A DSP Feedback System on a Mixed Signal Tester and its Application in ADC Test

by

Qinghua (Cindy) Zheng

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

February 1996

Author ____________________
Department of Electrical Engineering and Computer Science
December 15, 1995

Certified by ____________________
Hae-Seung (Harry) Lee
Thesis Supervisor

Accepted by ____________________
F. R. Morgenthaler
Chairman, Department Committee on Graduate Theses

A DSP Feedback System on a Mixed Signal Tester and its Application in ADC Test

by

Qinghua (Cindy) Zheng

Submitted to the
Department of Electrical Engineering and Computer Science

February 1996

In Partial Fulfillment of the Requirement for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

A DSP-based mixed signal tester with feedback capability provides great flexibility and versatility in real-time mixed signal device testing. In this project, a DSP-based feedback loop system is designed and implemented for the DSP module of Schlumberger's new mixed signal tester. This DSP-based feedback loop system provides a dynamic close-loop testing environment for waveform generation and data analysis. Also, an ADC servo-loop test for ADC output code transition edge search is developed to demonstrate the feasibility of this feedback loop system design.

Thesis Supervisor: Hae-Seung (Harry) Lee
Title: Professor of Electrical Engineering

Acknowledgment

Thanks to the Slingshot group at Schlumberger ATE Division for providing me such an exciting and challenging project: Daniel Rosenthal, for carefully framing the project before my arrival and guiding me through the project; Lakshmikantha Prabhu, for always helping me out on every design detail, and more importantly, being a such kind and patient friend; and the other Slingshot group members, for being friendly and being cool.

I am very grateful to my host families who have offered me their love and support: Joy and Nasser Madani in Wales, U.K., and Anne and Bob Berg in Boston, U.S. Thanks for providing me a lovely home during the holidays when I am far away from my parents, taking me out for dinners when I am stressed out of my studies,.... I could not have done it without them.

Also thanks to my college friends, Wenjie, Liang-wu, Guoling, Dawn, Anni, etc., for being there and being great.

And finally, many thanks to my caring parents and brother. Although they are living on the other side of the planet, their confidence and support have helped me all along the way.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Automatic Testing Equipment (ATE) industry has been playing an important role in device testing ever since the dawn of semiconductors in the 1960s. It is essential to fully quantify the performance and test the functionality of a semiconductor device before utilization. The traditional ATE industry concentrated mostly on developing equipment for testing and measuring either analog or digital devices. In recent years, a new class of semiconductor devices with significant contents of both analog and digital circuitries, namely, mixed signal devices, have increasingly become popular. This kind of device usually has one or more ADCs and/or DACs embedded in it to operate in both the analog and digital domains. The emergence and continuous growth of mixed signal devices stimulated the development of a new class of testing equipment which integrates both analog and digital test functionality: mixed signal ATE systems.

Schlumberger ATE division designs and manufactures automatic test equipment, diagnostic systems, and back-end automation systems for the testing of semiconductor devices[1]. The primary products of Schlumberger ATE division, ITS 9000 digital tester family, are geared to test high accuracy and high performance VLSI digital devices such as ASIC, RISC, or microprocessors, covering performance ranges from 10 MHz up to 200 MHz. For example, the ITS9000 FX and GX tester series are used to test the Intel Pentium microprocessors which have a running speed of up to 120 MHz. Recently, the development of a new mixed signal tester has been started at Schlumberger. This new mixed signal tester is based on the existing ITS 9000 family digital tester and meets the demands of testing not only high speed digital semiconductor devices such as microprocessors, but also complex mixed signal devices, such as Integrated Services

Digital Networks (ISDN) chips and modems.

The objective of this thesis project is to design and implement a feedback loop system for Schlumberger's mixed signal tester. This feedback loop system provides a new mixed signal testing scheme which enhances the flexibility and versatility of Schlumberger's mixed signal tester. After the feedback loop system is implemented, an ADC servo-loop test is developed to demonstrate the feasibility of the feedback loop system.

This thesis is comprised of seven chapters. In Chapter 2, an overview of mixed signal and ADC testing, and the description of Schlumberger's new mixed signal tester and its unique functionality are provided. An overview of the DSP module and existing hardware and software environment is given in Chapter 3. In Chapters 4 and 5, the feedback loop system of Schlumberger's mixed signal tester is described. In Chapter 6, an ADC servo-loop test using the feedback loop system is presented and its performance is analyzed. This thesis closes with its conclusion in Chapter 7.

Chapter 2

Background

2.1 Mixed Signal Testing

2.1.1 Overview

There are at least three ways to define a mixed signal device. One definition classifies a device as a mixed signal device if it has a certain ratio of analog to digital cells. Another definition classifies mixed signal devices as those with both digital and analog signals at their input, output or control terminals. The third definition classifies mixed signal devices as interfaces between the digital and analog domains[2]. These definitions have a common characteristic --- mixed signal devices handle both analog and digital signals.

The history of mixed signal testing is relatively short. Based on the conventional pure analog and pure digital testing equipments in ATE industry, various approaches have been proposed to tackle the task of mixed signal testing. The most intuitive approach is to test the analog functions of a mixed signal device with an analog tester and to test the digital functions with a digital tester. This method does not require any modifications to the existing test equipment, but it does not cover the testing of analog to digital or digital to analog conversions. Another approach is to modify the existing tester to accommodate the mixed signal testing capability, such as augmenting some digital functionality to an analog tester (big A - little D), or similarly, augmenting some analog functionality to a digital tester (big D - little A) [2]. This approach emphasizes a portion of the mixed signal functionality testing while diminishing the others.

The fast growing market of complicated telecommunication mixed signal devices has strived the ATE industry to provide test systems that tightly integrate the analog and digital functions together. For mixed signal devices with high integration levels, adequate testing is impossible without concurrent processing for analog waveforms and digital patterns in real-time[3].

The rapid development of digital signal processing technology has changed the ATE industry profoundly. Since a digital signal processor (DSP) handles complex data manipulations more flexibly and accurately than its analog counterparts, the device testing procedures are simplified and the testing accuracy is improved.

2.1.2 DSP-Based Mixed Signal Testing

Mixed signal device testing involves, in general, the generation of various signals to be sent to a device under test (DUT) and the measurement of the DUT response. Figure 2-1 schematically depicts a typical flow of a mixed signal device test.

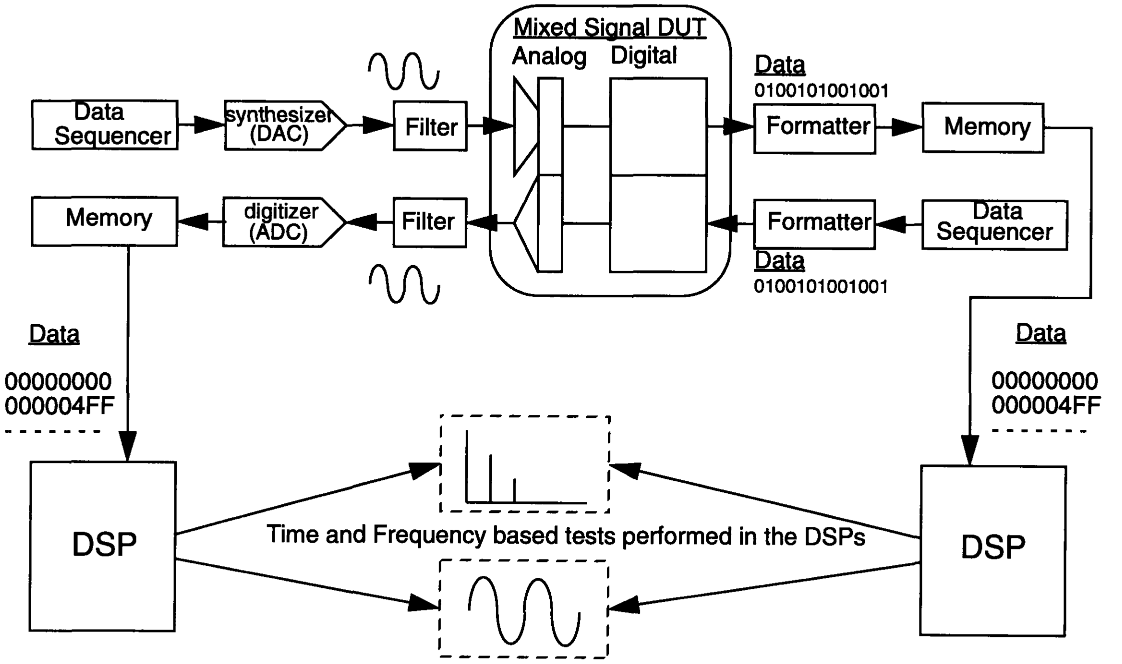


Figure 2-1: A typical mixed signal device test.

Several key components are involved in a test system during a mixed signal device test: a central processor unit for overall testing control, data sequencers for input/output storage and piping, formatters for digital input/output data format conversions, a waveform synthesizer and a waveform digitizer for analog and digital signal conversions, filters for analog noise reduction, and a DSP engine for data manipulation[4]. For example, suppose the DUT has an analog input and a digital output. A test can be performed by sending an analog signal to the DUT and measuring its digital output response. A set of digital data is first generated on the central processor unit of the tester and downloaded into the data sequencer (on the left of Figure 2-1). The digital waveform is then converted into an analog signal by the waveform synthesizer and the analog noise is filtered out before the signal is sent to the analog input terminal of the DUT. The DUT's digital output is converted by a formatter into a digital data format acceptable by the tester and then stored in a memory unit. The captured output is then sent to the appropriate DSP for data analysis. The test flow is reversed if the DUT has a digital input and an analog output.

The DSP plays an important role in mixed signal testing. It replaces most of the conventional analog instruments and acts as the core of a mixed signal tester. The DSP is responsible for processing data gathered from a DUT before it is uploaded to the tester's central processor unit[4]. For example, a signal from the analog output of a DUT is digitized by a waveform digitizer and sent to the DSP, which performs an algorithm such as an FFT, and sends the results to the central processor unit for further evaluation. Conversely, the DSP is able to generate any pre-specified waveforms in real-time to be sent to a DUT. This results in a faster test time because the DSP alleviates some of the responsibilities of the central processor unit. DSP also improves the repeatability and the accuracy of a test because it eliminates analog errors that would occur if analog instruments were in place. And more importantly, the ability to create and adjust testing programs on DSP vastly increases the flexibility and versatility of a tester.

2.2 A/D Converter Testing

2.2.1 Overview

ADCs and DACs are categorized as mixed signal devices because they handle both digital and analog signals. They are also often embedded in most of the complex mixed signal devices such as modems and ISDN. Hence, it is essential to fully test the performances of an ADC and a DAC.

An ADC and a DAC have their functionality in a reversed manner; however, due to their highly non-linear functionality, it is not feasible to just reverse the testing process for a DAC to produce a testing process for an ADC. DAC testing is relatively straightforward: A digital stimulus is sent to a DAC and its analog output is digitized and measured. A change in the digital input to a DAC will cause a corresponding change in its analog output. However, for ADC testing, a range of analog input is mapped to the same digital output, thus a change in ADC analog input may not cause any change in its digital output. Therefore it is necessary to develop testing methods specifically for an ADC.

In recent years various techniques have been developed for testing static and dynamic performance of an ADC at different accuracy levels and testing speeds. The servo-loop method and the histogram method can be used to test static ADC linearity at very high accuracy and resolution level[5][6]. The Walsh transformation presents a way to compute ADC error parameters[7][8]. Fourier analysis in the frequency domain is another approach to test an ADC[9]. A picosecond electron-beam tester has been developed to measure the dynamic performance of a high speed ADC[10]. Statistical analysis has also been applied to characterize the behavior of an ADC[11]. These testing methods approach the ADC testing problem from either theoretical or statistical point of view and concentrate on either static or dynamic aspect of an ADC's performance. Some of them are theoretically complex, such as the Walsh function application. Some are highly dependent on a tester's hardware functionality, such as the servo-loop method.

2.2.2 Linearity Test of an ADC

Integral-non-linearity (INL) is an important parameter for an ADC. It is a measurement derived from the ideal transfer function of an ADC and is typically expressed in terms of least-significant-bits (LSB)[12]. Figure 2-2 shows the concept of INL for a 3-bit ADC. A range of analog input is mapped to 8 digital output codes under the 3-bit ADC transfer function. The output of an ADC jumps from one code to the next when the analog input crosses a code transition edge.

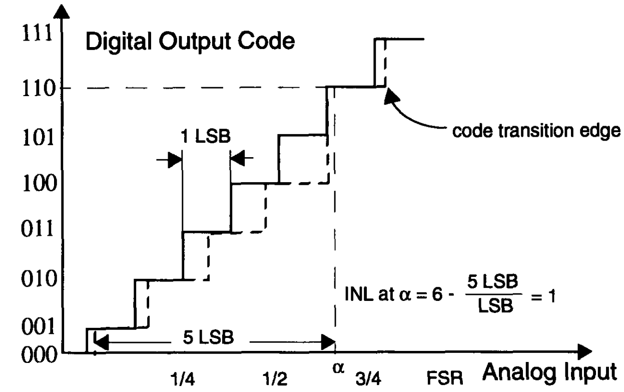


Figure 2-2: Integral non-linearity of a 3-bit ADC.
The solid line represents the ideal transfer function. The dash line represents the actual transfer function.

An ADC linearity test refers to the measurement of one or all code transition edges of an ADC. All the ADC linearity test methods can be classified into two categories: the servo-loop method and the histogram method[12]. The servo-loop method is used to find the input voltage at which a pre-specified output code transition occurs[5]. A servo-loop system is a closed loop system in which the digital output of an ADC is continuously measured and fed back to modify the input until a pre-specified code transition is located. This method requires closed loop circuits, and therefore additional hardware is needed for an open-loop test system to perform a servo-loop test. The special hardware involved in a servo-loop test is usually in one of the following three analog instrumentation architectures: analog dither with a digital volt meter (DVM), analog dither with a reference DAC, and DAC dither with a reference DAC[13]. Each of these variations addresses the same testing problem but has widely different execution time and accuracy due to the different analog instrumentations. The advantage of the servo-loop method is the ability of testing both pre-specified code transition edges and all code transition edges. Its disadvantage is the requirement of additional hardware if an open-loop system is used.

The fundamental difference between the servo-loop method and the histogram method is that the histogram method utilizes an open-loop system. A signal is sent to the input terminal of an ADC while the digital output is being monitored. An algorithm is applied to the set of digital data which counts the total number of occurrences for each code. The input signal is not modified in any way based on the digital output. The advantage of the histogram method is that it can be performed by ordinary test resources and does not require dedicated hardware.

2.3 Schlumberger's New Mixed Signal Tester

Schlumberger's mixed signal test system is designed based on the existing digital test system ITS9000 FX model. An analog system is designed and integrated into the existing ITS9000 FX tester to provide mixed signal functionality, while all digital functionality of the ITS9000 FX is preserved. This mixed signal tester provides testing capability for mixed signal devices at high speed range of 100 MHz for up to 448 pins[14].

2.3.1 Mixed Signal Tester's Analog System

The analog system of Schlumberger's mixed signal tester consists of four analog channels for parallel mixed signal testing. An analog channel is a complete test path for signal generation and response measurement. Each analog channel is comprised of three subsystems, namely, a digital subsystem, a DSP module, and an analog Pin Electronic (PE) subsystem[14]. Figure 2-3 illustrates the structure of an analog channel.

