Application Report
SPRACI3–September 2018

C2000™ CLA Self-Test Library

Salvatore Pezzino, Peter Ehlig, Bharat Rajaram, Ashish Vanjari

ABSTRACT

This application report describes Control Law Accelerator self-test for the TMS320F2837xD, TMS320F2837xS, TMS320F2807x series of C2000 devices, hereafter referred to as F28x7x devices. The code elements involved are referred to as Control Law Accelerator Self-Test Library (CLA STL).

Contents

Trademarks

1 Introduction

This application report provides an overview of the Control Law Accelerator Self-Test Library (CLA STL), and includes a brief introduction to the CLA and CLA STL. Also covered is the reasoning behind the development of the CLA STL and the problem it addresses. Subsequently, it describes the philosophy of the CLA STL implementation and how Design-for-Test (DFT) fundamentals are applied to this specific safety mechanism or diagnostic test. Finally, some considerations are provided to aid with the CLA system integration.

1.1 Acronyms and Descriptions

Table 1. Acronyms and Descriptions

Acronym	Description
CLA	Control Law Accelerator
STL	Self-Test Library
PSA	Parallel Signature Analysis
SOC	System On-Chip
HWBIST	Hardware Built-In Self-Test
CPU	Central Processing Unit
SA1	Stuck-At 1
ATPG	Automatic Test Pattern Generation
EDAC	Error Detection And Correction
FPU	Floating Point Unit
DFT	Design For Test
DC	Diagnostic Coverage
PEST	Periodic Self-Test
PIE	Peripheral Interrupt Expansion
POST	Power-On Self-Test
RAM	Random Access Memory
ROM	Read Only Memory
TMU	Trigonometric Math Unit
TRM	Technical Reference Manual
VCU	Viterbi and Complex Math Unit

2 CLA STL Overview

The purpose of the CLA STL is to monitor the CLA circuitry for degrading (or permanent d.c. random HW) defects that can cause failures in the circuitry and to provide the C28x CPU with indication of a fault occurring so that the C28x can take appropriate action.

2.1 Introduction to CLA

The CLA is an independent, fully programmable, 32-bit floating-point math co-processor that enables concurrent control-loop execution. It has a dedicated register set and 8-stage pipeline. It has a minimalist instruction set designed to enable the customer to accelerate the control-loop in the system. It is a task driven engine with up to eight task vectors available. Each of these task vectors can be triggered by software or some other configurable event on chip. The CLA has access to only a subset of memory blocks for program and data, including local shared RAM (RAMLS) between the C28x and CLA, message RAM between the C28x and CLA, as well as CLA data ROM. Additionally, the CLA has access to a subset of peripherals available on the device via hardware register access.

2.2 Introduction to STL

The functional safety concept of the CLA STL is that code executing on the CLA can verify if the CLA is functioning correctly. This self-test code exercises all of the circuitry in the CLA while monitoring the execution with embedded Parallel Signature Analysis (PSA) circuits that monitor program address and data write. These monitoring circuits capture incorrect execution in the CLA circuitry so that the C28x CPU code can identify the presence of a fault in the CLA circuitry and initiate appropriate actions.

The CLA self-test code is portioned into small tests so that the diagnostic testing can be executed in small time-slices within the system control loop.

2.3 CLA STL in the TMS320F28x7x Devices

The CLA STL testing of the CLA logic is designed to be executed in-system, meaning that it executes along with the real-time control operations. This means the control system code must provide some portion of the C28x CPU and CLA available cycles to the execution of these CLA tests.

The system integrator must provide a time slice in the C28x control loop for the C28x to load and execute a CLA test. The C28x CPU uses one or two (depending on wait states of the memory where the CLA code is stored) time slices to load the small test into a reserved memory range in the CLA program memory space. The next available time slice is used to execute the CLA code. These time slices are <5 µs long when operating the device at 200 MHz and executing from zero-wait state RAM.

The C28x will cycle through these CLA tests until all are completed and then start the diagnostic testing over again. This is referred to as periodic testing or periodic self-test (PEST). There are a few of the CLA tests that apply to CLA configuration circuitry, and are tested only at start-up or during maintenance cycles. This is referred to as start-up testing or power-on self-test (POST).

