A New, Flexible Sequencer Architecture for Testing Complex Serial Bit Streams

Based on a generic model of serial communication systems, this architecture dramatically reduces the time needed to program functional and in-circuit tests for devices with serial interfaces. It is implemented in a new Serial Test Card and Serial Test Language for the HP 3070 family of board test systems.

by Robert E. McAuliffe, James L. Benson, and Christopher B. Cain

As serial bit streams become more prevalent in electronic products, the need for high-quality, thorough tests for those products increases dramatically.[1] Traditional automatic test equipment (ATE) architectures have limitations that make it extremely difficult (if not impossible) to write such tests. In this paper we discuss those limitations and describe a new sequencer architecture specifically designed to address the challenges of serial bit stream testing.

The architecture has been implemented as an enhancement to the HP 3070 family of board testers and has been used to emulate many challenging serial protocols and formats, including: ISDN S- and U-interfaces, I^2C, HDLC control channels, generic 64-kbit/s streams, IOM-2, ST-Bus, and various time division multiplexed (TDM) backplanes. Customers using the architecture have experienced dramatic reductions (up to 25×) in test development time, as well as significant increases (8× or more) in test throughput.

We assume the reader has at least some knowledge of manufacturing test procedures and equipment. A basic knowledge of telecommunications concepts may also prove useful, as many of the examples given are related to telecomm applications.

Throughout the paper we will refer to a *device under test* (DUT). In general the discussion will be in the context of functional board testing, in which case the DUT would be a complete printed circuit assembly. In this time of rapid technological changes, however, functionality that once required entire boards is now implemented as small clusters of components or as single integrated circuits. We will therefore use the term DUT to refer to whatever is being tested, be it an IC, a cluster of components, a board, or a complete system.

First we give a brief overview of traditional ATE sequencers and their shortcomings, and then discuss in more detail some of the special challenges of serial bit stream testing. We will show that many test development problems are caused by a fundamental mismatch between the ATE capabilities and the features of DUTs with serial interfaces.

Next we introduce a generic model of serial communication systems. This is essentially a definition of the general characteristics shared by all serial bit streams. Using this model, we were able to develop a test sequencer architecture more closely matched to the characteristics of serial bit streams and the DUTs that use them.

We then describe the architecture as implemented in the HP 3070 board test family. The architecture of the *Serial Test Card* (STC) and the *Serial Test Language* (STL) are described in these sections.

Finally, we present several case studies showing how the STC solves real-life testing problems.

Evolution of ATE Sequencers

Traditional ATE systems feature a test pattern sequencer capable of driving and receiving many simultaneous digital signals. Sequencers of this type first appeared over a decade ago. Early versions of these sequencers excelled at testing SSI/MSI components and simple circuit boards, but had difficulty with microprocessors and other VLSI components.[2]

In response to the test problems posed by microprocessors, ATE manufacturers enhanced their sequencers. New features like formattable pins, algorithmic pattern generation, memory emulation, and bus emulation were added to make the test sequencers a better match to these microprocessor-oriented DUTs.

Today, DUTs embody faster and more powerful microprocessors, concurrent processing technologies, serial communication channels, mixed signal functions, and a variety of custom circuitry (such as ASICs and FPGAs). Each of these characteristics brings with it challenges for the test engineer, but the widest gap between the DUT and traditional ATE seems to be in the area of serial communication testing and its associated concurrent processing technology. The single-sequencer, massively parallel architecture of traditional ATE is not suited to the special problems presented by these DUT characteristics.

 © Hewlett-Packard Company 1995

Our goal was to design a sequencer architecture better matched to the special test requirements of serial-oriented DUTs. As a first step toward that goal, we surveyed a large number of DUTs and serial protocols to identify specific characteristics that make them difficult or impossible to test with a traditional ATE system.

Characteristics of Serial DUTs

Serial-oriented DUTs have many characteristics in common with many other modern electronic devices. Conversely, they also have many characteristics that are fundamentally (or sometimes subtly) different. Some of the more common features of serial-oriented DUTs are:

- Complex physical (electrical) interfaces
- Multiple interfaces operating at unrelated bit rates
- Nondeterministic bit streams
- Bit streams with embedded clocks
- Hierarchical bit streams.

Complex Physical Interfaces

To maximize utilization of a transmission medium, many serial protocols abandon traditional binary digital formats in favor of three-level or four-level digital interfaces. The ISDN S-interface operates with three logic levels,[3] and the ISDN U-interface (2B1Q) operates with four. Other standard telecomm protocols are similar.

Many of these protocols require the transmitted waveforms to correspond to exactly specified shapes, usually to limit the high-frequency components of the signal to a reasonable level. Traditional sequencers have binary stimulus and response capabilities with programmable high and low logic levels. Some even provide rudimentary slew-rate control, but none are designed to interface directly with complex, nonbinary bit streams.

Multiple Interfaces Operating at Unrelated Bit Rates

Many DUTs contain more than one serial interface. In many cases, each interface is asynchronous with respect to the others (there is no specified alignment between bit centers from one interface to the next). One can sometimes force alignment by running each interface from a common clock, but this technique does not necessarily work with asynchronous protocols (see “Bit Streams with Embedded Clocks,” below). Moreover, it is quite common to have interfaces running at entirely different bit rates.

The only viable testing approach for a traditional sequencer architecture is to apply vectors at a rate equal to the least common multiple of the clock rate of the two interfaces (assuming there are only two interfaces). The number of test vectors required for even simple tests can be formidable using this approach.

One of the authors has personally written a test for an ISDN S-interface device using a traditional sequencer and this “least common multiple” approach. The effort required three months and 13,000 lines of source code (roughly 90,000 compiled test vectors). Even with this huge effort, only a small fraction of device functionality was tested. Such long test development times are simply not acceptable in today’s competitive markets.

Nondeterministic Bit Streams

The response of a DUT to an applied stimulus is not always deterministic, that is, many different “correct” responses to the stimulus are possible. An analog-to-digital converter, for example, will not generally produce exactly the same sequence of digital samples in response to different applications of the same analog stimulus. This does not mean that any one sequence is more correct than any of the others. It simply means a different measurement technique must be applied to the problem.

Some ISDN data link activation procedures include the transfer back and forth of HDLC-like packets of information.† According to these procedures, the packet address is, under some circumstances, generated by a pseudorandom generator on the DUT. This presents no problem if the address generated by the DUT is deterministic in response to a particular initialization sequence. On the other hand, if the address cannot be predicted, the test sequencer must be able to capture whatever address was generated and save it for use later in the test. The authors have not encountered a traditional sequencer with such on-the-fly storage capabilities.

Bit Streams with Embedded Clocks

Some serial interfaces are asynchronous in nature, that is, the clock specifying the bit boundaries is embedded in the bit stream. Traditional sequencers typically sample at predetermined times and are unable to interface properly to such a bit stream.

Embedded clocks can take many different forms. Some protocols guarantee data transitions at regular intervals. Other protocols provide no such guarantee but depend on the transmitter and receiver operating at a prearranged bit rate (asynchronous protocols such as those typically used with RS-232 connections work in this way).

In addition, *framing* information can also be embedded in the bit stream. Asynchronous protocols, for example, mark a frame boundary with a start bit. The HDLC and similar synchronous protocols indicate framing with a special flag pattern, usually eight bits in length. A traditional sequencer may be able to handle such cases if it has sophisticated branching capabilities, but it is very difficult to write a test that can synchronize to a complex framing pattern while simultaneously applying test patterns to other interfaces of the DUT.

Hierarchical Bit Streams

Many serial interfaces contain bit streams within bit streams. We call these *hierarchical* bit streams. A basic-rate ISDN interface is an example of a hierarchical bit stream. A basic frame of this interface consists of 16 bits of B channel data and 2 bits of D channel data.†† If the D channel bits from each frame are extracted and assembled one after the other into a separate serial bit stream, they form an HDLC-like bit stream.

† See references 3 and 4 for a complete description of ISDN and associated activation procedures. Details of HDLC and other bit-oriented protocols can be found in references 5 and 6.

†† This is a simplified description of an ISDN basic rate frame. Actual frame length and content vary depending on which reference point (S, U, etc.) is being considered.

© Hewlett-Packard Company 1995

This situation is very difficult to handle with a traditional sequencer because the *logical* channel the test engineer wants to communicate with is surrounded by a great many other bits of little interest. Complicated subroutines must be written to extract the data of interest from the large frame of bits or insert the data of interest into the large frame of bits. Furthermore, the test engineer must somehow attain bit and frame alignment for both the main frame and the embedded logical frame. This may make the test impossible to implement with a traditional sequencer.

