

Data flow within an open architecture tester

Maurizio Gavardoni
NPTest
150 Baytech Drive
San Jose, California 95134
mgavardoni@nptest.com

Abstract

An open architecture tester allows a third party to develop its own instrument. Such a tester must be open in the sense that it needs to be able to integrate this instrument with minimal or no impact on the overall performance and cost. It is therefore fundamental that the data flow and synchronization among all the instruments in an open architecture tester be efficient. One of these two key topics, the data flow, is discussed in this paper. Some possible solutions that optimize the data flow are presented and one among them is given in full detail. Synchronization is a very wide topic whose discussion could be the subject for a separate paper.

1 Introduction

The optimization of the data flow in an open architecture tester is the key factor for the success of the tester itself. The flow should be optimized in order to address the following design requirements: first, the tester must be able to integrate a third party instrument. This integration must happen without having any impact on the performance of the tester itself and without constraining the design of such an instrument to an unacceptable level of complexity. A second basic requirement is that the data transfer from the CPU down to the Pin Electronic modules and the synchronization of data among instruments in the tester must be optimized for speed and latency. Then, the data flow should be able to manage many sites with several independent processes running on these sites (multi-thread environment). Last of all, the data flow should allow efficient loading and unloading of memories.

All these requirements can be met with several possible system configurations. One of these is a star connection, where a router manages the communication between the CPU and all the instruments individually through a high speed link. This solution may have minimal latency in the data flow if developed with a customized

protocol, but may require a very complex router structure.

Another solution is to connect all the instruments in a ring and have a special instrument inside this ring that interfaces with the CPU. This solution requires a lower level of complexity compared to the star configuration and, therefore, a lower cost, but may present some latency issues.

As explained in more detail later, the data flow in a ring topology is what has been developed and is the subject of this paper. The following sections explain how the ring can meet all the data flow requirements of an open architecture tester. Features of this data flow are presented, such as how to manage with sites and threads, how to perform Direct Memory Access (DMA), how to efficiently load a data pattern into a memory, how to allocate resources and functions for each instrument and finally how to be able to connect all the instruments together into a ring with any cable length.

2 Background

One of the most traditional approaches for data flow in a tester has been to connect all the instruments through a common back-plane using a shared data bus. This kind of approach is no longer suitable in an open architecture tester where multiple sites and data speed are among the most important requirements. Clearly, extending the back-plane and the common data bus to a high number of sites will result in serious speed and bandwidth limitations.

As already mentioned, the star and ring topologies in their various interpretations seem to best fit all the data flow requirements in an open architecture tester. Both configurations use point to point connections between two instruments, thus moving away from the concept of a shared data bus. The next two

figures show a star serial and a ring parallel topologies.