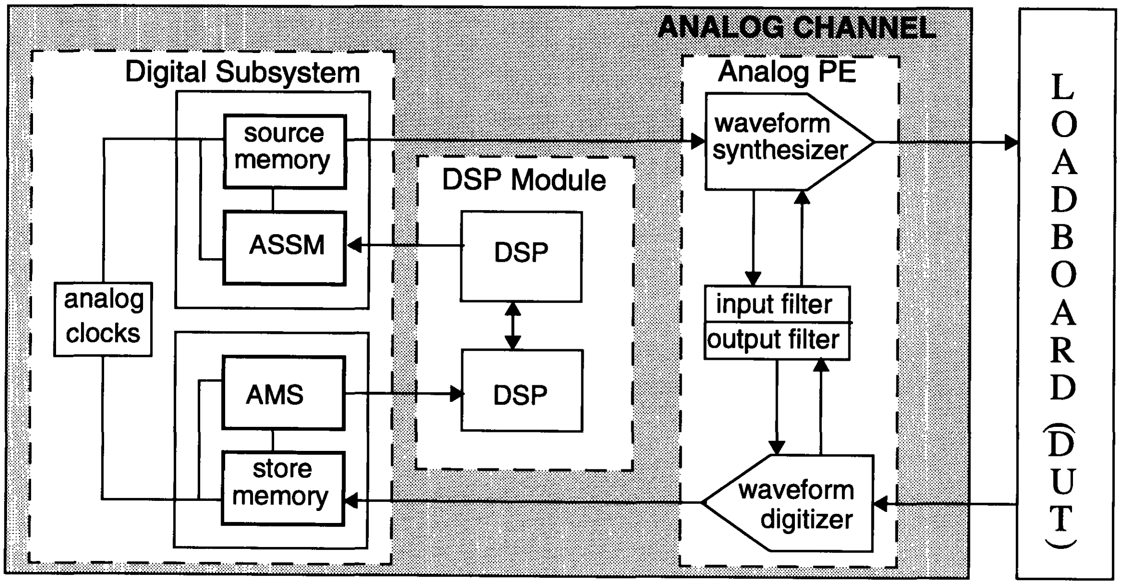


Figure 2-3: An analog channel of the new mixed signal tester.

The digital subsystem consists of memory units for source signal generation and captured data storage, an analog source sequencer module (ASSM) for high speed source signal piping, an analog measure sequencer (AMS) for captured signal piping, and analog clocks. This digital subsystem provides synchronization, data storage, and timing for an analog channel[14]. It also provides a bidirectional interface between the DUT and the DSP module.

The DSP module is the core of the mixed signal testing system. It is responsible for waveform generation and response analysis. The DSP module has a special dual-DSP architecture: one DSP is responsible for waveform generation and the other DSP is responsible for data analysis. This architecture provides a possibility of adding a feedback loop feature into the DSP module, which is discussed in greater detail in the later chapters.

The analog PE subsystem provides an interface between the DUT and the rest of the analog channel[14]. It contains a waveform synthesizer for digital to analog data conversions, a waveform digitizer for analog to digital data conversions, and filters for analog noise reduction. The waveform synthesizer and digitizer are essentially an ADC and a DAC respectively. If an analog output response of a DUT is to be measured, the analog output is first digitized by the waveform digitizer and is later sent to the DSP for analysis. A reversed procedure is performed on a waveform synthesizer if an analog input signal is required for a DUT.

Chapter 3

Existing DSP Environment

This thesis project concentrates on the feature enhancement of the DSP module of Schlumberger's new mixed signal tester. Prior to this project, the DSP module hardware setup has been completed along with some supporting software. A DSP test rig has been set up for the development and characterization of the DSP module. The DSP test rig is a complete system that isolates the DSP module from the tester, yet provides a suitable environment for DSP software development. An embedded DSP operating system has also been developed for the DSP module. This thesis project utilizes the DSP test rig as the base hardware. The goal is to modify the current DSP operating system architecture to support feedback based testing applications.

3.1 Available Hardware

The DSP test rig is comprised of a CPU board, a dual-DSP board, and a cage with 32-bit VERSAbus-E (VMEbus) for backplane connections. The DSP module of the analog system resides entirely on the DSP board. The DSP supporting software is to be developed and tested on the DSP test rig before being applied to the real tester. A diagram of the DSP test rig and its accessaries is sketched in Figure 3-1.

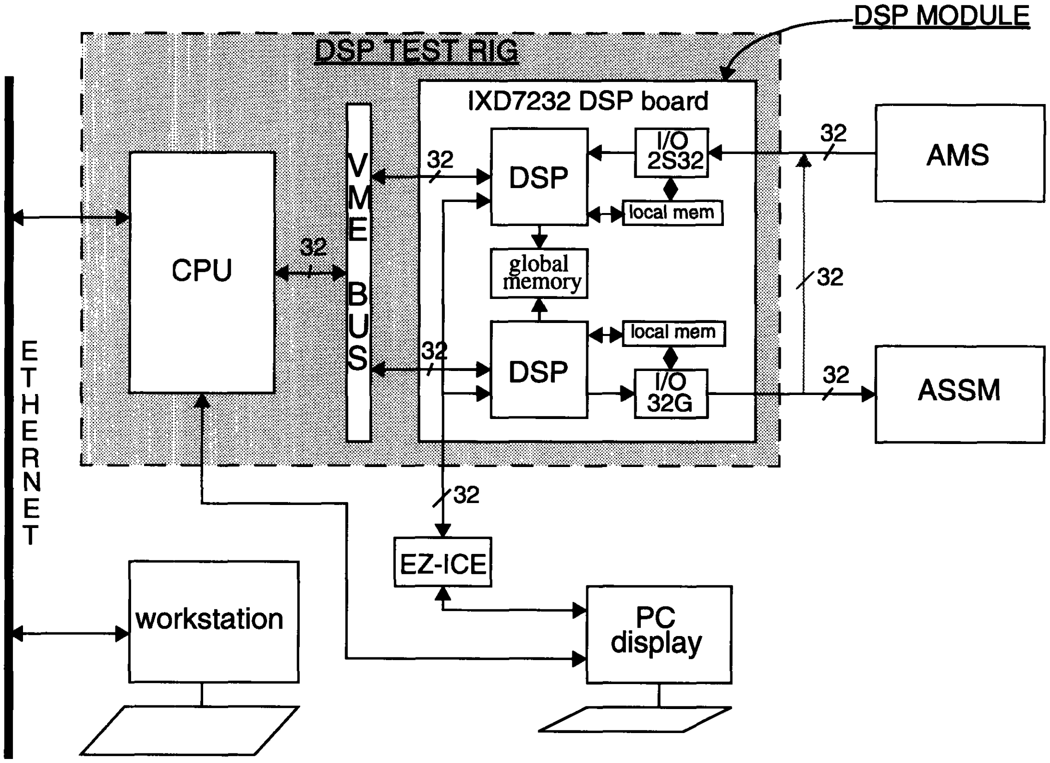


Figure 3-1: DSP test rig and its accessaries.

3.1.1 CPU

A CPU is a central processor unit that controls most of the activities in a test system. A SPARC CPU-1E/4 board by FORCE is selected to be the CPU for the mixed signal tester[14]. In a DSP test rig, a CPU is connected to the DSP module through a 32 bit VME bus to control its various activities. This hardware setting enables the CPU to access the DSP global memory for data communication. The CPU in the DSP test rig is connected to the local ethernet system so that a user can remotely log in to it from a workstation to control the DSP module. A PC is also attached to the CPU to monitor its status (Figure 3-1).

3.1.2 DSP Module: Ixthos IXD7232 board

The DSP module is built on an Ixthos IXD7232 board designed by Ixthos Inc[15]. IXD7232 is a real-time DSP VME bus board that provides the interfaces, architectural support, and board-level features required by real-time signal processing systems[16]. It utilizes two 33 MHz ADSP21020 DSPs developed by Analog Devices. A localized memory bank and an I/O port are attached to each DSP. One DSP (the upper one in Figure 3-1), called DSP_MEASURE, is used for data measurement and analysis. It is attached to an Ixthos 2S32 I/O port which has an internal memory bank. This Ixthos 2S32 I/O port captures a DUT's response and stores it locally. The other DSP (the lower one in Figure 3-1), called DSP_SOURCE, is used for waveform generation. It is attached to an Ixthos 32G I/O port which has only a register. This 32G I/O port passes data generated by DSP_SOURCE to the outside of the DSP module.

In an analog channel of the mixed signal tester, the DSP board interfaces with the analog sequencer boards (AMS and ASSM) and the CPU [15](Figure 3-1). The DSPs receive commands from the CPU through the VME bus and execute them. DSP_SOURCE interfaces with the ASSM through the 32G I/O port. Data generated on DSP_SOURCE is piped into the ASSM and is later sent to the DUT's input terminal. DSP_MEASURE interfaces with the AMS through the 2S32 I/O port. The DUT's output propagates through the AMS and is analyzed on DSP_MEASURE.

The two DSPs on the DSP module share a common global memory. This global memory is also accessible by the CPU via the VME bus. It serves as an interface among the two DSPs and the CPU so that a user is able to control the DSP activities from the CPU. This shared DSP global memory structure provides the required hardware environment for the feedback loop development.

3.2 Existing Software

3.2.1 Overview

The DSP module acts as a real-time data processor in the mixed signal tester. It generates waveforms to be sent to a DUT and measures the response from the DUT. It is essential to provide an interface that links the basic functions of a DSP with the user functions so that a user is able to program and control the DSP activities flexibly.

Schlumberger's new mixed signal tester utilizes ITS9000 software, Advanced Symbolic ATE Programming (ASAP) methodology, as the high level software tools[17].

Two levels of interface functions are created to interact ASAP functions with the basic DSP functions (Figure 3-2). Application Programming Interface (API) is a set of functions executed on the CPU. API translates user programs into a data structure that can be understood by a DSP[18]. DSP Executive is a specialized DSP operating system that schedules the tasks to be performed on a DSP. The two DSPs on the DSP board run as independent processors so each of the two DSPs carries its own DSP Executive.

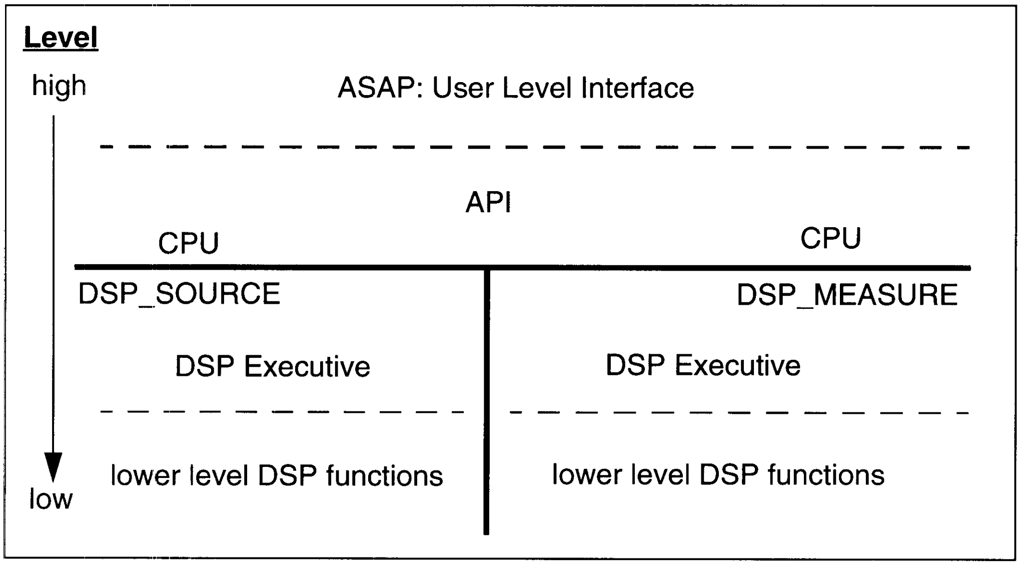

Level
high
ASAP: User Level Interface
API
CPU
CPU
DSP_SOURCE
DSP_MEASURE
DSP Executive
DSP Executive
lower level DSP functions
lower level DSP functions
low

Figure 3-2: Software interfaces between the CPU and the DSP module.

3.2.2 DSP Data Block (DDB) Structure

In order to control DSP activities from the CPU, it is essential that the information passed from the CPU is grouped into a structure that the DSPs can recognize. DSP Data Block (DDB) structure is designed to suit this purpose.

A DDB contains all necessary information describing a process to be performed on a DSP[18]. It is created on the CPU and passed onto the DSP. It contains information on algorithms to be executed on the DSP, buffers to be downloaded into the DSP, and the buffer to be captured from the DUT, etc. Before a mixed signal test is started, DSPs are initialized to an idle state waiting for inputs from the CPU. A test process is downloaded, in the format of a DDB, into the DSP global memory from the CPU through the VME bus and is later transferred into the DSP local memory. In the shared global memory, two memory blocks are allocated for the two DDBs, one for each DSP. The DSP Executive running on each DSP receives the DDB and decodes it into basic DSP functions. Figure 3-3 describes a process transfer from the CPU to a DSP using the DDB structure.