Solutions

For many serial interfaces, the above problems can be solved using custom electronics, ranging from special circuits in the test system fixture to a "hot mockup" of the final system application of the DUT, or by simplifying and eliminating tests so that they can be more readily implemented with a traditional sequencer. We propose a third solution: use of a test sequencer designed specifically to address the special challenges of serial bit stream testing. By using the right tool for the job, the test programmer can develop a thorough, high-quality test quickly and without the need for complex fixture electronics.

Generic Model of Serial Communication Systems

In the last section we discussed the difficulties of serial testing using traditional sequencer architectures. For the most part, these difficulties are caused by an inherent mismatch between the DUT and the tester.[7] In each case, the problems presented could certainly be solved with custom circuitry provided by the customer or by the ATE vendor. However, this approach undermines one of the key advantages of commercial ATE systems: the advantage of being able to use the same tester to test many different DUTs (or many different pieces of the same DUT).

A better approach is to design an architecture that is specific enough to handle the peculiarities of serial testing but general enough to be usable for many different types of serial DUTs and serial protocols. To aid in the definition of such an architecture, we looked deeper into the test problems described in the last section in an effort to understand the fundamental nature of each. This led to the development of our generic model of serial communication systems, described below.

We designed our new sequencer architecture using this model as a guide, so essentially any bit stream compatible with the model is compatible with our architecture. Our particular *implementation* of the architecture was targeted at telecomm applications, so cost/performance trade-offs appropriate to that market have been made. The model (and thus the architecture) is more general and could theoretically be implemented in other ways for other serial test markets.

Definitions

The following terms are used throughout this section:

- *Communication System.* A means of transferring information from one place to another.
- *Serial Communication System.* A communication system that encodes information into digital signals and transfers this digital information from one place to another in a time-serial fashion, that is, the bits of digital information are transferred sequentially one after another according to a prearranged protocol. A serial communication system is typically composed of subparts made up of various bit processors (see below), which process and transform serial communication bit streams (see below).
- *Serial Communication Bit Stream.* A physical transmission path connecting two bit processors (see below) in a serial communication system. Bits are transmitted in a serial fashion, that is, the bits of a message are transmitted sequentially one after another on a common transmission medium. "Serial communication bit stream" will be abbreviated to simply "serial bit stream" or "bit stream" throughout the following discussions.
- *Bit Processor.* A hardware or software device that connects one bit stream to another. A bit processor usually transforms or filters the bit stream in some manner, but can also serve as a *source* or *sink*. A source generates the information to be transmitted over the communication system, and a sink receives and analyzes that information at the other end.

A serial communication system is composed of numerous pieces, each piece defined as either a bit stream or a bit processor. Bit processors serve to connect (transform) one bit stream to another. Or, equivalently, bit streams serve to connect one bit processor to another. Each of these elements—bit streams and bit processors—has certain well-defined properties. These properties are discussed below.

Properties of Bit Streams

Every serial bit stream possesses the following four properties:

- Physical specifications
- Symbol synchronization algorithms
- Framing algorithms
- Logical channel identification (multiplexing).

Physical Specifications. The physical specifications describe the electrical properties of the bit stream, the number of logic levels defined, and any other properties related to the physics of transferring the bit stream from one place to another.

Symbol Synchronization Algorithms. Since a serial bit stream is inherently composed of bits, there must be a way of demarcating the bit boundaries within the bit stream. A device designed to interpret the bit stream would use a symbol synchronization algorithm to locate the bit boundaries. The synchronization property of the bit stream is either e*xplicit* (a dedicated signal path for clocking is provided) or *implicit* (symbol synchronization information is encoded within the serial bit stream†).

Framing Algorithms. A raw serial bit stream is capable of transferring very little information. For example, a binary bit stream can represent only one of two states at any given moment (1 or 0, on or off). However, if a time reference is provided with the bit stream (an indication of when the bit stream "starts"), then bits can be assembled into larger units capable of carrying more information (bytes, words, messages, etc.). This is the purpose of framing: to provide a

† Asynchronous protocols are also considered implicitly clocked. In that case the symbol rate is defined as part of the bit stream symbol synchronization algorithm.

© Hewlett-Packard Company 1995

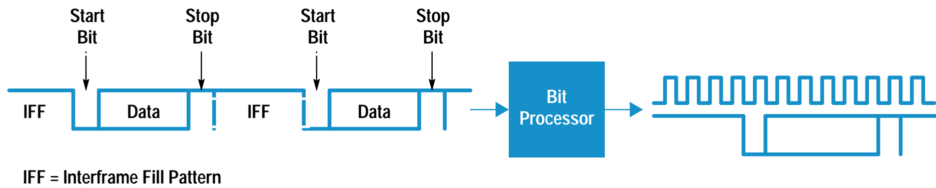

Fig. 1. Processing an RS-232 bit stream.

reference point so the bits of the serial stream can transfer more complex information.

There are two types of framing. Framing can be either *explicit* (a dedicated signal path for framing is provided), or *implicit* (framing information is encoded within the serial bit stream). An implicit framing algorithm may indicate framing using either a *bunched* framing pattern or a *distributed* framing pattern. A bunched pattern is made up of a group of contiguous bits. A distributed pattern is made up of a group of bits interspersed among the data bits.

The time interval between the end of one frame and the beginning of the next is called the *interframe gap*. This gap may be either zero-length (frames are back-to-back) or nonzero-length. If it is nonzero-length, the interframe gap is filled with an *interframe fill (IFF) pattern.*

Logical Channel Identification. Many serial bit streams simultaneously carry more than one independent logical channel of information. We say that these bit streams are *multiplexed* or *hierarchical*. If the independent logic channels are ever to be recovered from a multiplexed bit stream, there must be a means of uniquely identifying each individual logical channel. The multiplexing scheme specifies the method used to multiplex the logical channels and may be categorized as either *explicit* or *implicit*.

With explicit multiplexing, each frame contains a group of information from each multiplexed logical channel. The frame boundary provides a reference for the bit groupings. Time division multiplexed (TDM) highways, such as those found on telecomm line cards, and the ISDN S-bus are examples of explicitly multiplexed bit streams.

With implicit multiplexing, each frame contains a group of information from only one of the multiplexed logical channels. Information from other logical channels may follow in subsequent frames. In this case, logical channel identification is encoded in the bit stream (for example the address field of an HDLC frame or other packetized data.)

Properties of Bit Processors

The job of a bit processor is to convert one bit stream into another according to some specified algorithm. A bit processor therefore has two distinct properties: port definitions and transformation algorithms.

Port Definitions. A bit processor interfaces to bit streams through one or more *ports*. Most bit processors will have an *input port* and an *output port*, but some will have only one or the other.† Each port must have a *data interface* and must of course match the physical specifications of the bit stream. The directional sense of the data interface determines whether a port is an input port or an output port. Data always flows into an input port and out of an output port.

Other requirements are determined by the serial bit stream to which the port attaches. If the bit stream is explicitly clocked (uses an explicit symbol synchronization algorithm), the port must have a *clock interface*. Similarly, if the bit stream uses an explicit framing algorithm, the port must have a *frame synchronization interface*. These interfaces may consist of one or more physical signals that flow either into or out of the port. The implementation details can be determined by studying the bit stream specifications and the transformation algorithm.

Logical channel grouping algorithms do not directly affect the requirements of the port, although they may influence the design of the clock interface. For example, a bit processor designed to demultiplex a logical channel from a multiplexed bit stream could be implemented in one of several ways. One method might use gated clock signals: clock edges only occur when bits from the logical channel of interest are output from the data interface. Another method might use a continuous clock (*all* bits from the original bit stream are output at the data interface) and generate a separate "data valid" signal when bits from the logical channel of interest are present at the output data interface.

Transformation Algorithms. A transformation algorithm simply defines the function to be performed by the bit processor. The bit processor must manipulate the incoming bit stream in whatever manner is necessary to make it conform to the requirements of the outgoing bit stream.

A simple example will help to clarify the functions of bit processors and bit streams.

Applying the Generic Model

Fig. 1 shows an example of the generic model applied to a simple asynchronous communication system. The bit stream on the left is the incoming bit stream. Using conventional terminology, it would be described as an asynchronous bit stream using one start bit and one stop bit, no parity checking, and an eight-bit data field. The definition using the generic model would be something like this:

Physical Specifications:
- Logic levels: 2
- Voltage levels: RS-232-C levels; logic high = –3 volts; logic low = 3 volts

Symbol Synchronization Algorithm:
- Type: Implicit, known symbol rate
- Symbol Rate: 9600 per second

Framing Algorithm:
- Type: Implicit, bunched framing pattern
- Framing Pattern: "10" (at least one stop bit or IFF bit followed by the start bit)

† These are called *sources* or *sinks* and occur at the ends of a communication system. This is where the information sent back and forth in the serial bit stream is ultimately generated or analyzed.

