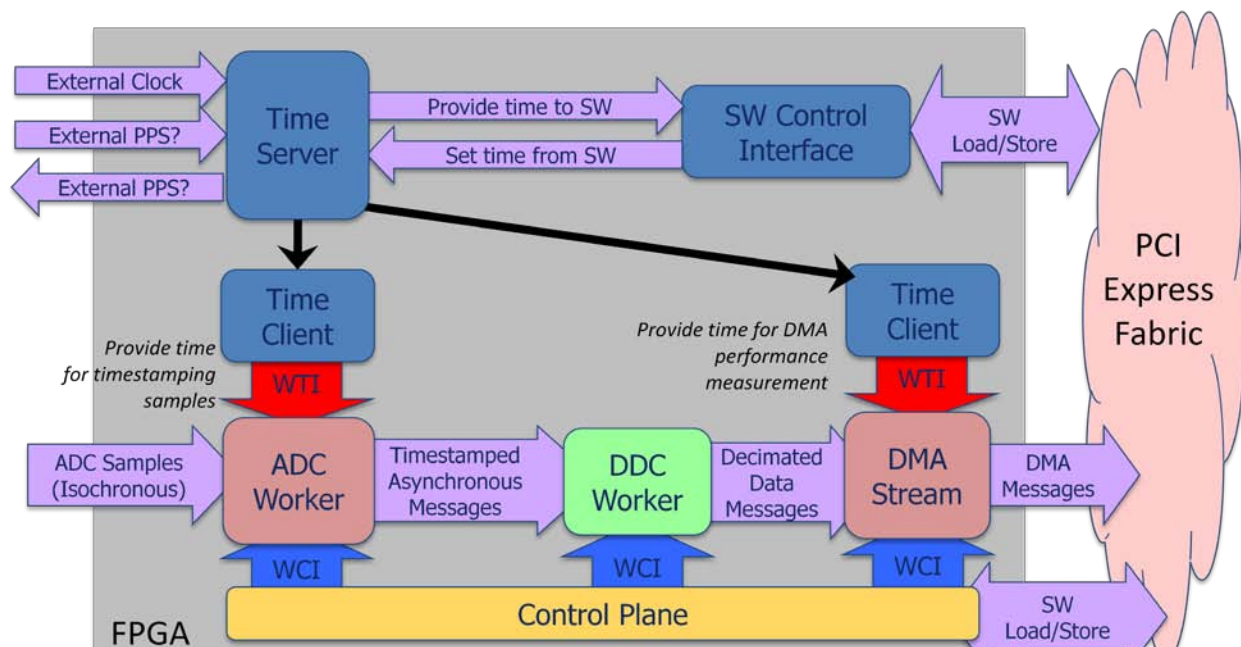
# RPL Time Server Properties

This section describes the properties of the RPL Time Service (RTS) from the

| Admin Offset | Name | Access | Description |
|---|---|---|---|
| +0x0030 | rplTimeStatus | RO | Time Status Register |
| +0x0034 | rplTimeControl | RW | Time Control Register |
| +0x0038 | rplTimeMS | [2] RW (stage) | RPL Time Integer Seconds |
| +0x003C | rplTimeLS | [2] RW (commit) | RPL Time Fractional Seconds |
| +0x0040 | rplTimeCompareMS | [2] RW (stage) | Time Compare Argument MS |
| +0x0044 | rplTimeCompareLS | [2] RW (commit) | Time Compare Argument LS |
| +0x0048 | rplTimeRefPerPPS | RO | Reference Clocks per PPS |

## Using the RPL Time Server

Requires re-write for host-based time control.

Take the worker out of reset. Set the configuration properties. Send the INIITALIZE and START control ops. (Optionally) Update the clock by writing the time in seconds.

At any time other than the interval between INITAILIZE and START, the RTS clock may be set by writing setSeconds. At the instant of writing setSeconds, the RTS time is transferred to a "WhenSet" holding register.

Clock time may be observed as a 64b (32.32) value at offset 0x10. This format of time is "binary coherent" across all 64 bits.

Clock time may also be read 32b Seconds and 32b Nanoseconds (IEEE-1588 ) at location 0x14. This form of time "count nanoseconds" in the 32 lsbs, which never exceed $10^9$.

# GCD Worker Configuration Properties

This section describes the configuration properties of the GCD worker.

| Worker Base Offset | Access | Description |
|---|---|---|
| +0x0000 | RW | GCD Argument "r0" |
| +0x0004 | RW | GCD Argument "r4" |
| +0x0008 | RO | GCD Result |
| +0x000C | RO | GCD Result Ready (bit0) |
| +0x0010 | RO | Worker Ordinal Number |
| +0x0014 | RO | Counter |
| +0x0018 | RW | B18 |
| +0x0018 | RW | B19 |
| +0x0018 | RW | B1A |
| +0x0018 | RW | B1B |
| +0x001C | RW | <No Response Generated> |
| +0x0020 | RW | Bit[31:1] - Reserved<br>Bit[0] - sFlagState |

# Using the GCD Worker

Take the worker out of reset. Send the INIITALIZE and START control-ops.

Write GCD arguments to r0 and r4 registers. A write to either property r0 or r4 will initiate GCD calculation when in the Operating state. The GCD Result Ready register will indicate with b0 True when the result is ready. When result is ready, read the GCD Result register. The GCD Result register will return the code 0xBADBADBA if there is not a valid GCD result ready.

# Other Features

The counter at location 0x14 will count when the control state is "Operating" and will reset when the control state is "Initialized".

Four discrete byte-sized registers with the power-on reset value equal to their address are packed at the DWORD 0x0018. Individual byte or word writes may be tested by writing these locations.

The location 0x001C will (on purpose) never generate a response and may be used to test the timeout handling.

