Visualizing Debugging Using Transaction Explorer In SoC System Verification

Alicia Strang, Robert C. Carden IV
Marvell Semiconductor, Inc. CA

MOVING FORWARD FASTER®

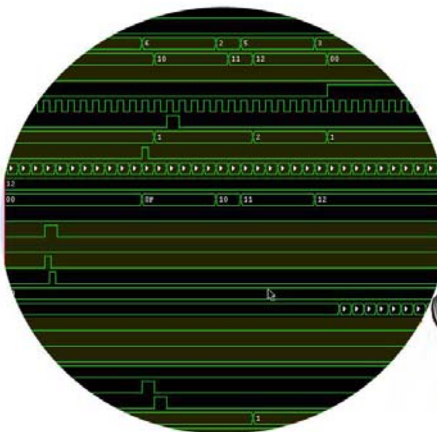
Overview

- The traditional debug process requires viewing large swaths of waveforms at one time.
- As semiconductor designs continuously grow in complexity, using a waveform viewer to debug a SoC system is like examining a forest from an ant's view: this becomes impractical.
- To move us from an ant's view to an eagle's view of the forest, data flow and bus transactions must be presented in way that will empower an understanding of the big picture of a system without unnecessarily cluttering the debug environment.

Too Much Or Not Enough Information?

The Traditional Debug Process

Using a waveform viewer to debug a SoC system is like examining a forest from an ant's view: this becomes impractical.

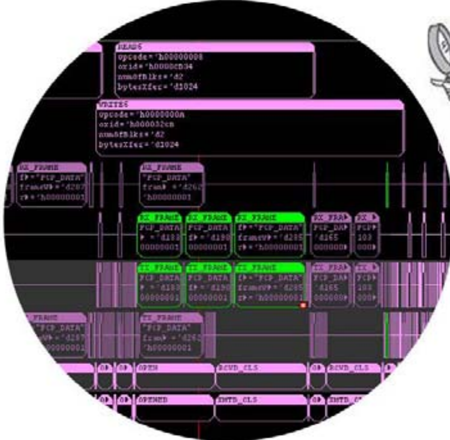


Traditional test benches typically will generate complex text based log file and huge waveform dumps. Waveform viewer implementations, such as Simvision and VirSim adhere to logic analyzer paradigm where information presentation is necessarily primitive, these files are of a raw format that is impractical to be used effectively for large systems.

Simply, the sequential log file form and the underlying massively parallel DUT presents cognitive dissonance; it is a distraction from the problem at hand.

Use High level of abstraction

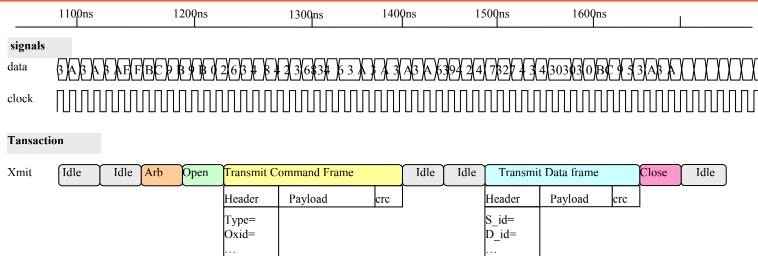
To move us from an ant's view to an eagle's view of the forest, data flow and bus transactions must be presented in way that will empower an understanding of the big picture of a system without unnecessarily cluttering the debug environment.



Cadence Transaction Explorer (TXE) enables advanced transaction viewing, navigating and organizing the waveforms.

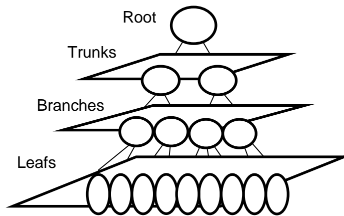
Use high level of abstraction – transaction based - linked hierarchical transactions instead of signals to track data flow through a SOC, debugging and signal exploration is simplified. The presentation of data at only the relevant layer of abstraction allows the engineer to quickly locate the problem area, find the fundamental problem.