© Hewlett-Packard Company 1995

Interframe Gap: >= 0 symbols
IFF Pattern: “1”

Logical Channel Identification:
Type: Explicit (none really, since there is a single eight-bit data field)

The bit processor receives this bit stream on the data interface of its input port. As shown in Fig. 1, the job of this particular bit processor is to generate a clocked, synchronous version of this bit stream on its output port. Here is the definition of the bit stream output from the bit processor (the bit stream on the right side of the figure):

Physical Specifications:
Logic levels: 2
Voltage levels: Standard TTL levels

Symbol Synchronization Algorithm:
Type: Explicit
Clock Rate: 9600 Hz

Framing Algorithm:
Type: Implicit, bunched framing pattern
Framing Pattern: “10” (at least one stop bit or IFF bit followed by the start bit)
Interframe Gap: >= 0 symbols
IFF Pattern: “1”

Logical Channel Identification:
Type: Explicit

Notice that the framing algorithm and logical channel identification specifications have not changed. The bit processor has merely generated a clock signal synchronized to the incoming bit stream. In this case another bit filter upstream of this one could further transform the bit stream and perhaps extract the data field. Since the output bit stream uses an explicit clock, it is not really necessary to specify the clock rate. Another bit processor that conforms to our generic model should be able to accept the bit stream knowing only that there is an explicit clock signal. Any practical implementation, however, will have a finite clock rate specification, so it is a good idea to specify all such performance requirements as part of the bit stream description.

The definition of the bit processor in our example looks like this:

Input Port:
Data Interface: Required. See bit stream definition for physical requirements.
Clock Interface: Not required, bit stream is implicitly clocked.
Frame Sync Interface: Not required, bit stream is implicitly framed.

Output Port:
Data Interface: Required. Standard TTL output specifications.
Clock Interface: Required, bit stream is explicitly clocked. The interface consists of a single clock output signal meeting standard TTL logic specifications.
Frame Sync Interface: Not required, bit stream is implicitly framed.

Transformation Algorithm:
1. Use a combination of the given baud rate, an internal high-speed clock, and data signal transitions to locate bit centers.
2. Sample the input data at the bit centers and retransmit on the output data interface. Transmit the sampling clock on the output clock interface.

The actual algorithm would probably need to be more sophisticated, but this simple example illustrates the process of mapping a real communication system onto the generic model. A more complex example is given in Fig. 2.

Modularity

The example just given was quite simple but can serve to illustrate an important feature of the model. As mentioned before, another bit processor upstream of the one described might further transform the bit stream, perhaps aligning itself to the framing markers and extracting the data field. The data field could then be passed to yet another bit processor, and so on. We call this feature of the model *modularity*. Using this idea, we can conceptually break up a serial communication system into smaller, simpler pieces consisting of a series of bit processors connected by a series of serial bit streams.

Modularity also aids the processing of hierarchical bit streams. The functions of demultiplexing and logical bit processing can be divided among more than one bit processor.

Hardware Architecture

As we have seen, real-world bit streams and communication systems can be described by carefully specifying each property of the generic model. In other words, the model can effectively mimic any serial communication system. A sequencer architecture that exactly implements the generic model would therefore be easily programmed to test *any* serial bit stream or protocol.

We knew of course that a variety of constraints (the laws of physics, for example) would limit our ability to implement the model exactly. We also knew an abstract model that could not be implemented well enough to solve real test problems would be useless, so we revisited the test problems described earlier. We examined each test problem, looking for issues that might affect the way in which we chose to implement the model.

Complex Physical (Electrical) Interfaces. The model handles this quite easily. One simply describes the characteristics on a sheet of paper. An actual implementation, however, is quite different. We decided early on that it was not feasible to implement hardware compatible with any possible electrical interface! Instead, we chose to implement hardware of several different classes, each class capable of handling a family of related physical bit streams.

Multiple Interfaces Operating at Unrelated Bit Rates. The generic model does not address this issue, although the model can be used to describe each individual interface. We decided that this implied a multichannel architecture in which each channel was capable of operating essentially independently of the others.

© Hewlett-Packard Company 1995

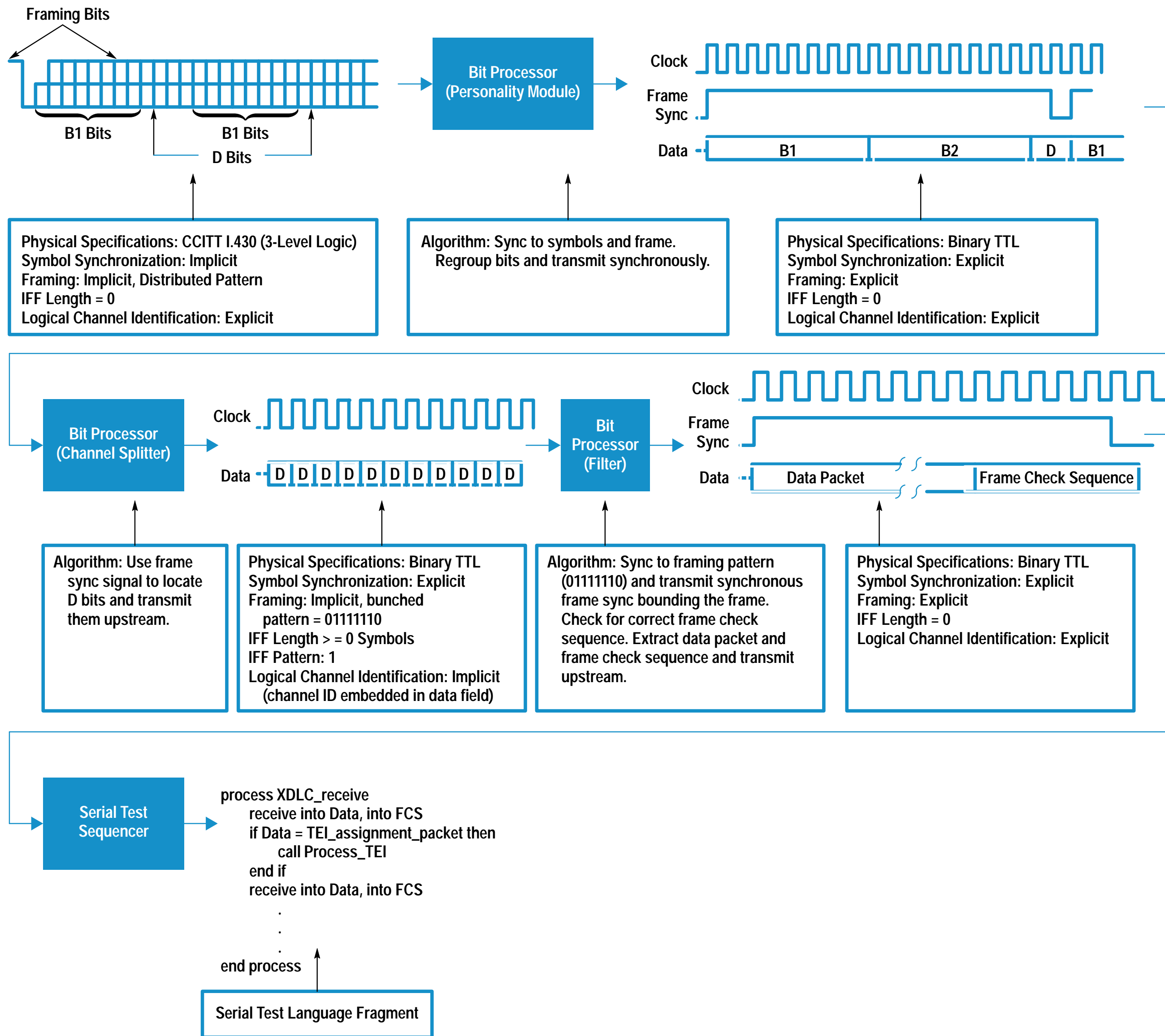

Fig. 2. Processing an ISDN S-bus D channel.

Although our original goal was to make test development faster and easier, we also realized that a multichannel, concurrent architecture could dramatically increase test throughput. For example, a long bit error rate test could be run simultaneously on all channels of a multichannel DUT.