The location 0x0020 allows SFlag to be set or cleared at Bit-0

# Stream-Message Adapter (SMAdapter) Worker Configuration Properties

This section describes the configuration properties of the SMAdapter

| Worker Base Offset | Property Name | Access | Description |
|---|---|---|---|
| +0x0000 | smaCtrl | RW | Control Register (see text) |
| +0x0004 | mesgCount | RO | WMI Rolling 32b message count |
| +0x0008 | abortCount | RO | WMI Rolling 32b abort count (imprecise) |
| +0x000C | | RO | Reserved |
| +0x0010 | thisMesg | RO | WSI-S {opcode[7:0], mesgLength[23:0]} |
| +0x0014 | lastmesg | RO | WSI-S {opcode[7:0], mesgLength[23:0]} |
| +0x0018 | portStatus | RO | Port Status: WSI-S[15:8], WSI-M[7:0] |
| +0x0020 | pMesgCount | RO | WSI-S |
| +0x0024 | iMesgCount | RO | WSI-S |
| +0x0028 | pMesgCount | RO | WSI-M |
| +0x002C | iMesgCount | RO | WSI-M |

## Using SMAdapter

The Stream-Message Adapter (SMAdapter) implements one of the five adapter topologies shown at right. A control register set prior to START determines the module's function. Six RO diagnostic registers measure the flow of messages past each port.



*WSI Pass*

*WMI to WSI Adapter*

*WSI to WMI Adapter*

*WSI Monitor/Split/Fork*

*WSI-WMI Merge/Join*

| Bits | Name | Description |
|---|---|---|
| 31:6 | | Reserved |
| 5 | impWsiM | 0=Normal; 1=Force Imprecise on WSI-M egress |
| 4 | nixWsiM | 0=Normal; 1=No Put to WSI-M port (/dev/null) |
| 3:0 | mode | 0 = Pass WSI-S to WSI-M<br>1 = WMI to WSI-M<br>2 = WSI-S to WMI<br>3 = Fork WSI-S to both WSI-M and WMI<br>4 = Merge WSI-S and WMI to WSI-M<br>5:15 = Reserved |
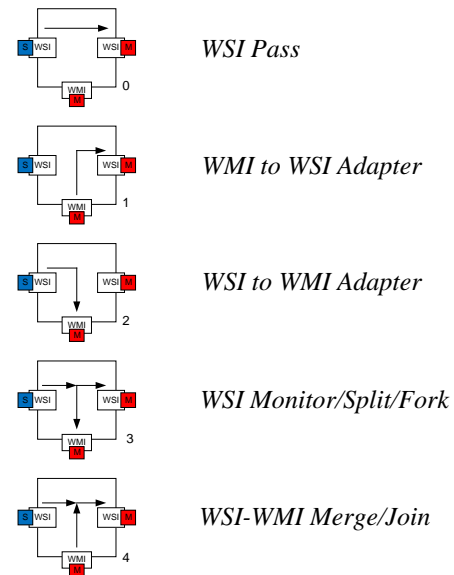
To start each worker, de-assert reset, and then issue the INITIALIZE and START control operations in sequence.

The following read-only (instrumentation) properties are initialized to zero at reset and on an INITIALIZE control-op. They are available for each of the three ports WMI, WSI-S, and WMI-M individually

The _MCnt registers maintain a 32b unsigned rolling message count (MCnt). This count is updated at message completion and indicate "how many messages have moved through this port".

The _MLast registers sample the opcode and mesgLength of the last message processed. This data is updated as soon as new information is available; but cannot be relied on to determine if a particular message has completed ingress or egress from any port. The mesgLength is given in Bytes.

# DelayWorker Configuration Properties

This section describes the configuration properties of the DelayWorker

| Worker Base Offset | Property Name | Access | Description |
|---|---|---|---|
| +0x0000 | dlyCtrl | RW | Bits[3:0] 4'h0=Bypass (no delay);  4'h7=Delay |
| +0x0004 | dlyHoldoffBytes | RW | Number of bytesWritten to holdoff before read |
| +0x0008 | dlyHoldoffCycles | RW | Number of cyclesPassed to holdoff before read |
| +0x000C | mesgWtCount | RO | |
| +0x0010 | mesgRdCount | RO | |
| +0x0014 | bytesWritten | RO | |
| +0x0018 | portStatus | RO | WMEMI-M[23:16],  WSI-S[15:8],  WSI-M[7:0] |
| +0x001C | 0 | RO | |
| +0x0020 | WSI-S: pMesgCount | RO | |
| +0x0024 | WSI-S: iMesgCount | RO | |
| +0x0028 | WSI-S: tBusyCount | RO | |
| +0x002C | WSI-M: pMesgCount | RO | |
| +0x0030 | WSI-M: iMesgCount | RO | |
| +0x0034 | WSI-M: tBusyCount | RO | |
| +0x0038 | wmemiWrReq | RO | |
| +0x003C | wmemiRdReq | RO | |
| +0x0040 | wmemiRdResp | RO | |
| +0x0044 | dlyWordsStored | RO | |
| +0x0048 | dlyRdCredit | RO | |
| +0x004C | dlyWAG | RO | |
| +0x0050 | dlyRAG | RO | |

# Using the DelayWorker

The DelayWorker delays messages by storing them in a DRAM-based circular buffer. Incoming messages are stored as they arrive. Outgoing messages do not begin until both the dlyHoldoffBytes and dlyHoldoffCycles constraints have been satisfied. The cycle counter, (nominally counting 125 MHz / 8nS cycles), begins counting when the first message arrives.

For example, to delay a message stream by 1 second, set the delayHoldoffCycles to 125E6; and leave dlyHoldoffBytes at its default of 0.

To use the DelayWorker, de-assert reset, and then issue the INITIALIZE and START control operations in sequence.

The DelayWorker delay function *requires* the WMemI interface, typically provided by an FTop' DramServer. It is essential to the message delay function that the WMemI service provider be initialized and started with an observed successful DRAM PHY training *before* using the DelayWorker for delay.

The yellow-lined, read-only instrumentation properties provide details of the internal state. These include: Multiplexing the metadata and message streams into a sequence of 16B words and write these words to DRAM memory. When the read criterion has been satisfied, begin reading these 16B words from DRAM memory and splitting them back into metadata and message streams.