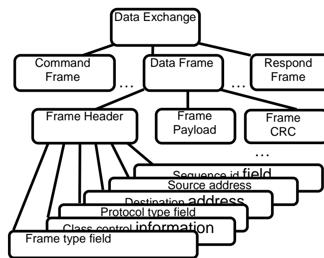
Getting Started to Record Transactions



To reduce the debugging environment clutter, it is practical to collapse data busses into a textual block to encode control signals into human readable text, and displayed bits as status messages. A few blocks of text can be used to replaces huge amounts of low level binary waveforms. The next step in the simulation is to move from the visualizing of absolute waveforms to context based transactions. These transactions can then be used to generate one or more higher level transactions. Thus a hierarchical tree of transactions can be constructed and traversed to view data flows, system configuration, and even inter-chip communications.

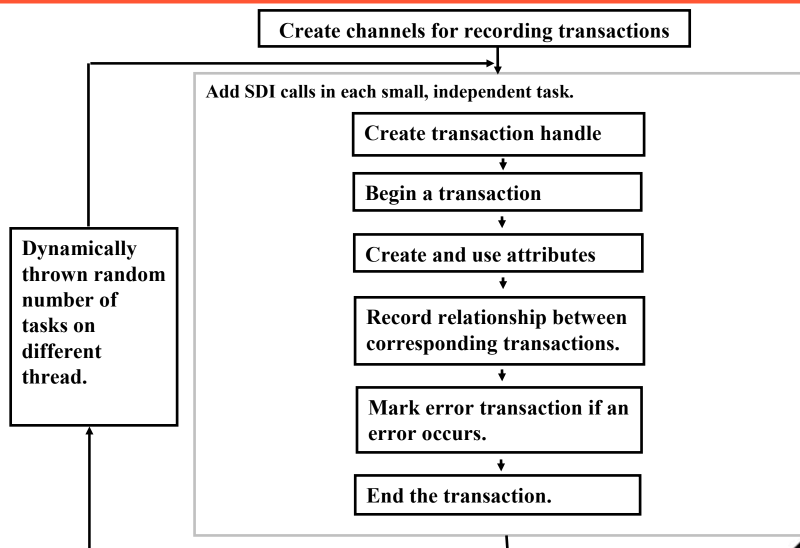
Hierarchical Transactions





- Transactions can be hierarchical
- Transactions can be linked together
- Linked hierarchical transaction methodology involves three dimensions:
  - Transaction hierarchies
  - Time
  - Stages of data flow (format changes)

Transaction Construction Flow Chart

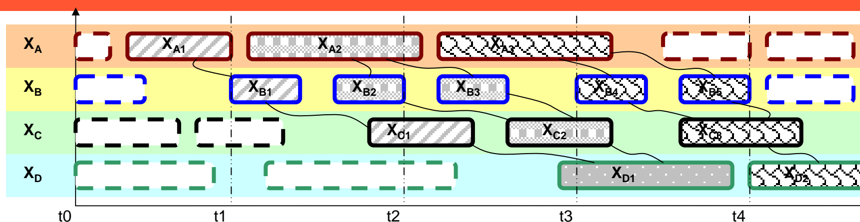


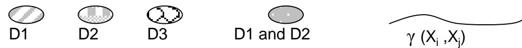
Methodology Implementation

Use high level of abstraction - transaction:

- **Record all concurrent tasks as overlapping transactions.**
- **Mark an error transactions.**
- **Debugging by viewing the overlapping transactions in waveform.**
- **Locate error by tracing the transaction links.**

Linking Across Functional Units



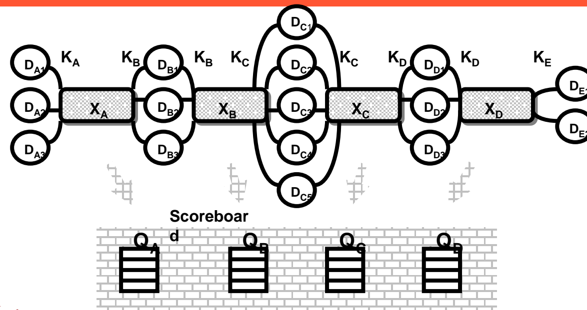


•Each transaction captures the processing done by indicated hardware and the data transformations at each stage