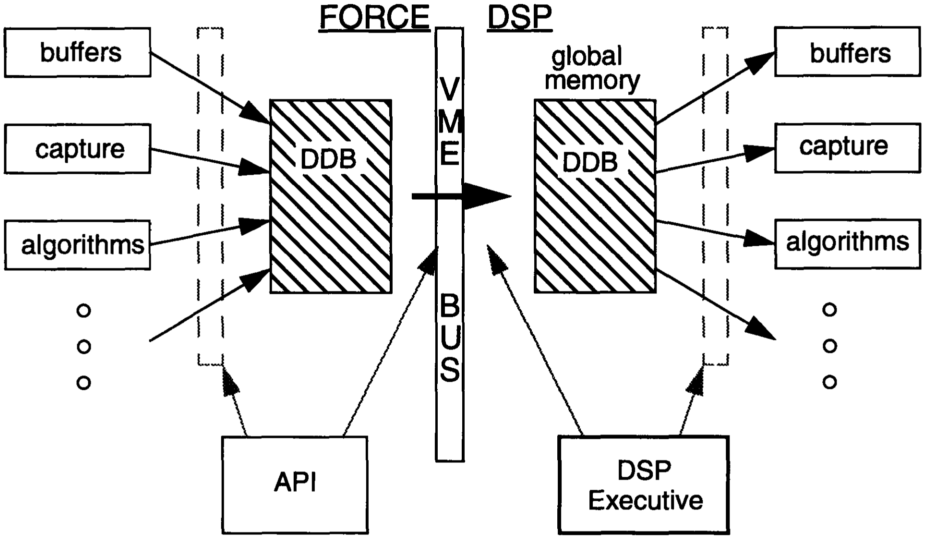
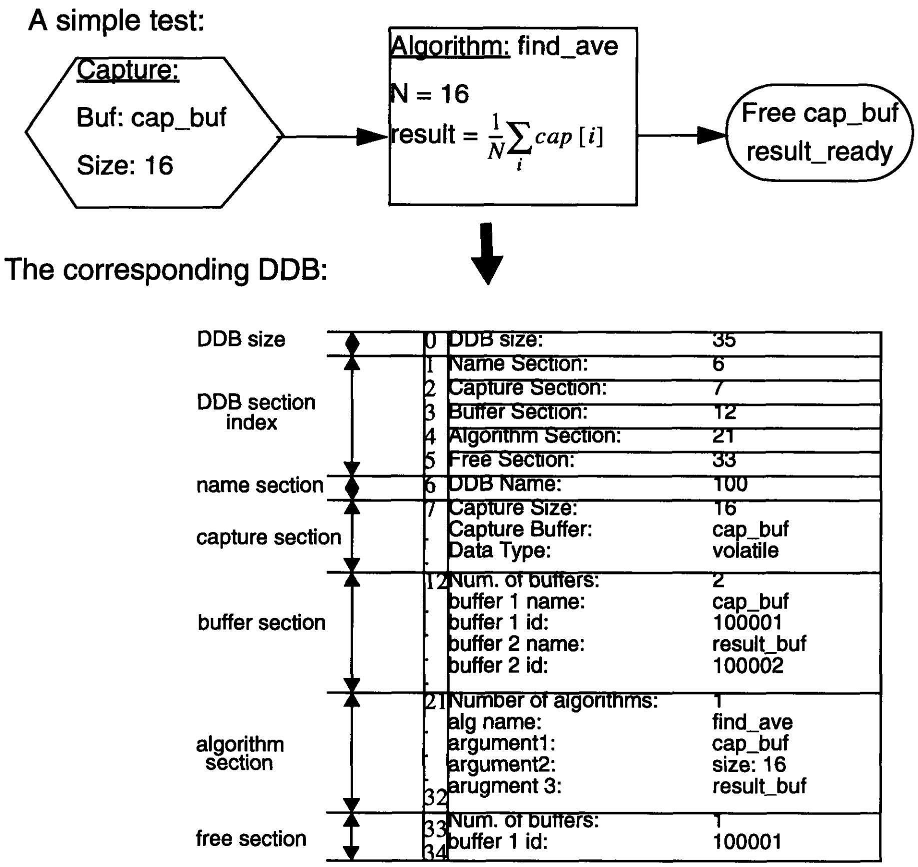


Figure 3-3: Information transfer from the CPU to a DSP through a DDB.

The current DDB structure contains six major sections, namely, Index section, Block Name section, Data Capture section, Buffer Allocation section, Algorithm section, and Free section. A DDB always starts with its size followed by the Index section which contains an index of the other five sections. The Block Name section provides the name associated with the current data block. The Data Capture section provides information on the size of the data block to be captured on the DSP from the AMS. The Buffer Allocation section provides a list of buffers that are currently allocated on the DSP and their specifications. The Algorithm section provides a list of algorithms to be executed on the DSP and their corresponding arguments. The Free section contains a list of buffers to be freed at the end of a test. These sections are created with a zero based offset; in other words, the pointer locations are based on the starting address of a DDB. Figure 3-4 gives an example of a simple test and its corresponding DDB. It is noted that each piece of the test information has been inserted into a designated section of a DDB.



DDB size	0	DDB size:	35
DDB section index	1	Name Section:	6
	2	Capture Section:	7
	3	Buffer Section:	12
	4	Algorithm Section:	21
	5	Free Section:	33
name section	6	DDB Name:	100
capture section	7	Capture Size:	16
	.	Capture Buffer:	cap_buf
	.	Data Type:	volatile
buffer section	12	Num. of buffers:	2
	.	buffer 1 name:	cap_buf
	.	buffer 1 id:	100001
	.	buffer 2 name:	result_buf
	.	buffer 2 id:	100002
algorithm section	21	Number of algorithms:	1
	.	alg name:	find_ave
	.	argument1:	cap_buf
	.	argument2:	size: 16
	32	arugment 3:	result_buf
free section	33	Num. of buffers:	1
	34	buffer 1 id:	100001

Figure 3-4: A simple test and its corresponding DDB.

Each test is translated into one or two sets of DDBs prior to the test process execution, depending on the number of DSPs involved. If only one DSP is involved, only one set of DDBs is generated and downloaded into that DSP sequentially. If both DSPs are involved, two sets of DDBs are generated, one set contains processes to be executed on DSP_MEASURE, and the other set contains processes for DSP_SOURCE. At any time during process execution, one and only one DDB exists on a DSP.

3.2.3 API

API is a higher level interface strictly used by ASAP for executing various services on a DSP. API can be divided into two sections by its functionality; namely, DSP API, which deals with DDB creation and DSP process control, and API Runtime Services, which handles I/Os from a DSP and DDB downloading at runtime.

The DSP API functions are used for the creation of a DDB and the insertion of test information into the DDB. API Runtime Services functions are called at runtime during a DDB execution. These functions include those that download and upload buffers to and from a DSP, check the status of a DSP process, etc. A lower level function is created for each of the API Runtime Service functions. This two-layer structure allows for possible future DSP module modifications. API provides a user friendly software environment for test program generation, which is helpful for the DSP module development.

3.2.4 DSP Executive

DSP Executive is a piece of software that runs on a DSP. It manages various software functions running on a DSP, such as executing algorithms stored in a DDB, downloading buffers from the CPU to the DSP, and sending result buffers from the DSP back to the CPU, etc[19]. Conceptually, DSP Executive can be categorized as an embedded operating system for a DSP. Having a specialized DSP Executive rather than an off-the-shelf generic operating system greatly improves the throughput of the mixed signal tester because it only implements the relevant functions.

The current DSP Executive can be divided into several subsets. Each subset contributes to a certain functionality of the DSP Executive. These subsets include Scheduler, Buffer Manager, DSP Host I/O, Sequencer I/O, DSP Block Handler, and Algorithm Symbol Manager[19]. Figure 3-5 shows the relationship among these DSP Executive subsets. DSP Scheduler is the heart of DSP Executive. It manages all the services provided by other subsets of DSP Executive. Buffer Manager manages all the data records used in the computations of a DSP, including buffer allocation, buffer update, and buffer deletion, etc[20]. DSP Host I/O is a mirror version of API Runtime Service functions. It executes runtime commands passed over from the CPU, such as downloading a DDB to the DSP, uploading buffers from the DSP, etc. Sequencer I/O handles data transfer between a DSP and a sequencer (AMS or ASSM); e.g., sequencer I/O sets up the I/O port on a DSP for data capture or data transmission. DSP Block Handler is responsible for algorithm executions on a DSP. Algorithm Symbol Manager keeps a list of algorithms that can be executed on a DSP.

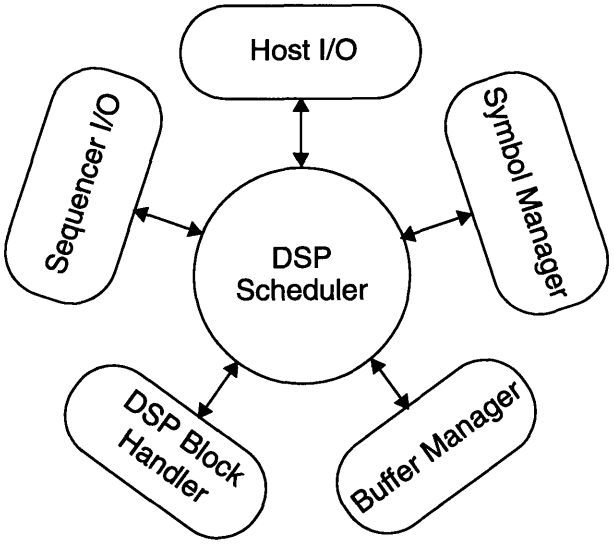


Figure 3-5: The structure of DSP Executive.

3.3 Existing Functionality

The API on the CPU and DSP Executive on the DSP module support some basic functions before this thesis project was started, including functions such as the data capture function, DSP algorithm execution, and the DSP time-out handler. Since DSP_SOURCE is not used in the first phase of the tester development, some of these existing functions can only be executed on DSP_MEASURE.

The data capture function enables DSP_MEASURE to capture a set of data piped from the AMS. This function is provided by the Sequencer I/O subset of DSP Executive. If a capture function is required for a test, details about the data to be captured is inserted into a DDB. DSP Executive then extracts the capture information from the Data Capture section of the DDB and calls the Sequencer I/O to handle the capture procedure. Data capture is done on the 2S32 I/O port of DSP_MEASURE. When the I/O port detects data flow from the AMS, it fills a pre-allocated buffer in its local memory bank with the incoming data and raises an interrupt to notify DSP_MEASURE that the buffer has been filled. The control is then passed back to the DSP Scheduler for the next process.

Algorithm execution is a major function performed on a DSP. Current DSP API and DSP Executive support the execution of about 60 algorithms. These DSP algorithms are previously defined and their algorithm symbols are stored in an algorithm symbol table. Algorithms needed for a test are inserted into a DDB. When a DSP receives the DDB, it retrieves the algorithm information from the Algorithm section of the DDB and calls DSP Block Handler for the corresponding algorithm execution. The buffer arguments for the algorithms are pre-allocated on the DSP using the Buffer Manager before the algorithm execution.

The DSP time-out handler is another useful feature developed for the DSP control. It provides the CPU the capability to abort any processes on a DSP. Under the circumstances when a DSP is waiting for data capture which never occurs, the CPU is able to stop the waiting activity and release the DSP back to its normal state[21].

Chapter 4

DSP Data Source

4.1 Overview

The objective of this thesis project is to design and implement a feedback loop system in the DSP module based on the existing structure. This feedback loop system enables real-time interactions between the two DSPs. A waveform is transmitted from DSP_SOURCE to a DUT's input terminal; the output of the DUT is captured and measured on DSP_MEASURE and then fed back to DSP_SOURCE for the next waveform generation. The current API and DSP Executive only support process executions on DSP_MEASURE. They are modified to support the process executions on DSP_SOURCE before the feedback loop system is developed.

DSP_SOURCE is responsible for waveform generation and the subsequent waveform data transmission. During a device test DSP_SOURCE creates a waveform according to the user specified parameters and sends it to the DUT through the ASSM and the waveform synthesizer (Figure 4-1).

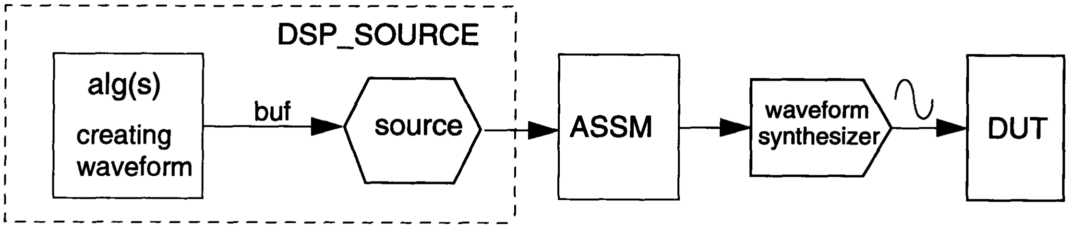


Figure 4-1: A data generation and transmission process.

A waveform is generated by executing several algorithms. Since DSP_SOURCE and DSP_MEASURE have an identical DSP structure, algorithm executions on both DSPs can be performed in a similar manner.

The data transmission process from DSP_SOURCE to a DUT is achieved by performing a data source function on the DSP. This data source function is parallel to the data capture function on DSP_MEASURE. Instead of capturing data into a DSP, the data source function takes a waveform buffer located on DSP_SOURCE and sends the buffer content sequentially to the ASSM at a high speed. The data source function is not just a reversed process of the data capture function. On DSP_MEASURE, the data capture procedure is handled by the Sequencer I/O subset of DSP Executive on the 2S32 I/O port. Since DSP_SOURCE only possesses a register based 32G I/O port, the data transmission process is performed on the DSP itself instead of on its I/O port.

4.2 DSP Data Source Function Design

4.2.1 Requirements

An ideal data source function on DSP_SOURCE should be able to handle the data transmission process flexibly. For example, in some cases it may be desirable to transmit waveforms of different frequencies using only one waveform buffer; while in some other cases it may be necessary to hold the data transmission process until the DSP receives a trigger from the CPU or from the other DSP. Therefore, the following requirements should be met:

- A user should be able to specify which buffer on DSP_SOURCE to be used to store the generated waveform data.
- A user should have an option to transmit a waveform buffer either for finite number of times or for infinite times until a test is complete.
- The source function should be able to transmit the data in a buffer selectively; e.g., to sample every n-th data in the buffer.
- A user should be given an option to specify under what condition the data transmission process should be started.

- A user should be able to specify a delay parameter to compensate the system delay in a real test.

4.2.2 A Data Source Algorithm

Similar to other DSP functions, the data source function specification must be passed from the CPU to DSP_SOURCE through a DDB. The current DDB designates a Capture section specifically for the data capture function on DSP_MEASURE and does not have a counterpart for the data source function on DSP_SOURCE. Since the data capture function is never performed on DSP_SOURCE, one option is to use the Capture section of a DDB to store the data source information for DSP_SOURCE. This strategy has its shortcomings. Since the data source function and the data capture function are completely two different functions, they require different types of information and hence cannot be converted into the same DDB format. Furthermore, it is conceptually incorrect to designate one section of a DDB for two totally separate functions.

As an alternative, the data source function is implemented as an algorithm. This strategy has a number of advantages. Firstly, the DDB need not to be reconstructed to accommodate the new information. The data source function is simply passed from the CPU to the DSP as an algorithm which can be stored in a DDB's Algorithm section. This avoids the creation of API functions for a new function handling. Secondly, no additional services need to be created on a DSP to manage the data source function. Since the function is formatted as an algorithm, the DSP Block Handler can be used to manage the data transmission process. This strategy results in a minimal change to the current DDB structure and current DSP software functions.

The prototype of the data source algorithm is

```
void source( float* source_buffer,
             int    source_buffer_size,
             int    number_of_loops,
             int    source_stride,
             int    trigger_type,
             int    delay_points).
```

This data source algorithm enables a flexible data transmission process and meet the requirements mentioned above.

- Source_buffer is a pointer to a buffer which contains the data to be transmitted. Before the source algorithm is executed, source_buffer is pre-allocated and filled with data.

- Source_buffer_size specifies the size of source_buffer.

- Number_of_loops specifies how many times the data in source_buffer is transmitted. If this argument is specified as FOREVER, the data is transmitted infinitely until being interrupted.

- The argument source_stride gives an option of transmitting data selectively. If the source_stride is specified as an integer *n*, only every *n*-th number from the source_buffer is transmitted. This parameter provides an option of transmitting waveforms of different frequencies from a single source_buffer.

- The trigger_type argument determines when the data transmission should be started. Three options are provided. TRIGGER _OFF indicates the transmission process is started immediately. If CPU_TRIGGER is requested, DSP_SOURCE holds the transmission process until a signal is received from the CPU. This option can be used to control the data transmission process from the CPU. If DSP_TRIGGER is requested, DSP_SOURCE waits for a signal from DSP_MEASURE before the data transmission process is started. This option can be used to synchronize the two DSPs during a feedback loop process.

- The argument delay_points specifies the number of data in the source_buffer to be sent before the real data is transmitted. This parameter is used to compensate the system delay during a device test.

Figure 4-2 describes the structure of the source algorithm. DSP_SOURCE first sets up the I/O port for data transmission and waits in an idle loop for the triggering signal if a trigger is required. Once a signal is received, DSP_SOURCE sends a number of data in the buffer specified by delay_points to the I/O port. Then the source_buffer pointer is reset and the real data transmission is started until the data has been sent for a pre-specified number of times or a stopping signal is received. The control is then returned back to the DSP scheduler for the next process.