Nondeterministic Bit Streams. In the model, any sort of calculation or adjustment of information in the bit stream is specified by the transformation algorithm. We thought there were two areas of implementation that could be affected by this issue. First, we thought it might sometimes be necessary for an algorithm to access information from both the transmit and receive bit streams simultaneously. This implied the need to process both bit streams within the same bit processor. Secondly, we knew that any actual implementation of a bit processor would have limits, so we wanted to be able to cascade or chain bit processors.

Bit Streams with Embedded Clocks. Our solution to the problem of complex physical interfaces applies to this problem as well: sets of different hardware each tuned to a class of embedded clock schemes.

Hierarchical Bit Streams. Hierarchical bit streams imply multiplexing, so we knew that our implementation needed to be good at handling multiplexed bit streams. This again implied the need for cascadable bit processors.

The architecture that eventually emerged from these considerations is shown in Fig. 3. The architecture is multichannel in nature. The figure shows a single channel of the architecture in the center with adjacent channels above and below. Each channel is designed to attach to a single bit stream of the DUT.

Each channel of the architecture contains two information sources/sinks called *serial test sequencers* (STS). In a typical application, each STS would process independent subchannels (logical channels) of the bit stream. In Serial Test Language, each logical channel is called a *substream* and is controlled by a *process* running on the STS. When required by the test program, STS resources from adjacent processing channels can also be attached to the bit stream, providing up to four substreams for each bit stream.

© Hewlett-Packard Company 1995

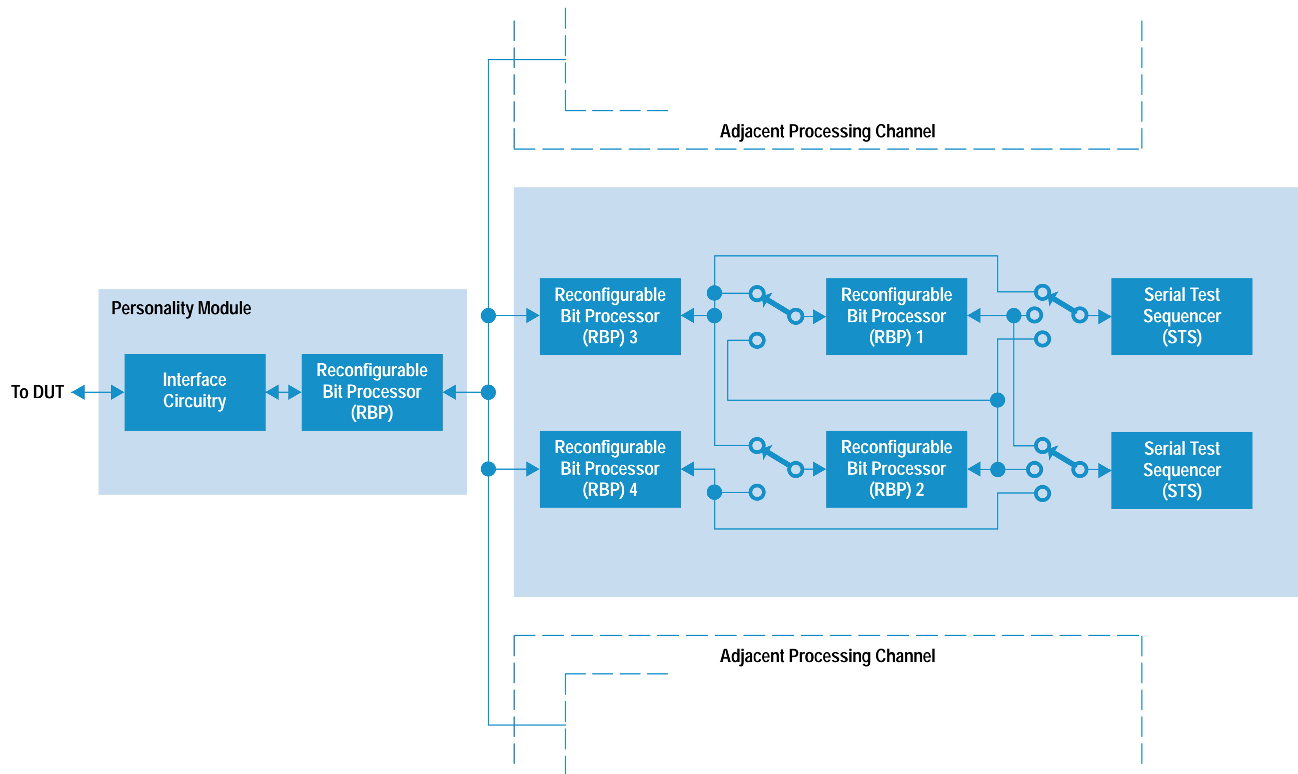

Fig. 3. A single processing channel of the Serial Test Card (STC) for the HP 3070 board test family.

Four channels of the architecture are implemented on the HP 3070 Serial Test Card (STC) and up to 12 STCs can be installed in a system.

Bit Processors

As shown in Fig. 3, each STC processing channel is composed of a series of bit processors connected by serial bit streams. The bit processors are implemented with the XC3000 series of field-programmable gate arrays (FPGAs) from XILINX Company. An important feature of the XC3000 series is their RAM-based configurability. The XC3000 can be programmed on-the-fly in the system, so no preprogrammed ROMs are needed. This feature allows the transformation algorithm of a bit processor to be changed on the fly and makes these devices ideal for this application. The transformation algorithms are implemented by circuits inside the XILINX devices. They are not really either hardware or software, so we have coined the word *circuitware* to describe this sort of reconfigurable circuitry.

Fig. 4 shows a more detailed view of our standard bit processor, called a *reconfigurable bit processor*, or RBP. In addition to the XILINX XC3042 FPGA, each RBP also includes a pair of 2K-by-8-bit RAMs. The RAMs are connected to I/O pins on the FPGA and are used as required by the circuitware designer. This RAM resource complements the architecture of the FPGA and provides a large, dense local storage element.† Each RBP has a downstream port (towards the DUT) and an upstream port (towards the STS), and each of these ports supports data transmission in both directions simultaneously.

† The XC3000 series of parts is structured as an array of D-type flip-flops fed by Boolean function generators. This structure makes them well-suited for state machine and random logic designs, but unsuitable for applications requiring a lot of storage elements.

The bit streams that interconnect the RBPs are defined according to the generic model. All internal bit streams are binary TTL logic signals and are explicitly clocked and explicitly framed. The RBP clock interface ports include *data valid* and *ready* signals to support multiplexed bit streams.

Personality Modules

We use *personality modules* to implement each interface class. There are currently personality modules available for TTL, ISDN S-bus and ISDN U-bus electrical formats. As a whole, the personality module serves as a bit processor and is responsible for converting the external serial bit stream into a format compatible with the internal STC serial bit stream definition.

Serial Test Sequencer

The serial test sequencer (STS) is the final bit processor in the chain. As such, it will often be an information source or

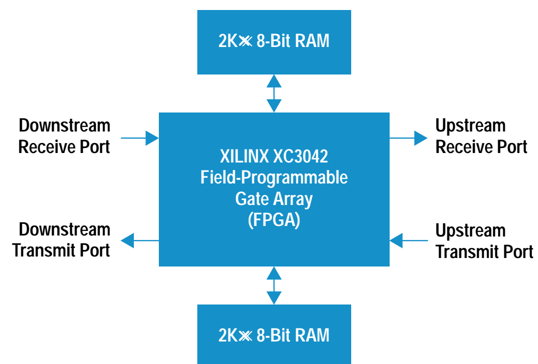

Fig. 4. Reconfigurable bit processor architecture.

© Hewlett-Packard Company 1995

sink. Recall that sources and sinks are where information is generated or analyzed by a communication system. In our case, sources and sinks are the means by which the test programmer communicates with the DUT (in traditional ATE terms, the sequencer or test pattern generator.)

The STS connects on one side to the internal STC bit stream format (as do all of our bit processors). The test programmer controls the transformation algorithm of the STS through a high-level programming language called the Serial Test Language (STL). The STS is implemented with a Motorola DSP56001 digital signal processor. The processor is well-suited to computationally-intensive transformation algorithms as well as general-purpose bit stream I/O.

