POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

System Level Test methodologies for complex timer peripherals

Supervisors

Prof. Paolo BERNARDI

Prof. Angione FRANCESCO

Candidate

Davide FOGLIATO

April 2024

Abstract

The thesis work will focus in the generation of SLT procedures aiming at complementing the weaknesses of structural test methods. The case of study will be the timer module of an industrial SoC.
This research will show different approaches adopted to produce programs suitable to test that parts of the component that are less stressed, and that can only be covered by specific functional procedures. Each method is described and detailed in order to provide a clear understanding of the motivations behind its choice, exploring solutions that are consistent with the analysis of the results obtained by the simulation to which each test is subjected.

Acknowledgements

I would like to thank my parents and my sister who have supported me throughout my years of study, providing me with the support I needed and allowing me to reach this significant milestone in my life.
I would also like to thank my supervisors, Professor Paolo Bernardi and Francesco Angione, who have assisted and supported me throughout this research work.

Table of Contents

List of Tables

List of Figures

Acronyms

SLT	System Level Test
SoC	System-on-Chip
CPU	Central Processing Unit
RAM	Random Access Memory
ALU	Arithmetic Logic Unit
GTM	Generic Timer Module
PWM	Pulse-Width Modulation
ARU	Advanced Router Unit

ATOM	ARU Timer Output Module
TIM	Timer Imput Module
MCS	Multi Channel Sequencer
MUX	Multiplexer
ATE	Automatic Test Equipment
DUT	Device Under Test
PSM	Parameter Storage Module
BRC	Broadcast
CMU	Clock Management Unit
TBU	Time Base Unit
ICM	Interrupt Concentrator Module

DTM	Dead Time Module
SPE	Sensor Pattern Evaluation
MON	Monitor Unit
CMP	Output Compare Unit
DPLL	Digital Phase Locked Loop
VCD	Value Change Dump

Chapter 1

Backgound

1.1 Introduction

This section concerns the research area of the research,focused on the concept of manufacturing test flow and on the explanation of the steps that composed it, with a particular attention on the System Level Test method. In addition, a part is dedicated to the structural and functional approaches to explain what advantages they offer and why they are crucial to reach a complete coverage of the system.

1.2 Manufacturing test flow

To introduce this background section, it is crucial to clarify the concept of "manufacturing test flow", a key aspect for the purpose of this research.
In the production of electronic devices, such as the integrated circuits, it has become essential to subject these kind of products to a functionality verification process in order to identify defects or malfunctions.
The manufacturing test flow is, therefore, a process to check whether the design of these semiconductor devices was fabricated in the right way,

since the production process is inevitably not without imperfections [1]. It's composed of several test phases, whose sequence order and organization could slightly change depending on the company context in which it is applied. The scheme in Figure 1.1 could be referred to the automotive field, scope of application for the board examined in this research.

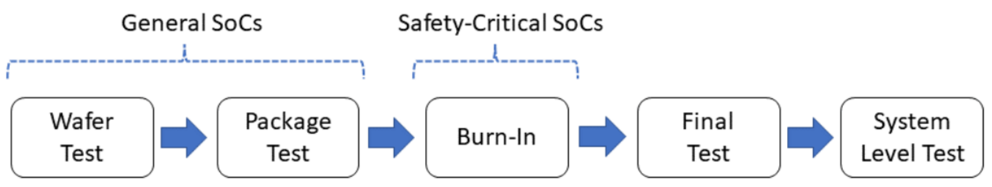


Figure 1.1: Manufacturing test flow.

Going into more detail, each step is performed targeting a specific defect for ensuring the quality of the product during the production phase.
Therefore the stages that compose the test flow are the following:

- Wafer Test, done at the wafer level, assess the individual integrated circuits on the wafer for any electrical defects of the chip through the application of several methods. During this phase, test patterns are applied utilizing probes.

- Package Test, done after the product has been packed, checks the integrity of the device and its basic functions to ensure the meeting of the electrical requirements.

- Burn-In exposes the device to some stress tests, such us high voltage and elevated temperature, in order to detect potential weaknesses of the component.

- Final Test is used to apply both structural and functional tests to identify possible faults
- System-Level Test includes complex functional programs to check the correct circuit behavior and the functionality of the entire system.

It is now crucial to make a distinction between structural and functional tests, explaining the advantages in their usage and how they have proven to be essential in SoC testing.
Structural tests exploits the knowledge of the internal structure of an integrated circuit, employing various techniques to enhance the automation of tests production. Functional tests, on the other hand, are focused on the overall behavior of the system, or a part of it, without necessarily taking its structure into account.
It might appear that test automation is the more favorable option to save time and effort, however, structural testing manages to cover only a part of the total defects, making the choice to adopt a combination of the two types of testing more comprehensive.

1.3 Scan-based structural tests

In this subsection, some information will be provided about the scan chain based methods, since it will represent a useful analysis element for an approach adopted during the program development.
These types of tests are named “scan" tests because they involve scanning test patterns into internal circuits within the device under test.
The flip-flops structure is changed to a new one, becoming elements capable of controlling the inputs and outputs of certain sections of the circuit under test, while always maintaining their original normal function.
Indeed, these elements, also called scan cells, allow to divide the whole

system in many parts, composed of combinational circuits not so difficult to test like the sequential ones [2].
In Figure 1.2 is depicted the the normal flip-flop compared to the one modified to scan the circuit.

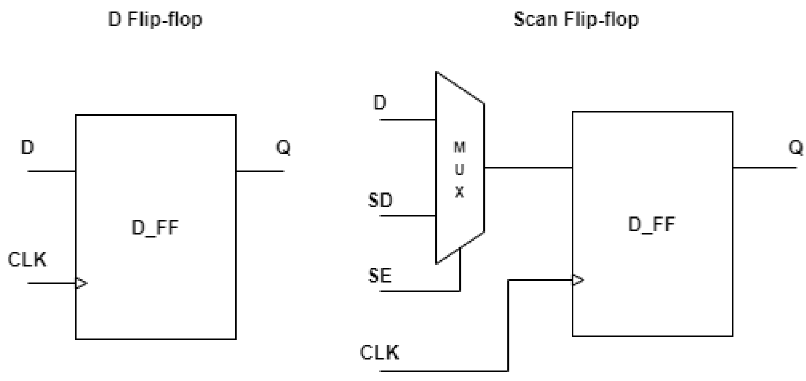


Figure 1.2: D Flip-flop and scan Flip-flop.

The scan cell consists on a normal D flip-flop with the input pin connected to the output of a multiplexer. The two inputs of the MUX refers to the signal used in Normal mode and the one used in Test mode. The signal SE is aimed to switch between these two modalities allowed by this architecture.
Each scan flip-flop is linked with an other one in order to form a "scan chain", as the one illustrated in Figure 1.3.

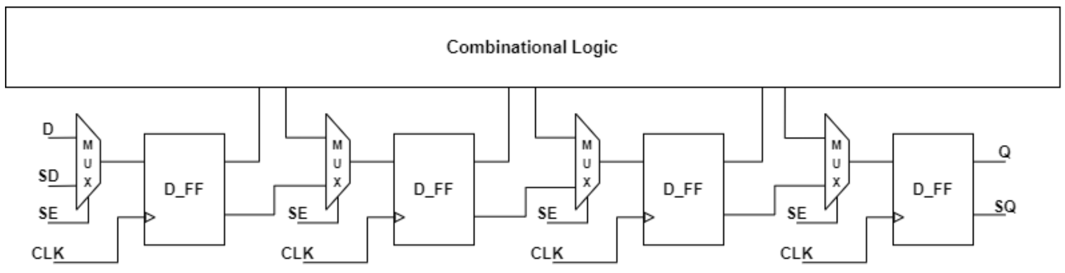


Figure 1.3: Scan Chain.

The input and output pins used in Normal mode are connected with the combinational logic of the circuit, that is part of its standard functional path. Instead, every scan input (SI) and scan output (SQ) pins are linked together, creating the testing path used in Test mode. This additional design modification is utilized to allow the insertion of test patterns generated by by an external automatic test equipment (ATE) during the Test mode of the circuit, targeting the faults in the combinational logic.
After the test patterns are loaded, the system function is switched into Normal mode and the circuit response is captured in one or more clock cycles. Then, again during Test mode, the captured test outputs are shifted out, and, at the same time, a new test pattern is ready to be shifted in, repeating the same process.
Finally, in order to detect possible defects, a dedicated component compares the test response with the expected one stored in its memory, to check if they match each other.
The advantages of the scan tests consist on their easy implementation in the system under examination, with the possibility to exploit this structure for debugging purposes.
However, it requires the utilization of a larger area and more components compared to a circuit lacking such functionality.

1.4 System-Level-Test

Over the years, semiconductor technologies have continued to grow in complexity, making it increasingly challenging for the industries to achieve a comprehensive set of tests for the modern electronic devices. As a consequence of this consideration, the System Level Test has become rather convenient to be utilized as an additional test process by the component manufacturers, providing a new tool to meet the strict customer failure rate requirements for improving the product quality

[3].
The System Level Test (SLT) consists on testing a device under test (DUT) by using it rather than creating test vectors, simulating an environment similar to the one in which it will be utilized, to evaluate the system as a whole.
SLT method is it is particularly suitable to detect chip failures and defects that are not scanned by the other steps of the industrial test flow, such as those tests applied at wafer level and the final test that don't reach all parts of the circuit.
However, as a drawback, it required an amount of time dedicated for each chip that tends to be significantly greater than the other phases, a negative aspect that can be somewhat balanced by applying levels of parallelism [4]. Therefore it is characterized by a significant cheaper cost in terms of equipment needed to operate the testing, which allows initiating multiple functional tests simultaneously.
To gain an overall understanding of the SLT-based approach, it might be helpful to compare it with the other existing testing techniques, especially the structural ones.
As described in [5], the key characteristics that define the SLT methods can be found in its lack of standard methods that involve the use of Automatic Test Pattern Generation (ATPG) procedures for producing test patterns and that rely on fault models.
Automated test equipment (ATE) or built-in self-test (BIST) techniques, that are based on these levels of automation, while considered often essential in the field of integrated circuit testing, are not without limitations, particularly when concerning the fault coverage. These methodologies primarily target specific defects outlined by the fault models taken into account, disregarding others that they may not be adequately covered, maybe due to the fact that are related to those parts of the circuit not reached by the ATPG patterns.
Moreover, SLT seems to be especially suitable to cover real-life cases, given that it includes testing procedures closely resembling how the

system would be used in the actual operational environment and subjected to real workloads.

Chapter 2

Generic Timer Module description

2.1 Component Introduction

In this section of the research, a brief overview of the Generic Timer Module (GTM) will be provided, outlining the fundamental principles underlying its creation and detailing the internal structure, with a particular focus on highlighting the primary applications of its submodules.
The GTM consists of a generic timer platform that serves different application domains, designed to reduce any significant interrupt load for an external peripheral core used to configure it. Most tasks within can operate independently and concurrently with the software. Indeed, although there could be specific scenarios requiring CPU intervention, the primary objective of this timer module is to minimize such occurrences.
The Figure 2.1 provides a general overview of the internal architecture of the main blocks and how they are organized. However, it is important to highlight that it represents only an approximate model of the actual structure, as it does not depict the exact number of instances for each module type that composed the timer.

