**OpenCPI HDL Device Workers Interface Specification**

Authors:

Shepard Siegel, Atomic Rules LLC ([Shepard.Siegel@atomicrules.com](mailto:Shepard.Siegel@atomicrules.com))

Jim Kulp, Mercury Federal Systems, Inc. (jkulp@mercfed.com)

*Revision History*

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Description of Change** | **By** | **Date** |
| 1.01 | Creation | jkulp | 2010-07-01 |
|  |  |  |  |

Table of Contents

1 Dram Device Worker 4

1.1 Configuration Properties 6

1.2 Using the DramServer 7

2 OpenCPI A/D and D/A Device Worker Control (General Topic) 9

3 ADC Device Worker 11

3.1 Configuration Properties 12

3.1.1 adcControl (ADCWorker +0x0\_000C) 13

3.1.2 fcAdc (ADCWorker +0x0\_0014) 14

3.1.3 adcSampleCount (ADCWorker +0x0\_0018) 14

4 DAC Device Worker 15

4.1 Configuration Properties 15

4.1.1 dacControl (DACWorker +0x0\_000C) 16

# References

This document depends on several others. Primarily, it depends on the “OpenCPI Generic Authoring Model Reference Manual”, which describes concepts and definitions common to all OpenCPI authoring models, and the “OpenCPI HDL Authoring Model Reference” which describes how HDL workers must be written.

|  |  |  |
| --- | --- | --- |
| **Title** | **Published By** | **Link** |
| OpenCPI Generic Authoring Model Reference | [OpenCPI](http://opencpi.org) | Public URL: <http://www.opencpi.org/doc> |
| OpenCPI HDL Authoring Model Reference | [OpenCPI](http://opencpi.org) | Public URL: <http://www.opencpi.org/doc> |

# Overview

This document describes OpenCPI HDL device workers, which are IP blocks for FPGAs that on one side present standard OpenCPI WIP-profile interfaces for use by applications, and on the “back” side attach to external devices via pins of the FPGA. They are essentially the “device drivers” for OpenCPI FPGA platforms. They are “optional” in the sense that for a given FPGA platform, they are only required to be included in the bitstream design if the application actually needs them. This is in contrast to the core OpenCPI infrastructure IP that is required as a basis for all OpenCPI application bitstreams (described in a different document).

Thus one view of these device workers is “just workers”, like application workers, in that they are controlled by WCI, have configuration properties and control operations, and, for device workers acting in the data plane, they have WIP data plane interfaces to connect to applications and consumer or produce data.

So this list of device workers is also the list of devices externally attached to FPGAs that are supported by OpenCPI for reasons other than the core bootstrapping devices (like PCI Express and basic timekeeping).

Configuration properties shaded in yellow are meant only for debugging and can be “compiled out” when not desired to save resources.

As with all HDL workers, all application workers must follow the control pattern defined in the HDL Authoring Model Reference [AMR]. First, workers are “taken out of reset”, by deasserting the OCP MReset\_n signal on its WCI OCP Slave interface. Next, if supported by the worker, the ***Initialize*** control operation is performed. Finally, the ***Start*** control operation is performed to cause the worker to begin operating (enter the operational state).

Device Workers are instanced outside the OpenCPI application container, in what is known as FTop’ (FTop-prime). This is the outer ring of IP bloks in the OpenCPI HDL Platform that attached to external devices via pins of the FPGA. While Ftop (not prime) contains core functions required by all applications, FTop’ (Ftop-prime) contains optional device workers that are needed only by some applications.

# Dram Device Worker

## Function Performed

This device worker allows access to DDR2 SDRAM via a WMemI profile interface. It is based on the MIG memory controller from Xilinx.

## Configuration Properties

This section describes the configuration properties of the DramServer

|  |  |  |  |
| --- | --- | --- | --- |
| **Property 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 Dram Device Worker

The Dram Device Worker provides a DRAM controller allowing a paged, flat-map as well as WMemI port for the user application. It is essential to any application workers attached to this device worker that it be successfully initialized and started before use. This allows the DRAM Device Worker to perform DRAM PHY, which is a process that cannot be done under reset.

# OpenCPI A/D and D/A Device 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 an asynchronous message domain. Described below are the methods and properties for controlling ingress (collection) and egress (emission) across the sample/message (isochronous/asynchronous) 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, entered by the ***Start*** control operation. 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 (from the HDL Time Service via an HDL Time Client)

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

A ***Trigger Output*** signal (that can be external to the FPGA) 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 (to start of dwell).

**“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 is 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 - 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)

# ADC Device Worker

## Function performed

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 a 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 configuration properties. The amount of data produced by the WSI-Master is deterministic based on configuration property 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 overrun error condition occurs, it is recorded.

The ADC worker is a client of the HDL Time Service. As such the ADC worker may be programmed by configuration properties to optionally insert time information at the start of ADC sample data. Independently, the ADC Worker may be set to send discrete, periodic “beacon” Time messages.

