OpenCPI
Capture v2 Component Data Sheet

OpenCPI Release: v2.4.7

Revision History

Revision	Description of Change	Date
v1.4	Initial Release	10/2018
v1.5	Version bump.	4/2019
v1.6	Converted Worker to version 2	11/2019
v1.7	Table of Worker Configurations and Resource Utilization Table removed	5/2020
v2.1	Capture v2 updated to use input port clock. New property called totalBytes added.	3/2021
v2.2	Capture v2 RCC version of capture_v2created.	7/2021

Summary - Capture v2

Name	capture_v2
Worker Type	
OpenCPI Release	v2.4.7
Last Update	11/2019
Component Library	ocpi.assets.util_comps
Workers	capture_v2.hdl
Tested Platforms	isim, Matchstiq-Z1(PL), xsim, ZedBoard(PL), centos 7

Functionality

The capture_v2 component provides the ability to store an input port's data. Two modes are supported:
* 'leading' - capture all messages from input port until the buffer is full. This has the affect of capturing messages from the start of an application.
* 'trailing' - capture all messages from input port and allow the buffer's content to be overwritten while application is in operation. This has the affect of capturing all messages (a buffers worth) near the end of an application.

This component provides an optional output port, so that, it may be placed between two components. For the capture_v2.hdl worker, the input messages are directly passed to the output port with a latency of the input port clock cycles.

The capture_v2 component takes input port messages and stores their data in a buffer via the `data` property as 4 byte words. Metadata associated with each message is stored as a record in a metadata buffer via the `metadata` property. It captures four 4 byte words of metadata. The first metadata word is the opcode of the message and message size (bytes); opcode 8 MSB and message size 24 LSB. The second word is the fraction time stamp for the EOM. The third word is the fraction time stamp for the SOM. And the fourth word is the seconds timestamp for the SOM. For capture_v2.rcc, the second and third words have the same value since there is no concept of start and end of message for RCC workers. So that the metadata can be read on a little-endian processor, the ordering of the metadata is as follows:

1) opcode (8-bit) & message size (24-bit),
2) eom fraction (32-bit),
3) som fraction (32-bit),
4) som seconds (32-bit)

Some example python code for reading in the `metadata` property values written to a file packed little-endian:

```
data = struct.unpack('<I', ifile.read(4))[0] # 32/8 = 4 bytes
```

For capture_v2.hdl, when the number of bytes sent for a message is not a multiple of 4, only the last (number of bytes modulus 4) least significant bytes of the last word in the data buffer represent received data. The (number of bytes modulus 4) most significant bytes will always be zero in the last word in the data buffer.
For example, given that last captured data buffer word is 0x0000005a:
- if number of bytes is 5, the last data received was 0x5a.
- if number of bytes is 6, the last data received was 0x005a.

The capture_v2 component counts the number of metadata records (`metadataCount`) have been captured, how many data words have been captured (`dataCount`), and the total number of bytes that were passed through the worker during an app run (`totalBytes`). It allows for the option to wrap around and continue to capture data and metadata once the buffers are full or to stop capturing data and metadata when the data and metadata buffers are full via the `stopOnFull` property.

When `stopOnFull` is true (leading), data and metadata will be captured as long as the metadata buffer is not full. The data buffer will loop if it is not full and the metadata buffer is not full. The metadata buffer will loop independently until it is not full. When `stopOnFull` is false (trailing), the data and metadata buffers loop

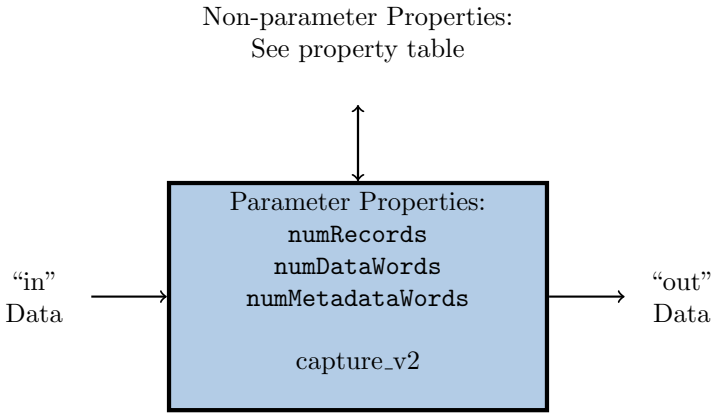
independently. There will be a wrap around when the data and metadata buffers are full and data and metadata will continue to be captured.

The component also has properties that keep track of whether or not the metadata and data buffers are full; `metaFull` and `dataFull`.

When using capture_v2.hdl, the `totalBytes`, `metadataCount`, `dataCount`, `metaFull`, and `dataFull` properties should be checked at the end of an app run because they won't be stable until then. The worker doesn't use cdc crossing circuits for them because it takes advantage that they will have a stable value by the time the control plane reads those values at the end of an app run.