Bit Processor Interconnect

When testing a multiplexed or hierarchical bit stream, the test engineer will typically choose to process (drive and receive) data for each logical channel independently of the others. At the same time, we have seen that it is often advantageous to be able to process a bit stream in a series of sequential steps. There is then a conflict, given a finite number of bit processors, between having a large multiplexing capability and having a large number of sequential processing stages. This is because multiplexed bit streams are best addressed by a wide, shallow array of bit processors, but multistep processing of a bit stream is best addressed by a narrower, deeper array of bit processors. Our RBP interconnect scheme and special resource assignment software minimize this conflict by allowing a wide variety of multiplexing and processing arrangements.

As shown in Fig. 3, there are several different interconnect paths between the RBPs and the STSs. Fig. 5 shows the topology of each possible interconnect combination. Multiplexing is supported by *channel splitter* circuitware loaded into two of the RBPs (the RBPs labeled 3 and 4 in Fig. 3). The channel splitters are labeled CS in Fig. 5. Resources can also be borrowed from adjacent processing channels, so additional branches can be attached to a personality module.

The system software manages this borrowing, minimizing the total resources required for a customer's test. If even wider multiplexing is needed, personality modules can be connected together at the same physical port of the DUT.

Performance

The STC serial processing pipeline was designed to accommodate maximum bit rates of up to 12.5 Mbits per second. There are obviously some applications that exceed this bit rate and cannot be addressed, but the performance of the architecture is a good match for proprietary telecomm PCM backplanes, automotive applications, LANs, asynchronous protocols at modem rates, ISDN basic rate interfaces, and many other similar applications.

Extensibility

The test capabilities of the STC are formidable, but there are still test cases that will require special capabilities or slightly different functionality. STC capabilities can be modified or increased by adding features to current circuitware, adding functionality to existing personality modules, developing new circuitware, or developing new personality modules. Since all of the circuitware is supplied with the system software and resides on disk, the first three of these can be achieved through simple software updates. The last requires a replacement of an existing personality module, but can easily be performed in the field.

Technologies

Several important technologies contributed to make the STC possible. The most important of these is the XILINX FPGA technology. Our architecture relies on the ability to load different circuitware (transformation algorithms) for each customer test.

To provide a great deal of functionality in as little space as possible, we wanted to use the smallest possible packages for the FPGAs and other parts. Surface mount technology was therefore a necessity, and fine-pitch surface mount was a very strong want. Unfortunately, because of mechanical registration limits, fine-pitch surface mount technology is extremely difficult to implement on large printed circuit assemblies like those used in the HP 3070 family of testers. To solve this problem, we decided to implement each channel of the STC architecture on smaller modules that would plug into the larger main printed circuit assembly. This provided two major benefits. First, we could use fine-pitch parts on these smaller modules, and second, trace routing on the main board was simplified considerably. Trace routing on the modules was still quite dense, but it only needed to be done once.

We also relied heavily on digital simulation technology. All of the circuitware developed for the STC was simulated and debugged before attempting bench turn-on. It was important to minimize debugging on the bench because of the highly

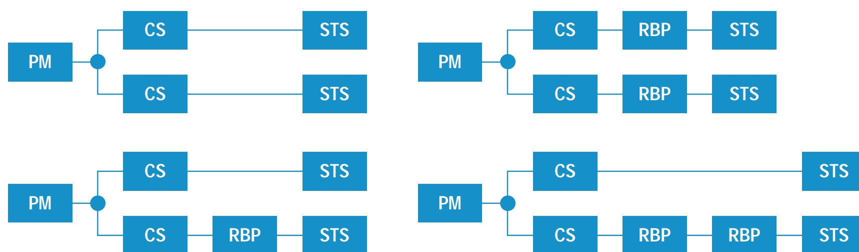

Fig. 5. Processing channel configurations.

© Hewlett-Packard Company 1995

integrated nature of the design. Although it is possible to probe internal FPGA nodes on the bench, it is much easier to probe during simulation.

Early in the project we also experimented with board-level simulation. The primary module, containing the STS and four RBPs, was simulated as a whole. These simulations were mostly intended to verify the circuitry surrounding the DSP56001 and to verify the RBP interconnect scheme. We decided not to simulate the entire STC card, mostly because of the lack of complete simulator models. In retrospect, this may have been a mistake because the vast majority of defects on the first STC prototype were in areas of the circuitry for which off-the-shelf models were available.†

† This part of the design consisted mostly of discrete logic gates and flip-flops, so it was quite similar in character to the RBP circuitware designs. Experience has shown that these types of designs are more prone to design defects than higher-level, "cookbook" designs.

Software Architecture

Up to now, we have discussed the generic model of serial communication systems and the hardware architecture based on that model. This section will provide a software overview of the Serial Test Language (STL).

Design Goals

To set the stage for the software discussion, we'll review the guiding design objectives for the STC and STL.

The process of studying serial bit streams, developing the generic model, and implementing our serial test architecture originally grew out of a desire to ease the test programming problems presented by serial-oriented DUTs. This led to our first design goal: to shorten the test development time by a factor of ten for DUTs with serial interfaces. As described earlier, existing test systems use a single parallel sequencer for controlling several serial bit streams. The resulting programs become very cumbersome and complex to create and maintain.

To address this problem, the STC uses a multiple-processor architecture to subdivide the overall programming task. With STL, we did not hide the hardware architecture from the programmer. Rather, we designed the STC user interface software to embody the generic serial protocol model. The software allows the programmer to segment a serial bit stream into logical pieces. Each logical piece can be programmed independently. This allows the programmer to concentrate on specific functions without keeping track of all the additional overhead found in the bit stream.

The hardware and software were designed concurrently. Several hardware changes were made to allow the software interface to better match the generic model. Many of these changes were facilitated by use of the FPGAs since they could be easily redesigned without a printed circuit board revision.

The concurrent nature of our architecture not only greatly eases the test programming burden, but also provides a clear test throughput advantage for multichannel DUTs. All channels of the DUT can be tested simultaneously, dramatically reducing the test time, especially for long bit error rate tests. We quantified this concurrent test strategy as our second design goal: to improve test throughput for multichannel DUTs by a factor of ten.†† We decided to address this problem through a parallel test strategy. The sequencer was carefully integrated into a complete ATE system which supports the use of many parallel sequencers. The software challenge was to provide an easy-to-program system capable of supporting many concurrently executing processors.

User Environment

Typical DUTs require system resources such as power supplies, nonserial digital drivers and receivers, and other signal sources and detectors to recreate the DUT's normal operating environment. The platform providing these system resources is the HP 3070 family of automatic test equipment.

The HP 3070 family supports both in-circuit and functional styles of testing.††† Both styles can be used together or separately on the HP 3070 system. The STC was designed primarily for board-level functional testing, but can be used for device-level functional test.

The software user interface of the HP 3070 supports the entire process of creating a suite of tests for a DUT. The in-circuit test development process involves describing the components and connections to the system. Using this information, the system generates the information required to build a mechanical interface between the DUT and the system. This information is also used to generate individual tests for each component.

The functional test process is similar. Only the edge connector need be described instead of all the components. The user then creates resource libraries describing the connection of HP 3070 resources to the DUT edge connector. The system uses the libraries and edge connector description to generate the mechanical interface description.

The highest-level user interface on the HP 3070 system uses the HP VUE environment running on the HP-UX* operating system. Specific portions of the test hardware are controlled by textual languages. These languages are similar in structure and syntax conventions and were designed for the specific purpose of testing DUTs. Overall test sequencing is controlled through a textual interface running an interpretive editor for the HP Board Test BASIC (BT-BASIC) programming language.

The standard digital sequencer is controlled by using the Vector Control Language (VCL). This sequencer provides parallel digital capability. The analog sources and detectors and external instrumentation access are controlled using the Analog Test Language (ATL). Both languages can be used together to test a particular DUT. A graphical Motif/X11

†† The choice of a factor of ten as a goal was somewhat arbitrary because the actual throughput improvement depends on the number of DUT channels. An eight-channel DUT might be tested only eight times faster, whereas a sixteen-channel DUT might be tested sixteen times faster.

††† In-circuit testing refers to methods for testing the various components of a DUT separately while they are in place on the board. This form of testing is based on the assumption that testing all components and connections between components ensures that the DUT as a whole has been properly tested. This form of testing produces excellent fault diagnosis to the failing component for repair. Functional testing refers to methods for testing a DUT by emulating the system environment into which it will later be integrated. Many DUTs have an edge connector interface, so this is also commonly called edge connector functional testing.