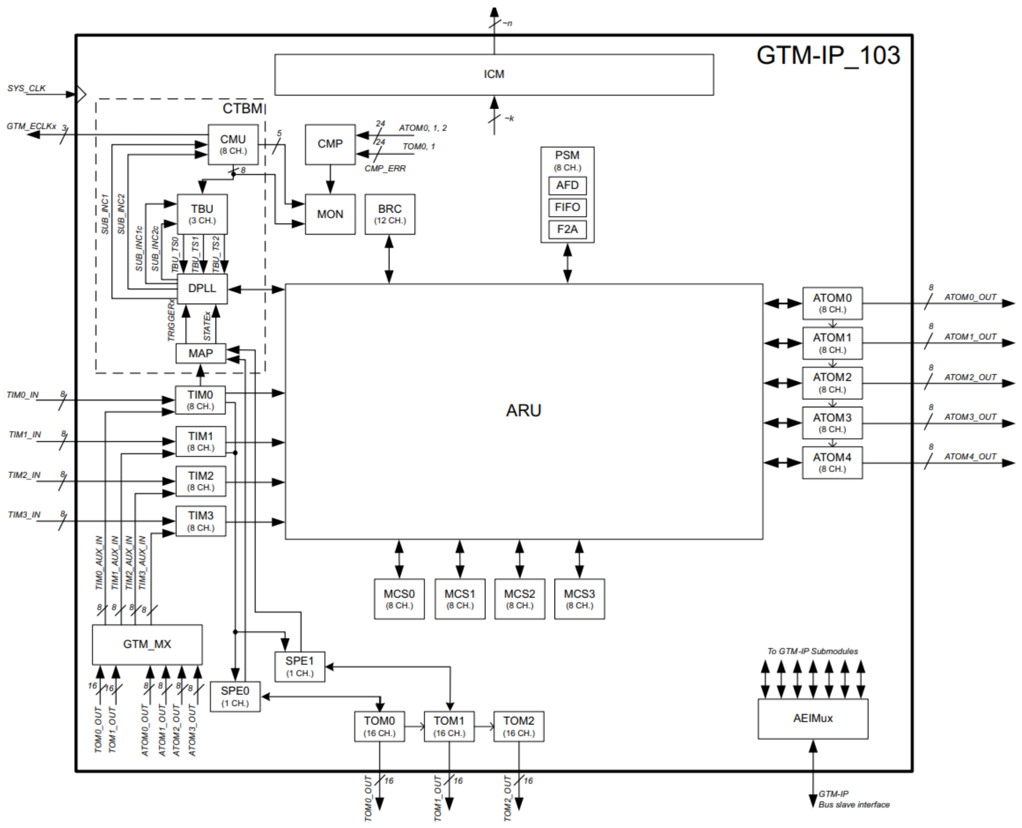


Figure 2.1: Generic Timer Module architecture.

The information about the GTM that will be provided in the next sections are taken most of all from the official manual [6] and from the guide [7].

2.2 Generic Timer Module principles

In order to give some details about the GTM, it's important to speak about the philosophy behind its design.
One possible approach to design a complex timer consists on adopting

a predominantly hardware-centric strategy, incorporating capture/-compare units and counters in the module, managed by an external processor or co-processor. In contrast, a second possible approach regards to follow a more processing-oriented methodology, employing a programmable micro controller that executes timer-specific tasks.
Of these two theories, the former faces some challenges, such as the external core having to manage numerous interrupts to control the timer module through the bus system; while the software-centric approach often encounters issues like lower signal processing resolution and programming difficulties due to the highly specialized instruction sets.
A potential third approach could involve combining the advantages of both the design philosophies, aiming to address the need for a flexible method to handle a substantial load on the CPU, while simultaneously ensuring high performance.
This objectives forms the foundation of the GTM, equipped with sub-modules capable of performing dedicated hardware functions, including a RISC-like processing engine integrated within the GTM that facilitates signal processing and flexible signal generation.
The presence of the Multi Channel Sequencer (MCS), a generic data processing module inside the timer, it is an evidence of how such an approach has been implemented, enhancing the GTM's flexibility. However the description of this component will be provided in a subsequent dedicated section.
Figure 2.2, taken from the manual [6], illustrates the timer philosophies and their alignment with the GTM.

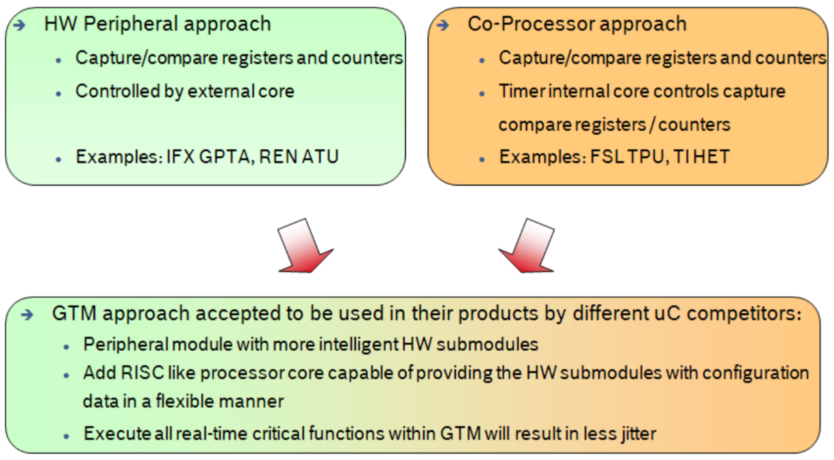

HW Peripheral approach
Capture/compare registers and counters
Controlled by external core
Examples: IFX GPTA, REN ATU
Co-Processor approach
Capture/compare registers and counters
Timer internal core controls capture compare registers / counters
Examples: FSL TPU, TI HET
GTM approach accepted to be used in their products by different uC competitors:
Peripheral module with more intelligent HW submodules
Add RISC like processor core capable of providing the HW submodules with configuration data in a flexible manner
Execute all real-time critical functions within GTM will result in less jitter

Figure 2.2: Market timer concepts.

2.3 Generic Timer Module architecure

This section will elucidate and analyze details of the GTM functionalities that have played a pivotal role in this research work for the development of certain theories, aimed at the creation of dedicated functional procedures. Consequently, a more comprehensive description of this system is presented.
The GTM is composed of dedicated hardware submodules that are placed all around a central routing unit, referred to as Advanced Routing Unit (ARU), that can combine inputs and outputs to produce more complex signals. Indeed, this specific circuit can route signals derived from different modules to processing units where an intermediate value, representing for instance an incoming signal frequency, can be calculated.
Such approach to designed the architecture system allow to link different components in a way that facilitates a flexible interconnection and, though the quantity of such components may differ from one device to another, they can be added or removed to optimize the area consumption.

Moreover, this type of connectivity is programmable via software and can be configured during runtime.
Broadly speaking about the submodules, there are some input components for timers, where incoming signals can be captured and analyzed, some modules that facilitate the implementation of intricate functions, and others that follow a more generic architecture performing standard timer functions, such as the PWM generation units.
A different category of submodules is dedicated to particular application domains, for instance the DPLL caters to engine management applications. Another set of them falls under a group that supports the implementation of functions in order to meet defined safety standards or with the purpose to handle interrupt services.
However, to get an overall picture of the system some main functionalities assigned to individual components will be described more in details.
Despite each version of this timer peripheral shows slight differences for what concerns the architecture, it always presents the same basic block structure, as that shown in the Figure 2.3, in which each block type is grouped based on its utility and identified with a distinct color.

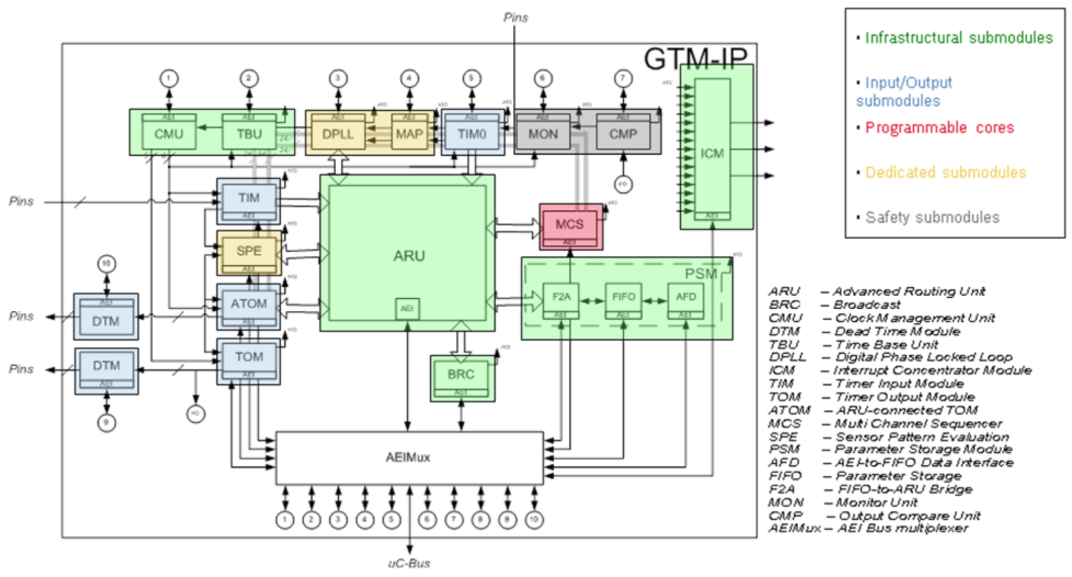

GTM-IP
Pins
CMU
TBU
DPLL
MAP
TIM0
MON
CMP
ICM
TIM
SPE
ATOM
TOM
DTM
ARU
MCS
PSM
F2A
FIFO
AFD
BRC
AEIMux
uC-Bus
• Infrastructural submodules
• Input/Output submodules
• Programmable cores
• Dedicated submodules
• Safety submodules
ARU – Advanced Routing Unit
BRC – Broadcast
CMU – Clock Management Unit
DTM – Dead Time Module
TBU – Time Base Unit
DPLL – Digital Phase Locked Loop
ICM – Interrupt Concentrator Module
TIM – Timer Input Module
TOM – Timer Output Module
ATOM – ARU-connected TOM
MCS – Multi Channel Sequencer
SPE – Sensor Pattern Evaluation
PSM – Parameter Storage Module
AFD – AEI-to-FIFO Data Interface
FIFO – Parameter Storage
F2A – FIFO-to-ARU Bridge
MON – Monitor Unit
CMP – Output Compare Unit
AEIMux – AEI Bus multiplexer

Figure 2.3: GTM block diagram.

In order to make a clear classification, the submodules represented can be categorized into the following types:

- the infrastructural components, marked with green color in Figure 2.3, pertain to those responsible for routing data coming from the CPU or the internal modules and directed towards other modules within (this is the case of the PSM, BRC, and ARU). This category includes also the TBU, providing a wide time base, the CMU, offering a clock prescaler for all the modules, and the ICM, responsible for bundling the interrupts.

- the Input/Output submodules, marked with blue color, with the purpose to sample external signals and produce complex output signals. This classification concerns the core components for the generation of PWM signals, such as the ATOM, the TOM, the TIM block and the DTM, able to combine also the outputs of the others.

- the application specific submodules, marked in yellow, representing those components which cover different application domains. The DPLL, MAP and SPE modules are part of this category.

- the MCS programmable core, a RISC-like processor, with an own internal RAM, that operates on the signals generated by the GTM or provided by the CPU.

- submodules to support functional safety, marked in gray, like the CMP, that can compare two adjacent channel outputs from ATOM or TOM, and the MON component, with the goal to monitors the internal clocks and MCS functionality.

2.4 GTM modules

In this section, the descriptions of some important modules for this research work will be provided.

2.4.1 TIM

The Timer Input Module provides a measurement and management system for GTM's input signals and it can be controlled by both the CPU and the ARU.
It is equipped with dedicated filters to eliminate any potential glitches in the signals and offers five configuration modes allowing measurements on the examined inputs.
At this point, it could be crucial to highlight how the instances of each internal component within the General Timer that will be described are actually composed of multiple channels, and therefore, multiple signal paths, which can operate independently of each other, enabling parallel functionality of the system and reducing data processing time.
For what concerns the TIM, there are six instances of the component,

each with eight channels, these ones characterized by a data measurement unit and a filter, as previously mentioned.
It is possible to set various time thresholds on the filter to determine how they should be utilized for 'cleaning' an input signal. For a clearer explanation, these thresholds can be regarded as values that dictate the time (controlled by an internal counter) during which the input signal remains constant, removing sudden changes (called glitches).
Once the input data is validated by the filter, it can be sent to the Signal Measurement Unit for the processing phase. Within this unit, some operations are applied, including counting edges or measuring the signal level time.
Each result from the measurement unit is stored in dedicated shadow registers that must be processed before the arrival of a new value. These registers, like the majority in the GTM, consist of 24 bits.
The general structure of the TIM is illustrated in the Figure 2.4, providing an overview of the organization of the aforementioned units.

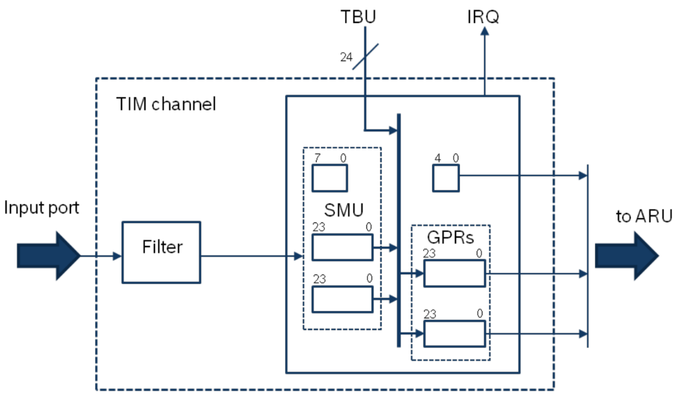


Figure 2.4: TIM channel diagram.

