Exercise 7: SystemC and Virtual Prototyping

Exercise on TLM Approximately Timed (AT)

Lukas Steiner

WS 2023/2024

Task 1

TLM Approximately Timed (AT)

This exercise is different. You don't have to write code. Surprised? The reason is simple: the AT protocol is pretty complex. Therefore, in this exercise you should study several code examples in order to understand how we can model in the AT coding style. To really learn the AT style you have to do a project where these code examples could be a starting point.

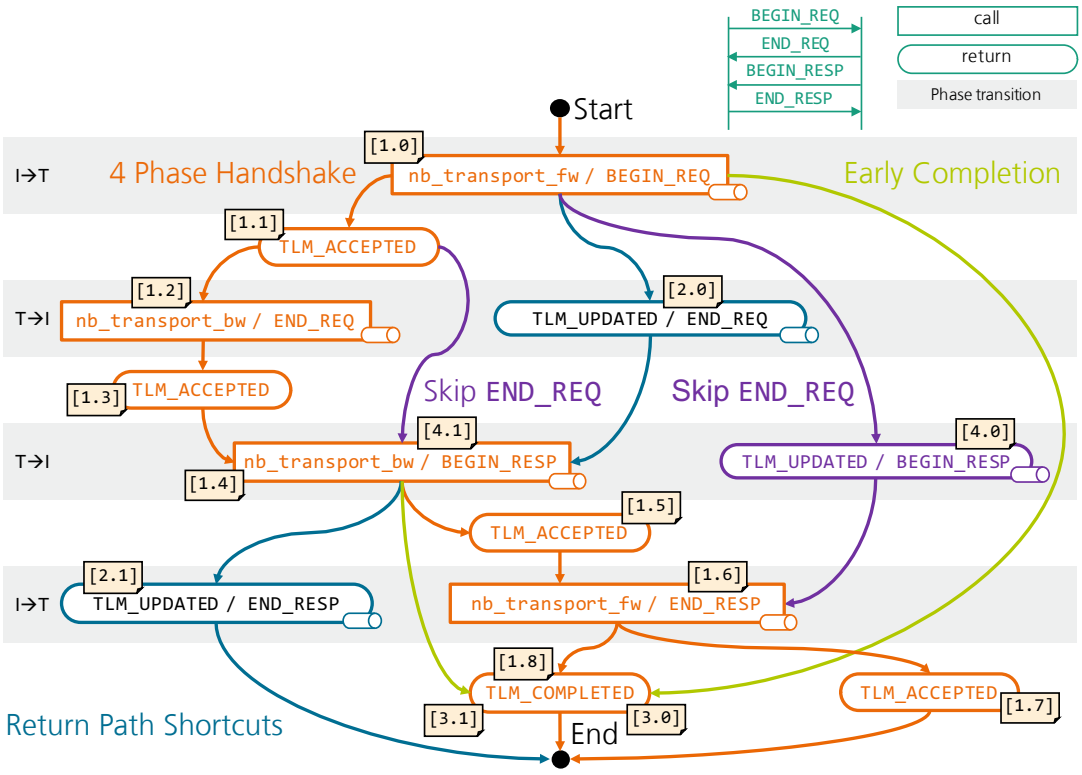

Fig. 1: The AT Cheat Sheet

Figure 1 shows a cheat sheet with the different possibilities that the TLM AT protocol offers. The annotated references `[X,Y]` are placed into the source code for a better orientation.

1. Four Phase Handshake

The code for the first example is available here:

`https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_at_1`

The example consists of an AT *Initiator* and *Target*, which both follow the *Four Phase Handshake*. The initiator will randomly send reads and writes to a random address and uses this random address also as data (also note how a `reinterpret_cast` can be used in TLM). The target will negate the data in the case of a read command. The intention is to show the concept and there is no deeper meaning behind the example. The target is only able to handle one incoming transaction and uses the backpressure mechanism (deferring END_REQ) to stop the initiator from sending new requests. Later we will see how targets could handle multiple incoming transactions.

Note that in general there are several ways to implement solutions that are standard compliant, this is just one way to do it.

In order to check if programmed models are standard compliant, the example uses the *TLM2 Base Protocol Checker* from John Aynsley:

`https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_protocol_checker`

This checker stops the simulation immediately if there is a non-standard-compliant behavior in the current simulation.

Please study the code of the AT initiator and target by following the four phase handshake shown in Figure 1.

2. Return Path Shortcuts

The following three examples are build on top of example 1 by inheritance from initiator or target. The code for example 2 is available here:

`https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_at_2`

Please study the code by following the *Return Path Shortcuts* in Figure 1. Note that the target decides to go the `[2.0]` or the normal path `[1.0]` and the initiator decides to go the path `[2.1]` or `[1.5]`. It is also possible that only one of the two shortcuts is taken.

3. Early Completion

The code for example 3 is available here:

`https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_at_3`

In this example the target ends the transaction by an early completion, which is similar to the `b_transport` of LT coding style. Please study the code by following the *Early Completion* path in Figure 1.

4. Skip END_REQ

The code for example 4 is available here:

`https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_at_4`

This example shows the two ways how the END_REQ phase can be skipped. Please study the code by following the *Skip* END_REQ paths in Figure 1.

5. Backpressure

The code for this example is available here:

`https://github.com/TUK-SCVP/SCVP.artifacts/tree/master/tlm_at_backpressure`

In this example the target has a request buffer for 8 elements. When the buffer is full, the target will apply backpressure to the initiator by deferring END_REQ. Please study the code and observe the output: Transactions will happen out-of-order!