© Hewlett-Packard Company 1995

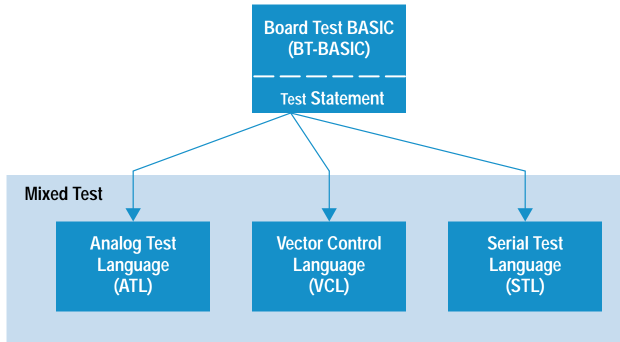

Fig. 6. Structure of HP 3070 testing languages.

environment supports development of VCL and ATL tests. Fig. 6 illustrates this software structure.

All textual sources are compiled before execution. The test is then executed from the BT-BASIC environment by use of the test statement. BT-BASIC is also used to control test resources like power supplies, test system operator interfaces, test results capture, and so on.

Serial Test Language Overview

STL was designed specifically for testing serial interfaces. Typically, these interfaces require the following capabilities:

- Simple, flexible input/output. All serial I/O can be segmented into a collection of bits called *frames*. The size of the frame varies depending on the application.
- Frame and bit manipulation features including comparison, modification, formatting, and concatenation.
- Complex conditional constructs that execute in real time. For many serial protocols, this means being able to perform 10 to 20 statements within 125 microseconds.
- Support for conversion of frame data to and from numeric data types. Many protocols require numeric processing of signals or numeric control.
- Support of multiple STCs running concurrently emulating a single serial bit stream or multiple serial bit streams.
- Triggering capabilities between concurrent processes (i.e., serial test sequencers) and the existing HP 3070 digital sequencer.
- Run-time control of STC personality modules and reconfigurable bit processors.
- Extremely flexible setup of STC personality modules and reconfigurable bit processors (RBPs). New RBP personalities and personality modules have been designed after initial product release. The user interface is designed to accept these with minimum software rework and without disturbing existing customer tests.
- Easy programming of the clock signal sources on the STC. These clock signal sources are used to simulate the bit and framing clocks found on many serial interfaces.

Each serial test for a particular DUT is contained within one source file. This source file controls all of the STCs required for that test. The number of tests depends on the level of diagnostics desired by the user. Generally, the user will write multiple tests to exercise the DUT fully.

All major units of the serial source code are organized as blocks. Each unit begins with a starting block statement and is terminated with an ending block statement. This greatly

```
serial;Count1,R1_frames

serial clock IOM_clock  is 2048 events
  events every 61.03515625n internal
  connect clk1 to ″DCL″
  connect clk2 to ″2048″
  at event 0 set clk1 to ″1100″
  at event 0 set clk2 to ″11110000″
end serial clock

stream IOM2_Master  type ″synchronous″
  connect ″tx_clock″ to ″2048″
  connect ″tx_data″ to ″DD″
  transmit length unknown
  substream TX_All
    filter ″xdlc CRC-CCITT″
    transmit bits all
  end substream
end stream

stream IOM2_Slave  type ″synchronous″
  connect ″rx_clock″ to ″2048″
  connect ″rx_data″ to ″DD″
  set ″rx_clock edge″ to ″falling″
  receive  length unknown
  substream RX_All
    receive bits all
    filter ″xdlc CRC-CCITT″
  end substream
end stream

process TX_All
 loop
   transmit ″\h5555″
   transmit ″\haaaa″
 end loop
end process

process RX_All;Count1,R1_frames
 dim AAAA$[16]
 loop
    receive into AAAA$,″\hxxxx″
    exit if AAAA$ = ″\hAAAA″
 end loop
 Count = 0
 loop
    receive ″\h5555″,″\hxxxx″
    receive ″\hAAAA″,″\hxxxx″
    Count = Count + 1
    exit if Count = Count1
 end loop
 R1_frames = Count
 initiate trigger digital
end process
```

Fig. 7. An example of a Serial Test Language (STL) program.

simplifies parsing and dramatically improves error diagnostic messages.

The overall structure for a serial test is illustrated in the example shown in Fig. 7.

The serial statement, serial;Count1,R1_frames, is used to define variables passed to or from the test execution environment (BT-BASIC). This allows flexibility in changing test execution parameters without needing to recompile. For instance, the user can pass in the number of seconds a bit error rate (BER) test is to be executed.

The optional serial clock block, serial clock IOM_clock ..., programs the STC clock sources to output specific clock signal patterns. Each clock section controls the four synchronous clock resources found on one STC. These clock generators can be synchronized to an internal or an external clock source. The clock signals can have variable lengths ranging from 2 to 65535 pattern changes (called events). The large number of events provides a very flexible format for defining

© Hewlett-Packard Company 1995

custom clock and frame sync signals. The width of each event is defined in nanoseconds or microseconds.

The stream block, stream IOM2_Master type "synchronous", is used to define the physical characteristics of the serial bit stream and protocol to the corresponding personality module. There are three personality module types: TTL, ISDN SBUS, and ISDN UBUS. The ISDN personality modules are used specifically for connection to ISDN interfaces. The TTL personality module is a flexible collection of programmable TTL-voltage level drivers and receivers used to interface to generic TTL-level serial bit streams. The stream type determines the personality module's mode of operation. In the above command, the keyword "synchronous" programs the personality module bit processor to expect an explicitly clocked bit stream. This type of serial bit stream requires a TTL personality module. An error would be signaled if this keyword were used and a TTL personality module didn't exist. The electrical levels of the personality modules are fixed, but the stream protocols are programmable. The example in Fig. 7 is an explicitly clocked and implicitly framed bit stream. Therefore, the TTL personality module uses two signal lines to receive data: rx_data and rx_clock. The explicit framing signal line, rx_frame_sync, isn't required by the bit processor because the HDLC format has the framing information embedded in the actual data transmitted. Different TTL modes are used to reprogram the bit processor to emulate a variety of serial protocols ranging from synchronous TDM interfaces to asynchronous interfaces like RS-232.

The mode of the ISDN personality modules can be changed as well. For example, the ISDN S-bus module can simulate terminal equipment or a network terminator. The automatic ISDN synchronization sequences are different depending on the ISDN mode selected. The mode type reprograms the personality module's circuitware to conform to different stream requirements.

The programmer connects the personality module to the DUT by using the connect statement, connect "tx_data" to "DD". This statement physically closes the correct STC relays to connect a personality module's resources to the DUT. The personality module's resources are multiplexed. During the test generation process, the system software will automatically determine the optimal connection method to the DUT. This connection method is then used to build the correct fixture interface to the DUT.

Each personality module mode has a set of programmable features. These are controlled by use of set statements: set "rx_clock edge" to "falling". The set statement is quite generic:

set "mode description" to <value>

The <value> parameter can be a numeric or string identifier like "enabled". Each mode has a specific set of features. If a feature is not explicitly set, then a default value is used.

By definition from our generic serial protocol model, each stream will have a structure of bits called a frame. A frame of bits is continuously (or on demand) transmitted or received. Depending on the protocol, we may or may not know the frame length. Certain serial protocols have the frame length embedded in the actual stream of logical bits; HDLC is an excellent example.

The transmit frame length and receive frame length statements are used to capture this information. This may be unknown (which is allowed as a keyword) or a number between 2 and 4096. This information is used by the channel splitter circuitware as it inserts or extracts bits from the stream.

This leads us to the next block structure, the *substream*. Each stream can have between one and four substreams. The substreams are defined as a block structure within each stream block. The substream programs the channel splitter to target specific bits within the frame. Each substream can receive and transmit bits from within each stream frame. The bits targeted by the channel splitter are therefore either extracted or inserted into the stream frame. Each substream has an associated serial test sequencer called a *process*. Associated substreams and processes have the same name. The substream defines which bits of the frame are passed to or received from the process. The process contains the actual program used to control the bit values.

If the stream frame length is known, then each substream is defined as a portion or all of the stream frame. Particular bits in the stream frame are enumerated from 1 to the frame length. The user tags certain stream bits for the substream to transmit or receive by use of the transmit or receive bits statements: transmit bits all, transmit bits 1 to 8, receive bits all. Multiple transmit or receive bits statements accumulate tagged bits. All these bits are concatenated (by the STC hardware) in order (bit 1 to bit n) and treated as a frame by the process associated with that substream.

If the stream frame length is unknown because it must be recovered from the bit stream, as is the case shown in Fig. 7, then the programmer must define a reconfigurable bit processor to transform the bit stream. This is identified in the substream block as a filter statement or block, filter "xdlc CRC-CCITT". Set statements can be used to control the filter in much the same way as set statements are used to control the personality module in the stream block.