# FrameGate Configuration Properties

This section describes the configuration properties of the FrameGate

| Worker Base Offset | Property Name | Access | Description |
|---|---|---|---|
| +0x0000 | frameGateCtrl | RW | |
| +0x0004 | frameSize | RW | |
| +0x0008 | sampleSkip | RW | |
| +0x000C | op0MesgCnt | RO | |
| +0x0010 | op1MesgCnt | RO | |
| +0x0014 | op2MesgCnt | RO | |
| +0x0018 | | RO | WSI-S[15:8], WSI-M[7:0] |
| +0x001C | WSI-S: pMesgCount | RO | |
| +0x0020 | WSI-S: iMesgCount | RO | |
| +0x0024 | WSI-S: tBusyCount | RO | |
| +0x0028 | WSI-M: pMesgCount | RO | |
| +0x002C | WSI-M: iMesgCount | RO | |
| +0x0030 | WSI-M: tBusyCount | RO | |

# Using the FrameGate

The FrameGate worker *frames* and *gates* an incoming stream of imprecise messages into precise sized messages of frameSize. It first *frames* the data with minimal buffering and latency by counting frameSize samples of opcode-0 as moves them from WSI-S to WSI-M. Then it *gates* off the input by discarding sampleSkip opcode-0 samples on the WSI-S input. Messages that are not opcode 0 are passed directly from input to output.

The yellow-lined, read-only instrumentation properties provide details of the internal state. These include the usual monitor and extended-monitor functions for WSI ports; as well as rolling counts of the number of opcode 0, 1, and 2 messages observed.

# DramServer Configuration Properties

This section describes the configuration properties of the DramServer

| Worker Base Offset | Property Name | Access | Description |
|---|---|---|---|
| +0x0000 | dramStatus | RO | |
| +0x0004 | drmCtrl | RW | |
| +0x0008 | dbg_calib_done | RO | |
| +0x000C | dbg_calib_err | RO | |
| +0x0010 | dbg_calib_dq_tap_cnt | RO | |
| +0x0014 | dbg_calib_dqs_tap_cnt | RO | |
| +0x0018 | dbg_calib_gate_tap_cnt | RO | |
| +0x001C | dbg_calib_rd_data_sel | RO | |
| +0x0020 | dbg_calib_ren_delay | RO | |
| +0x0024 | dbg_calib_gate_delay | RO | |
| +0x0028 | 32'hCODE_BABE | RO | |
| +0x002C | wmemiWrReq | RO | |
| +0x0030 | wmemiRdReq | RO | |
| +0x0034 | wmemiRdResp | RO | |
| +0x0038 | wmemi.status | RO | |
| +0x003C | Wmemi.ReadInFlight | RO | |
| +0x0040 | 0 | RO | |
| +0x0044 | 0 | RO | |
| +0x0048 | requestCount | RO | |
| +0x004C | 0 | RO | |
| +0x0050 | pReg | RW | |
| +0x0054 | 4B WRITE PIO | WO | |
| +0x0058 | 4B READ PIO | WO | |
| +0x005C | mReg | RW | |
| +0x0060 | wdReg[0] | RO | |
| +0x0064 | wdReg[1] | RO | |
| +0x0068 | wdReg[2] | RO | |
| +0x006C | wdReg[3] | RO | |
| +0x0080 | rdReg[0] | RO | |
| +0x0084 | rdReg[1] | RO | |
| +0x0088 | rdReg[2] | RO | |
| +0x008C | rdReg[3] | RO | |

# Using the DramServer

The DramSever provides a DRAM controller allowing a paged, flat-map as well as WMemI port for the user application.

To use the DramServer, de-assert reset, and then issue the INITIALIZE and START control operations in sequence.

The yellow-lined, read-only instrumentation properties provide details of the internal state

# Fabric Consumer (FC) and Fabric Producer (FP) Worker Configuration Properties

This section describes the configuration properties of both the FCWorker and FPWorker

| Worker Base Offset | Property Name | Access | Description |
|---|---|---|---|
| +0x0000 | r0 | RW | 4B GP Test Register "r0" |
| +0x0004 | mesgCount | RO | 32b message count |
| +0x0008 | zeroCount | RO | 32b count of zero-sized messages |
| +0x000C | mesgSum | RO | Rolling 32b sum of message data |
| +0x0010 | lastMesg | RO | {opcode[7:0], mesgLength[23:0]} |

# Using the FCWorker and FPWorker

The FCWorker is a "Fabric Consumer"; it consumes data from the infrastructure WMI interface by reading and produces a WSI message. The FPWorker is a "Fabric Producer"; it consumes data from its WSI interface and produces and writes a WMI message.

To start each worker, de-assert reset, and then issue the INITIALIZE and START control operations in sequence.

The "r0" register is available to test WCI access. It has no side-effect.

The following read-only properties are initialized to zero at reset and on an INITIALIZE control-op.

The "mesgCount" register maintains a 32b unsigned rolling message count. This count is updated at message egress and thus indicated "how many messages have fully left this worker".

The "zeroCount" register maintains a 32b unsigned rolling count of zero-sized messages. (The number of non-zero-sized messages may be derived by subtracting the zeroCount from the mesgCount, so long as the counters have not rolled over).

The "mesgSum" register maintains a 32b checksum of each valid 4B word that has advanced.

The "lastMesg" register samples the opcode and mesgLength of the last message processed. This data is updated as soon as new information is available; but cannot be relied on to determine if a particular message has completed ingress or egress from any port. The mesgLength is given in Bytes.

*Note: The FabricConsumer (FC) and FabricProducer (FP) IPs use has been depreciated.Use the Stream-Message Adapter (SMAdapter) instead.*

# BiasWorker Configuration Properties

This section describes the configuration properties of the BiasWorker

| Worker Base Offset | Property Name | Access | Description |
|---|---|---|---|
| +0x0000 | biasValue | RW | Bias (offset) to add to each 4B message element. 32b UInt |
| +0x0004 | controlReg | RW | |
| +0x0008 | messagePush | RO | Rolling 32b count of doMessagePush firing |
| +0x000C | lastOpcode0 | RO | Data associated with last Opcode 0 |
| +0x0018 | portStatus | RO | Unused[31:16]; Port Status: WSI-S[15:8], WSI-M[7:0] |
| +0x0020 | pMesgCount | RO | WSI-S |
| +0x0024 | iMesgCount | RO | WSI-S |
| +0x0020 | pMesgCount | RO | WSI-M |
| +0x0024 | iMesgCount | RO | WSI-M |

# Using the BiasWorker

The BiasWorker accepts a message from its WSI Slave input and passes it to its WSI Master output. It transforms all message data by adding the 32b UInt "biasValue" to each 4B element received. Both the message data and the biasValue are handled as 32b unsigned integers (unsigned 32.0 format). A biasValue of zero (the default) lets the message pass unaltered.

