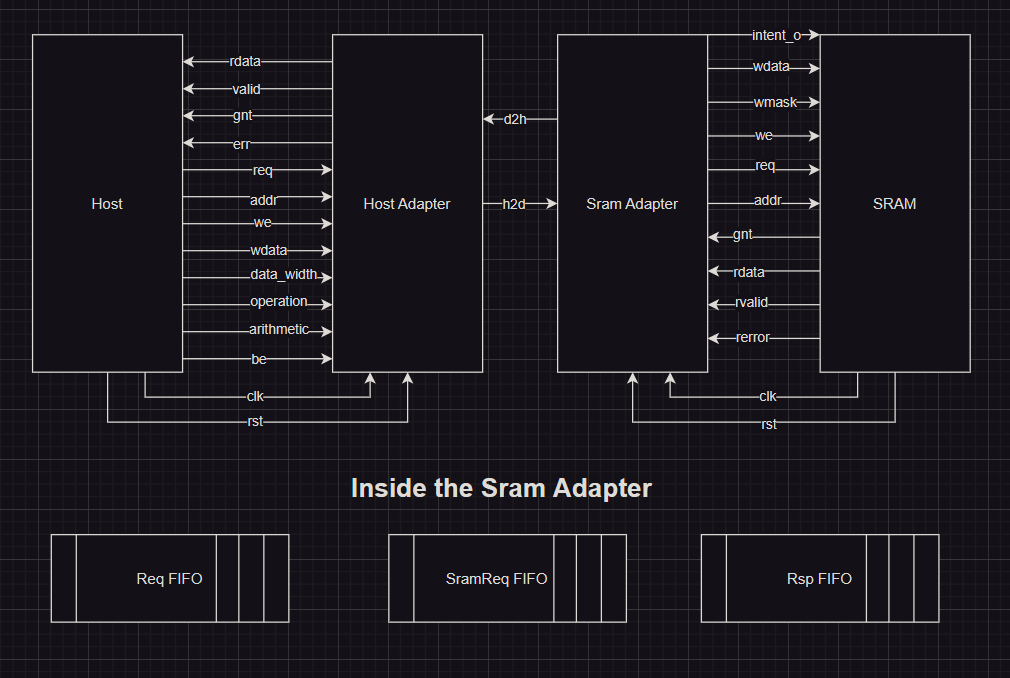
**Documentation for Upgrading TL-UL to TL-UH**

****

1. tl\_uh package
   1. I have made some changes to the enum that holds the opcodes of the request messages so that it includes the additional 3 new request messages: ArithmeticData, LogicalData and intent.
   2. The enum for the opcodes of the response messages has also been modified to include the hintAck message
   3. Additionally, I have introduced 3 new enums to encompass the various operations performed by the arithmetic and logical data, as well as, the intended operation in case of an intent request message.
2. tluh\_host\_adapter
   1. All request messages will be sent at the +ve edge of the clock.
   2. All response messages will be received at the -ve edge of the clock.
   3. Ports modification
      1. data\_width\_i
         1. input port.
         2. in the form of log2(datawidth / 8)
         3. will be assigned to the a\_size parameter.
      2. operation\_i
         1. input port.
         2. represents the arithmetic – logical – intent operation to be performed.
      3. arithmetic\_i
         1. input port.
         2. a bit indicates whether the operation is logical or arithmetic.
         3. 0: logical – 1: arithmetic.
   4. Supported features
      1. Burst feature
         1. How to detect it is a burst message
            1. First only the message that carries datapayload can be burst.
            2. As the size parameter contained in the tluh message content can take max value of 3 in our case (as it is only 2 bits width), this means that the data width can be up to 2^3 = 8 bytes
            3. The databus width is only 4 bytes. So, the max number of beats = 8 / 4 = 2 beats
         2. Now, we have 3 cases:
            1. Burst response only

E.g., Get 8 bytes data request 🡪 AccessAckData response.

Send the request message as non-burst message.

Raise the wait\_resp signal.

Then wait forever till the response arrives (no timeout).

Wait till the d\_valid signal becomes HIGH.

Count number of received beats (must be 2).

Once all beats are successfully received, pull the wait\_resp signal to 0 again.

* + - * 1. Burst request only

E.g., PutFullData – PutPartialData request with 8 bytes datapayload 🡪 AccessAck response.

Send each beat serially with incrementing a counter that count the number of sent beats.