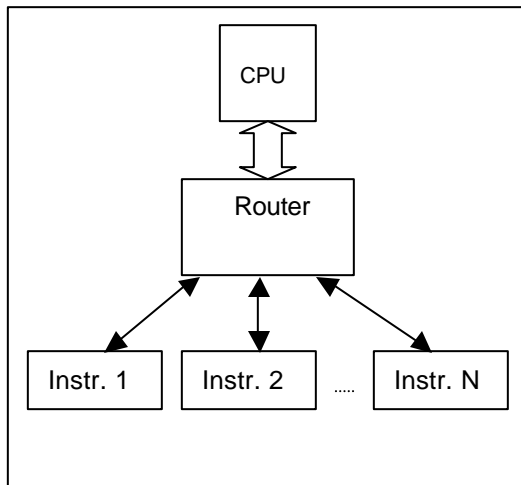


Figure 1: Data flow in star serial topology

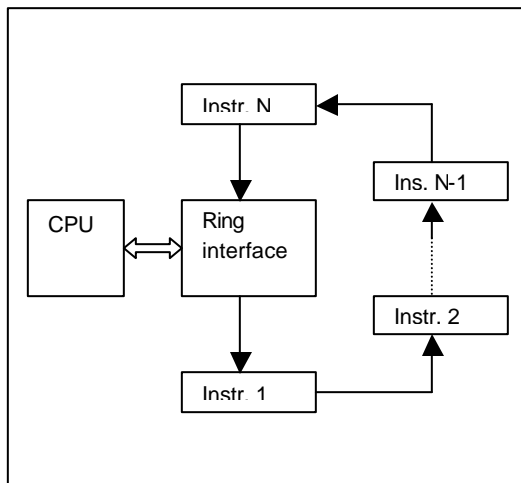


Figure 2: Data flow in a ring topology

The star serial topology may use a high speed serial link to connect each instrument with the router; the serial to parallel and parallel to serial data conversion can be performed inside modern and powerful FPGAs that integrate ser-des devices. The star serial topology may present minimal latency if the messaging traffic is low relative to system total bandwidth and a customized protocol is used, but the router can become a source of latency when the messaging traffic increases. Other drawbacks are the complexity of the design of the router and its likely elevated cost. If, instead of using a customized protocol, the protocol TCP/IP is used, then the latency increases because of the error correction mechanism. The increase in the latency may not be a problem from the data flow

point of view, but may be a serious limitation for the instrument synchronization.

Another solution that is conceptually similar to the serial star is to use several stars, one for each site. Each star controls all the instruments relevant to that site and serves as a site controller. All the site controllers may then be connected through a high speed serial link to a common hub that then interfaces with the PC.

The ring parallel topology connects all the instruments in a ring and uses a parallel bus to transfer data among instruments.

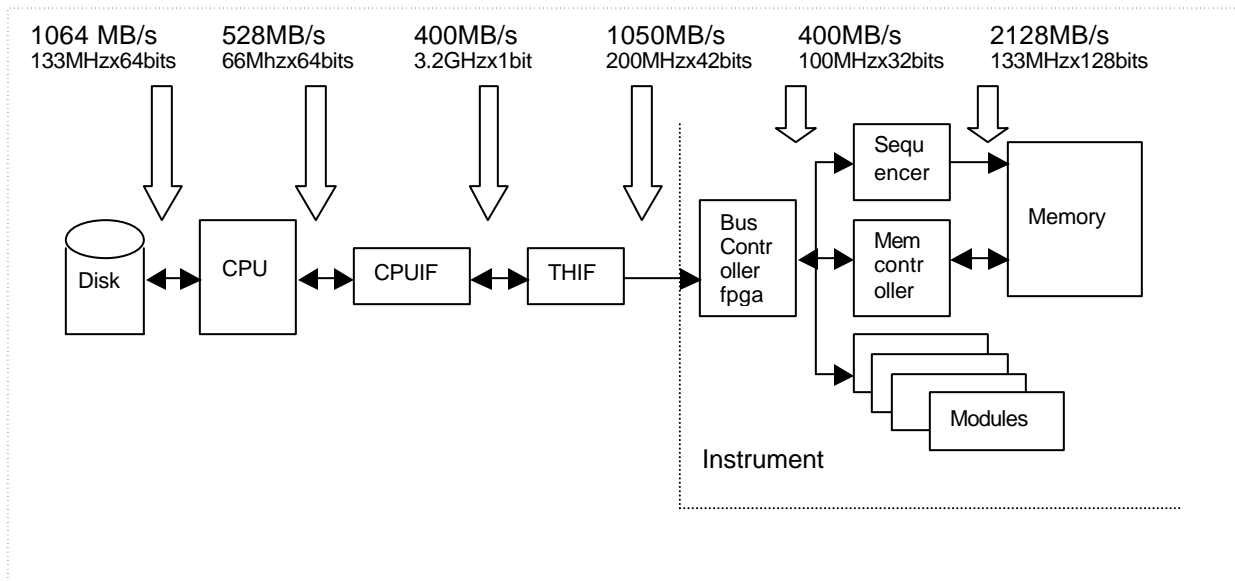
The ring has more latency than the customized star serial, but it is less complicated and does not present any cable congestion. Furthermore the latency, even though it can be significant, will not be a show stopper if it is kept to a known and fixed value. One big advantage of this structure is that the same bus that goes from one instrument to another can carry both the information relevant to the data flow and the synchronization.

