

Summary - Capture_v2

| | |
|-------------------|--------------------------|
| Name | Capture_v2 |
| Version | v1.5 |
| Release Date | 5/2019 |
| Component Library | ocpi.assets.util_comps |
| Workers | Capture_v2.hdl |
| Tested Platforms | xsim, isim, matchstiq_z1 |

Revision History

| Revision | Description of Change | Date |
|----------|---|---------|
| v1.4 | Initial Release | 10/2018 |
| v1.5 | <ul style="list-style-type: none"> Version bump. | 5/2019 |

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. The input messages are directly passed to the output port with a latency of the control plane 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. 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
```

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.

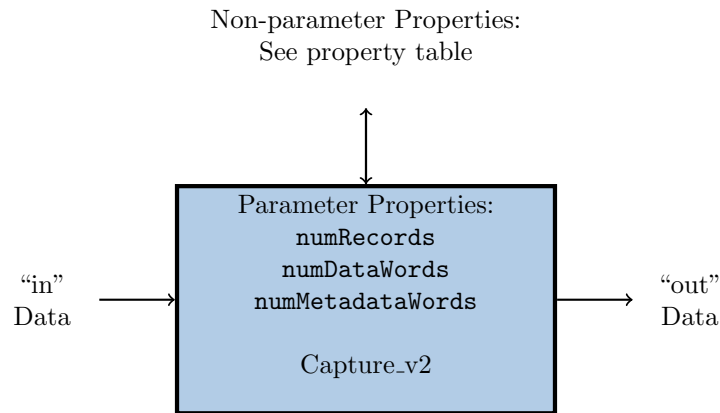
The Capture_v2 component counts the number of metadata records (`metadataCount`) have been captured and how many data words have been captured (`dataCount`). It allows for the option to wrap around and continue to capture data and metadata once the buffers are full or to stop capturing data and metadata 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. If the data buffer is full before the metadata buffer, metadata will still be captured until the metadata buffer is full. If the metadata buffer is full before the data buffer, no more metadata and no more data will be captured. When `stopOnFull` is false (trailing), 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`.

Block Diagrams

Top level



Source Dependencies

`Capture_v2.hdl`

- `assets/components/util_comps/Capture_v2.hdl/Capture_v2.vhd`
- `core/hdl/primitives/util/util_pkg.vhd`

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. If the metadata buffer is full before the data buffer, no more metadata and no more data will be captured. If the data buffer is full before the metadata buffer, no more data is captured, but continue to capture metadata until the metadata buffer is 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 | - | Number of metadata records/messages to be captured. |
| numDataWords | uLong | 1024 | - | - | - | true | - | Number of four byte data words 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 indicates the application is 'done'. |
| stopOnZLM | bool | false | - | - | - | false | Initial | Indicates stopping on ZLM of stopZLMOpcode. |
| stopOnEOF | bool | true | - | - | - | false | Initial | As of now, this indicates stopping on ZLM of opcode 0. In the future it is expected that OpenCPI will standardize a definition of EOF. |
| 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 | Data input to capture |
| StreamInterface | out | 32 | DataValueWidth = 8, NumberOfOpcodes='256', ZeroLengthMessages = true | 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 Control Plane and standard Control Plane signals.

Worker Configuration Parameters

Capture_v2.hdl

Table 1: Table of Worker Configurations for worker: capture_v2

| Configuration | numRecords | numDataWords |
|---------------|------------|--------------|
| 0 | 256 | 1024 |

Performance and Resource Utilization

Capture_v2.hdl

Table 2: Resource Utilization Table for worker "capture_v2"

| Configuration | OCPI Target | Tool | Version | Device | Registers (Typ) | LUTs (Typ) | Fmax (MHz) (Typ) | Memory/Special Functions |
|---------------|-------------|---------|---------|-----------------|-----------------|------------|------------------|--------------------------|
| 0 | stratix4 | Quartus | 17.1.0 | N/A | 524 | 695 | N/A | Block Memory Bits: 65536 |
| 0 | zynq | Vivado | 2017.1 | xc7z020clg400-3 | 519 | 742 | N/A | RAMB36E1: 3 |
| 0 | zynq_ise | ISE | 14.7 | 7z010clg400-3 | 516 | 890 | 309.245 | RAMB36E1: 5 |
| 0 | virtex6 | ISE | 14.7 | 6vcx75tff484-2 | 518 | 891 | 257.069 | RAMB36E1: 5 |

Limitations

Since the raw property address (`props_in.raw.address`) is currently 16 bits wide and `data` and `metadata` are raw properties that contain 4 byte words, the total address space for these properties is $2^{16}/4$. The framework does not throw an error when the total number of elements between the two array properties exceeds $2^{16}/4$ so the properties will not be filled correctly. This failure is run time and the application will successfully execute. Once AV-4254 is addressed to change the raw.address from 16 to 32, the new address space limit will be $2^{32}/4$.

Test and Verification

The `Capture_v2-test.xml` has a test property called `testScenario` that allows for four 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).

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 `metadataCount`, `dataCount`, `status(metaFull and dataFull)`, `metadata` and `data` match the expected results.

There are also some custom tests located in the `Capture_v2.test` directory that tests things not currently supported by the test suite. These tests are: testing stopping on an opcode other than 0 (need to be able to set `done='Capture_v2'` in the app xml to be able to test this) and testing the `Capture_v2` with no output connected. The tests are built and run manually via a `run_test.sh` script.

The custom tests also have `generate.py` and `verify.py` scripts but they are simpler versions than the ones in the top level `Capture_v2.test` directory or they are specific to the custom test.

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`.