3 Goal of CLA STL

Functional safety requires freedom from unreasonable residual risk originating from faulty behavior of safety related electronic and electrical systems. Faults can be either systematic or random hardware faults. Customers who use the CLA of F28x7x series of devices in functionally safe systems may require a specific diagnostic coverage (DC) in order to meet various functional safety standards or requirements. The CLA STL was designed and developed to achieve 90% DC coverage of the CLA logic to target ASIL B (per ISO 26262) and SIL 2 (per IEC 61508) for single point permanent fault coverage and ASIL D/SIL 3 for latent fault coverage. ASIL standards for “Automotive Safety Integrity Level”. SIL stands for “Safety Integrity Level”.

The F28x7x series of devices is equipped with various functional safety mechanisms. With regard to the processing elements on the SOCs, the F28x7x devices are equipped with a Hardware Built-In Self-Test (HWBIST) that targets the C28x CPU logic including the TMU, FPU, and VCU and is able to achieve up to 99% DC. For more details, see *C2000™ Hardware Built-In Self-Test*.

Additionally, all F28x7x memories are equipped with EDAC, or error detection and correction logic, except for the F28x7x ROMs. Furthermore, the F28x7x SafeTI Diagnostic Library (SDL) provides software for a RAM March13n algorithm that specifically targets the data, address and EDAC bits for permanent stuck-at faults, as well as worst-case path timing faults that can occur from system degradation over time. For more information, see *C2000™ CPU Memory Built-In Self-Test*.

This CLA STL is specifically design, developed, tested, assessed and approved to meet 90% DC of the remaining processing element on the F28x7x series of C2000 devices, namely the CLA. The CLA STL enables customers to utilize the CLA in their functionally safe control systems by providing functional safety diagnostic measures for the CLA. This allows customers concerned with functional safety to take advantage of the extra processing element of the F28x7x series of C2000 devices, thus enabling greater processing power.

4 CLA STL Philosophy

As mentioned previously, the CLA STL is a software self-test library that provides software for the C28x (master) and the CLA (slave) to diagnose the CLA logic for dangerous single-point faults. What follows is the philosophy of how the CLA STL was design and developed.

4.1 Design-For-Test Fundamentals

In order to detect a fault in some logic, two things must be accomplished together. The first requirement is that the fault be excited. In digital logic such as the CLA, this means that a test pattern must produce the complement of a stuck-at fault. For example, if a signal in the net list of the CLA is stuck-at one (SA1), then the test pattern must excite the SA1 fault by producing zero on that signal. In other words, the test pattern would produce a zero on that signal if it were not for the permanent fault forcing a one. This is the first requirement for detecting this fault. Secondly, the test pattern must also propagate that fault to an observable output by producing a sensitized path to the output from that fault signal location.

Using the above example and SA1 fault, the test pattern must produce a zero at that signal and then propagate that zero to an observable output. If that zero signal is an input into an OR gate and there is another input to that OR gate which is one, then that zero signal would no longer propagate. Therefore, the SA1 fault would not propagate along that path. That path would not be a sensitized path allowing for the propagation and ultimate detection of that SA1 fault.

Figure 1 shows the SA1 not being excited. Figure 2 shows the SA1 being excited but not propagated. Figure 3 shows the SA1 being both excited and propagated and therefore is detectable. Figure 4 is an illustration of how potential faults are detected for a block of combinational logic.

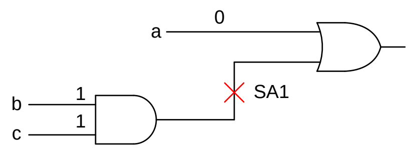

Figure 1. The Stuck-At-One Fault is Not Excited

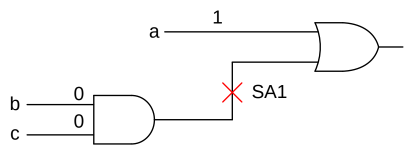

Figure 2. The Stuck-At-One Fault is Excited and Not Propagated

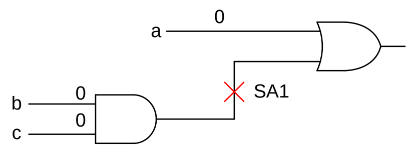

Figure 3. The Stuck-At-One Fault is Excited and Propagated (Detectable)

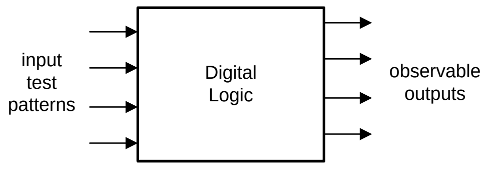

Figure 4. Input Test Patterns and Observable Outputs Testing Digital Logic

The goal of DFT is to create efficient test patterns that excite potential faults and propagate those faults to outputs. This is the approach of the Automatic Test Pattern Generation (ATPG). There are presently many software tools that can analyze a net list of digital logic and can determine optimal sets of test patterns for a DFT designer's set of criteria. For example, ATPG test patterns can be optimized to reduce the logic overhead of the DFT logic, or the ATPG test patterns can be optimized to achieve the greatest amount of coverage in the fewest number of test patterns. Further discussion of this is outside the scope of the document.