In addition, among the modes that can be set for an individual channel, and that primarily decide what operation has to be executed, two specific modes should be mentioned:

- the TIM PWM Measurement mode (TPWM), that measures the duty cycle and period length of an incoming signal in number of clock ticks, using an internal counter of 24 bits.

- the TIM Input Event Mode (TIEM), which enables the use of the register counter to keep track of the number of edges in the input signal.

These two settings will be utilized in the module's test programs explained in a dedicated section of this research.

2.4.2 ARU

The ARU is an important component to guarantee the flexibility of the GTM, since it's the core of the communication system for the submodules when the generic timer has to operate independently of the external processor.
The ARU routing data register is designed as shown in the Figure 2.5.

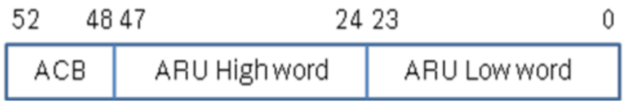


Figure 2.5: ARU data organization.

There is a register of 53 bits, divided into three parts. The first one, the ACB field of 3 bits, is used for the control signals, the other ones, both 24-bit wide, are used to store the values to be transferred.
A pivotal aspect of the GTM involves the routing mechanism employed within the ARU submodule to handle data streams. Grasping this concept is essential for effectively leveraging the resources of the timer. Every module connected to the ARU provide a variable number of ARU write channels, that they can be referred to as data sources, and ARU read channels, called data destinations. The underlying principle is based on the system of a time multiplex communication scheme;

consequently, at any given moment, there exists a unique path active in the ARU stream between two channels.
The choice to forego the implementation of a switch matrix, to prefer instead the employing of a serialized connectivity data router, takes the advantage to optimize the resource cost, but always ensuring the versatility for the GTM.
In Figure 2.6 it's represented the ARU data routing principle, where data sources are shown as green rectangles and data destinations as yellow rectangles. The dashed lines within the ARU delineate configurable connections between data sources and data destinations.

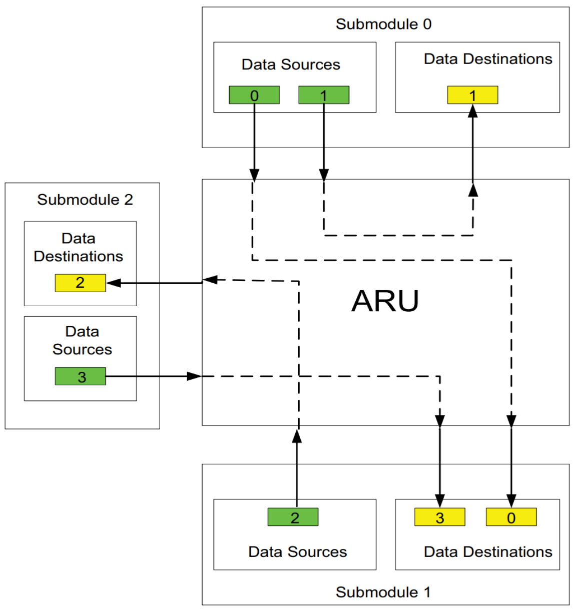


Figure 2.6: ARU routing mechanism.

Considering that each data source has its unique address, the ARU operates in the following manner: it systematically checks the data destinations of the linked modules in a round-robin sequence. When a destination requests new data from its configured data source, and this one is valid, the ARU transports the data to the destination, notifying

the successful processing of the transmission.
Then, the data source marks the delivered ARU data as invalid, indicating that the destination has consumed the data. If this not happened, for instance there isn't any data to be read yet, the destination waits until the source channel provides a new value, this process is called ARU blocking mechanism. The concept is shown in Figure 2.7.

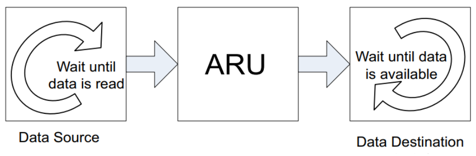


Figure 2.7: ARU blocking mechanism.

It is important to emphasize that each data source should be exclusively connected to a single data destination. Indeed, if two destinations pointed to the same source, one destination would consume the data before the other could do, causing a problem of conflict access to the resources.

2.4.3 MCS

The Multi Channel Sequencer (MCS) is a programmable data processing module connected to the ARU and it stands out as one of the most powerful component of the GTM.
It can be used for different kind of applications in order to reduce the CPU load, such as a complex engine position management [8].
Five instances of this submodule are present in the Generic Timer version used in this research, each with eight channels.
Its functionality involves the computation of complex output sequences, using data provided by the CPU or by the ARU in combination with other submodule, for instance values coming from the TIM module.

A picture of its internal architecture is presented in Figure 2.8.

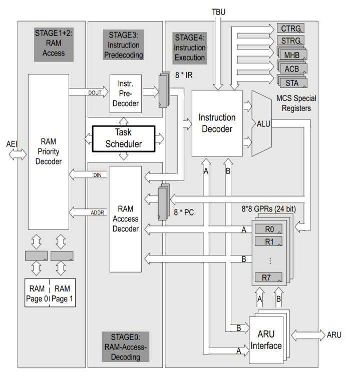


Figure 2.8: MCS architecture.

It executes tasks based on a RISC-like instruction set stored in its dedicated local RAM, in which, during the system startup, the MCS program is loaded by the CPU via the bus interface.
The RAM is divided into two separate blocks, each with an own interface for parallel access. Each RAM page has the capacity to hold code and data sections of arbitrary size, all of which are accessible by every MCS channel.
The memory layout contains up to 2^{12} locations each 32-bit wide, therefore a maximum byte range from 0 to $2^{14} - 1$.
All the eight channels can be active at the same time, for each instance of MCS, and they can operate as independent threads and exchange data. Each of them is equipped with its own program counter and register set, executing instructions from the same program or, instead, in parallel from different programs.
The MCS submodule incorporates an Arithmetic Logic Unit (ALU) with a RISC-like instruction set and multiple register sets, so that there is no need to save registers on the stack during a task switch.
Thanks to a pipelined approach of five stages, the processor achieves the execution of one instruction per clock cycle. In this way one task fetches a new command from the RAM, while another task decodes its

instruction, and a third task utilizes the ALU for the data execution. Figure 2.9 aims to elucidate the MCS pipelining.

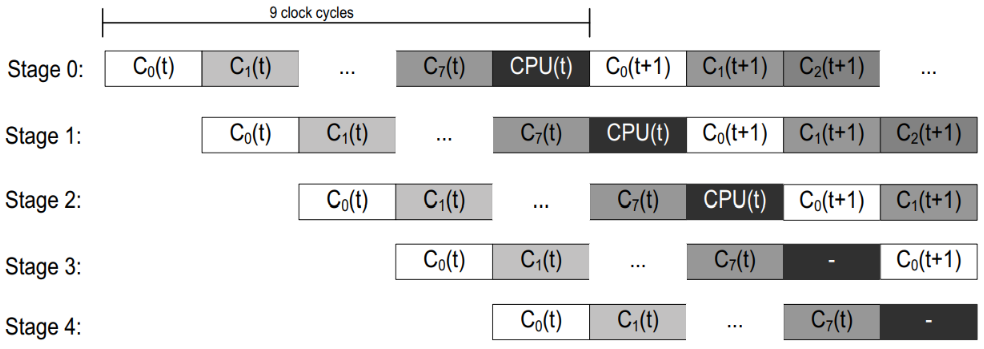


Figure 2.9: MCS scheduling.

The illustration delineates all the stages of the pipeline and the tasks arranged in a timing order.
It is evident that, when all tasks are active or when utilizing round-robin scheduling mode, it takes nine cycles until the subsequent instruction of a task can be executed. This last ninth cycle is allocated to provide the CPU an advantage in accessing the RAM section from which tasks retrieve their instructions.
Round-robin scheduling signifies that each task has its designated slot for execution.
In addition, there is also a second scheduling mode, called accelerated scheduling, that improves the computational performance of the MCS by skipping suspended MCS-channels.
Regarding the instruction set, each instruction is 32 bit wide but the duration varies in the range of 5 to 9 clock cycles, according to the number of suspended channels.
The instruction format is shown in the Figure 2.10.

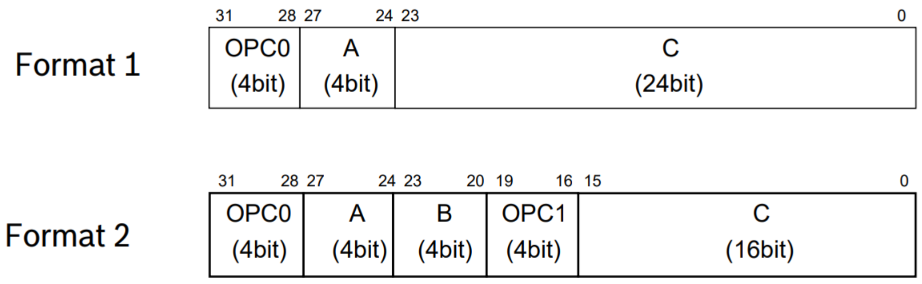


Figure 2.10: MCS Instuctions Formats.

As it's possible to observe there are two configurations for the bit alignment:

- the first one depicted in Figure 2.10 is the literal instruction format, for that instructions that accesses a 24 bit literal and a single 24 bit resister as operand.
- the second one is the double operand instruction format, for instructions that can access two operands stored in 24-bit registers

In addition, it is also possible to identify different categories within the instruction set based on the purpose of each operation. This classification is divided into:

- Data transfer functions, which can store data, provided either by the MCS or by the CPU, into internal registers or memory locations of the processor. Instructions such as POP and PUSH, used for stack interactions, also fall into this category.
- ARU instructions, involving all the write and read access operations of the ARU.
- Arithmetic logic instructions, covering all operations executed by the ALU component of the datapath.

- Test instructions, which regard some instructions used to compare different operands.
- Control flow instructions, representing the basic instructions for branching and function calls.
- Other instructions that do not fit into the aforementioned categories, such as NOP, which performs no operation, and WURM, used to wait for one or more trigger events generated by other MCS channels or the CPU.

It is important to highlight that, in order to program the MCS, it is necessary to write the code on a specific assembler source file, which has the dedicated extension ".mcs".
Using the Bosch ASM-MCS tool, the instructions are translated into machine code, generating a C-code array which is then processed by a compiler along with the rest of the program.
For helping the assembler converter on how to translate the instructions, some suitable directives are used for this purpose. All instructions and directives are listed in manual [6].
As described in [9], to make this conversion, from the channel sequencer code to the machine code, it has been used the command:

```
asm-mcs.exe -o out_file.c -odef out_file.h -olbl mcs_-
mem -I source_dir -I <mcs assembly source directory>
soucefile.mcs
```

The assembly tool takes as input the ".mcs" file, including the instructions that are used for the program, and a ".inc" configuration file, with some useful architecture specifications.
As outputs, it generates:

- a C-code file with the instructions array
- a header file with the label and memory offset definitions

Finally, these files can be integrate in a project, and the instructions can be executed according to the scheduling 2.9.

2.4.4 ATOM

The ARU-connected Timer Output Module (ATOM) is a component which can produce complex signals on the outputs of the GTM.
As shown in Figure 2.11, its channels are composed of two compare units, referred to as CCU0 and CCU1, that includes the CM0 and CM1 register and the associated shadow registers SR0 and SR1, used as a sort of terminal values for the counters CNT inside the channel.
In particular the compare registers CM0 and CM1 represent reference time values for the counter,in order to create the desired signal output.