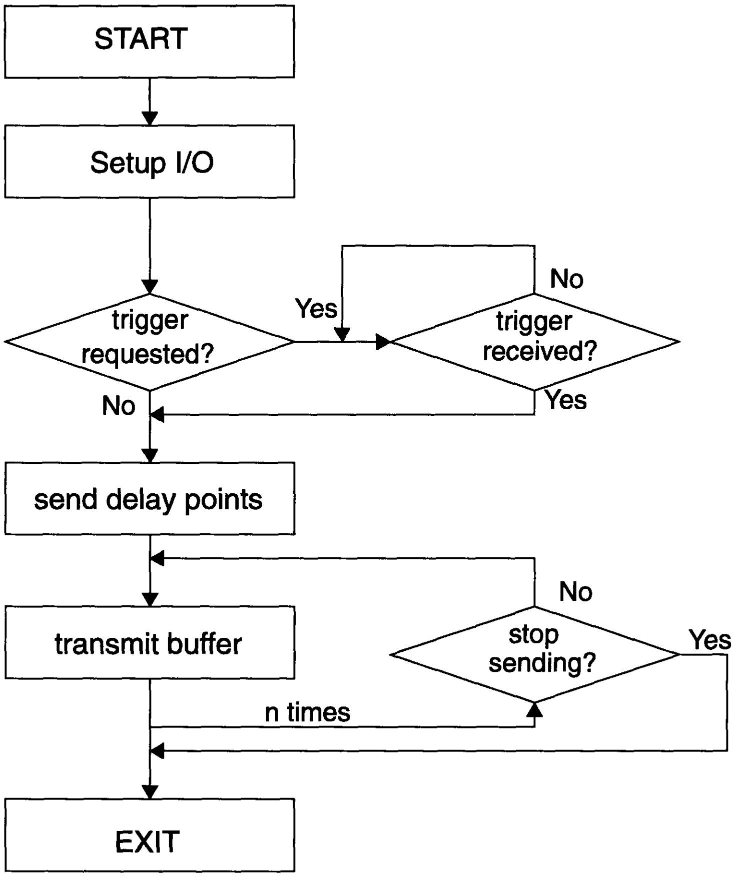


Figure 4-2: The structure of data source algorithm.

The data transmission rate is a primary parameter for the performance of the DSP module. The actual data transmission procedure in the data source algorithm is written in DSP assembly language to maximize the transmission rate. The current data source algorithm uses 8 DSP clock cycles to send one piece of data. This results in a data transmission rate of 4.125MHz, which satisfies the general specifications.

4.3 DSP Data Source Testing

An intuitive testing technique for the data source function is to measure the data sent from DSP_SOURCE with some measuring equipment. Given the dual-DSP structure of the DSP module and a working environment on DSP_MEASURE, a better strategy is to connect the I/O ports of the two DSPs together so that data transmitted from

DSP_SOURCE can be directly captured and measured on DSP_MEASURE (Figure 4-3). This strategy eliminates possible errors that may occur if an analog measuring instrument is used.

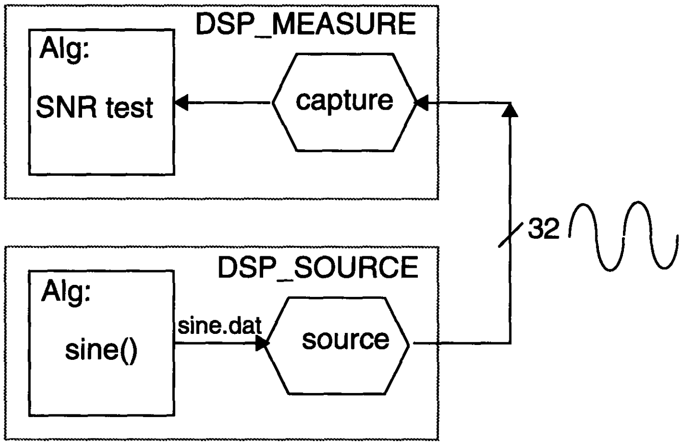


Figure 4-3: Hardware setup for data source function testing.

As shown in Figure 4-3, a sine waveform is generated and sent from DSP_SOURCE. The data is captured on DSP_MEASURE and an SNR test is performed to verify the validity of the captured data. The SNR test is a series of algorithms which calculate the signal-to-noise ratio of a data buffer. Its functionality has been previously verified. A sine waveform of 1024 points at frequency 16 cycles/buffer is sent from DSP_SOURCE and captured on DSP_MEASURE for the SNR test. The SNR test result for the captured data is calculated to be around 140 dB, which is in the same range as the input data being downloaded into DSP_MEASURE from the CPU. This test verifies that the data generated on DSP_SOURCE has been successfully transmitted into DSP_MEASURE with no missing data or data resolution loss. A number of waveforms of different sizes and frequencies are transmitted and analyzed using the SNR test to verify the flexibility of the data source function. Various testing programs are also created to test each individual data source parameter.

The data source function completes the implementation of DSP Executive for DSP_SOURCE. With a functioning environment on both DSPs, we are ready to design the DSP feedback loop system.

Chapter 5

DSP-based Feedback Loop System

5.1 Overview

In mixed signal device testing industry, there are circumstances that it is desirable to achieve a pre-specified response of a DUT and measure the corresponding input stimulus. For example, an analog filter's cut-off frequency test requires a test system to search for the -3 dB point of a filter's output. This type of device test can be performed efficiently with a feedback system. During a device test, the response from the DUT is fed back to the test system's waveform generator to create a new input stimulus such that the corresponding DUT response approaches the desired value. This feedback process is executed repeatedly until the desired result is reached.

The unique dual-DSP structure of Schlumberger's mixed signal tester is constructed to support this feedback scheme. In the DSP module DUT stimulus generator DSP_SOURCE and DUT response analyzer DSP_MEASURE share a common global memory. In other words, both DSPs have read and write permissions for this memory block. Hence, the global memory can be used as a bidirectional communication channel between the two DSPs during a feedback test.

The purpose of adding the feedback loop system to the DSP module is to provide users an option to program a closed-loop device test, such as an ADC servo-loop test, with no additional hardware instrumentation. The aim of this project is to develop a DSP module which supports both open-loop and closed-loop test applications. An open-loop test can be programmed and executed on the DSP module in which the two DSPs operate as independent processors. When a closed-loop test is executed on the DSP module, the two DSPs communicate with each other to achieve a desired test result.

5.2 Feedback System Hardware Setup

The current dual-DSP structure of the new mixed signal tester supports the feedback functionality (Figure 5-1). An analog channel of the mixed signal tester provides a path from the waveform generator DSP_SOURCE to the response analyzer DSP_MEASURE through the DUT and various digital and analog hardware such as the ASSM, the AMS, and the waveform synthesizer. To close the loop, a communication path needs to be created from DSP_MEASURE back to DSP_SOURCE. The global memory in the DSP module is used to close this feedback path. Figure 5-1 presents the closed-loop signal passage provided by an analog channel. No additional hardware is required for a closed-loop test application. However, the additional passage from DSP_MEASURE to DSP_SOURCE requires additional software functions for the system control.

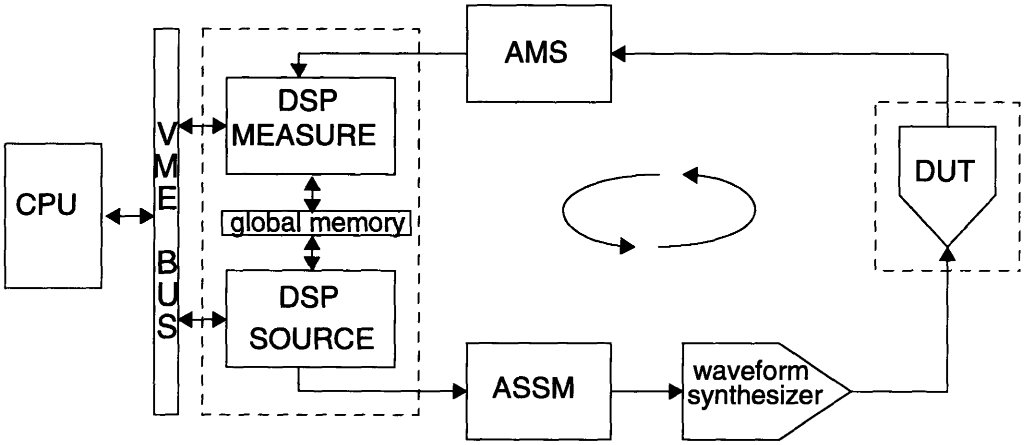


Figure 5-1: An analog channel provides a closed-loop path for feedback applications.

5.3 A Simple Example Using the Feedback System

The feedback loop system in the new mixed signal tester can be used in many applications. Many of today's complex mixed signal devices have one or more ADCs embedded in their circuitries. In order to fully measure the performance of these complex mixed signal devices, it is important that the embedded ADCs are tested extensively. A feedback loop system is useful for testing an ADC's linearity, a primary parameter of an ADC. This application is discussed in greater details in the next chapter.

The feedback loop feature can be applied in other applications also. An analog filter cut-off frequency test is a good example to demonstrate the power of the feedback loop system. The cut-off frequency of an analog filter is defined to be the input frequency at which the filter output signal achieves half of the input power, i.e., the -3 dB output power. This test searches for an input frequency at which a -3 dB power response is achieved at the output of the filter under test. This search process can be inaccurate and time consuming if an open-loop test system is used. A series of input signals of various frequencies are generated and the corresponding output power for each frequency is analyzed. The analysis result is returned to the CPU for the subsequent test pattern generation. If an accurate result is desired, a large set of input stimulus have to be generated, which may cause memory overflow. On the other hand, with a feedback loop feature added to the DSP module of a test system, the local DSP module is able to complete the search process without the CPU involvement. This results in an improved testing throughput. For this example, a user first creates a test program that describes the processes on the two DSPs, including the desired output response value 0.5 (Figure 5-2). These processes are then downloaded to the DSP module for execution. The process execution is carried out on the DSP module until the desired output is achieved. On DSP_SOURCE a low frequency sine wave is generated and sent to the filter under test. The filter output is captured on DSP_MEASURE and the input/output signal power ratio is calculated. This ratio information is fed back to DSP_SOURCE so that a waveform of a different frequency can be generated based on the feedback output power. This process is repeated until the power ratio calculated approaches 0.5 (-3 dB). The DSP module then notifies the CPU to upload the input waveform frequency, which is the actual cut-off frequency of the filter under test.

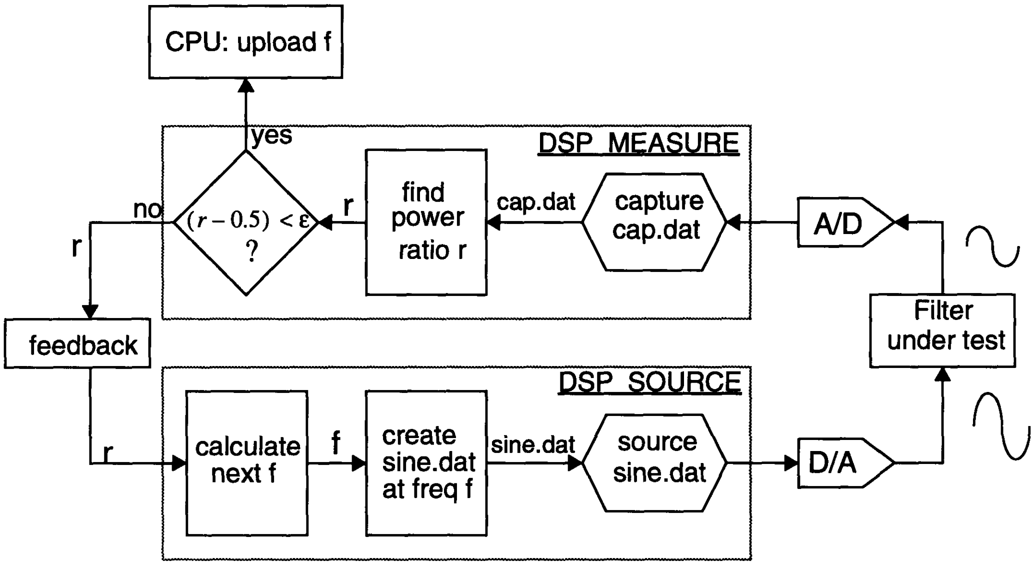


Figure 5-2: A filter cut-off frequency test example using the feedback system.

5.4 Feedback Loop System Design

5.4.1 Requirements

DSP API and DSP Executive control test program formation on the CPU and process execution on the DSP module. Currently, DSP API supports test program generation for individual DSPs. It arranges a test program into two processes, each containing test details to be performed on a DSP. DSP Executive schedules a process that resides on the current DSP and has no knowledge about the process on the other DSP. In order to support the feedback loop testing, DSP API and DSP Executive need to be modified so that the CPU and the two DSPs can interact.

The global memory shared by both DSPs and the CPU links the processors together. Serving as a communication channel, the global memory is used to pass DDBs from the CPU to the DSP module. In a feedback test the global memory is used to pass output information from DSP_MEASURE to DSP_SOURCE. This requires a close synchronization among the two DSPs and the CPU. Furthermore, since the global memory is accessible by multiple processors, it is important to restrict the memory usage to avoid memory overflow and memory corruption.

During a feedback process, information related to the current DUT response is fed back from DSP_MEASURE to DSP_SOURCE on each cycle. Since the CPU is not involved during the information feedback, DSP_MEASURE needs to be informed on what should be communicated to DSP_SOURCE and how the DSP_SOURCE should utilize the communicated data. This requires additional feedback related test information to be downloaded to the DSP through the DDB.

One of the responsibilities of a DSP is to determine when a feedback process should be terminated. The CPU should also be given the control to end a process when necessary, such as in the situation when a feedback process never meets the termination conditions in the DSP module.

5.4.2 DSP Feedback Loop Architecture

5.4.2.1 Events During a Feedback Test

A device test starts with the formation of DDBs for both DSPs on the CPU using DSP API functions. Relevant test information is inserted into the corresponding sections of a DDB for each DSP. If a test requires feedback, the feedback related details is also inserted into the DDB, including information to be passed to DSP_SOURCE and the loop termination conditions.

After a DDB is downloaded into a DSP, DSP Executive takes control of the process execution. It arranges the test details stored in the DDB and executes them in the pre-scheduled order. Table 5-1 lists the events that occur on DSP_SOURCE and DSP_MEASURE during a feedback loop based test.