The characteristics of this ring topology from the data flow perspective are described in the next sections.

3 Data flow in a ring system topology

As already mentioned, all the instruments are connected together in a ring with a parallel bus. There is a special instrument called the Test Head Interface (THIF) that is part of the ring and manages all the communication between the ring and the CPU. There is also another interface, called the CPU interface (CPUIF), that interfaces directly with both the PCI bus and the THIF. The THIF feeds the first instrument and receives the data from the last instrument in the ring.

Figure 3 shows the structure of the data flow in a ring topology from the CPU to the first instrument's data pattern memory. The Bus Controller inside each instrument decodes the data relevant to that particular instrument and passes all the other data along the ring. It also manages the communication with the data sequencer, the memory controller and with all the instrument's modules (which could be either a digital or analog Pin Electronic Card). The communication between the CPU and the CPUIF is performed by a PCI standard bus. Then the CPUIF and the ring THIF communicate through the special instrument THIF by a serial



link at 3.2 Gbit/s. All the instruments in the ring are linked by a 42-bit parallel bus that runs at 200 MHz. The Bus Controller then interfaces with the sequencer by using a 32-bit parallel bus running at 100 MHz.

3.1 Threads and sites

Each Device Under Test can be thought of as a site. A thread is an independent software process running on the tester. Multiple threads involving different sites can be run simultaneously on the tester. The CPUIF is the only part of the system which is aware of a thread switch: all the instruments in the ring, including the THIF are transparent to a thread switch. The CPUIF has a set of registers that allow it to manage, for each thread, a certain number, S , of sites: this set of registers is called the site-mask. The Bus Controller on each instruments receives an updated site-mask on each thread switch, which maps the instruments resources to different sites. Refer also the next paragraph for the concept of resources within an instrument.

Each time that there is a thread switch, the CPUIF is responsible to update the current resource code and site mask if needed. To implement this, each thread's register set, including the resource and site mask registers, start at a separate base address within the memory address space.

3.2 Allocation of resources and functions

Each instrument in the tester is capable of driving a certain number, N , of pins, each representing a resource for the tester. If M instruments are deployed in the tester, then the overall number of resources available will be $N * M$. All these resources are then divided into sites as explained in the previous sub-section 3.1. It is not required that the N resources of a given instrument belong to the same site, rather they can be spread out among several sites.

The system software dynamically allocates in its memory space all the N resources of a given instrument for a given tester configuration. In other words, the starting address value for the N resources of an instrument is flexible and can change depending upon the configuration of the tester at a given time.

The same concept also applies for the functions of each instrument. The function space of a given instrument is dynamically allocated in the memory space independently from how the resources are allocated.

This gives the open architecture tester extreme flexibility so that every instrument can be plugged into any slot of the Test Head with no limitation. The THIF uses a slow speed and, therefore, low cost serial star link (called the System Monitoring link) to connect to all the instruments in order to read the configuration eeprom of each instrument in the tester. This information is used by the software to recognize the type of each instrument (digital, analog, DPS etc.) and, therefore, the entire tester

configuration at a given time and then allocate dynamically the resource space.

The following tables show two examples where three instruments plus the THIF belong to the ring. Supposing that each instrument can manage up to 64 resources, the tables show how the tester can be configured in two different ways (actually any configuration up to the limit of the memory space can be imagined) and the instruments can use any slot number.

Slot number in the Test Head	Type of instrument	Resource space
0	THIF	0x0000 to 0x003F
1	DIGITAL	0x0040 to 0x007F
2	ANALOG	0x0080 to 0x00BF
3	DPS	0x00C0 to 0x00FF

Table 1: First possible system configuration

Slot number in the Test Head	Type of instrument	Resource space
0	DPS	0x0040 to 0x007F
1	ANALOG	0x0000 to 0x003F
2	THIF	0x0100 to 0x013F
3	DIGITAL	0x0200 to 0x023F

Table 2: Second possible system configuration

3.3 Pattern loading

Loading a data pattern into the memory of the sequencer requires millions of write cycles and, therefore, lots of system bandwidth. In order to make the pattern loading more efficient a special code to compress data is used in the communication between the CPU and the THIF. This reduces the amount of data sent to the THIF which then has to decompress the data and send them onto the ring. As shown in figure

3 the serial link between CPUIF and THIF is the slowest path between the CPU and the memory to be loaded. This is why the pattern compression is applied at this stage of the communication.