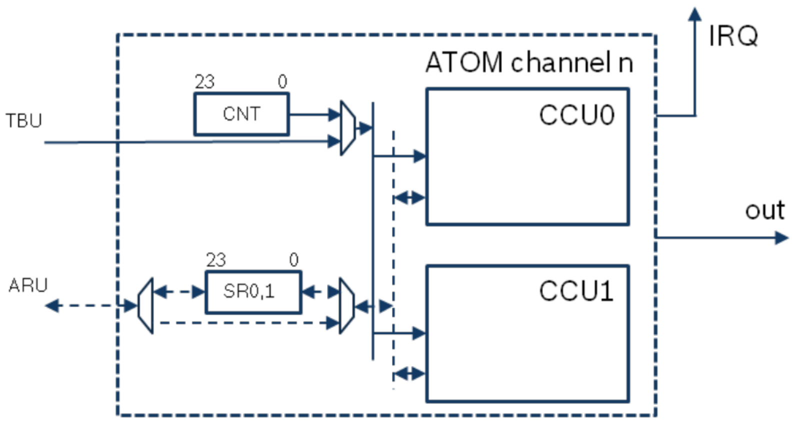


Figure 2.11: ATOM channel diagram.

This ATOM shows hardware characteristic similar to the Timer Output Module, with the only main difference that a source linked to

the ARU can be routed to feed the compare registers, both 24 bits wide.
Such as the TIM, it's also possible to change the functionality by setting the operation mode, choosing among four of them, and once enabled, each channel can operate independently from the others. These modes are the following:

- Signal Output Mode PWM (SOMP)
- Signal Output Mode Compare (SOMC)
- Signal Output Mode Immediate (SOMI)
- Signal Output Mode Serial (SOMS)
- Signal Output Mode Buffer Compare (SOMB)

As previously done for the TIM component description, only the modes of interest will be explained.
In SOMP mode, the ATOM channel is available for utilization as a PWM generator. The registers CM0 and CM1 store the period and the duty cycle values, respectively. Simultaneously, the shadow registers can be employed to concurrently define new characteristics for the upcoming PWM signal.
As observed from the Figure 2.12 the feature of this timer consists on the possibility to use the ARU; therefore, the shadow registers SR0 and SR1 can be reloaded with new data coming from the 53 bit wide ARU word, implementing a two-stage pipeline in which the period and duty cycle are set before the effective utilization.

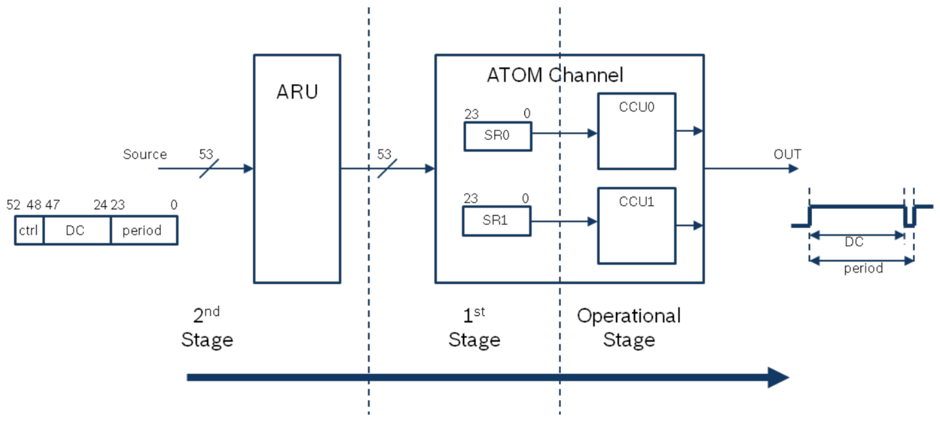


Figure 2.12: ATOM SOMP mode scheme.

The other mode to be mentioned is the SOMC modality, the most complex of this submodule.
In this context, the updates of the output signals are not determined by the independent counter registers within each channel; instead, they are implemented by the global time bases generated by the TBU submodule. In SOMC, either the CPU or a source connected to the ARU can supply compare values to the ATOM channel.
The CM0 and CM1 registers within the CCU0 and CCU1 units, or better their shadow registers, can be dynamically updated with new compare values until a match event is detected by one of the units. A match is raised when the counter CNT reaches a compare value, updating the output bit according to the options set.
Subsequently, it is feasible to input new compare values, but they will not take immediate effect, because before either the CPU or a destination connected to the ARU must read at least one value.
The menage of the compare mechanism within the two units involves several control bits, that has the role to decide in which way the output has to be updated, to define the compare strategy, such as comparing in both the units simultaneously, or even to use a different TBU time base values, for instance to facilitate the definition of an output signal

event on two distinct time bases.

Chapter 3

Hardware and software environments

3.1 Hardware instrumentation

This section of the research will introduce and describe the hardware instrumentation that has been employed.
The Device Under Test (DUT), for which the System Level Test programs have been developed, is identified as an industrial device named Bernina, produced by STMicroelectronics and belonging to the SPC58 device family. However it would be more appropriate to use the term of Component Under Test (CUT), since the target for which the tests have been created corresponds to a specific component of the microcontroller, a timer module named GTM344.
Developed by Bosch, this module is used for complex computations on digital inputs, captured in real-time, in order to generate any output signal shapes with Pulse Width Modulation (PWM).
To gain an overall overview, a reference has been made in Figure 3.1, which depicts the complete system setup.

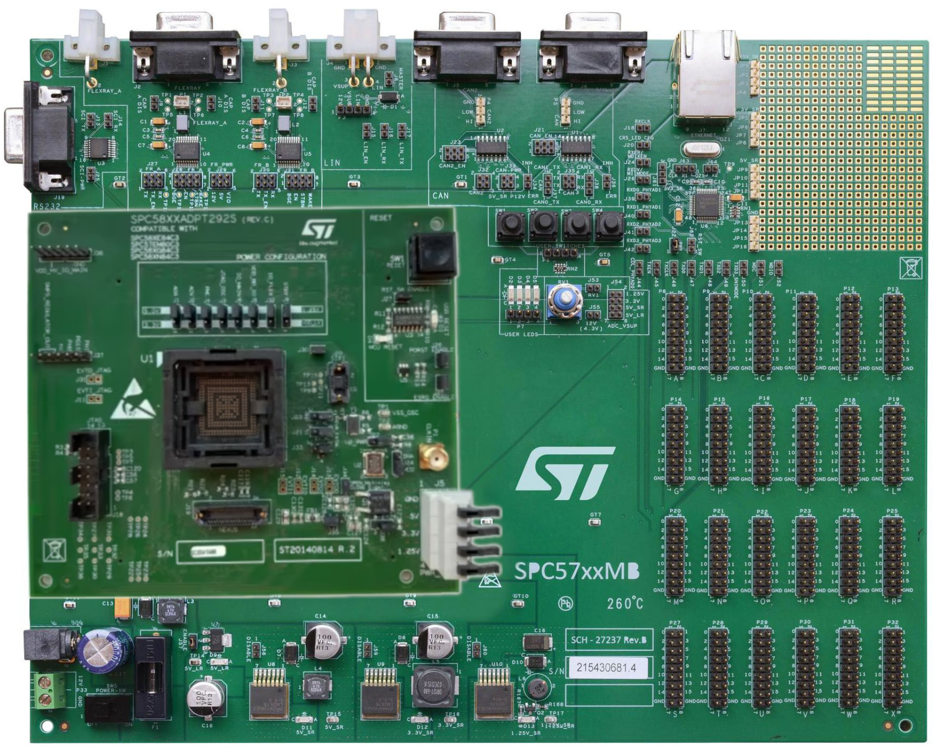


Figure 3.1: Hardware setup of the motherboard and daughterboard.

As can be seen, the hardware setup consists on a SPC57XXMB motherboard and a daughter card SPC58xxADPT, which plugs into the motherboard.
This configuration allows a full access to the CPU and all motherboard peripherals, serving as a support for the microcontroller device that is placed in a dedicated MCU socket.
The motherboard provides a single external 12V power supply, with the possibility of four on-board voltage regulators, and offers also a set of freely connectable pins and LEDs, which can be configured according to the requirements of the program being run.
Referring to the daughter board, the small one shown in Figure 3.1, it's possible to observe, on its left, the Jtag connector, which has been utilized to connect the USB/JTAG interface SPC5-UDESTK, depicted in Figure 3.2.

This component is required in order to enable the debugger functionalities and consequently verify the proper operation of the software.

Figure 3.2: SPC5-UDESTK debugger.

For what concerns the whole environment setup, instead, Figure 3.3 depicts a representative diagram of the system in which the research work has been performed.

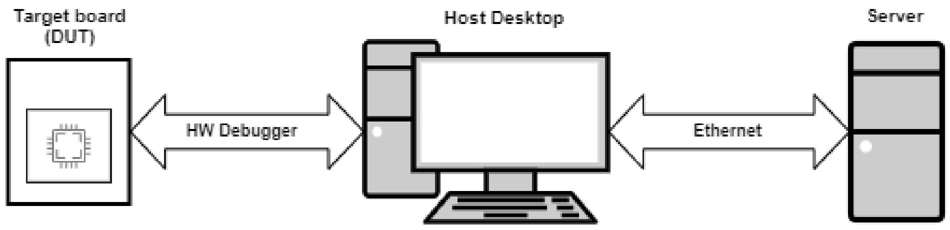


Figure 3.3: Hardware environment.

To sum up, the DUT is connected through the aforementioned debugger interface to a host computer, allowing interaction with the device, while a dedicated server has been provided in order to accomplish each operation related to the simulations of the functional programs. Indeed, it has been necessary as environment setup to apply all the analysis scripts needed, and to gain all the information about the coverage and the stimulated signals.

3.2 Software tools

In this section, the software platforms and tools used for the research are presented.
Regarding the code writing and debugger access, STMicroelectronics provides a free integrated development environment called SPC5-Studio. It consists of a built-on Eclipse plug-in development environment (PDE), fully customized by the user to define new components and to better control them [10].
It generates ANSI C compliant code, supporting the MISRA 2012 standard quality, and it provides an intuitive user interface and a comprehensive framework to design, build, and deploy programs for SPC5x family 32-bit architecture, such as the Bernina device.
In order to test the Generic Timer Module, several programs have been written using this software application, loaded into the board's flash memory, and tested through the debugger.
Additionally, some libraries provided by SPC5-Studio have been used to manage each module of the GTM at a high level. These libraries have been slightly extended during this research to adapt them to the needs of the testing procedures.
After developing and debugging a program, the files generated from its compilation have been uploaded to a dedicated server, that provided the environment for simulation and analysis.
The publication [11] can be a reference guide for what concerns the software used and the analysis procedure flow, because this research can be considered an in-depth study of a specific part of the experiments discussed in the article. Therefore the tools will not be explained in detail, but the ones of interest for this work will be briefly described.
The evaluation process for the SLT methods, that has been adopted, follows the steps outlined in the diagram of Figure 3.4.

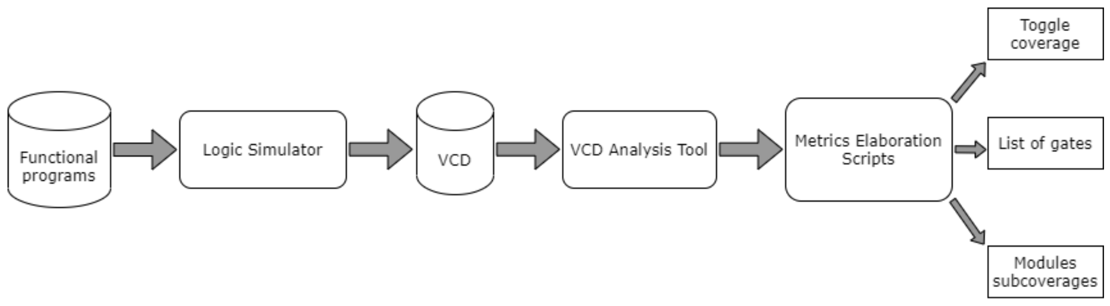


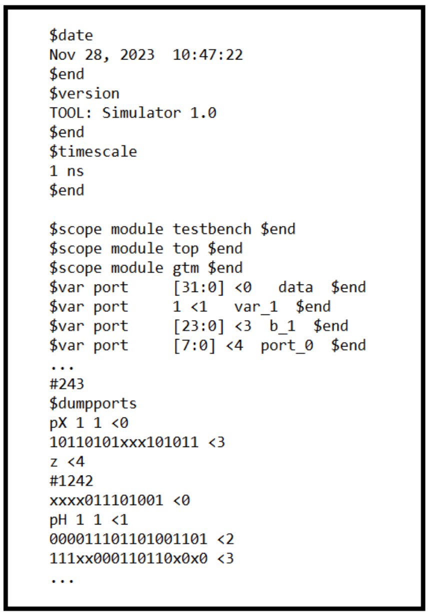
Figure 3.4: Software workflow.