Our generic model allows infinite levels of bit stream hierarchy. The STC hardware supports two levels (stream and a single layer of substreams). Additional levels can be emulated in the serial language. This has not proved too restrictive since most protocols require no more than two levels. Refer to Fig. 2 for an example of how this works.

Multiple substreams can receive the same bits, but only one substream can transmit a particular bit. A substream may have no bits being transmitted or received. This capability is used to create a process that simply controls other processes.

As stated previously, each substream has an associated process, process RX_All;Count1,R1_frames. Each process represents an independent program that is executed concurrently with all other processes defined in a particular serial source program. Variables passed into the serial test are subsequently passed to individual processes. The STL process statements were modeled after BT-BASIC. Therefore, the statements and structures allowed in STL processes include:

- If-then conditionals
- Logical operators
- Assignment
- Loop/exit if/end loop construct

© Hewlett-Packard Company 1995

- Subroutines (with data scoping control)
- Bit error rate test functions
- Numeric functions
- Access to stream and filter features through control and status functions.

Bits to be transmitted or received are formatted as a data type called a *frame*. Frame data can be stored in a variable, A$, or as a constant, such as 10001. Each frame variable is simply a linear array of packed bits. From a programming perspective, the frame variable is treated like a string data type. Each frame variable has a maximum length declared by the dim statement. The default length is 16 bits.

Data is transmitted by the transmit statement, transmit "\h5555", A$, or received into the process by the receive statement, receive "\h5555", into A$.

All process frames are buffered internally within the serial test sequencer. These buffers allow the serial test sequencer to continue to execute the program and not be tied directly to the DUT transmit or receive rate.

All string operations like substring, concatenation, equality, insertion, and deletion are supported with the frame data type. Each bit of a frame works just like a character in a string. The user can use various notations (binary, octal, hexadecimal, decimal, or ASCII) to assign a value to a frame variable. Since string operations were modeled after those in BT-BASIC, programmers familiar with BASIC languages can quickly learn STL frame operations.

An STL process also supports integer, real, integer array and real array data types. Conversion between integers and frame data types is supported. These data types are used to support complex control algorithms and digital signal processing of data. One application requiring this capability is testing analog line cards. Line cards interface subscriber equipment, such as a phone, to the public data network. Typical tests require a significant amount of digital signal processing for testing analog-to-digital transmission transfer functions.

Time-order sequence control between processes and the digital sequencer is handled by use of level-sensitive, named triggers. The triggering mechanism has been implemented using a simple mailbox approach. The process sending a trigger places the appropriate trigger number in its assigned mailbox. Processes receiving triggers are told in which mailbox and for which number to look. This implementation allows any process to send or receive a trigger to or from any other process. It also allows multiple processes to receive the same trigger. Ten different triggers can be sent to any other process defined in the serial source. In addition to interprocess triggering, the system supports triggering to and from the parallel digital sequencer. For example, Fig. 8 shows two processes, A and B, sending and receiving triggers.

This simple trigger scheme allows a wide variety of sequence control among concurrently executing processes and the HP 3070 digital sequencer.

All statements in the process section are optimized for speed. Testing and use have shown that STL can perform a fairly complex set of operations in 125 µs† to a few milliseconds.

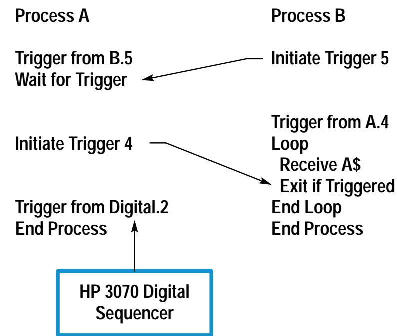

Fig. 8. Triggering in STL.

Debugging Serial Tests

The HP 3070 test debugging environment consists of a Motif/X11 interface that communicates with the HP 3070 system hardware through the BT-BASIC interactive editor.

The original debugging environment supports both the digital sequencer and the analog measurement subsystem. It provides an easy-to-use pull-down menu structure to control the analog and digital test functions, view digital test vectors graphically within a logic analyzer display, create measurement histograms, and so on. A serial mode was added to allow the user to view the status of variables, processes, connections, and so on. All modes support viewing of the textual source code and commands to execute the source program and view the data. If necessary, the source program can be modified, recompiled, and executed from the debug environment.

Fig. 9 shows the serial debug environment. The largest box contains the serial source program. It can be modified by the user in this pane. The compile-and-go button allows the user to quickly†† recompile just the serial source code and execute the test. The pane on the left side contains a list of STL processes. Clicking on one of these places the source program at the first line of that process.

The command pull-down shows the list of debugging commands available, such as viewing the current contents of variables, trigger log, current process status and line number, and status and contents of the transmit and receive buffers.

By various pull-down menu selections, specific variables or groups of variables of a particular type can be displayed. Frame variables can also be displayed in a variety of formats (binary, octal, hexadecimal, decimal).

The trigger log on each processor captures the last 20 trigger events. The events are displayed in chronological order. This helps resolve difficulties when triggering between processes or the digital sequencer. Each trigger event is displayed in the same syntax as the trigger statements in STL.

Each process keeps track of its own status and line number. The status debug command displays the current status and

† 125 µs is an important number in telecommunications applications because it corresponds to the basic 8-kHz frame rate used throughout the network.

†† Usually in under 15 seconds. The largest serial test to date takes 44 seconds to compile.

© Hewlett-Packard Company 1995

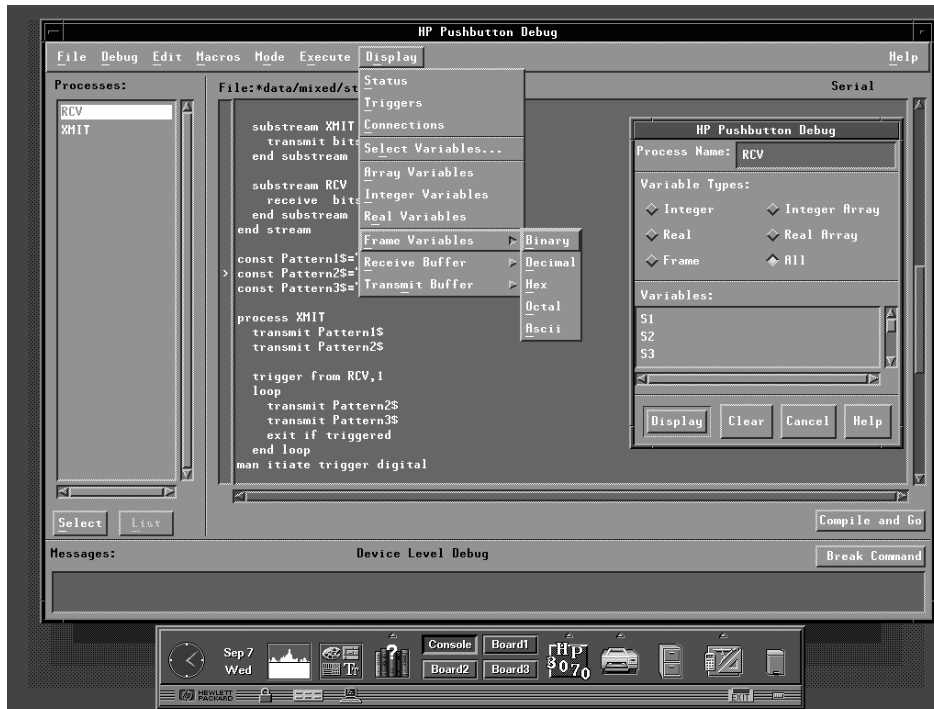

Fig. 9. Serial debug environment.

places the source pane at the line number the process was last executing.

Special commands are available to display the contents of the transmit and receive buffers. This allows the user to see what has been recently transmitted (in the transmit buffer), what was about to be received in STL, and what has been received by the STC hardware.

Breakpoints can be set by the user by means of the halt serial failing statement. Single-step execution is not possible because of the real-time nature of the STC. The serial test must execute until either the test ends normally or an exception occurs. If any exception occurs, all serial processes are immediately halted (this is implemented by a hardware signal) and the current status of all serial processes is available.

It is possible to switch to the digital (VCL) or analog (ATL) test modes by the mode pull-down menu. The interface for these modes is very similar to the STL interface.