To start the BiasWorker, de-assert reset, and then issue the INITIALIZE and START control operations in sequence.

# OpenCPI RPL A/D and D/A Worker Control (General Topic)

OpenCPI has generalized analog-to-digital (A/D) collection and digital-to-analog (D/A) emission capabilities that can be specialized for specific board and devices. Control of these ADC and DAC "device workers" are performed with a first-class notion of actual time, as provided by the RPL time service. Both the ADC and DAC workers share a similar control paradigm for gating the ingress or egress of sample data to or from a latency-insensitive message domain. Described below are the methods and properties for controlling ingress (collection) and egress (emission) across the sample/message barrier:

**Use of the Worker Control Operations** "Start" and "Stop": No samples shall cross the sample/message barrier, under any condition, unless the worker is in the "Operating" state. The movement of data is initiated across the barrier by a ***Trigger Event***, which may be any combination of:

**Software Trigger**:  A configuration property write

**Trigger Time Condition**: An absolute time that specifies when to trigger

**External Trigger In**: An input event to the system used as a trigger

A ***Trigger Output*** is provided to synchronize other devices to the trigger event.

**"dwell"**: A non-zero dwell specifies a duration over which data will cross the sample/message barrier. It can be thought of as "how long" to be collecting or emitting active signal samples before pausing. A zero dwell duration is a special case that describes infinite dwell.

**"start"**: A duration that specifies the delay from trigger to the start of the sample/message movement.

**"periodic"**: From the trigger, a non-zero periodic repeat duration is an interval that will pass before the repeat of the start/dwell pattern. A zero periodic repeat duration is a special case that describes no repeat at all. The end a non-zero periodic interval generates a trigger output.

To minimize latency, and latency-uncertainty (jitter); the effective sample/message bounds has been placed as close to the actual converter circuitry as possible.

The sample/message bounds are strict, transactional barriers. Failure to satisfy the ability to successfully ingress a sample to message (in an ADC Worker); or egress a message to sample (in a DAC Worker), will be recorded and will raise a WCI attention. Conversely, successful status indication at these device workers memorializes the messages and samples that have been processed.
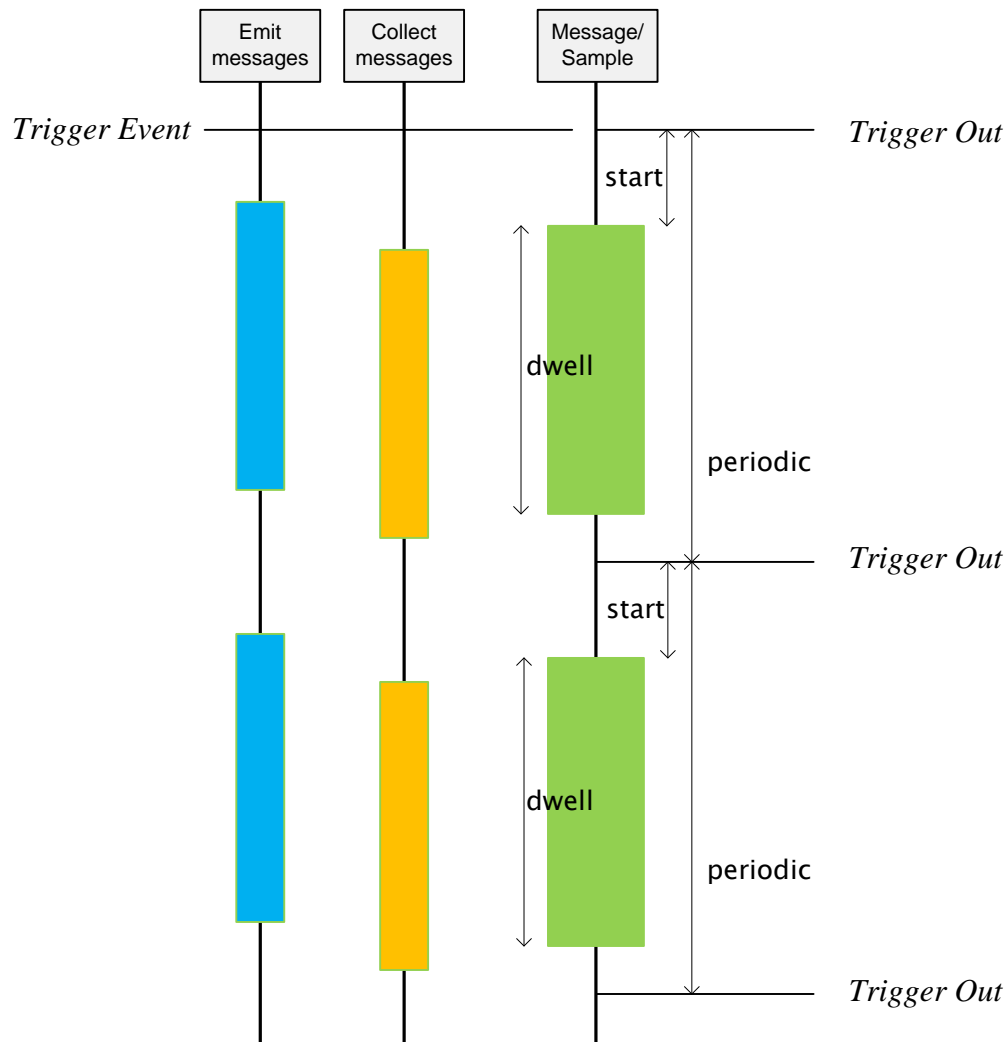
**Figure 1 - Sequence Diagram of Message/Sample Barrier**

The sequence diagram above shows the following key relationships:

- A trigger event causes a trigger output and establishes a start line
- A non-zero **periodic** creates an endless repetition of the start line
- An offset of **start** is inserted before data is allowed to cross the sample/message barrier
- Data crosses the sample/message barrier for the duration of **dwell** (Green)
- Messages arriving for emission need to arrive in time (Light Blue)
- Messages departing from collection leave as soon as possible (Orange)

# ADCWorker "Device Worker"

The ADCWorker is a type of worker known as a "device-worker", as it is instanced outside the OpenCPI application container in FTop'. Like any other OpenCPI worker, ADCWorker is controlled by its WCI interface (INITALIZE, START, etc); and its operation is configured by configuration properties. It accepts clock and data from an ADC (or ADCs); and produces one or more WSI-Master message stream outputs.