3.4 Direct Memory Access (DMA)

Reading a large chunk of memory in the tester, such as the capture memory after one test has been executed, is an operation that may take lots of bandwidth and may limit the performance of the tester, especially if the next test is being executed during this read.

The way the data flow has been designed in this ring topology minimizes the number of operations involved during a DMA between the CPU and the THIF. The CPU can issue a special request for DMA by providing only the memory start address and the number of locations to be read. The actual DMA expansion is then left to the THIF allow it to best optimize the flow traffic on the ring. A functional test can be performed while a DMA read is going on; it is the task of the THIF to manage the data flow in an efficient way.

3.5 Data flow control

As shown in Figure 3 the data flow in this open architecture tester presents different speeds within different areas of the tester. Some areas, such as the modules inside each instrument or the System Monitor serial link, are definitely much slower than the other communication buses. This implies that when these areas are accessed, the pace of the communication between the CPU and the Test Head may slow down to properly manage this kind of communication. Of course this approach is not efficient, especially if the software is the only tool managing this pace. An example of this situation is when the software reads the serial configuration eeprom of each instrument to identify its kind after power up.

To make all this efficient and transparent for the software, the hardware implements a flow control mechanism at various stages of the communication. The concept of flow control is illustrated by considering this example. When a receiving instrument has received inside its internal FIFO memory a number of data up to, for instance, 75% of its maximum capability, then a stop request is issued toward the

transmitting instrument which will stop sending data. When the data inside the FIFO are processed and the overall number goes down to, for instance, 70%, then the receiving instrument issues a resume request toward the transmitter which will resume sending data.

The two main data flow controls in this tester are in the communication between CPUIF and THIF and from the THIF and the Bus Controller FPGA of each instrument. The THIF assures that these two flows are kept independent of each other for performance reasons.

3.6 Error correction protocol

The serial link communication at 3.2 Gbit/s between CPUIF and THIF requires the use of a cable that might be several feet long. In fact, the CPUIF card is plugged into a PCI slot, while the THIF is one of the instruments in the ring and therefore is plugged into one of the slots of the Test Head. Besides this, the serial to parallel and parallel to serial data conversions may make this communication prone to possible errors. An error correction protocol is used by both ends simultaneously; the protocol operates symmetrically and independently on the uplink and downlink and, by using CRCs, it ensures that it is very improbable that the data has been corrupted in transmission. The data is divided into several packages or frame each of those having a frame marker associated with. If one package gets corrupted, the receiving end will ask for the re-transmission of the entire package.

3.7 Bus timing calibration

The parallel bus that connects all the instruments in a ring is 42-bit wide and runs at 200 MHz. Such a speed poses a challenge on the design of the cables: cable length and skew become critical factors to meet the proper timing requirement at the input of the Bus Controller when the available budget is only a multiple of 5 ns.

The solution presented here with the aid of the following picture overcomes this limitation and allows theoretical deployment of cables with any length.

Each Bus Controller in the ring (one for each instrument) uses at least three pipe stages to capture the incoming 42-bit word from the ring.

Each pipe stage is clocked with a variable clock whose phase can be controlled by programming the proper value into a Digital Clock Manager (DCM). The DCM is a feature available in modern FPGAs that allows the phase shift of its clock by using an internal phase lock loop.

Consider the example shown in the figure 4 where three pipe stages are used for this purpose: each pipe stage is clocked by the output of individual DCMs whose input clock is the main clock running at 200 MHz. Each DCM can have its phase shift programmed independently.