As the response message can arrive any time from the cycle of the first beat (in the same cycle) till any delay, then we have to check for it each cycle so that if it arrives early (i.e., before finishing sending all beats), then buffer the signal in the buffer and raise a signal read\_from\_buffer.

After sending all beats, check the read\_from\_buffer signal which indicates whether there is a data in the buffer or not.

If so, then out the data from the buffer to the master (assign it to rdata\_o).

If not, then wait infinitely (raise wait\_resp signal) for the response ack.

* + - * 1. Burst request and burst response

E.g., Arithmetic or Logical requests with 8 bytes datapayload 🡪 AccessAckData response.

Send all beats serially.

If a beat is received during sending the beats, then put it in the buffer.

If no beats were received during sending, then no need to buffer them. Put the data directly on the bus instead.

* + 1. Atomic feature
       1. As the operation is performed at the slave side, simply send the message and await the response.
    2. Intent
       1. Can’t be burst as it doesn’t carry datapayload.
       2. So, just send the request and wait for the ack.

1. tluh\_sram\_adapter
   1. Ports modification
      1. intent\_o
         1. Output port
         2. A 2-bit indicating the type of the intent message.
         3. 0: not intent message – 1: intent to read – 2: intent to write.
   2. The already implemented module
      1. 3 FIFO queues
         1. reqfifo (Request FIFO)
            1. signals

wvalid

before: a\_ack (slave ready & master valid)

after: no change

rready

before: d\_ack (slave valid & master ready)

after: (((beat\_no == 2) && (reqfifo\_rdata.size == 3) && ((reqfifo\_rdata.op == OpAtomic) || (reqfifo\_rdata.op == OpRead))) || (reqfifo\_rdata.size < 3)) && d\_ack

illustration

the first part to make sure that all beats are sent in case of burst response.

The second part represent the case that it is a non-burst response.

The third part to make sure the master is ready to receive and the slave puts valid data.

wready

it is an output signal from the FIFO.

rvalid

it is an output signal from the FIFO.

* + - 1. rspfifo (Response FIFO)
         1. push the response contained data payload only (i.e., in case of receiving atomic or get requests).
         2. signals

wvalid

before: rvalid\_i & reqfifo\_rvalid (read data arrives from the SRAM and the request is popped from the request fifo)

after: rvalid\_i & (reqfifo\_rvalid || burst\_enable)

illustration

we need to OR the burst\_enable signal because in this case, there are 2 responses in the rsp\_fifo corresponding to 1 request in the req\_fifo.

rready

before: ternary operator: (reqfifo\_rdata.op == OpRead & ~reqfifo\_rdata.error) ? reqfifo\_rready : 1'b0

after: ((reqfifo\_rdata.op == OpRead || reqfifo\_rdata.op == OpAtomic) & ~reqfifo\_rdata.error) ? (reqfifo\_rready || (beat\_no == 1)) : 1'b0

illustration

If the operation is either read or atomic, then raise the signal if:

Either: the rready of request FIFO is raised.

Or: there is already one beat sent in case of a burst response message.

wready

it is an output signal from the FIFO.

rvalid

it is an output signal from the FIFO.

* + - 1. sramreqfifo (SRAM request FIFO)
         1. push the response contained data payload only (i.e., in case of receiving atomic or get requests).
         2. signals

wvalid

before: sram\_ack & ~we\_o

after: no change

rready

before: rspfifo\_wvalid

after: no change

wready

it is an output signal from the FIFO.

rvalid

it is an output signal from the FIFO.

* 1. supported features
     1. Burst Feature
        1. Burst response
           1. The response may only be burst if the request is either a read request or an atomic request.
           2. Therefore, we will examine the size parameter to determine if the data will fit within the data bus width, or we need to transmit the response as a burst message.
           3. If the message is a burst, we will activate the burst\_enable signal. After sending all beats, the signal will be deactivated again.
     2. Atomic feature
        1. Thea atomic operation will be performed by the alu instance from ALU module.
        2. If the operation is atomic, then an op\_enable signal is raised until the operation is done and sent to the sram.
        3. If the atomic operation is burst, then we will perform the operation in 2 steps. Also, the carry out form the first step will be carry in to the second step.
     3. Intent
        1. If the request message is an intent message, assign the type of the intent to the intent\_o port.
        2. No need to put the response in any of the SRAM request fifo or the response fifo.
        3. Can’t be burst.