•A unique identifier at each stage tracks each logical entity through all data transformations

•Trace logical entities across hardware functional units, explicitly exposes causality

Process Diagram



Convert K_B to K_C :

Convert data from format B to format C. This process is recorded as Transaction B which has a handle B.

Set $\sigma_{BC} = (H_B, K_C)$ pair :

Set the transaction handle B associated with the identifier of new data format C.

Save σ_{BC} into Q_B :

Save the mapping pair into a queue on the scoreboard.

Search for σ_{AB} in Q_A using K_B as index:

Search the previous mapping queue on the scoreboard, using identifier of the old data format B as index to get the mapping to previous transaction handle A.

if (H_A, K_B) found then invoke $\gamma(X_A, X_B)$:

If a match is found, use the previous transaction handle A and current handle B to set up a causality link, where A is a predecessor and B is a successor.

else set error transaction associated with X_B :

Otherwise, if no match is found, an error is detected. Mark current transaction as an error transaction.

Transaction Usage Coding Example

```
//fiber and transaction handle declaration
integer   fiberHandle;
integer   xactHandle;

//create a stream
fiberHandle=$sdi_create_fiber(fiberName);

//+++++++++++++++++++++++
//task of command execution
//+++++++++++++++++++++++
task execute_command (cmmand cmdT)
{
  string cmdName  = cmdT.getName();
//begin parallel transaction
  xactHandle = $sdi_trans("Begin_No_Parent", FiberHandle, cmdName, cmdName,"");
//set transaction attributes
  $sdi_set_attribute( xactHandle, "OpCode",    cmdT.getOpCode(),   "h");
  $sdi_set_attribute( xactHandle, "oxid",      cmdT.getOxid(),     "h");
  $sdi_set_attribute( xactHandle, "numOfBlks", cmdT.getNumOfBlks(),"d");
  $sdi_set_attribute( xactHandle, "bytesXfer", cmdT.getBytesXfer(),"d");
  $sdi_set_attribute( xactHandle, "lba",       cmdT.getLBA(),      "h");
   …
```