Following initialization and configuration, the ADCWorker produces several types of messages, identified by opcode. The types of messages produced, and how often they are emitted, are controlled by WCI. The amount of data produced by the WSI-Master is deterministic based on WCI settings and the frequency of the ADC clock (which can be observed by reading a configuration property). A finite-sized FIFO buffers the short-term rate differences and clock domain crossing. This FIFO will overflow if the connected WSI-Slave cannot sustain consumption of the produced ADCWorker messages.  When that error condition occurs, it is recorded.

The ADCworker is a client of the RPL Time Service. As such the ADCWorker may be programmed by configuration properties to optionally insert Time information at the start of ADC sample data. Independently, the ADCWorker may be set to send discrete, periodic "beacon" Time messages.

The ADCworker may be triggered, or re-triggered, by several means, including setting specific configuration properties, or the arrival of enabled synchronization ("sync") signals. Each time the ADC worker is triggered to acquire, it will produce a capture frame of data. A capture frame of data is comprised of **numMesgPerFrame** messages, each of **fixedMesgSize**. The product of these two configuration properties are the number of Bytes in the capture frame. This is directly related to the number of ADC samples. And to the so-called "dwell time" of continuous acquisition through the ADC sample clock. While the ADCWorker exposes the fixedMesgSize through every message sent, the count of numMesgPerFrame is not exported. It is used locally to simply count off how many messages to send in a frame of dwell.

