

What are transactors, and how are they used?

Designs today commonly consist of a System on Chip (SoC) which includes embedded software, a CPU, memory, a system bus, various interfaces, and perhaps a special purpose processor like a DSP. The increasing complexity of the system means that the designers need to validate both the hardware and the software running together to be certain that the system is working. This places serious requirements on the performance of the system model and the efficiency and effectiveness of the verification environment.

As a way to address these needs, design teams are moving increasingly to the use of transaction level verification to improve verification performance and develop more powerful verification environments. Transaction level interfaces provide easier means to integrate software with the hardware model. In order to use the system environment to validate the hardware implementation, however, we need a way to integrate the transaction level models with the bit level gate level, RTL or cycle level models. This is where transactors come in.

A transactor is a module with two different APIs: one for the transaction level side and one for the RTL side. It translates the single function calls from the transaction side to sequences of transitions on the RTL side, and vice versa. The primary communication on the transaction side consists of reads and writes to specific addresses in the device under test. A write command is translated into the protocol specific signal sequences required by the signal level interface. When a read command is issued, the transactor issues the read command to the DUT, waits for the data to be ready, and then reads the data and returns it to the calling function.

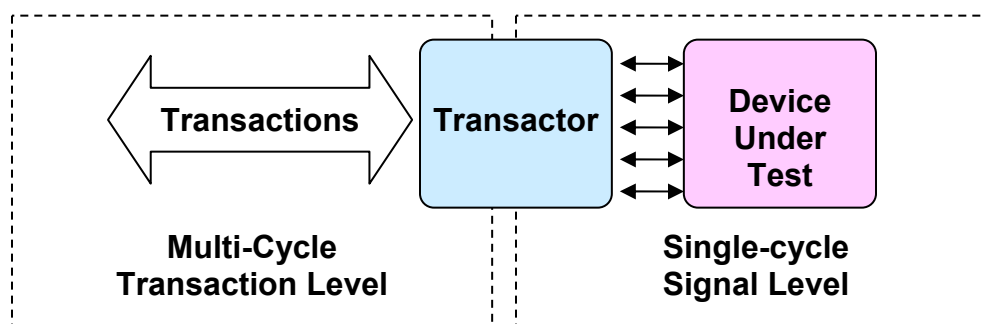


Fig. 1 – Transaction to signal level transition

Transactors also provide a way of isolating the basic communication functions from the actual communication protocol that is being implemented. Handshaking, timing, and other protocol specific features are added by the transactor and are not part of the transaction level communications. This helps to facilitate reuse by allowing components from the transaction side of the system to be reusable across multiple protocol

implementations. Different transactors are needed to implement the different protocols, but the system level components do not need to be re-written. This separation of the functional control from the protocol specific implementation also means that the developers writing the system level side of the environment do not need to become protocol experts.

Transactors can typically be modified easily to integrate with different system level tools or environments. The difficult part of the transactor is the signal level implementation and the state machine that implements the actual interface protocol. The transaction level side can be easily modified to support different APIs or feature sets because those are more abstract commands and also present a smaller set of functions to modify for a new environment.

Carbon provides a series of transactors to translate system level communications to individual signal level protocols. This speeds up the integration of the Carbon compiled cycle level models into the total system model. By using the compiled models in place of the actual RTL implementation, end users are able to realize the performance benefits of the cycle level model and execute their system model faster. Since the Carbon compiled model was generated directly from the actual RTL, however, the system model is still based on the actual implementation of the model and not a hand-coded implementation that may or may not actually match. This gives the design team both performance and confidence.

```
// carbonXAhbBurstRead (nbits, beats, incr, lock, address, *rdata);  
carbonXAhbBurstRead(32, 4, 1, 0, 0x500, &burstBuffer[0]);
```

Fig 2 - Example of a Transactor call

In the example above, a burst of data is read in a single transaction level call. Configuration information is passed in the call, including the data size (nbits) and number of data elements (beats). The address to read from is specified and a pointer to a buffer is passed into which the results are stored. Depending on the size of the burst read, this single call can represent hundreds of individual signal transitions. The single call is both more efficient to code and much easier to read. By using a vendor-supplied transactor, the user can spend their time on their actual design task instead of having to create their own transaction level interface routines. The transactor also provides a way to check to make sure that the actual bus interface has been implemented correctly in the device under test.

Transactors are available from a number of 3rd party vendors who specialize in the design and verification of particular protocols. Carbon provides a core set of transactors for all popular protocols including AMBA, AXI, MII, GMII, XGMII, PCI, PCI-Express, SATA, SPI3, SPI4, SRIO, and many others. New transactors are being developed on a regular basis.

System level models provide a fast path to high performance system exploration and validation. To confirm the results and close the design loop, however, requires being able to plug the design implementation models back into the system model. This requires a way of mixing the transaction level system model with the signal level RTL or equivalent cycle level model. Transactors provide the interface that enables this integration by translating the transaction level calls to the protocol specific signal level activity sequences. The designer now has the ability to mix and match system level transaction models with implementation level models, switching levels to provide the highest performance along with the implementation details when needed for implementation validation and debug. Carbon provides a complete set of transactors to ease the integration effort for all popular protocols, reducing development effort for your validation environment and letting you focus on the actual validation itself.