Table 5-1: DSP Executive events during a feedback test

STATE	DSP_MEASURE	DSP_SOURCE
IDLE STATE	DSP idle	DSP idle
	--	--
INIT STATE	--	Setup Source DDB
	--	Execute Initialization Algorithms
	--	Wait for DSP_MEASURE trigger
	Setup Measure DDB	--
	Execute Initialization Algorithms	--
	Trigger DSP_SOURCE	--
LOOP STATE	--	Execute Loop Algorithms
	--	Start data source
	Setup I/O for data capture	--
	Acquire data	--
	Execute Loop Algorithms	--
	Check for termination conditions.	--
	Jump to exit state if conditions are met	--
	Interrupt data source on DSP_SOURCE	Stop data source
	---	Jump to exit state if notified
	Send feedback info. to DSP_SOURCE	Receive info. from DSP_MEASURE
	Back to beginning of loop state	Back to beginning of loop state
EXIT STATE	Interrupt data source on DSP_SOURCE	--
	--	Stop data source
	DSP ready for results	DSP ready for results

During a feedback test, each DSP is in one of the four states, namely, idle state, init state, loop state, and exit state. The two DSPs are always in the same state at runtime. During tester initialization two identical copies of DSP Executive are downloaded into DSP_SOURCE and DSP_MEASURE, and the DSPs are initialized to an idle state waiting for inputs from the CPU. When the feedback test DDBs are downloaded into the DSPs, DSP Executive on each DSP enters the init state. In the init state, DSP Executive sets up the DDB in its local memory, allocates required data buffers, and executes initialization algorithms for initial buffer data generation. The two DSPs run asynchronously in the init state because no interactions between the DSPs are required. Once the initialization processes are completed on both DSPs, a trigger from DSP_MEASURE to DSP_SOURCE sends both DSPs into the loop state (Table 5-1). On DSP_SOURCE, the Loop Algorithms are first executed to generate a waveform, which is later transmitted infinitely to the DUT until an interrupt is received from DSP_MEASURE. On DSP_MEASURE, upon the capture of DUT output response, a series of Loop Algorithms are executed for output data analysis. The termination conditions are also examined during the Loop Algorithm execution. If the termination conditions are met, DSP Executive leaves the loop state and enters the exit state. If the termination conditions have not yet reached, an interrupt is sent from DSP_MEASURE to DSP_SOURCE to stop the infinite data source function to prepare for the information feedback. The two DSPs are then synchronized to transfer output information from DSP_MEASURE to DSP_SOURCE. After the data transfer, both DSPs resume their own processes in the loop state. In the exit state, DSP_MEASURE informs DSP_SOURCE to stop its current activities and prepare for the result uploading. DSP_MEASURE also notifies the CPU that the feedback process has been completed.

It is noted that the CPU is not involved during the feedback process execution on the DSP module. After the DDBs have been created and downloaded to the DSPs, the CPU is released from the process control until notified by the DSP module about the process execution completion. This arrangement enables the CPU to perform other processes in parallel with the DSP feedback process and thus increases the overall test throughput.

5.4.2.2 New DDB Structure

Feedback loop based testing calls for an extension of the existing DDB structure to accommodate the additional feedback related information. The new DDB structure is comprised of nine sections, of which the first six sections remain unchanged from the previous DDB structure. The three additional sections of the new DDB structure include an Initialization Algorithm section, a Source Feedback Algorithm section, and an Exit Algorithm section. The Initialization Algorithm section contains algorithms to be executed prior to the feedback loop process in the init state, such as algorithms that generate initial buffer data. The Source Feedback Algorithm section contains algorithms which exist on the corresponding source DDB and receive information from DSP_MEASURE during a feedback process. This Source Feedback Algorithm section is only valid for measure DDB. The Exit Algorithm section contains a list of algorithms which are used to determine the termination of a feedback loop process. Figure 5-3 shows the new DDB structure with the additional sections indicated.

5.4.2.3 Initialization Algorithms and Loop Algorithms

A feedback process requires a series of algorithms (Loop Algorithms) to be executed repeatedly on each DSP until a desired response is received from the DUT. It is also necessary to provide algorithms (Initialization Algorithms) that perform buffer initialization before the loop process is started. A new section called Initialization Algorithm section is created for this purpose. During the DDB formation of a feedback test, the specifications of the algorithms used for buffer initialization are inserted into this section. The regular algorithm section in a DDB is used to store the specifications of Loop Algorithms used in a feedback process. The specifications of Initialization Algorithms and Loop Algorithms are inserted into a DDB using two different DSP API functions.

The DSP algorithms executed in the DSP module must be defined and stored *a priori* in a tester. An algorithm symbol table is created to store information of all available DSP algorithms and their specifications. Three identical copies of algorithm symbol tables are generated: one copy on the CPU, one copy on DSP_SOURCE, and one copy on DSP_MEASURE. Hence, the DSPs can execute the algorithms specified by a user using the algorithm symbol table at runtime. If an additional algorithm is needed for a device test, this algorithm should first be inserted into the algorithm symbol table before it can be executed on the DSP. Initialization Algorithms and Loop Algorithms have an identical algorithm structure, and are both stored in the algorithm symbol table. During a process execution on a DSP, an algorithm is identified from the algorithm symbol table by its algorithm ID and is executed.

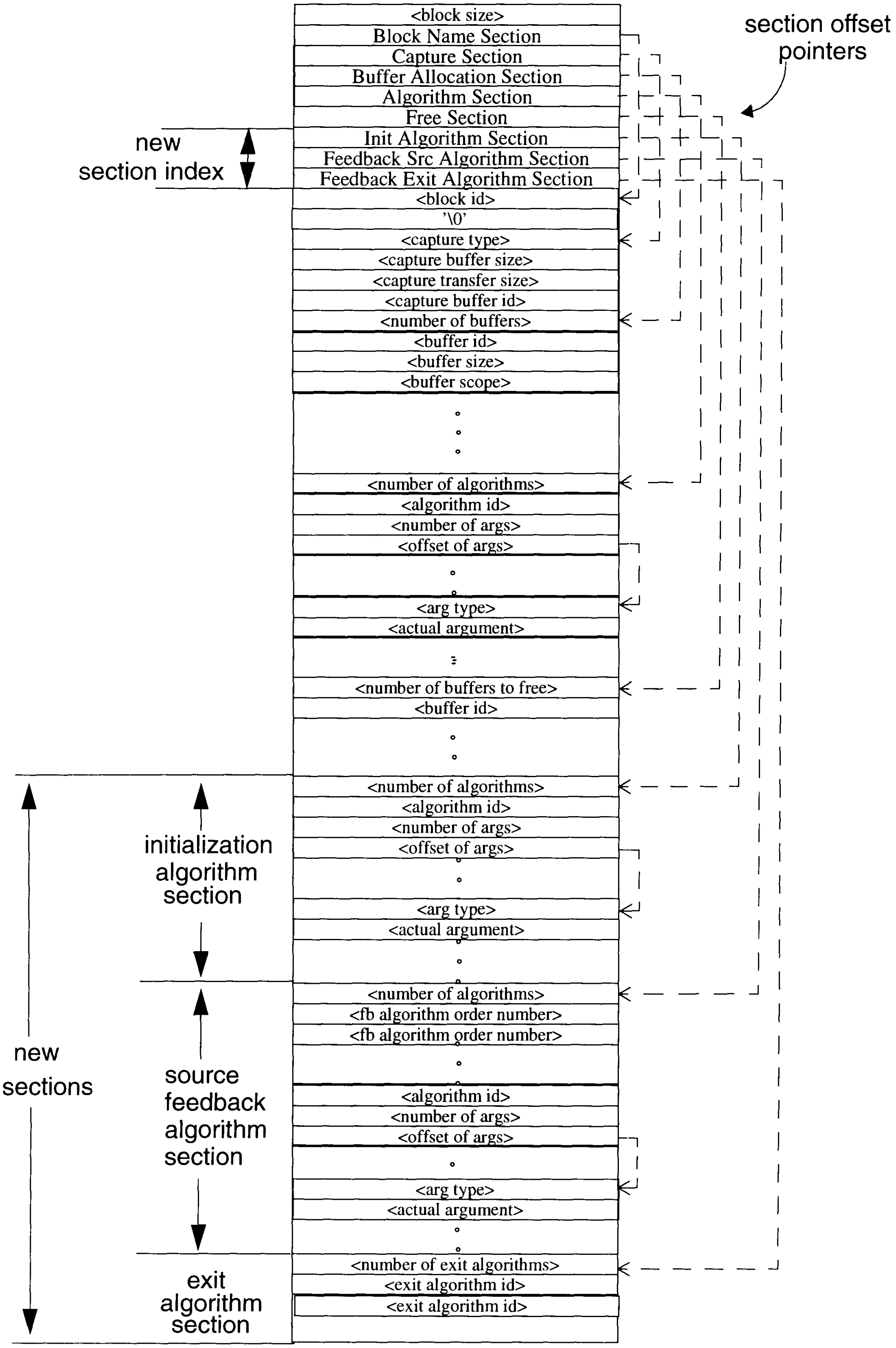


Figure 5-3: The new DDB structure.

5.4.2.4 Exit Algorithms

A special type of Loop Algorithm, called Exit Algorithm, is created as a decision maker for a feedback loop process. An Exit Algorithm determines when a feedback loop process should be terminated. It is analogous to the 'if' statement in computer programming, except that when the condition is met the only option is to terminate the process. An Exit Algorithm has the same algorithm structure as an Initialization Algorithm or a Loop Algorithm in the algorithm symbol table, with the exception that an Exit Algorithm always returns a value while a loop algorithm always returns void. When a feedback loop test is created, an Exit Algorithm in which the loop termination condition is specified is downloaded into the DSP module. During algorithm execution time, the Exit Algorithm is used to evaluate the DUT output status against the termination condition. If the termination condition is not yet met, a non-zero number is returned indicating the feedback process should continue. If the termination condition is met, a zero is returned indicating the feedback loop process should exit. In the case when there are multiple termination conditions, more than one Exit Algorithms can be used.

Similar to other regular Loop Algorithms, Exit Algorithms are inserted into the algorithm section of a DDB during a test creation. The algorithm IDs of these Exit Algorithms are inserted into the Exit Algorithm section of a DDB so that DSP Executive can distinguish an Exit Algorithm from a regular Loop Algorithm at runtime.

5.4.2.5 Source Feedback Algorithms

In order to pass information from DSP_MEASURE to DSP_SOURCE in the format that can be understood, it is necessary to provide DSP_MEASURE the information of corresponding process on DSP_SOURCE. A section is added into the DDB structure for this purpose. This section contains algorithms that run concurrently on DSP_SOURCE and whose arguments are influenced by the DUT response captured on DSP_MEASURE. DSP_MEASURE passes these algorithms with their updated arguments to DSP_SOURCE at each feedback cycle. DSP_SOURCE then executes these algorithms instead of the corresponding ones stored in its DDB so that the output information embedded in the arguments can be incorporated into the new waveform.

The structure of Source Feedback Algorithm section is identical to a regular algorithm stored in a DDB. This is essential because DSP Executive on DSP_SOURCE executes these algorithms instead of the corresponding Loop Algorithms in the DDB.

5.4.2.6 Feedback Loop Counters

Under some circumstances, a feedback process may not converge as a user expected. A loop counter is set up to ensure a process termination when the normal termination conditions have not been met after a finite number of cycles. A maximum cycle count is specified when a feedback test is formulated, and is decreased by one after each cycle during runtime. The feedback process is terminated when the counter reaches zero, no matter whether the termination conditions for the current process have been met or not.

5.4.3 Implementation

The implementation of the feedback loop system consists of modifications to the two existing software modules: DSP API and DSP Executive. DSP API should be modified such that it supports the additional sections in the new DDB structure as well as the existing functionality. At the same time, DSP Executive should be modified so that it supports all the functions and communications related to a feedback loop process, as well as existing non-feedback test processes.

5.4.3.1 DSP API

DSP_MEASURE is selected to be the driver of the feedback loop process because it captures the output of the DUT. All feedback related information is inserted into the measure DDB only. Several additional API functions are created for inserting feedback information into a measure DDB. An API function is developed to initialize a DDB for a feedback test. This function initializes the DDB with feedback related parameters, such as maximum feedback loop counter and the corresponding source DDB name. Other API functions are developed for inserting Initialization Algorithms, Exit Algorithms, and Source Feedback Algorithms into the corresponding sections of a DDB. Since all the feedback information is downloaded into DSP_MEASURE only, all these functions are for measure DDB generation only.

5.4.3.2 DSP Executive

DSP Executive manages all the DSP activities such as data transmission, data capturing, and algorithm execution. DSP Executive should be modified to manage the additional feedback related process embedded in a DDB. On the other hand, the modified DSP Executive should also support the existing non-feedback test processes.

DSP Executive is reconstructed to accommodate both feedback and non-feedback processes. Both situations can be handled easily by storing Loop Algorithm specifications in the regular algorithm section of a DDB. Figure 5-4 illustrates the handling of both feedback and non-feedback tests by the Scheduler of the DSP Executive.

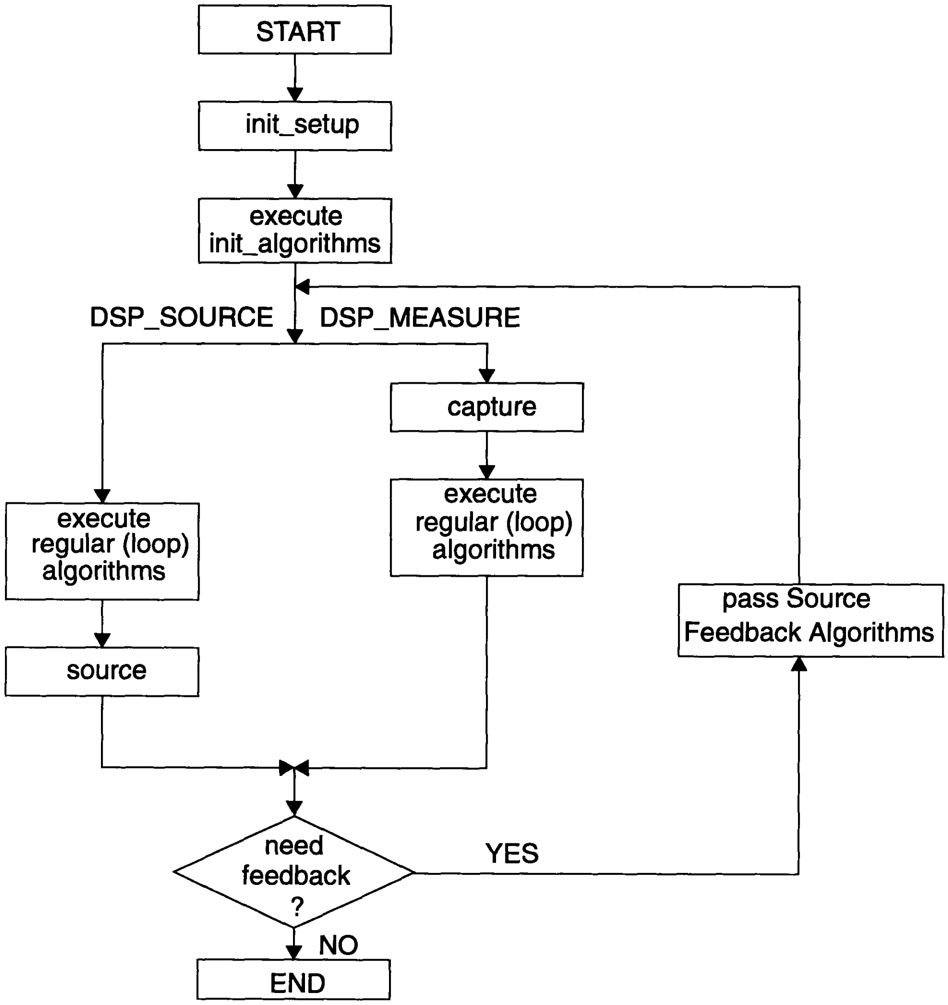

START
init_setup
execute init_algorithms
DSP_SOURCE
DSP_MEASURE
capture
execute regular (loop) algorithms
execute regular (loop) algorithms
source
pass Source Feedback Algorithms
need feedback ?
YES
NO
END

Figure 5-4: The structure of DSP Scheduler.