| Opcode | Message Type | Length | Description |
|---|---|---|---|
| 0 | Sample | Varies | On a 4B (32b) WSI link, two 16b ADC samples (N+1) and N are packed little-endian into a DWORD. $Sample_{N+1}$ is in bits [31:16], $Sample_N$ is in [15:0]. When the converter data is less than 16b, the data from the converter is MSB justified, with zeros in the LSBs.<br><br>These are (typically) imprecise messages whose length will not exceed the configuration property "maxMesgLength".<br><br>Transmission of this message is enabled by default; it may be inhibited by setting the control bit disableSample. |
| 1 | Sync | 0B | A Zero-Length message that is used to indicate a synchronization event. These events may be context and application specific. They typically include sampling discontinuities, such as at the start of acquisition.<br><br>Transmission of this message is disabled by default; it may be enabled by setting the control bit enableSync. |
| 2 | Timestamp | 24B | A 24B message that conveys the timestamp and supporting information.<br><br>```
typedef struct {
  Bit#(32) iSeconds;      // "now": integer Seconds
  Bit#(32) fSeconds;      // "now": fractional Seconds
  Bit#(32) dropCount;     // Rolling count of dropped samples
  Bit#(32) sampCnt;       // Rolling count of captured samples
  Bit#(32) dwellStarts;   // Rolling count of dwell starts
  Bit#(32) dwellFails;    // Rolling count of dwell failures
} SampMessage
```<br><br>Transmission of this message is disabled by default; it may be enabled by setting the control bit enableTimestamp. |

# ADCWorker Configuration Properties

This section describes the configuration properties of the ADCWorker.

| Configuration Property Offset (B) | Property Name | Access | Description |
|---|---|---|---|
| +0x0_0000 | wsiM Status | RO | WSI-M Port status in bits[7:0] |
| +0x0_0004 | adcStatusLS | RO | See ADC Status Bits[31:0] Table TBD |
| +0x0_0008 | maxMesgLength | RW | ADC Data maximum message length (in Bytes), multiples of 4B, $4 <= maxMesgLength <= 65532$ |
| +0x0_000C | adcControl | RW | See ADC Control Bits |
| +0x0_0010 | | RO | rsvd |
| +0x0_0014 | fcAdc | RO | Measured Frequency of ADC Sample Clock (in KHz, updated at 1KHz) |
| +0x0_0018 | adcSampleEnq | RO | Rolling 32b count of ADC Samples ENQ in capture domain |
| +0x0_001C | sampleSpy0 | RO | Last two samples from ADC0 (little endian) {second:first} |
| +0x0_0020 | sampleSpy1 | RO | Last two samples from ADC1 (little endian) {second:first} |
| +0x0_0024 | spiClock | WO | Issue indirect-IO SPI command to Clock |
| +0x0_0028 | spiAdc0 | WO | Issue indirect-IO SPI command to ADC0 |
| +0x0_002C | spiAdc1 | WO | Issue indirect-IO SPI command to ADC1 |
| +0x0_0030 | spiResponse | RO | Last Response from an SPI Read Command |
| +0x0_0034 | mesgCount | RO | Rolling 32b count of WSI messages sent (all opcodes) |
| +0x0_0038 | | RO | |
| +0x0_003C | Stats.dwellStarts | RO | Number of Dwell intervals Started |
| +0x0_0040 | Stats.dwellFails | RO | Number of Dwell intervals Failed |
| +0x0_0044 | lastOverflowMesg | RO | Value of mesgCount when buffer overflow last occurred |
| +0x0_0048 | phaseCmdAdc0 | WO | ADC0 Phase Shift (b1=ENA, b0=INC) |
| +0x0_004C | phaseCmdAdc1 | WO | ADC1 Phase Shift (b1=ENA, b0=INC) |
| +0x0_0050 | extStatus.pMesgCount | RO | Precise Messages Generated |
| +0x0_0054 | extStatus.iMesgCount | RO | Imprecise Messages Generated |
| +0x0_0058 | Stats.dropCount | RO | Samples Dropped |
| +0x0_005C | dwellFails | RO | Dwell intervals where samples were dropped |
| +0x0_0400 – 0x0_04FC | ADC0 SPI | RW | Memory Map of 1KB ADC0 device control space |
| +0x0_0800 – 0x0_08FC | ADC1 SPI | RW | Memory Map of 1KB ADC1 device control space |
| +0x0_0C00 – 0x0_0CFC | Clock SPI | RW | Memory Map of 1KB Clock device control space |

## adcControl (ADCWorker +0x0_000C)

This register controls the overall behavior of the ADC device worker.

| Bits | Name | Access | Description |
|---|---|---|---|
| 31:4 | | Reserved | 0 |
| 3 | inhibitOnDrop | RW | 0=Continue acquire if samples dropped<br>1=Inhibit acquire if samples are dropped |
| 2 | enableTimestamp | RW | 0=Inhibit Transmission of Timestamp messages<br>1=Allow Transmission of Timestamp messages (Opcode 2) |
| 1 | enableSync | RW | 0=Inhibit Transmission of Sync messages<br>1=Allow Transmission of Sync messages (Opcode 1) |
| 0 | disableSample | RW | 0=Allow Transmission of ADC data messages<br>1=Inhibit Transmission of ADC data messages (Opcode 0) |

## fcAdc (ADCWorker +0x0_0014)

The measured frequency (in KHz) of the ADC sample clock.

## adcSampleCount (ADCWorker +0x0_0018)

A rolling 32b count of ADC Sample clocks observed.

# DACWorker Configuration Properties

This section describes the configuration properties of the DACWorker.

| Configuration Property Offset (B) | Property Name | Access | Description |
|---|---|---|---|
| +0x0_0000 | dacStatusMS | RO | See DAC Status Bits[63:32]   [7:0] WSI-S port status |
| +0x0_0004 | dacStatusLS | RO | See DAC Status Bits[31:0] |
| +0x0_0008 | | RO | rsvd |
| +0x0_000C | dacControl | RW | See DAC Control Bits |
| +0x0_0010 | fcDac | RO | Measured Frequency of DAC Sample Clock (1/16 $F_{SAMP}$) (in KHz, updated at 1KHz) |
| +0x0_0014 | dacSampleDeq | RO | Rolling 32b count of DAC Samples DEQ in emitter domain |
| +0x0_0018 | mesgCount | RO | Rolling 32b count of WSI messages received (all opcodes) |
| +0x0_001C | samplesSinceSync | RO | 32b count of DAC samples deq since last sync event |
| +0x0_0020 | underflowCount | RO | 32b count of sourceless DAC emitter superwords |
| +0x0_0024 | lastUnderflowMesg | RO | Value of mesgCount when buffer underflow last occurred |
| +0x0_0028 | popCount | RO | Rolling 32b count of DEQ cycles from WSI Slave |
| +0x0_002C | unrollCount | RO | State of WSI Message Unroll |
| +0x0_0030 | syncCount | RO | Rolling 32b count of "Opcode 3" Sync Messages Received |
| +0x0_0034 | mesgStart | RO | |
| +0x0_0038 | underFlowCount | RO | |
| +0x0_003C | stageCount | RO | |
| +0x0_0040 | | RO | |
| +0x0_0044 | | RO | |
| +0x0_0048 | extStatus.pMesgCount | RO | WSI-S Precise Messages Received |
| +0x0_004C | extStatus.iMesgCount | RO | WSI-S Imprecise Messages Received |

## dacControl (DACWorker +0x0_000C)

This register controls the overall behavior of the DAC device worker.

| Bits | Name | Access | Description |
|---|---|---|---|
| 31:4 | | Reserved | 0 |
| 5 | | RW | |
| 4 | emitDisable | RW | 0=emit Enabled 1=emit Disabled |
| 3 | dacClkDiv | RW | 1=DAC outputs $F_{SAMP}$/8 (normal) |
| 2 | dacDelay | RW | 0 = Normal |
| 1 | dacRz | RW | 0 = |
| 0 | dacRf | RW | 0 = |

# Data-Plane Control Registers

This section describes the control registers used by the data plane. As an example, an application "OC1001" as shown in the following diagram has been constructed.



We describe the registers and memory used in the infrastructure DPx blocks. Note that in this application DP0 feeds the "fabric consumer" worker (by WMI path 1) and DP1 is fed from the "fabric producer" worker (by WMI path 4).

The following table shows the BAR0 worker assignments in the OC1001 application:

| BAR0 Offset | Region | OC1001 Assignment |
|---|---|---|
| +0x0_0000 | Admin | Admin |
| +0x1_0000 | Worker 0 | GCD Worker 0 |
| +0x2_0000 | Worker 1 | GCD Worker 1 |
| +0x3_0000 | Worker 2 | FC Worker 2 |
| +0x4_0000 | Worker 3 | Bias Worker 3 |
| +0x5_0000 | Worker 4 | FP Worker 4 |
| +0x6_0000 | Worker 5 | Unused |
| +0x7_0000 | Worker 6 | Unused |
| +0x8_0000 | Worker 7 | Unused |
| +0x9_0000 | Worker 8 | Unused |
| +0xA_0000 | Worker 9 | Unused |
| +0xB_0000 | Worker 10 | Unused |
| +0xC_0000 | Worker 11 | Unused |
| +0xD_0000 | Worker 12 | Unused |
| +0xE_0000 | Worker 13 | Data Plane 0 (FC) |
| +0xF_0000 | Worker 14 | Data Plane 1 (FP) |

# BAR1 – 64KB Memory Space for Data Movement

BAR1 is a 64KB memory space used by the data plane. Only 32b DWORD access to BAR1 is allowed. Each data plane adapter occupies 32KB. This 32KB space may be partially- or fully populated- with BRAM buffer memory. The 64KB space is suitable to hold two data plane adapters.

| BAR1 Offset | Region |
|---|---|
| +0x0000 | Data Plane 0 Buffer Memory (16KB) |
| +0x4000 | Data Plane 0 Buffer Memory Expansion (16KB) |
| +0x8000 | Data Plane 1 Buffer Memory (16KB) |
| +0xC000 | Data Plane 1 Buffer Memory Expansion (16KB) |

We show and example of a 16KB data plane memory-mapped BRAM used to hold messages and metadata. The infrastructure can be programmed the following organization for a 7-buffered storage in each BRAM. There are seven 2KB message buffers and seven 16B metadata buffers.

| BRAM Offset | Region |
|---|---|
| +0x0000 | Message Buffer 0 (2KB) |
| +0x0800 | Message Buffer 1 (2KB) |
| +0x1000 | Message Buffer 2 (2KB) |
| +0x1800 | Message Buffer 3 (2KB) |
| +0x2000 | Message Buffer 4 (2KB) |
| +0x2800 | Message Buffer 5 (2KB) |
| +0x3000 | Message Buffer 6 (2KB) |
| +0x3800 | Metadata 0 (16B) |
| +0x3810 | Metadata 1 (16B) |
| +0x3820 | Metadata 2 (16B) |
| +0x3830 | Metadata 3 (16B) |
| +0x3840 | Metadata 4 (16B) |
| +0x3850 | Metadata 5 (16B) |
| +0x3860 | Metadata 6 (16B) |
| +0x3870~3FFF | Unused (16KB – 14KB – 112B) |

The 16B data structure for the metadata is

