Cheshire eXchange Bus (CXB)

Cheshire eXchange Bus (CXB) is the bus protocol used by Dragon. It is designed for high throughput, low latency communication and supports multi-master and multi-slave configurations.

Overview

CXB is based on Wormhole Switching. All data transfers are in units of a Flit. A Flit is the amount that can be transferred across one bus link in a single cycle and is equal to the link width of 68 bits. Of the 68 bits, 64 bits are used for data transfer and 4 bits for routing / control metadata.

Transfers are unidirectional and organized into packets. Each packet consists of one or more flits. The first flit is always a header and describes the type of transfer. The header is followed by payload data that has a different interpretation depending on the transfer type.

Flit Formats

Common Metadata

The common flit metadata is defined in SystemVerilog as:

```
typedef struct packed {
    logic       header;
    logic       last;
    logic       _unused;
    logic       reset;
} CXB_FlitInfo;
```

Of the 4 bits of metadata, only 3 bits are currently used. These are `header`, `last`, and `reset`.

The first two bits are used for flow control. A packet's initial Flit is always a Header and must have the `header` bit set. This is cleared for all other flits. The `last` bit is set to indicate this is the last Flit in the packet and allows switching to be done without tracking packet sizes.

The `reset` bit is used to reset the bus. Within the bus logic, `reset` is automatically propagated to all master / slave devices and internal switch nodes. Devices should split their reset logic so that logic interfacing with the bus can be reset separately from the device's internal logic where possible. A bus reset allows the system to recover from partial reset. For example: the SDRAM controller may be reset without reseting the CPU, but the CPU can use a subsequent bus reset to know previously in-flight requests will not receive a response. The core logic of a device can be reset by sending it a Reset command (not yet implemented), allowing the device to be controlled at a low level via bus commands.

Packet Headers

Packet header flits are defined in SystemVerilog as:

```
typedef struct packed {
    CXB_FlitInfo common;

    /* Routing */
    logic        high_priority;   /**< Give this transmission priority. */
```

```
    logic        address_decode;  /**< If true, target ID must be decoded from address. */
    CXB_Id       source_id;       /**< Source device ID. */
    CXB_Id       target_id;       /**< Target device ID. */

    /* Command */
    logic       [1:0]  opcode;    /**< Packet command type (see above) */
    CXB_Tag            tag;       /**< Request tag */
    logic       [1:0]  unused;

    /* Read / Write Parameters (ignore for other opcodes) */
    logic       [1:0]  unit;      /**< Transfer size unit (8b, 16b, 32b, 64b) */
    CXB_Length         length;    /**< Request length in flits (minus one) */
    logic              increment; /**< If set, increment address by 'unit' after each flit */
    CXB_Address        address;   /**< Target request address */
} CXB_Header;
```

The `common` field holds the 4 bits of CXB metadata described above.

Every packet must start with a header in the above format. A new header cannot be sent until the previous packet has finished transmission (indicated by the `last` bit).

The `source_id` field generally does not need to be provided by devices. It is automatically filled in by the bus as the Header flit enters its first switch node[1].

If `address_decode` is set the bus will decode `address` as a global address and update `target_id` with the device that owns that address and update `address` with the device-local address (offset within the device's address space). In this case `target_id` does not need to be initialized[2].

The size of each transfer is set by `unit`. Although the size of each flit is fixed at 68 bits, it is possible for only a subset of bits to be used by each payload flit. Transfers must always be naturally aligned (i.e. a 32-bit transfer must be 32-bit aligned). Smaller transfer units allow more granular control of modified addresses at the expense of wasted bus bandwidth.

If `increment` is set, the receiving device should internally increment the effective `address` after each payload flit is received by the size of `unit`. If it is not set all read / writes will be executed against the same address provided in the Header. `increment` is normally set for memory-like access, but disabling this can be useful for interacting with memory-mapped structures where repeated accesses to the same register are useful.

In its current configuration, CXB IDs are 5 bits (maximum of 32 devices on a single bus), CXB tags are 8 bits, CXB lengths are 5 bits (maximum payload size of 32 flits), and CXB addresses are 32 bits.

The value of fields in the final section (`unit`, `length`, `increment`, and `address`) should be ignored for non Read / Write opcodes (including replies). They may be set to 0 or left uninitialized by devices.

Packet Payload

Packet payload flits are defined in SystemVerilog as:

```
typedef struct packed {
    CXB_BeatInfo common;
    logic [63:0] data;
} CXB_Payload;
```

The `common` field holds the 4 bits of CXB metadata described above.