The two DSPs execute a non-feedback process in an asynchronous manner. On DSP_SOURCE a waveform is generated by executing several algorithms and is later transmitted to the input terminal of the DUT; while on DSP_MEASURE the DUT's output is captured and evaluated using several algorithms. If a feedback process is to be executed, an additional feedback process is performed. After the output analysis, DSP_MEASURE passes new waveform specifications to DSP_SOURCE, along with the associated algorithm names, and DSP_SOURCE executes the corresponding algorithms with the new parameters.

Information feedback from DSP_MEASURE to DSP_SOURCE is the most essential part of a feedback process. Source Feedback Algorithms are passed from DSP_MEASURE to DSP_SOURCE in each cycle. The output arguments of the Loop Algorithms and the input arguments of the Source Feedback Algorithms share the same buffers. So when the Loop Algorithms are executed on DSP_MEASURE, their output argument buffers are filled with the captured output information. These buffers are later transferred to DSP_SOURCE as input argument buffers of the Source Feedback Algorithms. The Source Feedback Algorithms are then executed on DSP_SOURCE to produce the next waveform to be sent to the DUT.

Due to the memory limitation of the global memory, feedback information can not be transferred all at once. It is done sequentially. The Source Feedback Algorithm specifications are transferred first, followed by the corresponding buffer arguments. It is very important to keep the two DSPs in synchronization so that the transfer process is performed without encountering dead lock situations. Two memory cells are set up as communication channels for handshake synchronization in the global memory.

Figure 5-5 shows the information feedback operation on the two DSPs and the global memory. When DSP_MEASURE completes the Loop Algorithm execution and is ready for the Source Feedback Algorithms transfer, the following procedures are performed on the two DSPs:

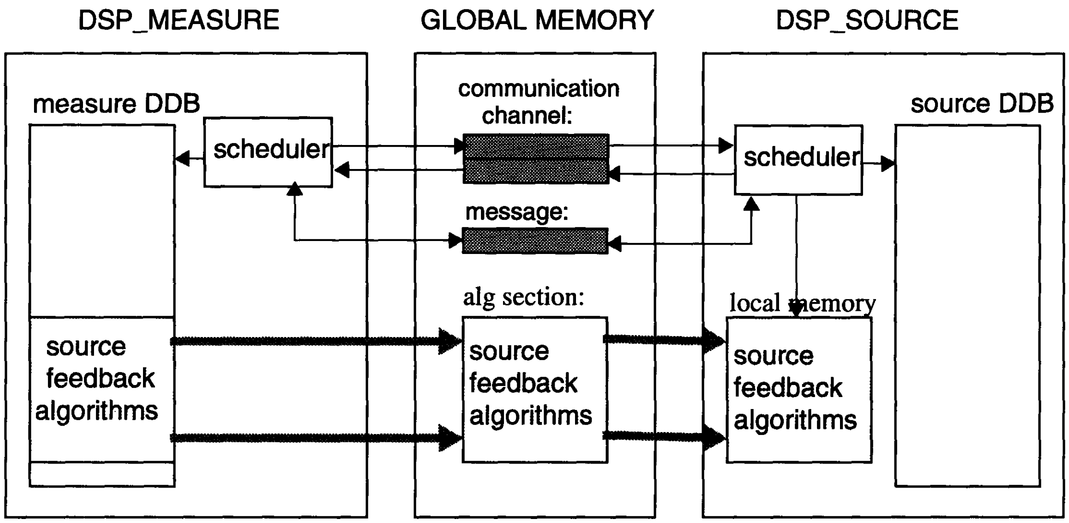


Figure 5-5: Information feedback process.

1. DSP_MEASURE writes "stop source" in the communication channel to stop the data transmission process on DSP_SOURCE.
2. DSP_SOURCE receives "stop source" from the communication channel and stops the data source function and prepares for the information transfer.
3. DSP_MEASURE writes Source Feedback Algorithm specifications into the specified algorithm section in the global memory and signals DSP_SOURCE "feedback algorithms ready" through the communication channel.
4. DSP_SOURCE receives the "feedback algorithms ready" signal and copies the Source Feedback Algorithm specifications into its pre-allocated local data memory.
5. DSP_SOURCE checks each argument of the Source Feedback Algorithm specifications. If an argument is a buffer, it writes the buffer ID in the message location and signals "buffer request" in the communication channel.
6. DSP_MEASURE receives "buffer request" signal and retrieves the requested buffer ID from the message location of the global memory. If the buffer is found, the content of the buffer is copied into a temporary location in the global memory. The buffer location address is then inserted into the message location and a

"buffer ready" signal is sent to the communication channel. If the buffer is not found (unchanged by DSP_MEASURE), "buffer not exist" is written to the communication channel.

7. If a "buffer ready" is received on DSP_SOURCE, the corresponding buffer content is moved from the global memory into the location where the buffer with the same ID is stored. If a "buffer not exist" is received, step 5 is repeated until all the arguments have been updated.

During the algorithm transfer process, when one DSP writes a message into the communication channel, the other DSP will be always waiting for the message. This avoids any dead lock situations.

5.5 Feedback Loop Testing and Analysis

A real-time error correction module is developed to test the feedback functionality of the DSP module. The structure of the real-time error correction module is shown in Figure 5-6. This module utilizes the feedback architecture of the DSP module and a differential amplifier to correct error signals generated by an external signal generator.

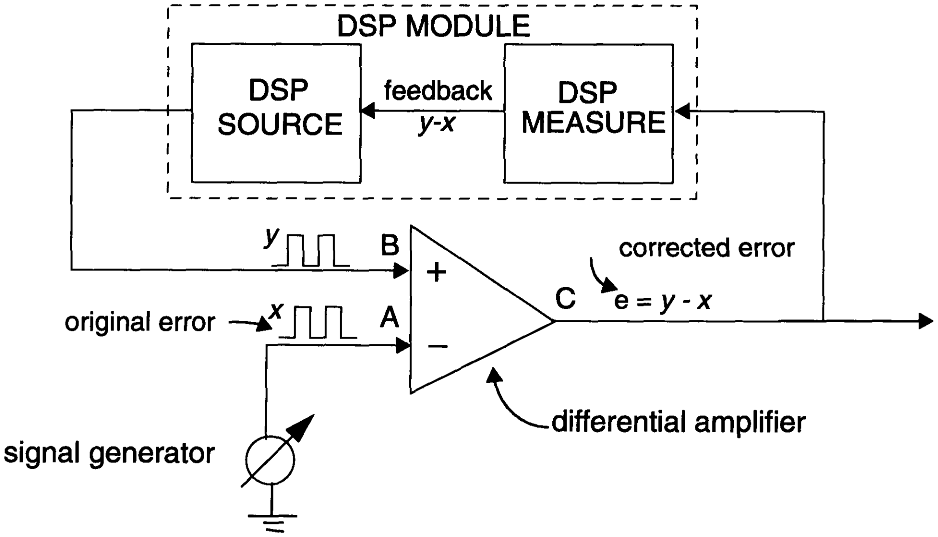


Figure 5-6: A real-time error correction module.

A signal generator simulates an error signal into the negative input terminal (A) of the differential amplifier. The goal is to control the signal into the positive input terminal (B) of the differential amplifier so that the corrected error signal at the output terminal (C) is as small as possible. The error correction process is accomplished by monitoring the corrected error at output C and accordingly modulating the signal to the input B using the DSP module. If the DSP feedback system is implemented properly, the signal into input B should trace the behavior of the time-varying error signal at input A. For example, if a triangular wave is driven into input A, a triangular wave of same frequency should appear at input B with some time delay. Appropriate algorithms need to be designed for the DSP module so that the signal into input B always corrects the original error signal into input A and produces a very small error at output C.

Under the simplest case, assume during the *n*-th cycle the signal generator is generating a constant error signal *x[n]* into point A. Given that the DSP is sending a constant signal *y[n]* into point B, the corrected error at point C is then *e[n] = y[n] - x[n]* (Figure 5-6). DSP_MEASURE captures this error and passes it to DSP_SOURCE. Based on this error *e[n]* and the previous source data *y[n]*, a new source data *y[n+1]* is generated by evaluating *y[n+1] = y[n] - e[n] = y[n] - (y[n]-x[n]) = x[n]*, thus produces a new error of *e[n+1] = y[n+1] - x[n] = x[n] - x[n] = 0*. If the signal generator is driving a time-varying signal at some fixed frequency into point A, this error correction module functions in a similar manner until the original error signal frequency reaches the Nyquist frequency of the feedback system, which is half of the updating frequency of the DSP module.

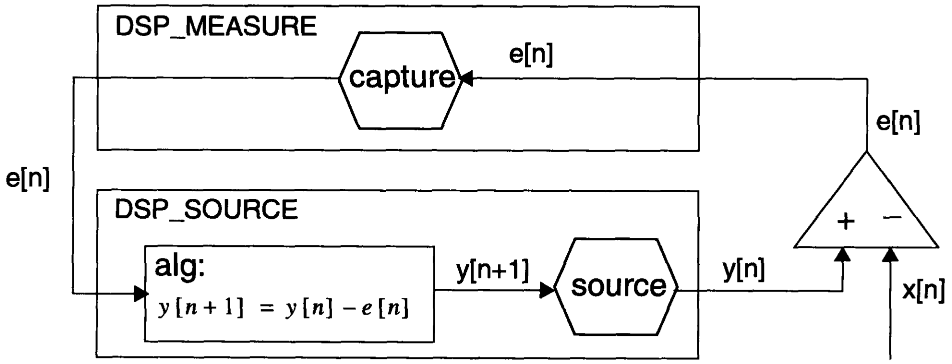


Figure 5-7: Error correction test block diagrams.

The test program developed on the DSP module for real-time error correction is given in Figure 5-7. On DSP_MEASURE, only the data capture function is needed for error collection. DSP_MEASURE passes the captured data e[n] into DSP_SOURCE. Based on this error and the previous data y[n], a new data y[n+1] is calculated and sent to the positive terminal of the differential amplifier.

This test is carried out by driving a low frequency triangular wave into the point A from the signal generator. The output of the DSP_SOURCE is converted into an analog signal and displayed on an oscilloscope. Figure 5-8 is a sketch of the waveforms at point A and B displayed on the oscilloscope. It is noted that the signal sent out from DSP_SOURCE to point B is tracing the triangular wave at point A with some time delay. The jigsaw pattern of signal at point B is caused by the latency of the DSP feedback. The frequency of the signal driven into point A is manually increased and the response at point B is monitored. The signal at point B tracks the signal at point A until a frequency of 600 Hz is reached. This implies that the current feedback loop system has an average updating rate of 1200 Hz (600 x 2). The updating rate varies depending on the complexity of the algorithms on each DSP and the associated buffer sizes.

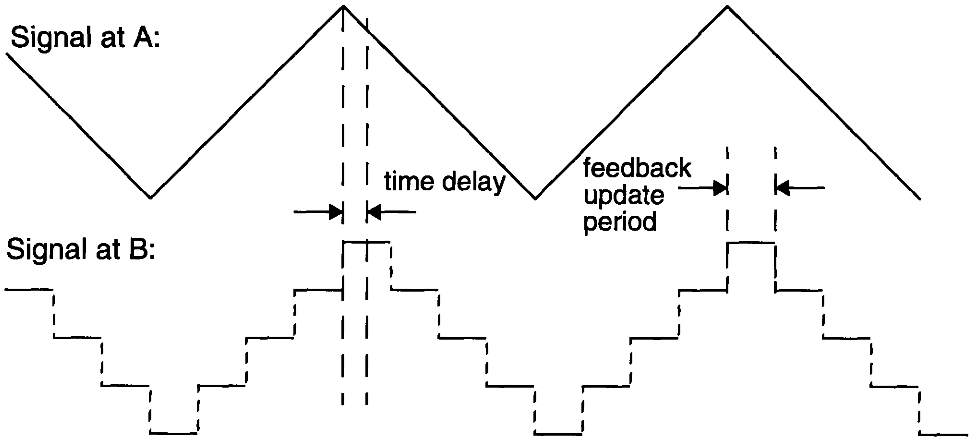


Figure 5-8: Oscilloscope display of error tracking signals.

In order to verify that the modified DSP module supports both open-loop and closed-loop test applications, the programs designed for testing the data source function in Chapter 4 are executed using the modified DSP software functions. The test results show that the DSP module still supports the existing open-loop applications.

Chapter 6

Feedback Application: An ADC Servo-loop Test

In this chapter, an ADC servo-loop test is developed to verify the feasibility of the feedback loop system in Schlumberger's mixed signal tester. An ADC test is one of the primary applications of mixed signal device testing. The goal of an ADC servo-loop test is to search the transition edge for a particular ADC output code. Based on the characteristics of an ADC, a range of analog input is mapped into a particular digital output code. The ADC code transition edge search can be very difficult and time consuming if an open-loop test system is used. However, if a feedback enabled test system is present, the ADC servo-loop test can be implemented in a much simpler and faster way.

6.1 Hardware Setup

To develop an ADC servo-loop test, an ADC under test and a DAC are required as well as the DSP test rig. The DAC is used to convert the waveform generated on DSP_SOURCE into an analog signal to be sent to the input terminal of the ADC under test. For simplicity, the ADC embedded in the waveform digitizer of the tester's analog PE (Pin Electronic) subsystem is used as the ADC under test. This eliminates a loadboard to be setup for the DUT. The waveform synthesizer of the analog PE subsystem is selected to perform the digital to analog signal conversions for the test. Figure 6-1 shows the hardware setup for the ADC servo-loop test.

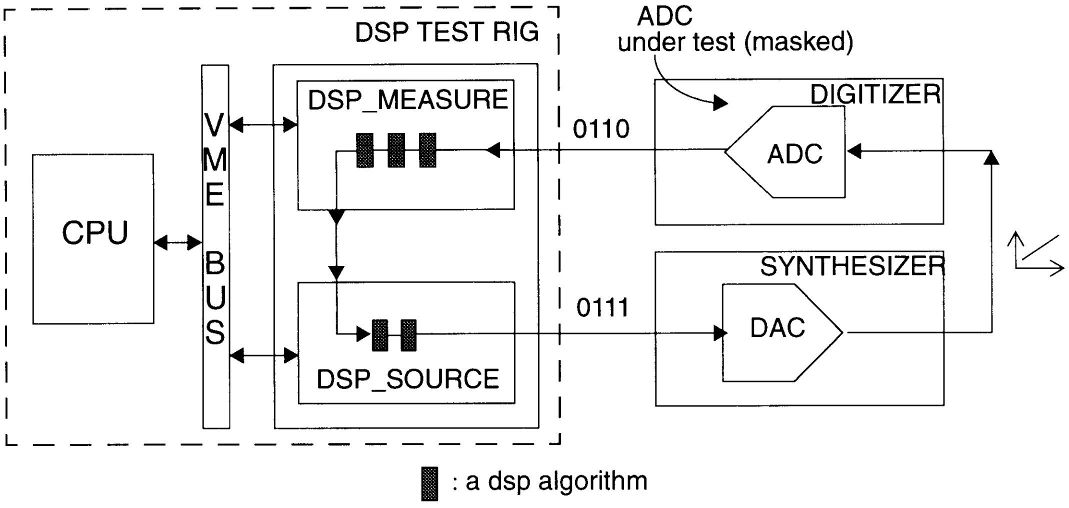


Figure 6-1: An ADC servo-loop test process.

