Chapter 3
OCP Write Operations

Summary This chapter examines progressively intricate OCP write operations to introduce additional OCP signals, timings, and capabilities.

3.1 Posted Write Illustrating SCmdAccept Request
Handshake Command Pacing

Figure 3.1 illustrates the basic command accept flow-control mechanism using SCmdAccept. The master issues three posted writes and each experiences a different request accept latency.

3.1.1 Sequence Description by Time Point

1. The master presents the first posted write request by asserting
   (a) WR on MCmd
   (b) A valid address on MAddr
   (c) Data on MData.

   The OCP configuration files for both the master and the slave include the SCmdAccept signal. For this request, the slave asserts SCmdAccept to accept the transfer request in the same cycle the master asserted its command, the request accept latency for this transfer is 0.

2. The slave captures the write address and data and the first transfer ends when the master detects SCmdAccept asserted at the beginning of the next cycle. The master initiates a second posted write transfer request in the new cycle by leaving MCmd unchanged. The slave is unable to accept the new posted write request,
so it de-asserts (negates) SCmdAccept, indicating to the master it is not ready for
the new request.
3. Recognizing SCmdAccept is not asserted, the master must continue to hold all
request phase signals (MCmd, MAddr, and MData) steady through the next
cycle. The slave asserts SCmdAccept before the end of the cycle, the request
accept latency for this transfer is 1.
4. The slave captures the write address and data. And the master indicates that it has
no request for the slave by asserting IDLE in the MCmd signal.
5. After 1 idle cycle, the master presents a third non-posted write request. The slave
de-asserts SCmdAccept because it cannot accept the request.
6. The slave captures the write address and data. Since SCmdAccept is now asserted,
the transfer ends. SCmdAccept was negated (low) for 2 cycles, so the request
accept latency for this transfer is 2.

3.2 Non-Posted Write with Response Enabled

Figure 3.2 is similar to the previous example except the OCP interface provides
responses to non-posted writes. This is achieved using the resp configuration param-
eter that includes the two-wire SResp signal in the OCP interface. Table 3.1 shows
the four values SResp can assume.

For posted-writes operations, the only two valid responses are
1. DVA which indicates success
2. ERR which indicates an unsuccessful transfer (failure).

Providing a response naturally partitions a completed operation into more than
one phase
- Request phase
- Response Phase
Each phase has an independent latency:

1. A request accept latency for requests

This example shows how to determine these latency values.

In addition, the existence of different phases allow OCP to place signals in an OCP configuration into a signal group. With signal groups, all signals in a group are active together.

In the following example, note that prior to the first write operation, the slave has asserted SCmdAccept before the master asserts its first non-posted write request. This naturally results in a request accept latency of zero.

Moreover, because the slave typically asserts the command response in the same cycle, the request-to-response latency is also 0. Hence such transactions comprise 0-wait state write transactions.

Finally, note that in this example, the slave has asserted SCmdAccept prior to the master asserting its first non-posted write request. Since this example illustrates a fully-synchronous handshake approach, this is only possible when a slave’s ability to accept any command issued depends solely on its internal state. Same-cycle SCmdAccept may also be achievable using combinational signal logic.
3.2.1 Sequence Description by Time Point

(a) The master presents a first non-posted write request by asserting WR on MCmd, a valid address on MAddr, and associated data on MData. The slave, having already asserted SCmdAccept, immediately signals it accepts the request for a request accept latency of 0. Simultaneously, the slave drives DVA on SResp to indicate a successful transaction for a request-to-response latency of 0. This was a 0-wait state write transfer.

(b) The master starts a second non-posted write operation on the next cycle. The slave captures the write address and data and de-asserts SCmdAccept, indicating it cannot accept a new request. With SCmdAccept negated, the master must hold the MCmd, MAddr, and MData request phase signals.

(c) The slave asserts SCmdAccept in the next cycle, for a request accept latency of 1 and drives DVA on SResp to indicate a successful transaction for a request-to-response latency of 0.

(d) The slave captures the write address and data.

(e) After 1 idle cycle, the master starts a third non-posted write request. The slave de-asserts SCmdAccept.