The process starts from the functional programs loaded on the server. It is important to mention that an additional initial step in the represented flow can be considered. Indeed, it might be necessary to check the actual functionality of the program through an RTL simulation, before the gate level analysis, performed in this case by a commercial tool. This operation allows a kind of debugging in a high-level abstraction environment to verify the correct behavior of the system in a short amount of time, since it employs relatively low computational resources.
On the contrary, the netlist-based gate simulation, to which the functional procedures are subjected, represents the step that requires the highest amount of time, not only because it involves complex models of circuit interconnections and logic gates, but also because it has to write the VCD file, containing detailed information about signal value changes.
Two types of VCD documents exist:

- Four state, to represent variable changes in 0, 1, x, and z, if in binary format, with no strength information.
- Extended, to represent variable changes in all states and strength information.

This file begins with a header section with the date, the version of the simulator utilized for the simulation, and the timescale used.

Then, the definition of the scope and variables being recorded are reported, succeeded by the changes at each simulation time increment [12].
An example is shown in Figure 3.5.



```
$date
Nov 28, 2023  10:47:22
$end
$version
TOOL: Simulator 1.0
$end
$timescale
1 ns
$end

$scope module testbench $end
$scope module top $end
$scope module gtm $end
$var port     [31:0] <0   data  $end
$var port     1 <1   var_1  $end
$var port     [23:0] <3  b_1  $end
$var port     [7:0] <4  port_0  $end
...
#243
$dumpports
pX 1 1 <0
10110101xxx101011 <3
z <4
#1242
xxxx011101001 <0
pH 1 1 <1
000011101101001101 <2
111xx000110110x0x0 <3
...
```

Figure 3.5: VCD file example.

The file of interest for this research is the extended version one, briefly named eVCD.
As described in [11], this file is a crucial element of the toolchain, as it's used as input for the dedicated designed analyzer tool which may take a considerable amount of time for performing the analysis, since the eVCD could be very large (hundreds of GBytes). This tool aims to provide some useful stress metrics information; among them, the full single-point metric (toggle coverage) represents an important data point for this research. Indeed, it gives an indication of the total number of gates that the programs, executed on the physical device, needs to cover.
To better understand this concept, it is essential to highlight what is

meant by "covering a gate".
As explained in the article [11], a gate is considered covered when it operates both the transitions (i.e., from 0 to 1 and vice versa, from 1 to 0). Such coverage ensures that a logical node within the circuit has been thoroughly tested.
The developed toolchain involves also the pruning of that signals which do not affect the stress metrics during the evaluation phase. This filtering method is achieved by matching the information in the VCD file with the list of the actual gates that make up the physical device, thus providing an accurate coverage of the SoC under examination.
Finally, in the last steps of this workflow, some scripts are provided to collect all the data needed to characterized a test, such as the number of gates not toggled, the coverage referred to each submodule of the GTM and the list of signals with the corresponding executed transitions.
After each evaluation of a single program, the results are merged with the previous ones, using a dedicated script, in order to gain a more meaningful incremental coverage to keep track of progress.

Chapter 4

Workflow

4.1 Workflow introduction

The work done for this research aimed to identify the most suitable method for testing a specific complex peripheral timer, taking into account the Generic Timer Module as subject of study.
In order to follow a logical work flow consistent with the results obtained from simulations, the first programs have been based on simple logical insights, to proceed then with the different methods explored after a more accurate analysis of the available data. For instance, a deep study of the covered and not covered signals has allowed to consider some solutions over others.
Each test is detailed and followed by some examples to understand better its goal and the ideas that led to its designed.

4.2 Research proposal

As starting point, it has conducted an analysis of the physical areas of the Bernina microcontroller, using the heatmaps, provided in [11], in order to identify possible regions of the hardware insufficiently stimulated by the structural tests, with a primary focus on the toggle coverage. The picture in Figure 4.1 represents the final result, obtained from the analyzer tool, of the superimposition of different stress patterns, such as:

- 32 scan based ATPG patterns
- 12 selective ATPG patterns
- 1024 scan based pseudo-random patterns
- LBIST patterns
- MBIST patterns
- some functional patterns

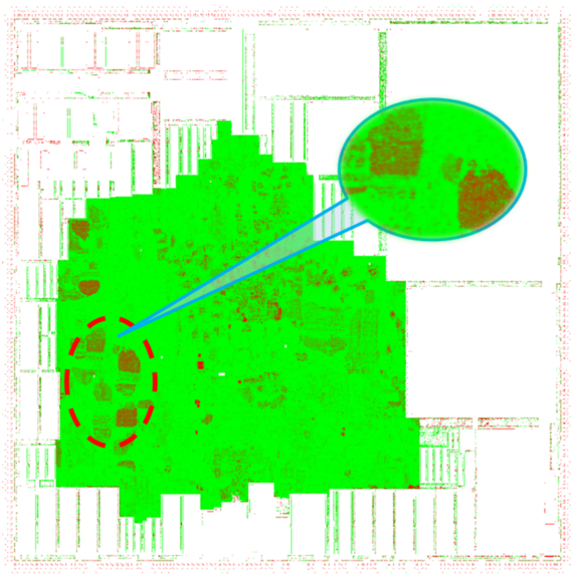


Figure 4.1: Heatmap of the overall stress provided by the superimposition of all stress patterns.

The red colored areas in Figure 4.1 represent the regions of interest that did not achieve a sufficient stress level after the microcontroller

underwent all the structural and functional tests listed. The region inside the red circles is an example of these areas.
Therefore, analyzing the image, some "shadow zones" have been identified, located in those circuit locations where, among the components that composed Bernina, the timer module GTM has been implemented. One possible explanation for the lack of coverage in this module could lie in its implementation.
Therefore, the Generic Timer Module, designed by Bosch, has been interconnected using a dedicated hardware interface, different from the rest of the device, that potentially has hindered the specific automatic tests from effectively exciting the signals and gates within the component itself.
The conclusion drawn has been the need to prioritize the development of functional programs targeting the area of interest. This has been done not only to achieve a more comprehensive coverage from a signal testing perspective, but also to ensure the proper functioning of the key features of such a complex module not originally part of the main system.
So, after having identified the problem and some potential resolutions, the initial phase would undoubtedly involve an in-depth study of the device under test, in order to understand its usage and pinpoint the weaknesses.
The work carried out has followed the scheme in Figure 4.2.

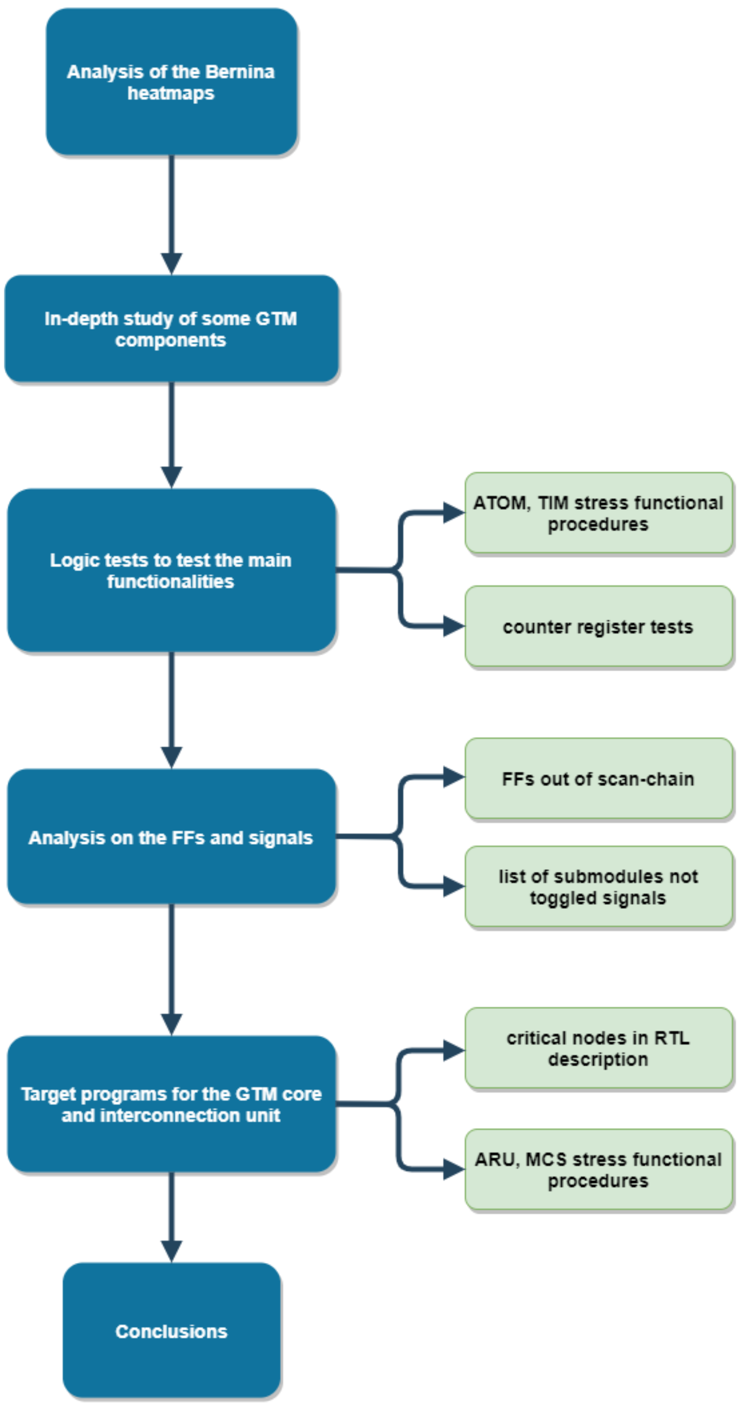

Analysis of the Bernina heatmaps
In-depth study of some GTM components
Logic tests to test the main functionalities
ATOM, TIM stress functional procedures
counter register tests
Analysis on the FFs and signals
FFs out of scan-chain
list of submodules not toggled signals
Target programs for the GTM core and interconnection unit
critical nodes in RTL description
ARU, MCS stress functional procedures
Conclusions

Figure 4.2: Work process flowchart.

4.3 In-depth study of the GTM

The first step has involved the studying of the manual [6] to establish a basic understanding of the GTM's functionalities, identifying the more complex modules and those that could be used more frequently.
This initial phase also served as a sort of "testing ground" to become familiar with the module, developing some programs with the sole purpose of utilizing the GTM and studying its behavior.
In this regard, also the guide [7] has been consulted, which has allowed to make use of some useful examples, such as the procedures that leveraged the interaction between the different submodules within the timer.

4.4 Logic test procedures

The first case of study has been focused on the input and output modules of the GTM.
The programs developed for testing, that follow a System Level Test approach, aimed to simulate the behavior of a system as close as possible to the actual component's usage.
The idea behind these first attempts has been to find a circuit element of these types of components that, when tested, could provide a significant increase in coverage and could toggle as many gates as possible.
This critical element has been identified in the counter registers, those most commonly used for the operations of a timer, but, most of all, those with the highest number of bits, so theoretically with a high probability of reaching a good number of untoggled signals.

4.4.1 ATOM SOMP tests

For the initial test has been involved the ATOM module, taken individually, and it has been decided to focus on a single channel of it, configured in SOMP mode by setting the appropriate control bits, in order to generate PWM signals, as described in Section 2.4.4.
Specific functions have been created ad-hoc to speedup its programming and configuration procedure.
The program was aimed to provide values to be stored in the comparison registers CM0 and CM1 of the CCU0 and CCU1 units, serving as a sort of terminal counter values for the 24-bit counter registers.
Since the focus has been limited on the use of the ATOM, the comparison values has been provided via the CPU to the shadow registers SR0 and SR1. These ones have the function of continuously supplying values to the CM0 and CM1 registers after each output period time.
The operation is illustrated in Figure 4.3.

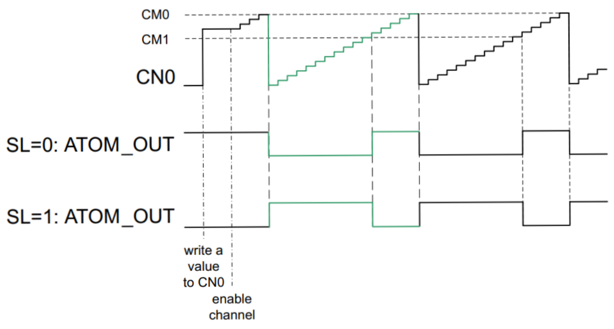


Figure 4.3: ATOM SOMP beahvior.