The waveform synthesizer utilizes a 20-bit DAC with an adjustable voltage range for high resolution digital to analog signal conversions. The waveform digitizer instead utilizes a 20-bit ADC. One of the requirements for an ADC servo-loop test is that the analog signal into the input terminal of the ADC under test must have a higher resolution than the ADC's LSB size. Therefore, in order to meet this requirement, the last several bits (the exact number of bits can be specified by a user) of the ADC under test in the waveform digitizer are masked using a DSP algorithm to give a bigger LSB size. Several algorithms are generated and downloaded into the two DSPs in the DSP module. A waveform is generated on DSP_SOURCE, converted into an analog signal, and sent to the input terminal of the ADC under test in the waveform digitizer. The digital output of the ADC under test is captured and evaluated on DSP_MEASURE, and later fed back to DSP_SOURCE for the next waveform generation.

6.2 ADC Servo-loop Test Design

Figure 6-2 is a flow chart for an ADC code transition edge search using the servo-loop method.

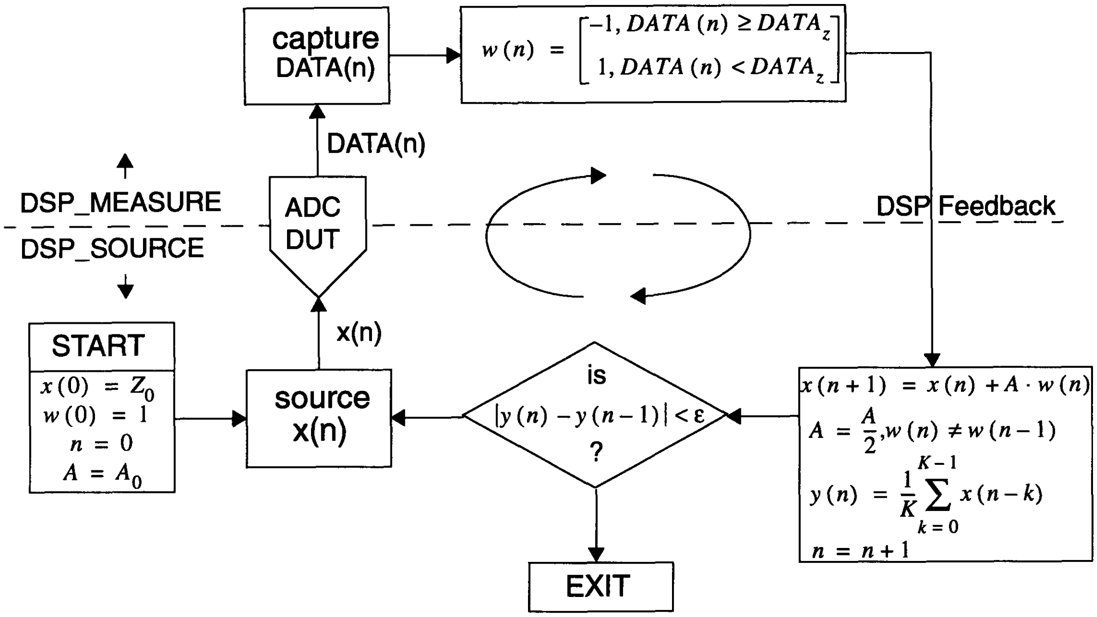

capture
DATA(n)
$w(n) = \begin{bmatrix} -1, DATA(n) \geq DATA_z \\ 1, DATA(n) < DATA_z \end{bmatrix}$
DATA(n)
DSP_MEASURE
DSP_SOURCE
ADC
DUT
DSP Feedback
START
$x(0) = Z_0$
$w(0) = 1$
$n = 0$
$A = A_0$
x(n)
source
x(n)
is
$|y(n) - y(n-1)| < \varepsilon$
?
$x(n+1) = x(n) + A \cdot w(n)$
$A = \frac{A}{2}, w(n) \neq w(n-1)$
$y(n) = \frac{1}{K}\sum_{k=0}^{K-1} x(n-k)$
$n = n+1$
EXIT

Figure 6-2: An ADC servo-loop test flow chart.

Suppose a transition edge search for output code $DATA_Z$ is required. DSP_SOURCE first starts with sending an initial data value *x(0)* to the ADC under test. During the feedback process, for example the n-th cycle, a source data value *x(n)* is sent to the ADC under test and the corresponding output code *DATA(n)* is captured by DSP_MEASURE. The captured code *DATA(n)* is compared with the designated code $DATA_Z$ and the result *w(n)* is fedback to DSP_SOURCE. A new source data *x(n+1)* is generated on DSP_SOURCE based on *w(n)*. If the previous code response *DATA(n)* is less than the specified code response $DATA_Z$ (i.e., *w(n)* = 1), the next data to be sent to the DUT, *x(n+1)*, will be one step higher than the previous value, and vice versa. The step size *A* is determined by the change in *w(n)* from cycle to cycle; i.e., if *w(n)* changes from 1 to -1 or -1 to 1, the step size *A* is halved. A changing step size in a feedback process allows for the input source data to converge under any termination threshold levels. The step size is decreased when the input data crosses the code transition edge so that a finer search can be performed. An average of last *K* samples of *x(n)*, *y(n)*, is calculated in each cycle. The change in the moving average *y(n)* from *y(n-1)* serves as the process termination threshold (ε) for the ADC servo-loop test. As the source data *x(n)* and the moving average *y(n)* converge towards the code transition edge, the output code starts oscillating between $DATA_Z$ and the code below due to the fact that the input signal *x(n)* crosses the code transition edge at each cycle. Figure 6-3 illustrates the converging process of the ADC code transition edge search using the scheme described above. It is noted that while *x(n)* converges towards the desired code transition edge, the corresponding output codes oscillates between the two adjacent codes.

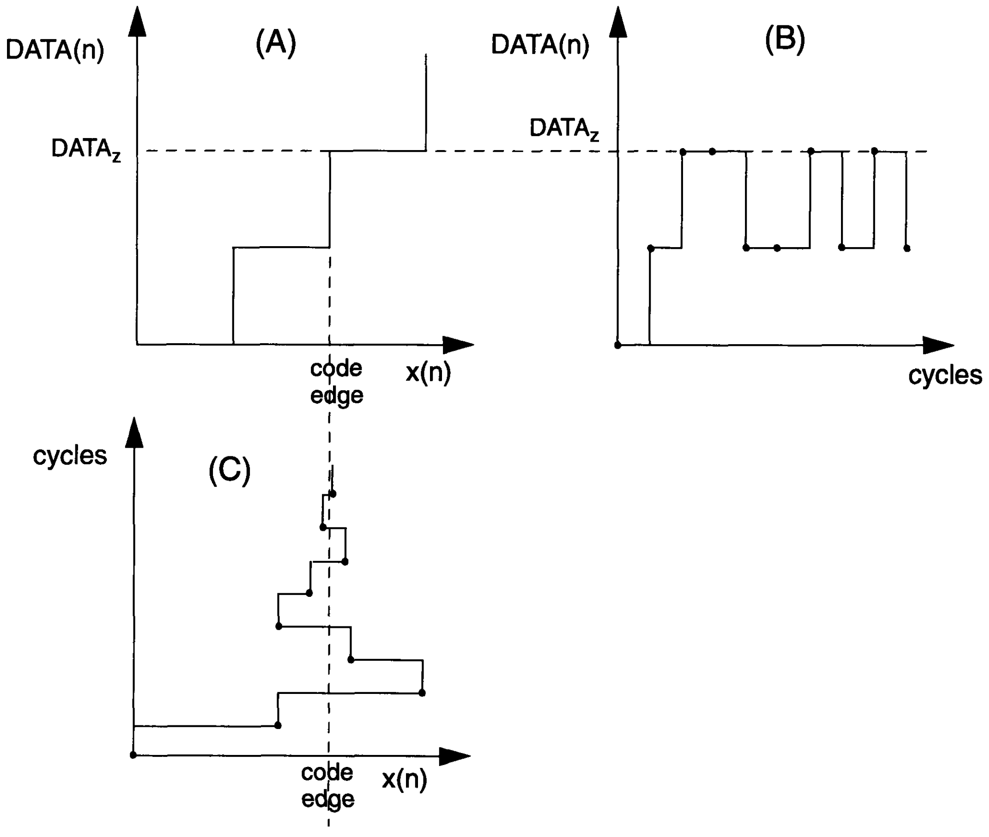
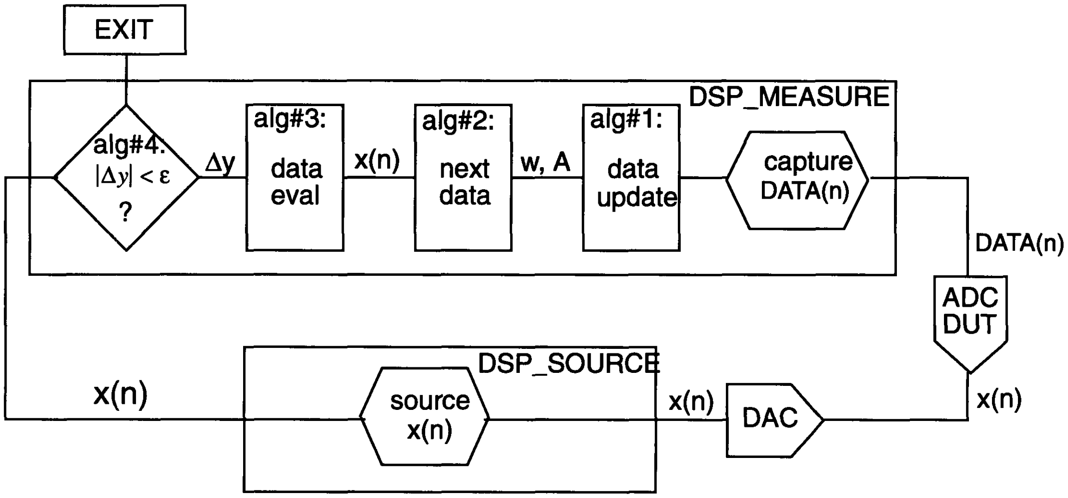


Figure 6-3: Convergence of the ADC servo-loop test. Figure (A) is the actual DUT transfer function. Figure(B) is a trace of DUT output data. Figure (C) is a trace of DUT input data.

The implementation of the ADC servo-loop test described above is straightforward. Since DSP_MEASURE is the DSP feedback driver and is responsible for intensive algorithm execution, some algorithms such as data evaluation, new source data generation and process termination decision, etc., are moved from DSP_SOURCE to DSP_MEASURE. At each cycle, a new source data is generated on DSP_MEASURE and passed to DSP_SOURCE. Figure 6-4 shows the DSP algorithms created for the ADC servo-loop test on the DSP module.



alg #1: $w = \begin{cases} -1, DATA(n) \geq DATAz \\ 1, DATA(n) < DATAz \end{cases}$

$A = A/2, w(n) \neq w(n-1)$

alg #2: $x(n) = x(n-1) + w \cdot A$

alg#3: $\Delta y = \frac{1}{K}\sum_{k=0}^{K-1} x(n-k) - \frac{1}{K}\sum_{k=0}^{K-1} x(n-k-1)$

Figure 6-4: DSP algorithms for the ADC servo-loop test.

The sole activity on DSP_SOURCE is the infinite data source function. At the first cycle DSP_SOURCE sends an initial data downloaded from the CPU to the DUT. For the subsequent cycles DSP_SOURCE receives source data from DSP_MEASURE and sends it to the DUT. On DSP_MEASURE four algorithms and the data capture function are linked for output response capture and evaluation. For each response captured, four algorithms are executed to calculate the next source data (*x(n)*) and check the termination condition. The first three algorithms are regular Loop Algorithms and algorithm #4 is an Exit Algorithm. When the termination condition is met in algorithm #4, the feedback process is terminated on both DSPs and the CPU is notified for result uploading.

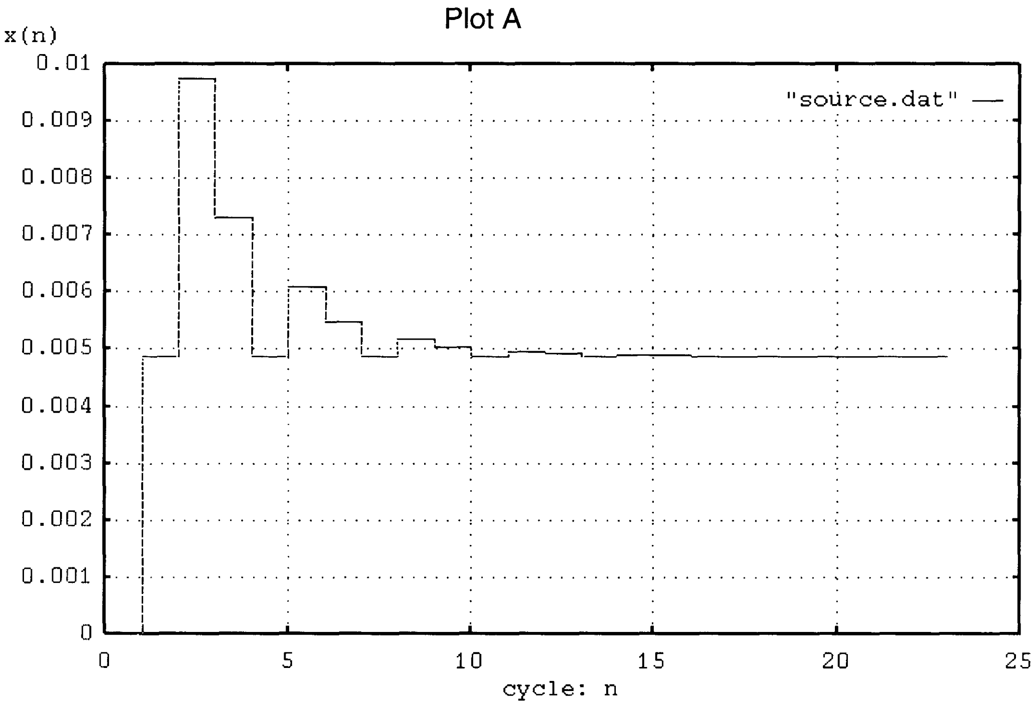
6.3 Result Analysis

An ADC servo-loop test program is created and tested on the DSP test rig and the analog PE subsystem. Two history files are generated for result analysis. One file records data transmitted from DSP_SOURCE to the ADC under test at each cycle, while the other file records the ADC output codes captured on DSP_MEASURE at each cycle. Figure 6-5 shows the plots of these two history files for a 12 bit ADC code transition edge search. The output code under search is 0.0048828125, with a termination threshold level of 10^{-7} volts. It is noted that as the feedback process proceeds, the data sent to the ADC under test converges to the transition edge for the desired output, while the data captured oscillates between the desired output code 0.00048828125 and the adjacent code 0.000439453125.