When the system is powered up and configured, a special software program is run in order to calibrate the entire bus by finding the proper phase value for each pipe stage of the Bus Controller and for each instrument in the ring. A generic PRBS data pattern can be used by this program to transmit data on the bus and validate the bus timing

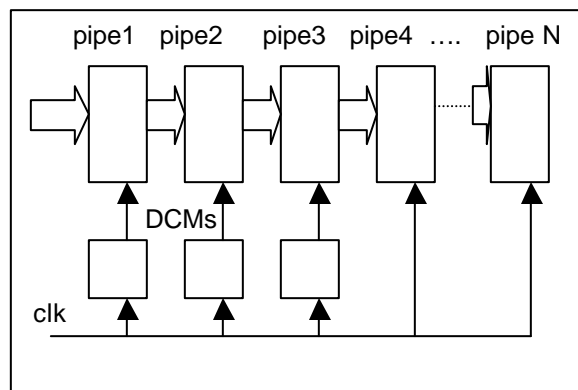


Figure 4: Bus timing calibration

4 FPGAs challenge

In order to minimize the overall cost of the tester, the latency and the real estate, most of the data flow functionality is implemented inside powerful FPGAs. This also allows the system to be more flexible since these devices are re-programmable.

This approach pushes the limit at which these devices can be used: all the functions described above that make the data flow efficient increase the complexity of each FPGA in the tester, pushing the number of logic gates beyond one million. Furthermore even the speed at which some buses run, like the 200 MHz ring bus, pushes the performance of these devices very close to the limit of the current technology.

Despite these technical challenges though, this approach allows the reduction of the cost of the tester since just one device, the FPGA, includes all the functions that could have been performed by deploying lots of devices, and therefore reducing the part inventory. Furthermore the FPGA provides more flexibility since the device is re-programmable.

5 Experimental data

The maximum ring latency is 1024 clocks at 200 MHz, therefore 5.12 us.

Assuming that each instrument can pass information along the ring within 4 pipes, then the maximum number of instruments supported is 256.

The overall number of resources (or pins) that can be addressed is 32K: each instrument can support up to 128 resources.

The overall number of threads is 256, each of those can control up to 1024 sites.

CPU resource and data writes normally operate as fast as the PCI interface allows. Reads, however, take approximately 30 ns * total number of instruments + instrument-specific delays per addressed instrument.

The following table shows some maximum latencies for different number of instruments. Data relevant to synchronization are not reported because the topic is outside the purpose of this paper.

INSTRUMENTS	CPU WRITE	CPU READ OF 1 INSTRUMENT	CPU READ ALL INSTRUMENTS
5	811 ns	1277 ns	2081 ns
10	961	1427	3236
20	1261	1727	5546
50	2161	2627	12476
100	3661	4127	24026
200	6661	7127	47126
250	8161	8627	58676

6 Conclusion

The optimization of the data flow in an open architecture tester is a critical point in building an efficient tester with high performance.

Summarizing the key requests discussed in the introduction, the data flow should be able to:

- Integrate a third party instrument into the tester
- Manage many sites with several independent processes running on these sites
- Load and unload memories efficiently
- Minimize the latency
- Allow an efficient synchronization among instruments

All these requirements have been met with a ring topology. Advantages of the ring topology are:

- The structure is relatively simple and, therefore, maintainable
- The ring does not present cable congestion
- Data flow and synchronization share the same data bus
- The latency may not be minimal if lots of instruments are deployed in the ring, but is fixed for a given system configuration.
- The ring bus is fast (200 MHz, 42-bit wide)

Almost all the data flow related features are implemented inside powerful FPGAs: this reduces the cost of the tester and therefore provide a better return of investment. Furthermore it makes the tester be more easily maintainable and flexible due to the re-programmability of these devices.

7 Acknowledgements

Thanks to everybody in NPTest for their own contribution to make this happen.

References

- [1] H. Hall - W. Hall, High-Speed Digital System Design, Hardcover
- [2] H. Johnson – M. Graham, High-Speed Digital Design, Hardcover
- [3] J. Hui, Success By Learning, Mass Market Paperback