Block Diagrams

Top level



Source Dependencies

capture_v2.hdl

- assets/components/util_comps/comp.hdl/comp.vhd
- assets/components/util_comps/comp.hdl/capture_implementation.vhd
- core/hdl/primitives/util/util_pkg.vhd
- core/hdl/primitives/util/BRAM2.v

capture_v2.hdl

- assets/components/util_comps/capture_v2.rcc/capture_v2.cc

Component Spec Properties

Name	Type	Default	SequenceLength	ArrayLength	ArrayDimensions	Parameter	Accessibility	Usage
stopOnFull	bool	false	-	-	-	false	Initial	True - Stop capturing data and metadata when the data and metadata buffers are full. The data buffer will loop if it is not full and the metadata buffer is not full. The metadata buffer will loop independently until it is not full. False - Wrap around and continue to capture data and metadata once the buffers are full. This stop functionality is independent of both the control plane 'stop' operation and 'finished' worker state.
metadataCount	uLong	-	-	-	-	false	Volatile	Counter of metadata records written.
dataCount	uLong	-	-	-	-	false	Volatile	Counter of words captured.
numRecords	uLong	256	-	-	-	true	-	This is the maximum number of metadata records that may be captured for messages received (not necessarily the content).
numDataWords	uLong	1024	-	-	-	true	-	Maximum number of 32 bit data words that may be stored in the data buffer. If stopOnFull is true, meaning no wrap around, no more data will be captured once the data buffer is full.
numMetadataWords	uLong	4	-	-	-	true	-	Due to a limitation, cannot use constrained elements in unconstrained array declarations, so cannot directly set the second dimension for the metadata property to 4. The number of metadata words must always be 4, since there are four 4 byte words that are captured. The first metadata word is the opcode for the message and message size in bytes;opcode 8 MSB and message size 24 LSB. The second word is the fraction timestamp for the EOM. The third word is the fraction timestamp for the SOM. And the fourth word is the seconds timestamp for the SOM. So the default value must not be changed.
metaFull	bool	false	-	-	-	false	Volatile, Initial	Metadata buffer full flag.
dataFull	bool	false	-	-	-	false	Volatile, Initial	Data buffer is full flag.
stopZLMOpcode	uChar	0	-	-	-	false	Initial	Opcode associated with the ZLM which causes the worker to become finished.
stopOnZLM	bool	false	-	-	-	false	Initial	Indicates causing the worker to become finished on ZLM of stopZLMOpcode.
stopOnEOF	bool	true	-	-	-	false	Initial	Indicates causing the worker to become finished on EOF. stopOnEOF is always regarded as true now since the only reason to be false was the opcode-0-ZLM aliasing that no longer exists.
totalBytes	uLongLong	-	-	-	-	false	Volatile	Total number of bytes that passed through the worker during an app run..
metadata	uLong	-	-	-	numRecords, numMetadataWords	false	Volatile	Multidimensional array containing metadata records.
data	uLong	-	-	numDataWords	-	false	Volatile	Data buffer containing data words.

Component Ports

Name	Protocol	Producer	Optional	Usage
in	-	false	false	Data input to capture
out	-	true	true	Data from input passed to output unchanged

Worker Interfaces

capture_v2.hdl

Type	Name	DataWidth (b)	Advanced	Usage
StreamInterface	in	32	DataValueWidth=8, NumberOfOpcodes='256', ZeroLengthMessages=true, clockDirection="in"	Data input to capture
StreamInterface	out	32	DataValueWidth=8, NumberOfOpcodes='256', ZeroLengthMessages=true, clock="in"	Data from input passed to output unchanged
TimeInterface	time	64 (32b sec MSW and 32b frac LSW)	-	Allows worker to capture timestamps

Control Timing and Signals

The capture_v2 worker uses the clock from the Data Plane and Control Plane. And it uses standard Control Plane signals.

Worker Configuration Parameters

capture_v2.hdl

Performance and Resource Utilization

capture_v2.hdl

Test and Verification

The capture_v2-test.xml has a test property called `testScenario` that allows for five different test cases: testing sending no data; testing making only metadata full; testing making data full; testing sending multiple zlms (with different opcodes), a single word message, filling data and filling up metadata (for configurations where there are at least six metadata records); and test sending stopZLMOpcode opcode and no output port connected.

An input file is generated via `generate.py`. The `generate.py` script will output different input data based on the `testScenario` chosen.

The tests are verified by `verify.py` script. It checks that the `totalBytes`, `metadataCount`, `dataCount`, status(`metaFull` and `dataFull`), `metadata` and `data` match the expected results.

Applications

For an example of the capture_v2 component used in an application, please reference the tb_bias_v2 application located in `assets/applications/tb_bias_v2`.

Something to note is that for the hdl worker, the BRAM2 module that is used to store the data and metadata initializes the BRAM to an initial value of 0xAAAAAAAA (2863311530 in base 10) in simulation. This means at the start of an application, in simulation, the `data` and `metadata` properties will have an initial value of 0xAAAAAAAA.

There is also a script that outputs the capture_v2 properties in a more readable format, to a file. This is primarily formatting the metadata property. This can script can be used when trying to interpret the capture_v2 properties when running an app. It requires the user to use the dumpFile option when running an app and the provide that dump file to the script. A capture_v2 unit test .log file can also be used as an input to the script. The script is located in `assets/scripts/format_capture_v2_property_dump.py`.