(f) Since SCmdAccept is now asserted, the request phase ends. SCmdAccept was low for 1 cycle, so the request accept latency for this transfer is 1. The master detects that Sresp signals NULL, indicating the slave has not responded to the request yet. The slave captures the write address and data. The slave then drives DVA on SResp to indicate a successful transaction.

(g) The master samples the DVA response on Sresp and the third response phase ends with a request-to-response latency of 1.

3.3 Non-posted Write with Commit Response

OCP supports two types of basic writes:

1. Posted writes that optionally return a response depending on the OCP configuration indicating whether commands are accepted

2. Non-Posted writes that must return a response indicating that non-posted requests are either successfully or unsuccessfully committed

Figure 3.3 illustrates for non-posted write example transfers. As just indicated, the response must be returned to the master once the write operation commits. There is no command acceptance difference from posted writes, but the response may be significantly delayed. Thus, using non-posted writes for all posting-sensitive transactions can provide higher system reliability, a side effect can also be decreased data throughput.
### 3.3.1 Sequence Description by Time Point

(a) The master presents a first non-posted write request by asserting WRNP on MCmd, a valid address on MAddr, and data on MData. The slave *combinationally* asserts SCmdAccept and the request phase ends with a *request accept latency* of 0.

(b) The master asserts IDLE on the MCmd signal and waits for a commit response. The slave drives DVA on SResp in the same cycle to indicate a successfully-committed first transaction.

(c) The master samples DVA on SResp and the first response phase ends with a *request-to-response latency* of 1. The master then presents a new non-posted write request. The slave de-asserts SCmdAccept, indicating it is not yet ready to accept the second non-posted write request. The master holds the MCmd, MAddr, and MData signals.

(d) The slave asserts SCmdAccept for a *request accept latency* of 1.

(e) The slave captures the write address and data.

(f) The slave drives DVA on SResp to indicate a successfully-committed second transaction.

(g) The master samples DVA on SResp and the second response phase ends with a *request-to-response latency* of 2.

### 3.4 Posted Write with the Datahandshake Extension

Figure 3.4 shows three posted writes with no responses using the datahandshake extension. This extension adds the datahandshake phase, which is completely independent of the request and response phases. Two signals, MDataValid and SDataAccept, are added, and MData moves from the request phase to the datahandshake phase.
3.4.1 Sequence Description by Time Point

(a) The master starts a the first posted write request by driving WR on MCmd and a valid address on MAddr. It does not yet have the write data, however, so it de-asserts MDataValid. The slave asserts SCmdAccept. It does not need to assert or de-assert SDataAccept yet, because MDataValid is still de-asserted.

(b) The slave captures the write address from the master. The master is now ready to transfer the write data, so it drives the data on MData and asserts MDataValid, starting the datahandshake phase. The slave is ready to accept the data immediately, so it asserts SDataAccept. This corresponds to a data accept latency of 0.

(c) The master de-asserts MDataValid since it has no more data to transfer. (Like MCmd and SResp, MDataValid must always be in a valid, specified state.) The slave captures the write data from MData, completing the transfer. The master starts the second posted write request by driving WR on MCmd and a valid address on MAddr.

(d) Since SCmdAccept is asserted, the master immediately starts a third posted write write request. It also asserts MDataValid and presents the write data of the second posted write on MData. The slave is not ready for the data yet, so it de-asserts SDataAccept.

(e) The master sees that SDataAccept is de-asserted, so it holds the values of MDataValid and MData. The slave asserts SDataAccept, for a data accept latency of 1.

(f) Since SDataAccept is asserted, the datahandshake phase ends. The master is ready to deliver the write data for the third posted write request, so it keeps MDataValid asserted and presents the data on MData. The slave captures the
data for the second posted write from MData, and keeps SDataAccept asserted, for a \textit{data accept latency} of 0 for the third write.

(g) Since SDataAccept is asserted, the datahandshake phase ends. The slave captures the data for the third write from MData.

The next chapter discusses OCP signals, how to configure them, and their natural groupings.