The test execution time varies with the process termination threshold level (ε) and the voltage resolution of the ADC under test (number of bits). A number of bits of the ADC in the waveform digitizer are masked to create an ADC of various voltage resolutions. Table 6-1 lists the average number of cycles consumed to search a transition code edge of an ADC at different voltage resolution levels and different termination threshold levels. The table shows that as the test threshold level and voltage resolution decrease, the number of cycles taken to search a code transition edge increases.

Table 6-1: Number of cycles required for an ADC code transition edge search

ε	4 bits	6 bits	8 bits	10 bits	12 bits	14 bits
10^{-4}	19	16	12	11	--	--
10^{-5}	23	19	17	14	11	--
10^{-6}	30	25	23	20	16	12
10^{-7}	38	30	29	23	22	19
10^{-8}	41	38	34	31	26	25
10^{-9}	46	42	38	35	31	26



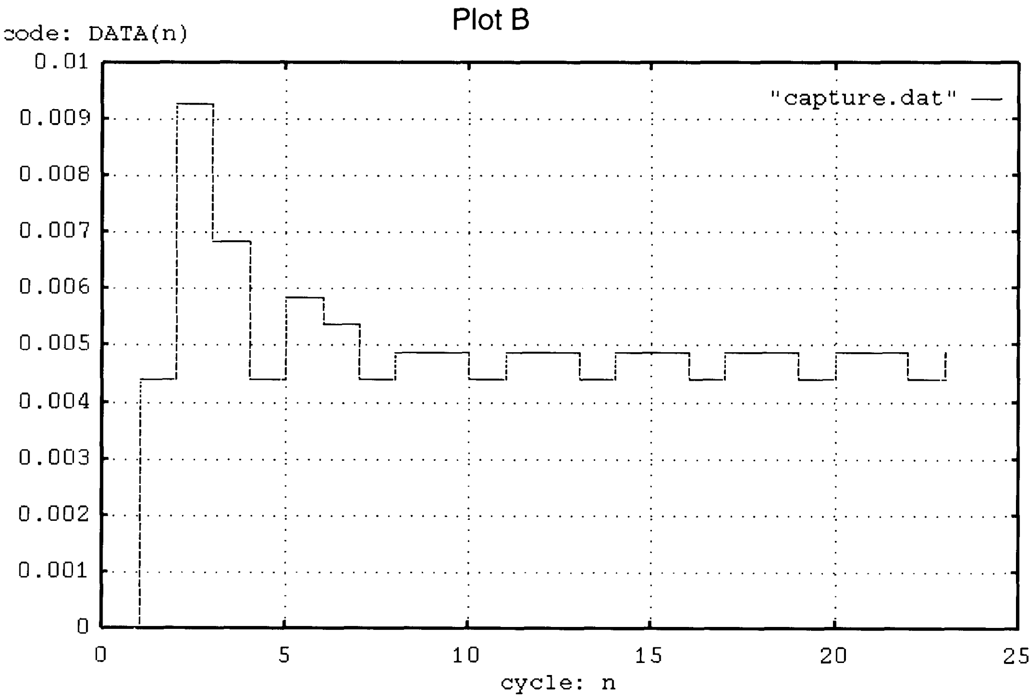


Figure 6-5: History file plots of sourced data and captured data. Plot A is a trace of data sent to the ADC under test. Plot B is a trace of ADC output data captured.

This ADC servo-loop test scheme is suitable for searching any pre-specified ADC code transition edges. An additional DSP algorithm can be added into the ADC servo-loop test to search the entire set of code transition edges of an ADC. Instead of downloading a pre-specified output code, the voltage resolution specification (number of bits) of the ADC is provided to the DSPs. Using the additional algorithm, a complete set of output codes are generated on DSP_MEASURE. An output code is retrieved from the set and a transition edge search is performed for that output code. When the search reaches the threshold level, the next output code is retrieved from the set until the entire set of output codes have been searched. Figure 6-6 shows the history files of data sent and data captured during a set of code transition edges search for a 4-bit ADC (positive output codes only). It is noted that data sent to the ADC under test converges to a code transition edge before the search for the next code edge starts; correspondingly, the data captured oscillates between two adjacent codes under each edge search.

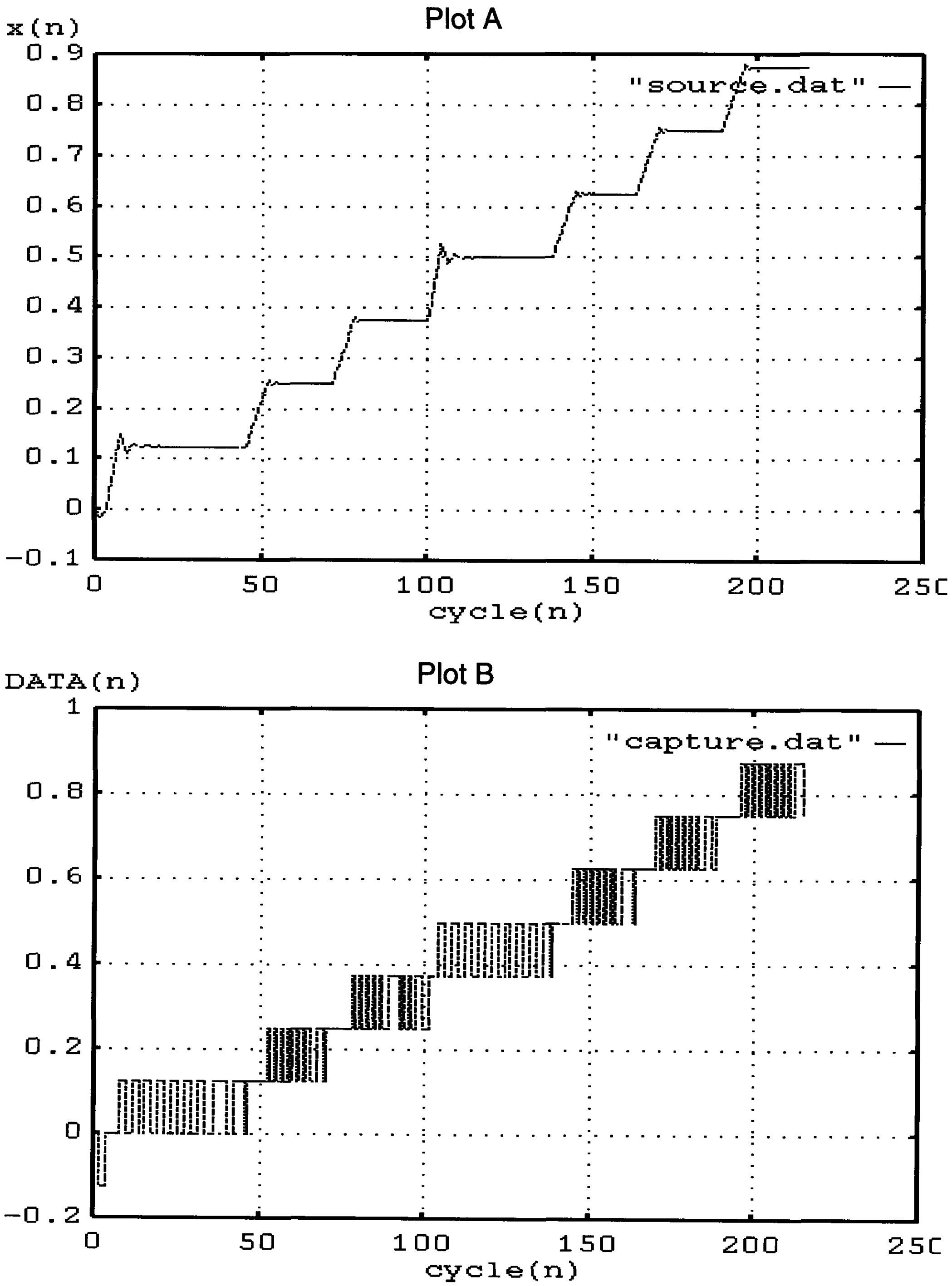


Figure 6-6: History file plots of sourced data and captured data in a ADC servo-loop test for all code transition edges search. Plot A is a trace of data sent to the ADC under test. Plot B is a trace of ADC output data captured.

Chapter 7

Conclusion

In this thesis project, a feedback loop system is designed and implemented in the DSP module of Schlumberger's "big D - little A" mixed signal tester. This feedback loop system enables the mixed signal tester to control the input waveform generations depending on the output response from the DUT in real-time. With this feedback loop system built in the DSP module of the mixed signal tester, the tester is able to support both the open-loop and the closed-loop mixed signal test applications. The real-time error correction program developed for the DSP feedback system, as well as the programs developed previously for the DSP module testing, have confirmed that the modified DSP software is able to support both open-loop and closed-loop test applications.

A typical application of this DSP-based feedback loop system is the ADC servo-loop code transition edge test. Schlumberger's mixed signal tester eliminates any additional hardware and provides a clean solution for the ADC servo-loop testing. The DSP algorithms applied in the ADC servo-loop test are simple to use and easy to modify. A test can be designed to verify some pre-specified ADC code transition edges, and with a slight modification to the algorithms in the test, the entire set of ADC code transition edges can be verified. This DSP-based ADC servo-loop test can be performed at high accuracy by specifying a feedback loop termination threshold at a value as low as the voltage resolution of the DAC used in the test. The ADC servo-loop test developed proves that the feedback loop system incorporated in the DSP module is feasible.

Appendix A

DSP API Feedback Functions

A.1 dsp_api_fb_init_block

```
int dsp_api_fb_init_block    (char    *fb_block_name,
                              char    *src_block_name,
                              int     loops)
```

A.2 dsp_api_add_init_algorithm

```
int dsp_api_add_init_algorithm   (char *algorithm_name,
                                  int num_args, ...)
```

A.3 dsp_api_add_fb_src_algorithm

```
int dsp_api_add_fb_src_algm   (char   *src_blk_name,
                               int    order_number,
                               char   *algorithm_name,
                               int    num_args, ...)
```

A.4 dsp_api_add_fb_exit_algorithm

```
int dsp_api_add_fb_exit_algm   (char  *algorithm_name,
                                int   num_args, ...)
```

Appendix B

ADC Servo-loop Test Algorithms

```
#include <stdlib.h>
#include <math.h>


/* Algorithm #1: Data Update                               */
/* It performs output response analysis                    */

void data_update(float *cap_data, float ref_value, int *w,
                  float *step)
{
   int i;
   if ((*cap_data >= ref_value) && (*w == 1))
       {
         *w = -1;
         *step = 0.5 * (*step);
       }
   if ((*cap_data < ref_value) && (*w == -1))
       {
         *w = 1;
         *step = 0.5 * (*step);
       }
}


/* Algorithm #2: Create the next data source               */

void crt_nxt(int *w, float *old_buf, float *new_buf,
             float *step)
{
   *new_buf = *old_buf + (*w) * (*step);
   *old_buf = *new_buf;
}
```

```
/* Algorithm #3: Data Evaluation                            */
/* It calcuates the difference in the input moving average */

void data_eval(float *new_src_data, float *old_array,
               float *new_array, int size, float *diff_y)
{
   int i;
   float tmp;
   float old_ave;
   float new_ave;
   float old_sum = 0.0;
   float new_sum = 0.0;

   for(i=0; i<(size-1); i++)
    new_array[i] = old_array[i+1];

    new_array[size-1] = *new_src_data;
     for(i=0; i<size; i++)
       {
         old_sum = old_sum + old_array[i];
         new_sum = new_sum + new_array[i];
       }
     old_ave = old_sum / size;
     new_ave = new_sum / size;

     if (new_ave >= old_ave)
        *diff_y =  new_ave - old_ave;
     else
        *diff_y = old_ave - new_ave;

     for(i=0; i<size; i++)
        old_array[i] = new_array[i];
}



/* Algorithm #4: Threshold comparison                      */
/* It determines if a servo-loop test has completed        */

int vcompare(float *in_val, float threshold, float dum1,
             float dum2)
{
   if (*in_val < threshold)
          return 0;
   else
          return 1;
}
```

Bibliography

[1] Schlumberger Automatic Test Equipment Corporate backgrouder, July, 1995

[2] E. Norton. Digitized analog testing. *Evaluation Engineering,* The Magazine of Electronic Evaluation and Test. November, 1992

[3] K. Karube, Y. Bessho, T. Takakura, K. Gunji. Advanced mixed signal testing by DSP localized tester. *International Test Conference,* pp1055-1060, IEEE 1991

[4] M. Mahoney. Tutorial: DSP-based testing of analog and mixed-signal circuits. *The Computer Society of IEEE,* 1987

[5] A. Maeda. The advanced test system architecture provides fast and accurate test for a high resolution ADC. *International Test Conference,* pp68-75, IEEE 1992

[6] S. Pei, S.P. Chan. DSP application in integral-non-linearity testing of A/D converters. *24th Asilomar Conference on Signals and Computers* vol.1, pp516-519, 1990

[7] A. Brandolini, A. Gandelli. Testing methodologies for analog-to-digital converters. *IEEE Transactions on Instrumentation and Measurement,* vol. 41, No. 5, pp595-603, October, 1992

[8] A. Brandolini, A. Gandelli. High-precision analog-to-digital converter testing using Walsh transform. *Conference on Precision Electromagnetic Measurements,* pp384-385, IEEE 1990

[9] L. Benetazzo, C. Narduzzi, C. Offelli, D. Petri. A/D converter performance analysis by a frequency-domain approach. *IEEE Transactions on Instrumentation and Measurement,* vol. 41, No. 6, pp834-839, December, 1992

[10] R. Hagelauer, F. Oehler, G. Rohmer, J. Sauerer, D. Seitzer, R. Schmitt, D. Winkler. Investigations and Measurements of the Dynamic performance of high-speed ADC's. *IEEE Transactions on Instrumentation and Measurement.* vol. 41, No. 6, pp829-833, December, 1992

[11] E. Liu, A.L. Sangiovanni-Vincentelli, G. Gielen, P. Gray. A behavioral representation for Nyquist rate A/D converters. *IEEE International Conference on Computer-Aided Design*, pp386-389, 1991

[12] S. Pei, S.P. Chan. New approach to linearity testing of A/D converters. *International J. Electronics*, Vol. 70, pp1049-1062, 1991

[13] J. Weimer, K. Baade, J. Fitzsimmons, B. Lowe. A rapid dither algorithm advances A/D converter testing. *International Test Conference*, pp498-507, IEEE 1990

[14] Schlumberger Technologies, ATE/Components Division, ITS 9000 Slingshot Hardware Engineering Requirements Specification.

[15] Schlumberger Technologies, ATE/Component Division, ITS 9000 Slinghsot DSP High Level Design Specification

[16] Ixthos IXD7232 User's Manual, Ixthos, VME Solutions for Advanced DSP

[17] Schlumberger Technologies, ATE/Components Division, ITS 9000 Slingshot Software Engineering Requirements Specification.

[18] Schlumberger Technologies, ATE/Component Division, ITS 9000 Slinghsot DSP API & Runtime Services Low Level Design Specification.

[19] Schlumberger Technologies, ATE/Component Division, ITS 9000 Slinghsot DSP Executive High Level Design Specification.

[20] Schlumberger Technologies, ATE/Component Division, ITS 9000 Slinghsot DSP Buffer Manager Low Level Design Specification.

[21] Schlumberger Technologies, ATE/Component Division, ITS 9000 Slinghsot DSP Timeout Handling Low Level Design Specification.