```
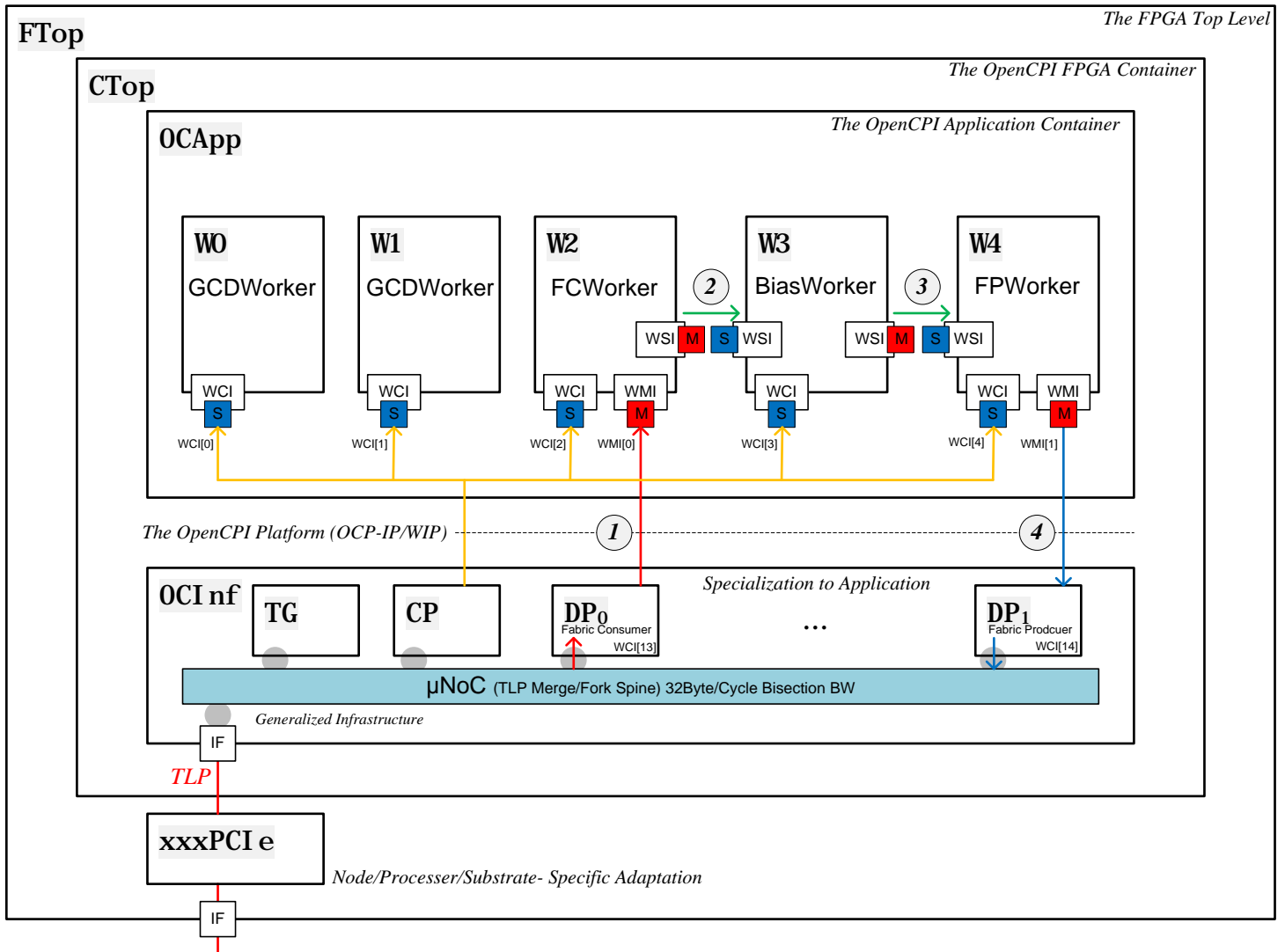typedef struct {
  Bit#(32) length;   // Message Length in Bytes
  Bit#(32) opcode;   // Opcode in bits [7:0]
  Bit#(32) tag;
  Bit#(32) interval;
} MesgMeta deriving (Bits, Eq)
```

# Data Plane Configuration Properties (DPCP)

The data plane configuration properties (DPCP) allow specialization of the data plane infrastructure IP behavior at runtime. Each data plane attachment has its own DPCP register set.

The color coding of the rows in the table below delimits the initialization properties, from the message-advancing events, from the buffer status, from the side-effect-free status and debugging probes, from the registers directly used in active message and doorbell movement.

| DPCP Base Offset | Property Name | Mode | Default | When Used | Description |
|---|---|---|---|---|---|
| +0x0000 | lclNumBufs | RW | 1 | initialize | Number of Local Buffers (This Node) |
| +0x0004 | fabNumBufs | RW | 1 | Initialize | Number of Fabric Buffers (Far Node) |
| +0x0008 | lclMesgBase | RW | 0x0000 | initialize | Local Message Buffer Base Address |
| +0x000C | lclMetaBase | RW | 0x3800 | initialize | Local Metadata Buffer Base Address |
| +0x0010 | lclMesgBufSize | RW | 0x0800 | initialize | Local Message Buffer Size |
| +0x0014 | lclMetaBufSize | RW | 0x0010 | initialize | Local Metadata Buffer Size |
| +0x0018 | fabDoneAvail | WO | - | MesgAdv | Fabric Done Available Scalar (See Table Following Describing Usage) |
| +0x001C | regionNum | RO | | initialize | To which memory region this DP belongs |
| +0x0020 | bufReady | RO | - | Poll | Lcl Buffers Committed/Freed (Number of buffers available for remote) |
| +0x0024 | rsvd | RO | - | Debug | 32'hF00D_FACE |
| +0x0028 | bmlAccu | RO | - | Debug | { Lcl Buffers Available/Ready[31:16], RemoteBuffersAvailable[15:0]} |
| +0x002C | bufIndex | RO | - | Debug | {remIndex [31:16] , lclIndex [15:0]} |
| +0x0030 | lclCounts | RO | - | Debug | {lclStart [31:16] , lclDone [15:0]} |
| +0x0034 | remCounts | RO | - | Debug | {remStart [31:16] , remDone [15:0]} |
| +0x0038 | thisMesg | RO | - | Debug | {opcode [31:24] , length [23:0]} |
| +0x003C | lastMesg | RO | - | Debug | {opcode [31:24] , length [23:0]} |
| +0x0040 | v[1] | RO | - | Debug | req/wrt Count |
| +0x0044 | v[0] | RO | - | Debug | wrtData |
| +0x0048 | rsvd | RO | - | Debug | 32'hDADE_BABE |
| +0x004C | bufferExtent | RO | - | Debug | Size of Local Memory Buffer |
| +0x0050 | fabMesgBase | RW | FFFF_0000 | DMA | Fabric DMA Message Base Address |
| +0x0054 | fabMetaBase | RW | FFFF_3800 | DMA | Fabric DMA Metadata Base Address |
| +0x0058 | fabMesgSize | RW | 0000_0800 | DMA | Fabric DMA Message Buffer Size |
| +0x005C | fabMetaSize | RW | 0000_0010 | DMA | Fabric DMA Metadata Buffer Size |
| +0x0060 | fabFlowBase | RW | FFFF_0018 | DMA | Fabric DMA Flow Control Base Address |
| +0x0064 | fabFlowSize | RW | 0000_0004 | DMA | Fabric DMA Flow Control Base Size |
| +0x0068 | dpControl | RW | 0 | DMA | Data Plane Control (See Text) |
| +0x006C | flowDiagCount | RO | 0 | Debug | Number of Flow Control Events Sent |

Notes: All "sizes" and "addresses" in Bytes unless otherwise stated.

# fabDoneAvail (DPCP +0x0018)

This write-only DWORD location treats each write as signaling either a "done" or "avail" event. Depending upon the data plane dpDirection and dpRole set in the dpControl register (DPCP+0x68), the event signaled here will cause a specific internal action within the DPCP, as described in the table below:

| Local Role | Fabric Producer (FP) | Fabric Consumer (FC) |
|---|---|---|
| Passive | (source) done – near buffer empty | (sink) done – near buffer full |
| Active Message | (pusher) avail – far buffer empty | (puller) avail – far buffer full |
| Active Flowcontrol | (pullee) done – near buffer empty | (pushee) done – near buffer full |