Implementation

The STL compiler was written using C++ and object-oriented programming techniques. In an attempt to improve our software development process, the STC/STL group set several goals:

- Learn and use object-oriented programming development techniques (using C++ as a development language).
- Leverage libraries and tools as much as possible.
- Dramatically improve the quality of our products.

Object-Oriented Programming. When we started, object-oriented programming was a new technique in the realm of software development. After investigation and some training, we knew this technique was not going to be easy or immediately rewarding. In retrospect, the definition of classes is a time-consuming task, but is critical to the success of a project.

Use of an object-oriented design provides two primary advantages: better hardware abstraction and extensibility. An object-oriented abstraction models a given hardware construct like the STC quite naturally. Object-oriented programming allows a hierarchy of classes defining the various hardware levels to be created easily.

The abstraction of object-oriented programming made the STL compiler design much easier and more readable. Both the language constructs and the STC hardware are encapsulated in a hierarchy of classes. They meet at the code-generation phase of the compile, where language constructs (which are simply an instance of the statement class) invoke a message to the hardware code-generation classes.

An enhancement to the original software was the support of real and array data types. Typically, the addition of these data types would have taken approximately one engineering year. Because of the object-oriented programming benefits, the addition took only three engineering months.

Leverage. Leverage of existing tools and libraries saved a great amount of time and effort. We leveraged many areas of the STL/STC software development. We used the Codelibs library of C and C++ classes for data abstractions and algorithms. We used a third-party assembler for the STS sequencer programming. Debugging assembly code during development of the STL compiler was far easier than editing binary downloads. We eventually replaced the assembler with a direct binary output to speed up the compiler, but we left the assembly output mode as a debugging tool. We used wacco, a top-down recursive descent parsing tool, to simplify the parser and improve error handling, and we used and improved internal C++ classes for error reporting and file I/O.

The quality of the existing tools or libraries should be investigated carefully before leveraging them into the product. We

© Hewlett-Packard Company 1995

did not suffer any problems, but we were careful with our dependency on leveraged code.

Software Quality Assurance. There is no shortcut to quality. 30% of the product development time was spent in quality assurance.

There is no doubt that defects are far more easily fixed in earlier phases of software development, but a high-quality product is not ensured until the boundary conditions have been tested. The object-oriented programming design process and leverage of high-quality software components contributed greatly, but we also spent a great deal of effort in testing our software.

Our testing effort focused on two areas: early users (alpha sites) and automated regression testing. We developed tests on several customer DUTs and used two alpha sites to build our confidence in the ability of the STC to test serial DUTs and meet our project goals. We also created a set of tools that allowed our group to develop over 900 automatic regression tests. We used branch flow analysis† to refine these tests to get to 92% coverage within the STL compiler. The regression test suite is also used to verify that a particular change or defect fix has not introduced any other defects.

STL Summary

The key STL objectives were to shorten test development time and to increase test throughput for DUTs using serial devices. The ability to divide and conquer the serial bit stream using multiple processors has proved very successful in reducing the time to implement tests. In some cases, test development has been reduced from 4 to 6 months to 1 to 2 weeks. The ability to easily test multiple bit streams concurrently increases test throughput dramatically, especially in BER tests.

Customer Application Case Studies

The sequencer architecture we have described in this paper was derived from studies of our generic model of serial communication systems, which was itself developed from the study of a wide variety of serial protocols and DUTs. To test the effectiveness of the new architecture, we wrote functional tests for many serial protocols and customer boards. Two customer boards used in these case studies are described in more detail below.

Case Study I

The first customer board was a telecomm multiplexer. The board is used to multiplex and demultiplex four 2.048-Mbit/s bit streams to and from a single 8.448-Mbit/s bit stream. Converters in the fixture were used to translate the HDB3 signals in these streams to TTL-compatible levels. An additional serial bit stream is used to send and receive control information to and from the board. Each of the six serial bit streams was attached to an STC processing channel.

The customer had been manufacturing this particular board for five years, and had brought up their suite of functional tests for the board on two previous testers, both of which used a traditional pattern sequencer. Their test suite consisted of 15 functional tests, and they reported test development times of nine months and five months using the two previous board testers. Using the STC, they implemented their 15-test suite and 16 additional tests in four days—a 25-fold decrease from their best previous test development time. The tests implemented included a BER test on all four channels simultaneously and other tests they simply could not perform with the sequencer architecture available on the other testers.

Case Study II

The second customer board was an ISDN U-interface central office line card. This board has an ISDN U-interface for each of four subscribers and a serial backplane interface. Each of the five bit streams was attached to an STC processing channel. At the time this test was developed, we had not yet introduced our U-interface personality module, so commercial network terminators were used to translate the subscriber channels to ISDN S-interface format. The backplane of this board is a 2.048-Mbit/s serial bit stream conforming to the IOM-2 protocol.

Each subscriber port was split by the STC processing chain into two substreams, one to handle data transfer and the other to handle activation control of the ISDN interface. The backplane bit stream was also split into substreams, but in this case we chose to use a single process to handle all four control channels. This reduced the total number of STC channels that would otherwise have been required and did not overly complicate the test program. We also handled the four data channels with a single process, using a special, optimized BER function built into STL. This function can receive up to 32 independent BER data streams simultaneously with no intervention or special programming required of the test engineer.

Other Applications

During product development, we tested the architecture against many other serial protocols and formats, including ISDN S- and U-interfaces, RS-232-style asynchronous protocols, CEPT-30, T1, automotive serial interfaces, I^2C, HDLC control channels, generic 64-kbit/s bit streams, IOM-2, ST-Bus, and various TDM backplanes. In each case we were able to communicate with the bit stream with a minimum of programming time and effort.

Summary and Conclusions

Based on the case studies and other applications described above, we have found that when testing serial-oriented DUTs, the new architecture offers the following advantages over traditional sequencers:

- Much faster test development
- Much better test coverage (more functionality of the DUT can be tested more easily)
- Much better throughput (because of the ability to test multiple channels of a DUT simultaneously)
- A reduction in fixture electronics.

† This tool inserts probes into the source code that allow reporting of coverage of particular execution paths within a program.

© Hewlett-Packard Company 1995

The only potential disadvantage of the architecture is a slight increase in the capital cost of the test system. The relative weight of the advantages and disadvantages is determined by the type of DUT being tested and the type of test being run on that DUT. When testing boards with multiple identical channels, especially when the board tests include long conformance tests like BER, the n:1 increase in throughput one can achieve using n STC channels easily offsets the slight price premium of the hardware. Board tests that do not include long conformance tests and that involve significant overhead because of handling or heavy in-circuit testing will not see such a clear cost advantage, but even then the significant reduction in test development time may offset the cost of the hardware.

Acknowledgments

We would like to thank the following people for their work on the STC/STL project: John Siefers for the design and development of the STC main board and personality modules, John Algiere for his system design work, Greg Stander and Bud Cribar for the design and implementation of the serial test compiler, Eric Waldheim for the design and implementation of the DSP56001 assembler routines, Sunit Bhalla for writing the hardware confirmation and diagnostics test routines, Wilson Spence for his always enthusiastic suggestions and feedback, and Cullen Darnell and Lynn Schmidt for their management leadership.

References

1. R.E. McAuliffe, "Practical Production Testing of ISDN Circuit Boards," *Proceedings of the IEEE International Test Conference,* 1988, pp. 39-46.

2. J.T. Healy, *Automatic Testing and Evaluation of Digital Integrated Circuits*, Reston Publishing Company Inc., 1981.

3. CCITT, "Integrated Services Digital Network (ISDN), Overall Network Aspects and Functions, ISDN User-Network Interfaces," *CCITT Blue Book, Recommendations I.310-I.470, Volume III, Fascicle III.8*, 1988.

4. W. Stallings, *ISDN and Broadband ISDN*, *Second Edition*, Macmillan Publishing Company, 1992.

5. H.S. Stone, *Microcomputer Interfacing,* Addison-Wesley Publishing Company, Inc., 1982.

6. D.N. Chorafas, *The Handbook of Data Communication and Computer Networks,* Petrocelli Books, Inc., 1985.

7. R.E. McAuliffe, "Board Testing Modern DUTs: Solving the ISDN Test Challenge," *Hewlett-Packard internal communication*, 1988.

HP-UX is based on and is compatible with Novell's UNIX® operating system. It also complies with X/Open's* XPG4, POSIX 1003.1, 1003.2, FIPS 151-1, and SVID2 interface specifications.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

X/Open is a trademark of X/Open Company Limited in the UK and other countries.

Motif is a trademark of the Open Software Foundation in the U.S.A. and other countries.

© Hewlett-Packard Company 1995