The 24-bit wide counter register CN0, starting from zero, allows to toggle the output, each time it reaches the value stored in CM1 and, subsequently, the value stored in CM0, returning then to the initial value. This results in the generation of a square wave, with the possibility to customize the duty cycle and period time based on the data stored in the compare registers. Instead, the Signal Level bit (SL)

influences the starting value of the output.
The CM0 and CM1 registers have been cyclically changed to test them with various possible values.
These functional procedures, however, have yielded unsatisfactory results, with an increase in coverage of only 0.001% and a simulation time of approximately 30 hours.
The problem of this approach consisted on the excessive repetitions in toggling the same signal, for instance the same bits in the counter register, as this would slow down the program execution, especially in simulation, due to the high number of signal changes.

4.4.2 ATOM Counters tests

Another issue to consider was that, by using the counters as they actually operate in a real environment, the registers were not uniformly tested.
A high number of bit transitions did not contribute to increasing the coverage, reproducing the same toggles in the same gates.
It has been crucial to aim for a uniform toggle activity across all the 24 bits of each register, in order to avoid waste of simulation time and achieve an adequate level of program efficiency.
Indeed, as outlined in Figure 4.4, the least significant bits appeared to be highly stimulated by the counter's usage, gradually decreasing towards the most significant bits.
This condition is plausible even for a non-testing-oriented use of the circuit.
Since the main functionalities of the Generic Timer Module also include the generation of complex outputs, such as PWM signals, it has been reasonable to assume that a counter would rarely reach high values in its maximum range, or at least less frequently than lower values that are revisited every time a comparison value was reached, as illustrated in the example of Figure 4.3.

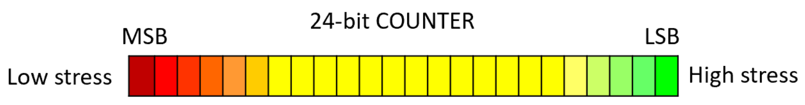


Figure 4.4: Scheme of the stress for a 24-bit wide counter register.

To achieve this goal, several techniques has been explored, testing different approaches.
Firstly, a "walking bit" method has been applied to the registers under examination, which consists on a standard technique ensuring a complete uniformity in bit transitions.
It involves the setting of the initial value to '1' (in the case of the "walking one") and then the "shifting" of this bit to the left, so that each bit undergoes both possible transitions only once. This process is illustrated in Figure 4.5.

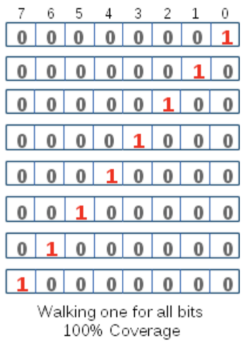


Figure 4.5: Walking bit of a register.

However, the application of this technique assumes that the counter should increment by a different value each time, which is not supported by the GTM settings.

Given this consideration, a possible solution has been to insert the values "manually", so by directly writing into the target registers without actually running the counter. This could only be achieved by disabling the ATOM channel and cyclically updating the CNT register with the correct values to perform the walking one.
In this case, the ATOM has been set in both SOMP and SOMC mode, meaning both with the generation of a PWM output and relying solely on triggering the output toggle (see Section 2.4.4 for an explanation in detail of these modes).
Another test has been then considered, more geared towards an approach that included both uniform coverage in the register and a real system utilization. This strategy has been aimed to use the counter more effectively, keeping the counter active only long enough to perform a single increment, preventing it from overloading excessively the simulation.
Specifically, it has planned to set the CNT register to optimal values that, with an increment, would toggle an high number of bits. The counter would then be stopped (or better reset to zero) immediately after this increment by correctly setting the comparison registers CMx. This procedure has been possible by configuring the ATOM channel in one-shot mode, which involved generating a single period of PWM signal. This choice has prevented the counter from automatically restarting once reaching the initial value, which was to be avoided for the purpose of this approach.
The diagram in the Figure 4.6 provides a clearer explanation of how this approach works.

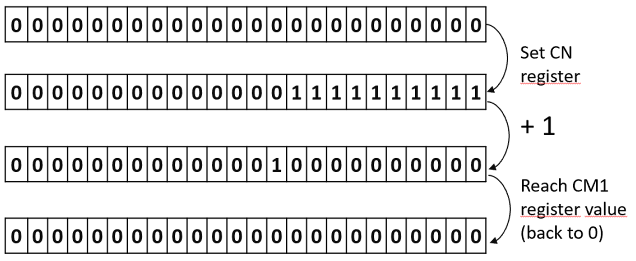


Figure 4.6: Example of stress counter technique.

The program described has been finally applied to all the eight channels of all the six instances of the ATOM module.
Furthermore, it is worth considering that other registers have been stressed to expand the testing area of the program, such as that dedicated for the control operations, that aimed for the configuration, and some shadow registers

4.4.3 TIM tests

Then, it has been the turn of the Timer Input Module of the GTM, which has proved to be a more challenging component to test.
In fact, applying the same approach as for the ATOM to this component, presents a problem due to the design of the GTM, since it is a module that does not offer much freedom in managing the counters, at least not for testing purposes.
A emblematic case is the TIM, as it has many counter registers within each channel that do not allow the possibility of loading values into them from the CPU.
To get a clearer idea, let's consider an example by looking at the internal structure of the TIM channel, as shown in Figure 4.7.

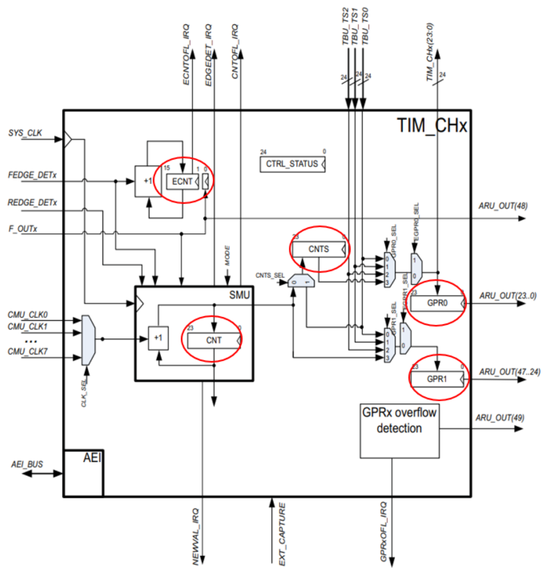


Figure 4.7: TIM channel structure.

The registers circled in red in the image are the channel counters (CNT, ECNT, and CNTS) and some registers where output values are temporarily stored (GPR0 and GPR1). They all represent examples of 24-bit registers, and therefore, they are some of the largest registers in the entire GTM, but they cannot be overwritten directly by the CPU. Testing them can only be done by starting the normal functionality of the counters, resulting in long simulation times.
Therefore, the option of trying to test the registers for reduced bit ranges, smaller than the maximum one corresponding to 24 bits, has been considered.
Subsequently, the results from the simulations would be collected to decide whether, given the increase in coverage achieved, it would be advantageous to spend a lot of time reaching the most significant bits of these register counters.
The most suitable mode for configuring the TIM for this purpose has been found to be the TIEM mode, already cited in Section 2.4.1. This timer option allows to count the edges of the input signal (rising, falling

or both, depending on the preference), using the CNT register, shown in the Figure 4.7, to keep track of the count.
So, for instance, if the goal was to stimulate the last 4 bits of the counter, it would have been sufficient to toggle the input $(2^4 - 1)$ times, so that the register would reach the MSB of the chosen range with the value "1111".
Given that the TIM allows to provide input not only from the components outside the GTM, but also directly from the CPU, the latter option has been chosen, iteratively changing a value so that, during the program execution, a number of edges equal to the desired value have been created. This procedure has been tested for 4, 8, and 12-bits counters.
After several tests, the results in this case have showed a slight increase in coverage; however, increasing the number of bits used for a counter has proved to be an inefficient strategy, since, contrary to expectations, the toggle coverage did not increase.

4.5 ATOM and TIM tests results

The simulation results have been reported in Table 4.1 and Table 4.2.

Table 4.1: Table of ATOM and TIM tests coverages.

SLT application	**Toggle coverage** [%]					
	Total ports	Toggled ports	Not toggled Ports	Single coverage	Incremental coverage	Delta
structural(32 atpg)	3785917	2951554	834363	78.13	78.130	0.000
structural(1024 pseudo random + 12 selective + 3 delay + mbist + lbist)	3785917	3616021	169896	95.51	95.510	17.380
TIM(4-bit counter)	3785917	3616207	169710	8.77	95.518	0.008
TIM(8-bit counter)	3785917	3616215	169702	8.78	95.518	0.000
TIM(12-bit counter)	3785917	3616218	169699	8.78	95.518	0.000
ATOM(walking_bit)	3785917	3616269	169648	10.17	95.519	0.001
ATOM(stress_test)	3785917	3616356	169561	10.86	95.523	0.004

Table 4.2: Table of ATOM and TIM test simulation data.

SLT application	**Simulation**		
	Clock Cycles [cc]	Execution Time [ms] @80Mhz	Simulation Time [h]
TIM(4-bit counter)	1675118	20.94	19
TIM(8-bit counter)	1727466	21.59	20
TIM(12-bit counter)	1846378	23.08	20
ATOM(walking_bit)	726642	9.08	11
ATOM(stress_test)	934642	11.68	14

4.5.1 Tables format description

It is now necessary to briefly explain how the table 4.1 and 4.2 are organized, because the same format and fields will be used to represent the results in the subsequent sections as well.
The tables present the System Level Test (SLT) programs developed in each phase of this research, dividing them into rows and showing the most significant information obtained from both their simulation and from the elaboration scripts discussed in Section 3.2.
In detail, the collected information includes:

- Total ports, the total number of gates within the GTM.
- Toggled ports, the number of signals stressed by the specific program combined with all the previous ones.
- Not toggled ports, the total number of gates that have not yet been toggled.
- Single coverage, the coverage with respect to the Generic Timer Module and obtained only from the individual program, without considering the previous ones.
- Incremental coverage, the coverage produced by the intersection between the single program and all the previous ones.
- Delta, the incremental difference between the single functional test coverage and that belonging to the previous test.
- Clock Cycles, the number of clock cycles of the test simulated.
- Execution Time, the actual duration time of the program (in ms), with a clock frequency set at 80 MHz.
- Simulation Time, a value indicating the time taken by the tool to simulate the test (expressed in hours).

In addition, it is important to consider that in all the table types, as 4.1, that will be presented, the first two rows have been dedicated to the structural tests mentioned in the reference article [11], both for the test patterns generated through the ATPG and for the others scan-based techniques.
Regarding these two cases, only some significant information has been reported, such as the total toggle coverage, which is essential to understand the extent of the increment achieved by the subsequent functional tests. However, all the information about the simulations are not reported (Not Available), since these fields refer to functional tests, instead of structural ones.

4.6 Analysis of the GTM Flip-flops and signals

After attempting several other programs applied to multiple submodules of the Generic Timer, it has been decided to proceed with a more analytical approach, seeking to derive useful information from the RTL structure of the component and from the signal lists, before operating the actual tests.
First of all, since the purpose of these System Level Test procedures was to complete a work already started with a structural testing approach, a specific consideration was given to the scan chain used in these tests, as described in Section 1.3.
In particular, a useful information could be the total number and the location of the flip-flops not included in the chain, and thus maybe not yet tested.
Indeed, after obtaining the list of all flip-flops in the Bernina microcontroller, divided into two files in order to distinguish between those transformed into scan cells and those not, a specific Python script has been created to recursively search through this list for all flip-flops

included in the GTM.
Then, a second script has been devised to compare the flip-flop names, in these lists, with the names of the toggled and non-toggled gates present in those files that have been generated by the toolchain after applying the structural tests.
So, in the end, the results of this process have been collected and organized into a table, similar to the one described below.

Table 4.3: Table of GTM Flip-flops.