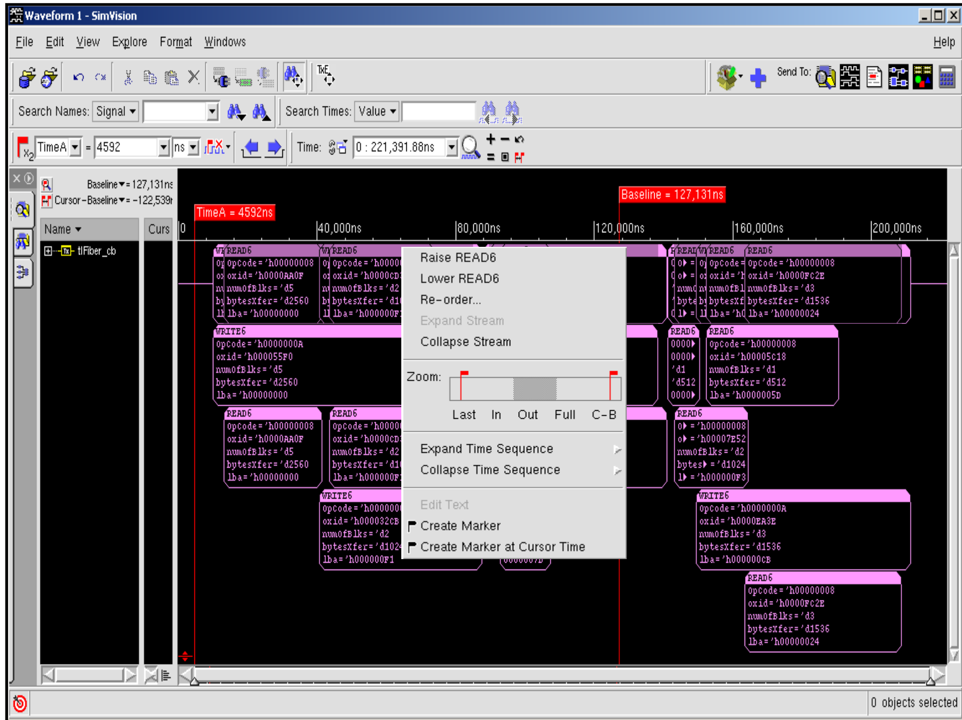
The Coding Example - continue

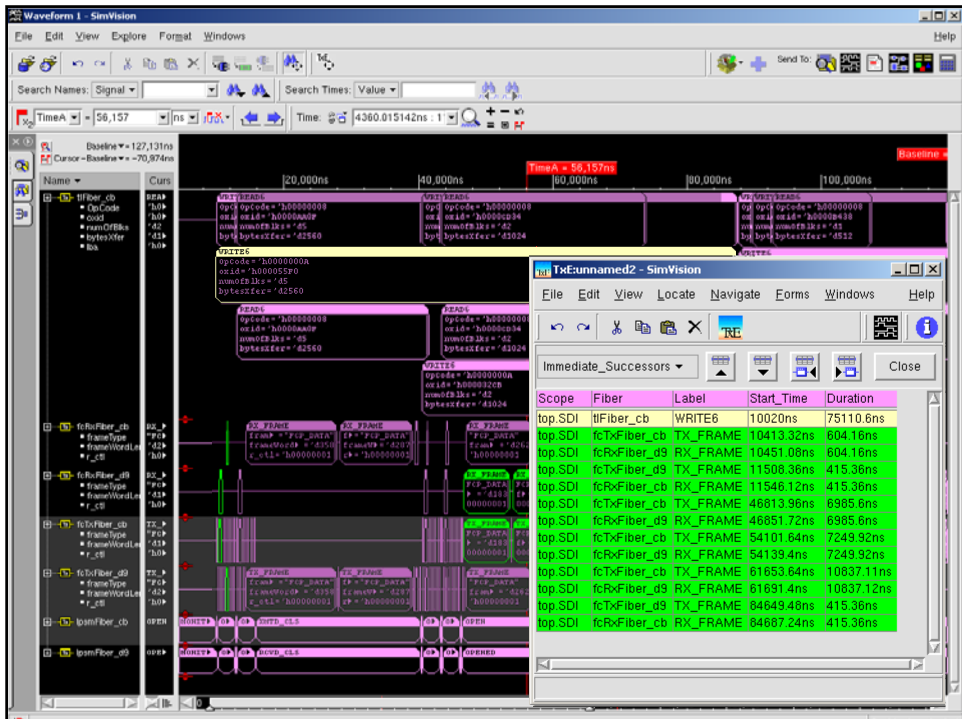
```
//command execution code goes here
   …
  frameR.print_header_info();
  frameR.set_header_attribute(fcRxXactH);

//link transactions
  $sdi_link_trans(scsiCmdH[frameR.getOxid()],fcRxXactHandle);
   …
//mark errors
   if(mismatch)
      xactHandle = $sdi_trans("Error", FiberHandle);
  …
//end transaction
  $sdi_end_transaction(XactHandle);
}

//++++++++++++++++++++++++++++++++++++++++++++++++
//random commands are executed at the same time
//++++++++++++++++++++++++++++++++++++++++++++++++
cmdNumber = random();
while(cmdNumber)
 {
    fork execute_command (nextCmd);  join none
    suspend_thread();
    cmdNumber--;
 }
```