The ADC worker 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 ADC Worker 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. SampleN+1 is in bits [31:16], SampleN 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.  These are (typically) imprecise messages whose length will not exceed the configuration property “maxMesgLength”.  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 indicate sampling discontinuities, such as at the start of acquisition, or when acquisition is gated.  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.  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  Transmission of this message is disabled by default; it may be enabled by setting the control bit enableTimestamp. |

## Configuration Properties

This section describes the configuration properties of the ADC Device Worker.

|  |  |  |  |
| --- | --- | --- | --- |
| **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. The initial value is zero.

|  |  |  |  |
| --- | --- | --- | --- |
| **Bits** | **Name** | **Access** | **Description** |
| 31:5 |  | Reserved | 0 |
| 4 | average4 | RW | 0=Normal operation  1=The ADC outputs one averaged sample for every four digitized |
| 3 | inhibitOnDrop | RW | 0=Continue acquire if samples dropped  1=Inhibit acquire if samples are dropped |
| 2 | enableTimestamp | RW | 0=Inhibit Transmission of Timestamp messages  1=Allow Transmission of Timestamp messages (Opcode 2) |
| 1 | enableSync | RW | 0=Inhibit Transmission of Sync messages  1=Allow Transmission of Sync messages (Opcode 1) |
| 0 | disableSample | RW | 0=Allow Transmission of ADC data messages  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.

## Using the ADC Worker

With its default settings, the ADC worker will start producing data messages (Opcode 0) as soon as the Start control operation is performed. Other bits in the control property enable the two other opcodes: Sync (sample discontinuity) and Time (Time of following sample). Whether an overrun (samples dropped) stops the acquisition is another option.

Dwell, start offset, and period time intervals are programmed by TBD.

# DAC Device Worker

## Function Performed

The DAC device worker accepts data from a WSI input and send data samples in the messages to the DAC.

## Configuration Properties

This section describes the configuration properties of the DACWorker.

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |  |
| +0x0\_000C | dacControl | RW | See DAC Control Bits |
| +0x0\_0010 | fcDac | RO | Measured Frequency of DAC Sample Clock (1/16 FDAC)  (in KHz, updated at 1KHz) |
| +0x0\_0014 | dacSampleDeq | RO | Rolling 32b count of Samples DEQ in emitter domain |
| +0x0\_0018 |  | RO |  |
| +0x0\_001C |  | RO |  |
| +0x0\_0020 |  | RO |  |
| +0x0\_0024 | firstUnderrunMesg | RO | mesgStart where underflowCount first became non-zero |
| +0x0\_0028 |  | RO |  |
| +0x0\_002C |  | RO |  |
| +0x0\_0030 | syncCount | RO | Rolling 32b count of “Opcode 3” Sync Received |
| +0x0\_0034 | mesgStart | RO | Rolling 32b count WSI message Starts Received |
| +0x0\_0038 | underflowCount | RO | Rolling 32b count of underflows (16-samples per LSB) |
| +0x0\_003C | stageCount | RO | Rolling 32b count of enqueues (16-samples per LSB) |
| +0x0\_0040 |  | RO |  |
| +0x0\_0044 |  | RO |  |
| +0x0\_0048 | pMesgCount | RO | WSI-S Precise Messages Received |
| +0x0\_004C | iMesgCount | RO | WSI-S Imprecise Messages Received |
| +0x0\_0050 | tBusyCount | RO | WSI-S Rolling Count of SThreadBusy backpressure |

### dacControl (DACWorker +0x0\_000C)

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Bits** | **Name** | **Access** | **Description** |
| 31:8 |  | Reserved | 0 |
| 7 | toneEn | RW | 0=Disable; 1= FDAC/16 CW Tone (Write 0x88 to dacControl) |
| 6 | invertMSBs | RW | 0=NOP; 1= Invert MSBs (dual-sample b31 and b15) |
| 5 | replicate16x | RW | 0=replicate2x; 1=replicate16x |
| 4 | emitEnable | RW | 0=emit Disabled; 1=emit Enabled |
| 3 | dacClkDiv | RW | 0=Not Supported; 1=DAC outputs FSAMP/8 (normal) (this bit always set) |
| 2 | dacDelay | RW | 0 = Normal |
| 1 | dacRz | RW | 0 = Normal |
| 0 | dacRf | RW | 0 = Normal |

Following START and emitEnable==1; the DACWorker will not begin emission until 128 4B words (256 samples) are received at the WSI-S port.

## Using the DAC worker

After it is started, the DAC worker actually starts sending samples to the DAC upon receiving the first WSI message (actually 256 samples are available). After than it watches for an underrun condition where there is no incoming data to put on the isochronous output. By looking at the underrun indication (the first underrun message property), it is easy to determine how much data passed before underrun.