Flip-Flops	**Total FFs**	**Scan FFs**	**nScan FFs**	**Scan FFs nt**	**nScan FFs nt**
GTM	88081	83488	4593	233	35
gtm_control	2364	2302	62	58	5
adc_interface	496	474	22	45	0
ip_gtmdi_syn	5774	5556	218	42	11
gtm_ip	79347	75063	4284	88	19

The table 4.4 displays the total number of flip-flops present in the Generic Timer Module, followed by the count of the scan FFs, the number of those not included in the scan chain, and, finally, in the last two columns, those of both types that haven't reached a toggle coverage equal to 100%. The latter, in particular, are considered "uncovered" since not all signals associated with these gates have been toggled, or have not been tested with both the bit transitions (from '0' to '1' and from '1' to '0').
While the data of the second row are referred to the GTM, the other ones show the FFs included of all the four sub-entities that composed the timer.
The first three rows refer to components dedicated to control bits, the external interrupt management, and the ADC interface used in

conjunction with the timer.
However, it's the last row (named gtm_ip) that represents the actual RTL enitity descibing the structure of the GTM, with all the modules already mentioned in Section 2.4.
What is easily evident from the data is that this part has the majority of the non-scanned FFs, making it the most problematic.
Additionally, an aspect that was not expected from these results has been the number of scan FFs not fully covered, greater than previously thought, most of all with respect to the total of those that are not part of the scan chain.
The signals related to these "untoggled gates" have been listed in the following table:

Table 4.4: Table of GTM Flip-flops not toggled signals.

Flip-Flops signals	Total	Input(D)	Reset(SN)	Enable(E)
Scan FFs Not Toggled	233	162	25	46
FFs Not Toggled	35	0	0	35

The nodes in the circuit not covered by the scan chain are primarily related to Enable signals of some FFs. However, concerning the others, there is a significant amount of nodes corresponding to the FFs inputs. This may suggest that there are parts of the system that have not been reached by previous tests, and so not adequately tested during simulation.
However, determining the exact location of these flip-flops within the timer has been quite challenging. Therefore, an attempt was made to gather more information through a detailed analysis of the files containing the lists of toggled and untoggled gates within the GTM.
For the analysis, a script whas been used to provide information about the signals, dividing them by submodules.

Table 4.5: Table of GTM not toggled signals divided into submodules.

Not toggled signals	Total nt signals	ATOM	TIM	TOM	DPLL	TBU	CMU	AEI, ARU	MCS
structural (1024 pseudo random, 12 selective, 3 delay, mbist, lbist)	169896	1151	449	127	610	114	49	645	166751
ATOM(walking_bit)	169648	999	449	127	606	114	49	553	166751
ATOM(stress_test)	169561	895	449	127	606	114	49	570	166751
ATOM(ctrl_regs)	169581	935	449	127	606	114	49	550	166751
TIM(counter 4-bits)	169807	1151	440	127	606	114	49	569	166751
TIM(counter 8-bits)	169804	1151	439	127	606	114	49	567	166751
TIM(counter 12-bits)	169803	1151	439	127	606	114	49	566	166751

The values reported in the Table 4.5 show the the number of the untoggled signals for the most significant modules that composed the GTM; each row represent the results collected by some example tests for ATOM and TIM, already explained in the dedicated subsection. In addition, in the second row, the respective values obtained from the application of the structural tests are reported.
It was thus possible to observe that the initial tests did not stimulate a sufficient number of gates. However, the most significant finding concerned the module where almost all the signals were located, namely the Multi-Channel Sequencer (described in Section 2.4.3), a component that had not been considered up to that point.
Therefore, the subsequent efforts has been focused on this part of the generic timer, initially studying its functionalities and then developing ad-hoc programs in order to reach an higher coverage.

4.7 SLT programs for the Multi Channel Sequencer

The first step has involved ad in-depth study of the MCS instruction set and the writing of some test programs, in order both to gain familiarity with the module's usage and to obtain the first simulation results. The aim has been to verify whether this component represented the critical part of the GTM.

4.7.1 First MCS functional programs

The initial functional procedure has been applied to the first of the five channel sequencers (referred to as MCS0), focusing on only one of the eight channels that allow parallel execution of the program (referred to as Channel 0).
The chosen approach has involved a comprehensive test of the entire instruction set, systematically trying each operation supported by the MCS.
The process has begun with the creation of the ".mcs" file, containing the dedicated program. Subsequently, following the procedure described in Section 3.2, this document has been translated into the corresponding array representing the program in C-code, placed in the ".c" file, along with the respective header file ".h", which included the directive definitions. For this purpose, the assembler tool provided among the plug-ins of the SPC5-Studio application has been utilized.
Given the necessity to ensure that the program executed by the CPU did not finish before the code executed by the Multi-Channel Sequencer, as this would prematurely terminate the simulation, a check has been inserted at the end of the program. This check involved the CPU verifying that a specific register internal to the module stored a key value assigned by a final instruction in the MCS code, in this case the

R7 register has been chosen among the eights provided.
The procedure has been applied also for the first three channels of MCS0 and the results of the simulations have been collected.
However, the coverage values and the number of toggled signals, obtained from the simulations, were consistent with the previous tests, failing to provide a significant increase in the coverage of the GTM component.
These values are reported in the first rows of Table 4.6, and they are compared to the results of the structural tests that initiated this research work, while the Table 4.7 shows the data about the execution time of the simulations.

Table 4.6: Table of first MCS tests coverages.

SLT application	**Toggle coverage** [%]					
	Total ports	Toggled ports	Not toggled Ports	Single coverage	Incremental coverage	Delta
structural(32 atpg)	3785917	2951554	834363	78.13	78.130	0.000
structural(1024 pseudo random + 12 selective + 3 delay + mbist + lbist)	3785917	3616021	169896	95.51	95.510	17.380
MCS0_prova	3785917	3616389	169528	5.72	95.523	0.013
MCS0_test(1 channel)	3785917	3616782	169135	8.18	95.533	0.010
MCS0_test(3 channels)	3785917	3617288	168629	8.50	95.546	0.013
MCS0_all_channels	3785917	3618109	167808	9.03	95.568	0.022
MCS1_all_channels	3785917	3619688	166229	9.09	95.610	0.042
MCS2_all_channels	3785917	3621214	164703	9.34	95.650	0.040
MCS3_all_channels	3785917	3621388	164529	8.92	95.655	0.005
MCS4_all_channels	3785917	3621702	164215	8.89	95.663	0.008
MCS012(alu_test)	3785917	3621633	164284	16.69	95.664	0.001
MCS34(alu_test)	3785917	3621753	164164	13.16	95.665	0.001

It should be considered, however, that the low incremental coverage values may be justified by the fact that only a portion of the MCS0 channels has been stimulated.
Therefore, the next step has been to extend the test to all the eight channels of the first channel sequencer, and then to subject the same process to the other MCS modules.
The new tests, also reported in Table 4.6 (labeled as MCSx_all_channels), show much more promising results, with increments that reach up to 0.042% for what concerns the delta value.
Although this has represented a slight improvement in the overall system view, since it has indicated that the direction taken could have been the right one.
For the sake of completeness, two additional functional procedures have been added, focusing on specific tests for the ALU of the datapath and on the general-purpose registers of the five MCS. The goal has been to apply more intense stress on the bits of these registers and to specifically utilize only those instructions involving the arithmetic unit.

Table 4.7: Table of first MCS test simulation data.

SLT application	**Simulation**		
	Clock Cycles [cc]	Execution Time [ms] @80MHz	Simulation Time [h]
MCS0_prova	243581	4.10	5
MCS0_test(1 channel)	291281	4.70	5
MCS0_test(3 channels)	364475	5.61	6
MCS0_all_channels	373149	5.72	7
MCS1_all_channels	465671	6.88	7
MCS2_all_channels	467471	6.90	7
MCS3_all_channels	471568	6.95	7
MCS4_all_channels	471568	6.95	7
MCS012(alu_test)	1101877	14.83	15
MCS34(alu_test)	952977	12.97	14

4.7.2 MCS0-ARU test programs

Following the simulation of these tests, a more specific analysis has been conducted on the signals related to these RISC-like processors.
In particular, from the lists obtained from the respective elaboration scripts, it has been noticed that a significant portion of the untoggled gates were located in the circuitry area comprising the connection interface with the ARU, a component described in Section 2.4.2.
This discovery has been crucial since it has allowed to shift the target of this research towards a specific zone of each MCS, focusing the attention on a potential weak point of the whole system.
From this analysis, it has been possible to identify in the RTL description of the circuit the processes related to the gates to be stressed. These processes handle the execution of those instructions, belonging to the

MCS, that are used for data transmission through the ARU component. These instructions are the following:

- AWR/AWRI, that execute a write access to the ARU to transfer two 24-bit values from two general registers to the ARU, using a literal value to define the write address port. They block the current MCS channel until the operation is finished.
- ARD/ARDI, which, on the contrary of the previous instructions, execute a blocking read access to the ARU, storing two 24-bit values in the MCS registers. Also in this case a literal value is used, but to define the ARU read address.
- NARD/NARDI, that have the same purpose of ARD/ARDI instructions, but they suspend the MCS channel only for a maximum of one ARU round trip cycle.

Therefore, the functional procedures have been programmed using only these instructions. However, to execute them, it has been necessary to enable other modules through which an MCS core could communicate via ARU.
For this purpose, the TIM and ATOM components have been chosen, as they were those submodules with the greater number of gates to stimulate and the greater number of ARU addresses available.
So, as a first step, each channel of the MCS0 component has been subjected to dedicated programs aimed at writing to the buffer register of the ARU, a register that has been subsequently read by every channel of all available ATOMs in the GTM.
This has been achieved by trying various combinations of MCS general registers used to transfer data and by using all associations between the MCS addresses and the addresses of the ATOMs.
In order o optimize the simulation times, for each of the eight channels of the MCS0, a program has been created. In this way more than one code could be simulated at the same time, without wasting time in case of issues.

Simultaneously, an ad-hoc procedure has been developed for the TIM components, this time for reading instructions performed by the selected channel sequencer, since, unlike the ATOMs, the TIM components did not allow the writing.
For this test, only one program has been simulated, as it fell within acceptable timeframes.
The same reading procedure has been then applied to the ATOM submodules.
A simple scheme of the program approach has been provided in Figure 4.8.

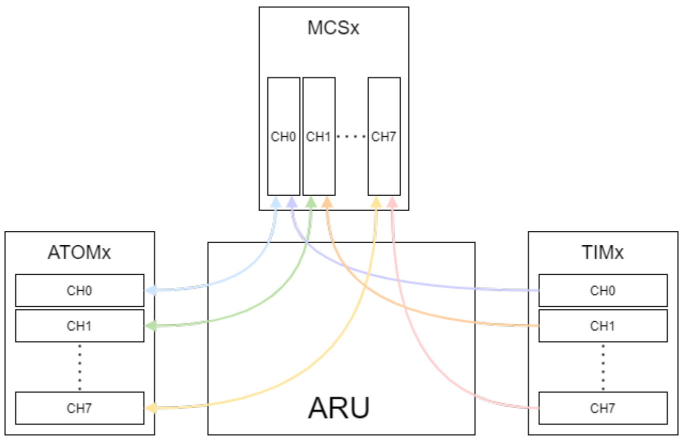


Figure 4.8: MCS and ARU scheme.

Furthermore, it is worth to highlight that a key point for these tests has been the use of the WURM instruction, detailed in the manual [6], as a synchronizing method between the program executed by the CPU of the board and the small program executed by the MCS0 core, a necessary measure to allow the simulation of the entire code.
The results has been collected in the Table 4.8 and Table 4.9.

In the tables, a stress test applied only to the buffer register of the ARU has been also reported; however it did not yield promising results. On the contrary, the eight programs focused on writing data from

Table 4.8: Table of MCS0 tests coverages.