Transaction (Event) Sequences





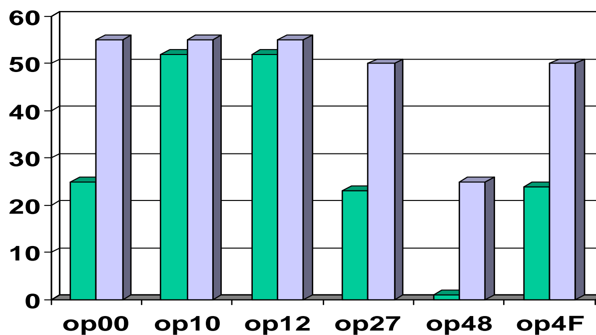
Scope	Fiber	Label	Start_Time	Duration
top.SDI	tlFiber_cb	WRITE6	10020ns	75110.6ns
top.SDI	fcTxFiber_cb	TX_FRAME	10413.32ns	604.16ns
top.SDI	fcRxFiber_d9	RX_FRAME	10451.08ns	604.16ns
top.SDI	fcTxFiber_d9	TX_FRAME	11508.36ns	415.36ns
top.SDI	fcRxFiber_cb	RX_FRAME	11546.12ns	415.36ns
top.SDI	fcTxFiber_cb	TX_FRAME	46813.96ns	6985.6ns
top.SDI	fcRxFiber_d9	RX_FRAME	46851.72ns	6985.6ns
top.SDI	fcTxFiber_d9	TX_FRAME	54101.64ns	7249.92ns
top.SDI	fcRxFiber_cb	RX_FRAME	54139.4ns	7249.92ns
top.SDI	fcTxFiber_cb	TX_FRAME	61653.64ns	10837.11ns
top.SDI	fcRxFiber_d9	RX_FRAME	61691.4ns	10837.12ns
top.SDI	fcTxFiber_d9	TX_FRAME	84649.48ns	415.36ns
top.SDI	fcRxFiber_cb	RX_FRAME	84687.24ns	415.36ns

Transaction-Based Functional Coverage

Exporting Coverage Data to Excel

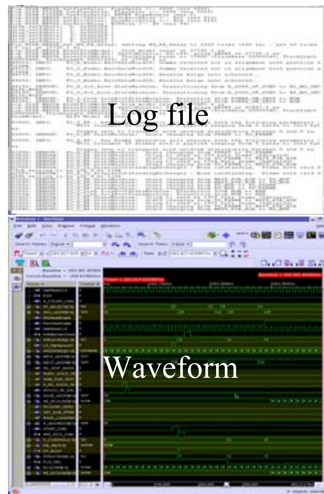
Use TxE to export result to Microsoft Excel:

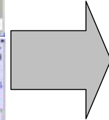
- **Write the result table to comma separated values (csv) file**
- **Import the csv file to Excel**

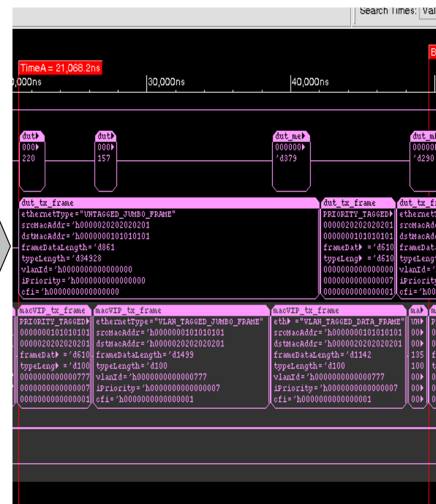


MARVELL

From Ant's View To Eagle's View







Conclusion

- **Higher Levels of Abstraction take the first step in the revolutionary process.**
- **Error Tracing Across Functional Units**
- **Going Down Hierarchy for More Detail**
- **Abstract level display making viewing hardware function in big picture possible. This opens a new world to the EDA company to create performance analysis tool function, which can help architects in the early design stage.**

What's next?

We suggest:

- Enhanced graphical representations
- Error flag propagation through multiple levels
- Advanced waveform and transaction browsing via multiple windows and layouts.
- Transaction color coding based on data and customer specifications.