The key fundamental to understand is that in DFT, test patterns are generated as inputs to the logic under test in order to detect (excite and propagate) potential faults within that logic.

4.2 Design-For-Test Fundamentals Applied

The CLA STL applies these above DFT fundamentals in order to diagnose dangerous faults in the CLA. The CLA STL applies these fundamentals to achieve 90% DC by detecting (exciting and propagating) permanent stuck-at faults.

In the case of the CLA STL, however, test patterns are not scanned into scan registers as is done in ATPG. Rather, small test programs have been designed and developed to target the various digital logic components and functional units of the CLA. These small test programs can also be referred to as test vectors. This is because these test programs are series or vectors of opcodes and data being fetched and executed by the CLA. When these test vectors are being fetched and executed by the CLA, they exercise the different logic components and functional units of the CLA and yield unique results that are written to the internal CLA registers, and then written by the CLA to the data bus. These are the observable outputs of the input test vectors.

As previously stated, the CLA STL makes use of Parallel Signature Analysis (PSA) circuitry that monitors the CLA program address bus and the CLA data write bus. There is one PSA that monitors the CLA program address bus and another PSA that monitors the data write bus. These PSAs have the ability to monitor these two system buses on every cycle. They monitor the buses on a cycle-by-cycle basis and calculate unique signatures based on the address and data being driven on the buses, respectively. This PSA circuitry is a key element of the efficient (cycles and memory) diagnostic technique of the CLA STL.

In order to take advantage of the PSA circuitry on the F28x7x series of C2000 devices, a collection of specifically targeted test vectors were designed and developed. As these test vectors are fetched and executed by the CLA, the CLA logic components and functional units are exercised and the results are written to the message RAM. Concurrently, the PSA circuitry generates unique signatures based on the program fetched as well as the results written on the data bus to the message RAM. Subsequent to the test vector completion, the master C28x reads the PSA signatures and compares them to the golden or expected PSA signatures. If the PSA signatures are as expected, then no fault was detected by that test vector. However, if the PSA signatures are not as expected, then a dangerous fault was detected by that test vector. In the CLA STL, an error message is returned to the system so that appropriate measures can be taken to safely shut down or repair the system. Figure 5 illustrates the DFT fundamentals applied to the CLA STL.

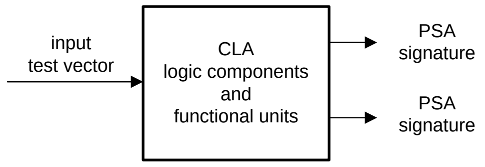

Figure 5. Input Test Vector and PSA Signature Outputs Testing the CLA

5 System Integration Considerations for the CLA STL

While the control loop is active, the validation of the CLA circuitry presents challenges that are not an issue at start-up. This section does the following:

- Presents system challenges to CLA self-test
- Explains how this application addresses these challenges
- Presents system considerations when executing the CLA STL

5.1 CPU/CLA Execution Time

The CLA must have execution time available to execute the self-test code in conjunction with the real time control code. It must do so with little or no effect on the timing of the sensor reads, processing and actuation updates. So the CLA test must be executed as a time slice that avoids or minimizes impact to the control loop. For example, in a typical control loop the sensor data is read based upon a timer event and then the sensor data is processed to provide an actuation update. Following the high priority sensor interrupt, processing and actuation update, there are available system cycles that can be used for background tasks, including diagnostic testing. Here, the system integrator has a choice. On the one hand, the CLA STL diagnostic testing can be integrated to execute at the end of the high-priority interrupt service routine and included in the routine. On the other hand, the CLA STL diagnostic testing can be integrated to execute during the background loop of the system. In the first instance, the CLA STL executes with a high priority. In the second instance, the CLA STL operates as a background task. This decision is the responsibility of the system integrator.