The meaning of payload flits depends on the packet's opcode (as provided in the packet's Header).

[1] Insertion of `source_id` is not implemented yet. It is currently necessary for the sender to manually provide its ID in the header.

[2] Address decoding is not implemented yet. It is currently necessary for both `target_id` and `address` to be filled out correctly by the sender.

Packet Command Types

There are 4 packet command types. The type of packet is determined by the `opcode` field of the packet's Header. These are:

1. Read Address
2. Write Address
3. Other (TBD - Will include Reset / Interrupt / etc.)
4. Reply

Read / Write commands allow master devices to transfer data from / to slave devices. All Read / Write operations are address oriented, but the meaning of addresses within a device's assigned address range is up to the device. Addresses may represent memory-mapped IO (MMIO) or normal RAM.

Because packets are always routed to exactly one destination device, the address ranges defined by a Read / Write packet must be entirely within that device's address range. Addresses outside the device's normal address range may either wrap or be ignored.

Read commands consist of a Header with no Payload, and these packets are always exactly one Flit. The slave device must always respond with exactly the number of transfers requested.

Write commands consist of a Header followed by a Payload of exactly the number of flits specified by the `length` field (plus one). Write commands do not receive a response from the slave[3].

Reply commands are used to respond to other command types. They are not sent unsolicited, and there will be exactly one or zero Reply commands sent from a Slave for each command received by the Slave device. The Master device implementation must be able to identify the returned data from the `tag` and `source_id`. The value of the `address` in responses is Slave device-implementation defined.

Packet Routing

The CXB network routes packets using one or more CxbSwitchNode instances. Each CxbSwitchNode has two inputs (labeled A / B) and two outputs (labeled X / Y). There may be multiple valid paths through the network for each pair of `source_id` and `target_id`. However, the network must *not* have any cycles[4].

All switch nodes are unidirectional. There must be a separate set of node instances for communication in each direction. Best performance is achieved if all devices are strictly classified as either Master or Slave, as this requires significantly less paths through the network (i.e. no Master -> Master or Slave -> Slave paths are necessary).

Routing is done entirely with local decisions. The X and Y output ports each have a static bitmap (set via module parameter) that controls whether the port can be used to route packets destined for a device. The `target_id` in a packet's header is used to lookup into these bitmaps. If the corresponding bit for the output port is set, the packet is allowed to be routed through that port. If both ports are allowed, the X port will be preferred but the Y port will be used if the X port is busy with another transmission.

If each input port can be routed to a different output port, two packets can be transmitted in parallel. Internal routing logic adds a 1 cycle delay to packet transmissions plus 1 bubble cycle[5] (used for calculating the packet route) unless there is contention.

As an example, this will instantiate a node that will route packets destined for devices 2-7 through port X and 8-15 through port Y:

[3] Considering using a Header bit to request confirmation of write completion to solve ordering problems?
[4] If there are cycles in the network and a packet is routed through that cycle, it is possible for the switch network to deadlock.
[5] The current implementation adds 2 bubble cycles, but will be optimized to 1 cycle at a later date.

```
CxbSwitchNode #(
    .OUT_MASK_X(32'b00000000_00000000_00000000_11111100),
    .OUT_MASK_Y(32'b00000000_00000000_11111111_00000000)
) cxb_switch_node (
    .clock(clock_cpu),
    .resetn(sys_resetn),

    /* Input A -> Node */
    .in_rbus_A(brcxb_wbus),
    .in_rbus_A_valid(brcxb_wbus_valid),
    .out_rbus_A_ready(brcxb_wbus_ready),

    /* Input B -> Node */
    /*
    .in_rbus_A(cpu_wbus),
    .in_rbus_A_valid(cpu_wbus_valid),
    .out_rbus_A_ready(cpu_wbus_ready),
    */

    /* Node -> Output X */
    .out_wbus_X(cpu_rbus),
    .out_wbus_X_valid(cpu_rbus_valid),
    .in_wbus_X_ready(cpu_rbus_ready),

    /* Node -> Output Y */
    /*
    .out_wbus_X(brcxb_rbus),
    .out_wbus_X_valid(brcxb_rbus_valid),
    .in_wbus_X_ready(brcxb_rbus_ready),
    */
);
```

General Precautions

- Ordering of packets is not guaranteed. Two packets sent from Master A to Slave A are likely to arrive in order, but might not.
- Devices may buffer command requests and respond out of order.
- There is currently no way to guarantee write ordering on the bus, because commands can be processed out-of-order and there are no write responses.
- If an attempt is made to send a packet with no valid route to its destination, the bus will deadlock.