This single-location for all inbound data plane event signaling can reduce the complexity of control software.

Internally to the DPCP, "done" events generate a "remoteDone" signal which is used to
    Increment LBAR
    Decrement LBCF
    Increment remBuf
    Increment fabBuf

Internally to the DPCP, "avail" events generate a "fabBufAvail" signal which is used to
    Increment FBA

# regionNum (DPCP +0x001C)

This read-only location indentifies to which dataplane memory region this DP belongs

| Bits | Name | Description |
|---|---|---|
| 31:4 | rsvd | '0 |
| 3:0 | regionNum | DP memory region |

# dpControl (DPCP +0x0068)

This register controls the behavior of the Data Plane. For each DP adapter in the infrastructure that is to be actively used, a choice of data flow *direction* and *role* must be made.

The dpDirection field requires a specific selection of Producer vs. Consumer, if the module is to act at all. The dpRole field refines that behavior into a Passive, Active Message, or Active Doorbell role.

| Bits | Name | Access | Description |
|------|------|--------|-------------|
| 31:3 | | Reserved | 0 |
| 7 | dpMoveData | RW | 0=Inhibit Active Message Movement<br>1=Enable Active Message Movement |
| 6 | dpMoveMeta | RW | 0=Inhibit Active Metadata Movement<br>1=Enable Active Metadata Movement |
| 5 | dpSendTail | RW | 0=Inhibit Transmission of Active Message Tail Event<br>1=Enable Transmission of Active Message Tail Event |
| 4 | dpSendInterrupt | RW | 0=Inhibit Interrupt at end of Message Movement<br>1=Enable Interrupt at end of Message Movement |
| 3:2 | dpDirection | RW | 00=Disabled DP Module<br>01=**Fabric Producer (FP)**<br>10=**Fabric Consumer (FC)**<br>11=Reserved |
| 1:0 | dpRole | RW | 00=**Passive**<br>01=**Active Message**<br>10=**Active Flowcontrol**<br>11=Reserved |

| Local Role | Fabric-Producer (FP) | Fabric-Consumer (FC) |
|------------|----------------------|----------------------|
| **Passive** | 1. Partner polls status and reads message data by any means | 2. Partner polls status and writes message data by any means |
| **Active Message** Movement | 3. We receive fabric Flowcontrol for far-side "partner-buffer-free"; we **produce message data by Active-Push** DMA | 4. We receive fabric Flowcontrol for far-side "partner-buffer-full"; we **consume message data by Active-Pull** DMA |
| **Active Flowcontrol** Transmission | 5. We **Transmit** DMA "local-full" Flowcontrol to our partner; Partner reads message data by any means | 6. We **Transmit** DMA "local-free" Flowcontrol to our partner; Partner sends message data by any means |

These settings are applied to each DP component in accordance with the generic control proxy on incidence of a control operation START event while in the INITIALIZED state.

Unused DP adapters in the infrastructure will remain disabled due both to the default "disabled" in the dpDirection field; as well as control system not initializing and then starting that component.

**Multi-buffer Initial Conditions**

The initial conditions of the multi-buffer accumulators has been chosen so that no precharge or priming is required to start data movement.

```
if (dpControl.dir==FProducer)  begin
  lclBufsAR    <= lclNumBufs;  // Producer starts all  Available (for egress from FPGA)
  lclBufsCF    <= 0;           // Producer starts none Committed (for egress to Fabric)
  fabBufsAvail <= fabNumBufs;  // Producer starts knowing all fabric buffers are Available
end else begin
  lclBufsAR    <= 0;           // Consumer starts none Ready (for ingress to FPGA)
  lclBufsCF    <= lclNumBufs;  // Consumer start  all  Freed (for ingress from Fabric)
  fabBufsAvail <= 0;           // Consumer starts knowing no fabric buffers are Available
end
```

## Multi-Buffer Logical Block Diagram

The diagram below shows a simplified version of the logic connecting the various produce/consume events to the accumulators which track buffer availability and position.