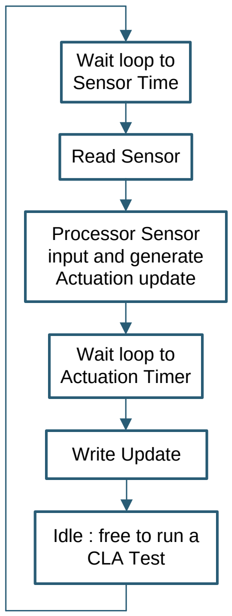
Wait loop to Sensor Time
Read Sensor
Processor Sensor input and generate Actuation update
Wait loop to Actuation Timer
Write Update
Idle : free to run a CLA Test

Figure 6. Integrating CLA STL into Control Loop

During the background diagnostic testing, the C28x and CLA will execute the software provided in the CLA STL. The CLA STL provides the software to:

- Load the CLA test vector into a specific address range in LS0
- Initiate and monitor the execution of the CLA test vector

During the execution of the CLA test vector, both the C28x and the CLA are isolated from the system control operations (all maskable interrupts masked). While the C28x is loading the CLA test code, the CLA can be executing system code so long as it is executing from a different LS SRAM block than the block into which the CLA. test code is being loaded (RAMLS0).

As the execution of the CLA test code requires full attention of both the CLA and the C28x, it is important for the system integrator to provide this level of synchronization. Stated differently, there must be the “idle: free to run the CLA test” box shown in Figure 6 aligning in both the C28x and the CLA. This might be done by making use of one of the timer interrupts or possibly via the communication over the message SRAMs.

5.2 Memory Usage

The CLA STL requires a specific range of RAMLS0 memory space be reserved for the CLA STL tests. This is because the monitor of the CLA program address will fail if the code does not start at the correct address. Prior to the C28x initiating execution of a CLA test vector, it remaps or reconfigures RAMLS0 to CLA program memory. Upon completion of the CLA test vector, the C28x remaps or reconfigures the memory to CLA data space. The system code running on the C28x or the CLA can make use of the remainder of LS0 as data memory.

The CLA control code tasks should reside in a different RAMLS block other than RAMLS0. The remaining RAMLS block usage is up to the system integrator. Additionally, there is a small amount of message SRAM usage on the CLA side.

5.3 Safety Interval

The minimum safety interval of the CLA STL is determined by the number of control loop periods it takes to complete all of CLA test vectors. This safety interval is determined by the system integrator and the system's availability to execute the CLA STL. For example, suppose a system is able to load one test vector and execute one test vector per control loop iteration. Considering there are 45 in-system test vectors, the system would achieve 90% DC in 45 control loops. If we suppose that a system is only able to load one half of a test vector in a single control loop and is only able to execute one test vector in a single control loop, then considering there are 45 in-system test vectors, the system would be able to achieve 90% DC in 135 control loops. This system would require 90 control loops for loading the test vectors (45 x 2) and 45 control loops for execution, resulting in 135 control loops.

5.4 Interrupt Latency

Interrupts to the C28x and the CLA are masked during the execution of the CLA test vector. This is for approximately 2 µs when operating at 200 MHz device frequency. This results in the need for an efficient synchronization between the C28x and CLA availability to execute the CLA STL.

5.5 Integrating Start-Up Tests for Periodic Testing

The CLA STL also includes a number of tests that are intended for start-up testing or power-on self-testing (POST). This is because these tests are more invasive to the system, whether that includes using more RAMLS memory, more message RAM, using a SPI to trigger CLA tasks, or just executing longer than the in-system or periodic self-tests (PEST). If the system integrator plans to integrate these into the control loop, then more effort is required to more thoroughly understand the CLA STL software and properly integrate them into the control loop.

6 References

- F28x7x SafeTI Diagnostic Library (SDL)
- Quality and Reliability: Soft Error Rate FAQs
- *TMS320F2837xD Dual-Core Delfino Microcontrollers Technical Reference Manual*
- *TMS320F2837xD Dual-Core Delfino™ Microcontrollers Data Manual*
- *TMS320F2837xD Dual-Core Delfino™ MCUs Silicon Errata*
- *TMS320F2837xS Delfino Microcontrollers Technical Reference Manual*
- *TMS320F2837xS Delfino™ Microcontrollers Data Manual*
- *TMS320F2837xS Delfino™ MCUs Silicon Errata*
- *TMS320F2807x Piccolo Microcontrollers Technical Reference Manual*
- *TMS320F2807x Piccolo™ Microcontrollers Data Manual*
- *TMS320F2807x Piccolo™ MCUs Silicon Errata*
- C2000™ SafeTI™ Diagnostic Software Library for F2837xD, F2837xS, and F2807x Devices
- *C2000 Hardware Built-In Self-Test*
- *C2000 CPU Memory Built-In Self-Test*
- *Error Detection in SRAM*