SLT application	**Toggle coverage** [%]					
	Total ports	Toggled ports	Not toggled Ports	Single coverage	Incremental coverage	Delta
structural(32 atpg)	3785917	2951554	834363	78.13	78.130	0.000
structural(1024 pseudo random + 12 selective + 3 delay + mbist + lbist)	3785917	3616021	169896	95.51	95.510	17.380
ARU_test	3785917	3621753	164164	5.89	95.665	0.155
MCS0_awr_ch0	3785917	3623104	162813	11.26	95.700	0.035
MCS0_awr_ch1	3785917	3624816	161101	11.24	95.745	0.045
MCS0_awr_ch2	3785917	3626782	159135	11.14	95.797	0.052
MCS0_awr_ch3	3785917	3628571	157346	12.59	95.844	0.047
MCS0_awr_ch4	3785917	3629994	155923	12.59	95.882	0.038
MCS0_awr_ch5	3785917	3631297	154620	12.56	95.916	0.034
MCS0_awr_ch6	3785917	3632704	153213	11.24	95.954	0.038
MCS0_awr_ch7	3785917	3633982	151935	12.24	95.987	0.033
MCS0_TIM_ard	3785917	3637740	148177	12.24	96.087	0.100
MCS0_ATOM_ard	3785917	3637964	147953	12.64	96.100	0.013

the MCS to the ATOM (denoted as MCS0_awr_chx) have showed better results compared to all previous tests, demonstrating that the stimulated part of the circuit was the right target to achieve a significant increase in coverage.
Indeed, as can be observed in Table 4.8, each of the test, dedicated to a single MCS channel, has produced an increase in toggle coverage ranging from 0.033% to 0.047%, for a total delta equal to 0.322%. Regarding the tests on the ARU reading (denoted as MCS0_TIM_ard and MCS0_ATOM_ard) which involves the TIM, they have shown an increase of 0.113%, but with a disparity in the contribution provided. Indeed, the test related to the TIM have showed a delta of 0.100%, while the one related to the ATOM have provided a delta of 0.013%.

Table 4.9: Table of MCS0 tests simulation data.

SLT application	**Simulation**		
	Clock Cycles [cc]	Execution Time [ms] @80Mhz	Simulation Time [h]
ARU_test	2440924	31,57	25
MCS0_awr_ch0	5414259	68,74	37
MCS0_awr_ch1	5414263	68,74	37
MCS0_awr_ch2	5365663	68,13	37
MCS0_awr_ch3	5365713	68,13	37
MCS0_awr_ch4	5414313	68,74	37
MCS0_awr_ch5	5414313	68,74	37
MCS0_awr_ch6	5414213	68,74	37
MCS0_awr_ch7	5365563	68,13	37
MCS0_TIM_ard	6979398	88,30	42
MCS0_ATOM_ard	7155803	90,50	51

This could be attributed to the fact that both, despite using different ARU addresses, have stimulated the same part of the circuit that executes the read instructions (ARD/ARDI and NARD/NARDI), providing almost identical single coverage contribution and stressing the same gates.
The total increase in toggle coverage has been equal to 0.435%.
Having received a positive feedback from the previous tests, it has been reasonable to make the hypothesis that similar outcomes could be attained also by the other MCS instances.
To validate this theory, the programs have been adapted and executed on the remaining channel sequencers.

4.7.3 MCS1-ARU test programs

Thus, the same functional procedures which have been applied previously to the first MCS, have been replicated on the second MCS (referred to as MCS1), and the results of the simulations have been collected in Table 4.10 and Table 4.11.

Table 4.10: Table of MCS1 tests coverages.

SLT application	**Toggle coverage [%]**					
	Total ports	Toggled ports	Not toggled Ports	Single coverage	Incremental coverage	Delta
structural(32 atpg)	3785917	2951554	834363	78.13	78.130	0.000
structural(1024 pseudo random + 12 selective + 3 delay + mbist + lbist)	3785917	3616021	169896	95.51	95.510	17.380
MCS1_awr_ch0	3785917	3639917	146000	11.81	96.150	0.640
MCS1_awr_ch1	3785917	3641394	144523	11.78	96.191	0.041
MCS1_awr_ch2	3785917	3643391	142526	11.86	96.237	0.046
MCS1_awr_ch3	3785917	3644995	140922	11.87	96.285	0.048
MCS1_awr_ch4	3785917	3646365	139552	11.86	96.321	0.036
MCS1_awr_ch5	3785917	3647860	138057	11.88	96.358	0.037
MCS1_awr_ch6	3785917	3649064	136853	11.87	96.390	0.032
MCS1_awr_ch7	3785917	3654719	131198	11.86	96.438	0.048
MCS1_TIM_ard	3785917	3654201	131716	12.28	96.528	0.090
MCS1_ATOM_ard	3785917	3654315	131602	12.56	96.533	0.005

The tests have been organized in the same order as in Table 4.8, and, as anticipated, very similar values have been obtained with respect to those observed with MCS0.
Indeed, the total increment obtained from the eight ARU writing programs amounted to 0.319%, relative to the coverage of the last simulated test.
Additionally, the other two ARU reading tests, combined with ATOM

Table 4.11: Table of MCS1 tests simulation data.

SLT application	Simulation		
	Clock Cycles [cc]	Execution Time [ms] @80MHz	Simulation Time [h]
MCS1_awr_ch0	5629635	71,43	37
MCS1_awr_ch1	5629639	71,43	37
MCS1_awr_ch2	5629789	71,43	37
MCS1_awr_ch3	5629789	71,43	37
MCS1_awr_ch4	5629789	71,43	37
MCS1_awr_ch5	5629939	71,43	37
MCS1_awr_ch6	5629789	71,43	37
MCS1_awr_ch7	5629635	71,43	37
MCS1_TIM_ard	7078974	89,50	49
MCS1_ATOM_ard	7316179	92,50	54

and TIM modules, have contributed with an additional 0.095%, resulting in a total delta of 0.414%.

4.7.4 MCS2-ARU test programs

The same process, already tested for the MCS0 and MCS1, has been applied to the third MCS, called MCS2.
The results are shown in Table 4.12 and Table 4.13, organized with the same format of the previous tables.

Table 4.12: Table of MCS2 tests coverages.

SLT application	**Toggle coverage [%]**					
	Total ports	Toggled ports	Not toggled Ports	Single coverage	Incremental coverage	Delta
structural(32 atpg)	3785917	2951554	834363	78.13	78.130	0.000
structural(1024 pseudo random + 12 selective + 3 delay + mbist + lbist)	3785917	3616021	169896	95.51	95.510	17.380
MCS2_awr_ch0	3785917	3655459	130458	11.88	96.555	1.045
MCS2_awr_ch1	3785917	3657366	128551	11.84	96.605	0.050
MCS2_awr_ch2	3785917	3659377	126540	11.86	96.658	0.053
MCS2_awr_ch3	3785917	3660671	125246	11.85	96.692	0.034
MCS2_awr_ch4	3785917	3662262	123655	11.86	96.734	0.042
MCS2_awr_ch5	3785917	3663787	122130	11.85	96.775	0.041
MCS2_awr_ch6	3785917	3664925	120992	11.86	96.805	0.030
MCS2_awr_ch7	3785917	3666210	119707	11.85	96.839	0.034
MCS2_TIM_ard	3785917	3670167	115750	12.27	96.943	0.104

In this case, the coverage has been incremented of a delta value equal to 0.410%, aligned with the previous results.

Table 4.13: Table of MCS2 tests simulation data.

SLT application	**Simulation**		
	Clock Cycles [cc]	Execution Time [ms] @80MHz	Simulation Time [h]
MCS2_awr_ch0	5629635	71,43	37
MCS2_awr_ch1	5629639	71,43	37
MCS2_awr_ch2	5629789	71,43	37
MCS2_awr_ch3	5629789	71,43	37
MCS2_awr_ch4	5629789	71,43	37
MCS2_awr_ch5	5629939	71,43	37
MCS2_awr_ch6	5629489	71,43	37
MCS2_awr_ch7	5629939	71,43	37
MCS2_TIM_ard	7076274	89,51	49

4.7.5 MCS subcoverages

In order to better understand how much the performed tests have contributed to improving the reliability of these individual modules, the components of the Multi-Channel Sequencers have been further analyzed using a dedicated processing script suitable to gain the different sub-coverages of the components taken individually, distinguishing between the parts (RTL description entities) that compose them.
From this study's results, the coverage of each MCS and their respective toggled gates have been derived, focusing on the comparison between those gained before the application of the stress programs and those collected after the System Level Testing.
The results have been organized in the Table 4.14.
The table is organized so that the first five rows refer to all those gates

Table 4.14: Table of MCS modules coverages

MCS signals analysis	Total ports	Toggled ports	Not toggled Ports	Not toggld ports (MCS012 tests)	Single coverage	Single coverage (MCS012 tests)
MCS0 (AEI)	12472	12459	13	13	99.90	99.90
MCS1 (AEI)	3404	3386	18	18	99.47	99.47
MCS2 (AEI)	3497	3464	33	33	99.06	99.06
MCS3 (AEI)	3211	3185	26	26	99.19	99.19
MCS4 (AEI)	3335	3323	12	12	99.64	99.64
Total	**25919**	**25817**	**102**	**102**	**99.61**	**99.61**
MCS0(ARU,registers)	302764	271414	31350	13442	89.65	95.56
MCS1(ARU,registers)	307517	276154	31363	13888	89.90	95.48
MCS2(ARU,registers)	318968	285122	33846	16041	89.39	94.97
MCS3(ARU,registers)	316440	280827	35613	35467	88.75	88.79
MCS4(ARU,registers)	309410	275649	33761	33429	89.20	89.20
Total	**1555099**	**1389166**	**165933**	**112267**	**89.33**	**92.80**

that are included in the part of the circuit representing the interface between the MCS components and the AEI (called Generic Bus Interface), especially regarding the data transfer with the CPU.
The five categories in the second part of the table, instead, show the signals belonging to the internal system of each MCS and to the communication with the ARU.
Toggled and non-toggled gates, along with their respective toggle coverage, are collected.
At the end of each of the two parts, the total count is computed for the gates, and the average is calculated for what concerns the coverage percentage.
These data have been obtained prior to the application of the dedicated programs, unlike the two columns that include the statement "(MCS012 tests)", which instead show the results subsequent to the tests on the first three MCS.
Going into detail, in the first group there are few signals to be toggled, with individual channel sequencer coverages that remain very high, averaging 99.61%.
As expected, the situation is different in the second group, which, as already explained, includes the main gates of the circuit of all five MCSs, including the interface with the ARU. Indeed, in the initial results, no

component exceeded the 90% coverage threshold, demonstrating an unacceptable stress level.
However, the increase provided by the functional tests corresponds to 4.17% for the MCS0, to 5.33% for the MCS1 and to 5.58% for the MCS2, reaching an average of 92.80%, above an acceptable threshold but still improvable by applying the same tests to the remaining modules.

Chapter 5

Results and conclusions

5.1 Final results analysis

In this section, a summary table of all the tests performed from the beginning to the end of this research will be shown, with the different programs sorted in chronological order of development. In addition, some final results regarding the toggled signals and the heatmaps will be discussed, accompanied by a brief reflection on the work carried out.
In Table 5.2, it is possible to observe how the most significant results coincide with the phase of the workflow following the in-depth analysis of the system, where the Multi Channel Sequencer has been identified as the weak point of the circuit in terms of gate and signal stress level. These results, collected using the elaborations scripts, demonstrate the difficulty of trying to increase a coverage percentage that is already high for the previous performed structural tests, however they also highlight the challenges in stimulating the correct nodes of a complex module such as the Generic Timer Module.
The final toggle coverage obtained has been of 96.943%, which is an acceptable value that can still be increased by developing specific tests

for the remaining MCS modules.
Considering that the results of each tested MCS showed an incremental delta ranging between 0.4% and 0.5%, it is plausible to assume that the others modules would also show similar values, reaching an ideal coverage between 97.743% and 97.943%.

Table 5.1: Summary table of GTM tests coverages (Part1).

SLT application		**Toggle coverage [%]**					
		Total ports	Toggled ports	Not toggled Ports	Single coverage	Incremental coverage	Delta
0	**structural(32 atpg)**	3785917	2951554	834363	78.13	78.130	0.000
1	**structural(1024 pseudo random + 12 selective + 3 delay + mbist + lbist)**	3785917	3616021	169896	95.51	95.510	17.380
2	**TIM(4-bit counter)**	3785917	3616207	169710	8.77	95.518	0.008
3	**TIM(8-bit counter)**	3785917	3616215	169702	8.78	95.518	0.000
4	**TIM(12-bit counter)**	3785917	3616218	169699	8.78	95.518	0.000
5	**ATOM(walking_bit)**	3785917	3616269	169648	10.17	95.519	0.001
6	**ATOM(stress_test)**	3785917	3616359	169558	10.86	95.523	0.004
7	**MCS0_prova**	3785917	3616389	169528	5.72	95.523	0.000
8	**MCS0_test(1 channel)**	3785917	3616782	169135	8.18	95.533	0.010
9	**MCS0_test(3 channels)**	3785917	3617288	168629	8.50	95.546	0.013
10	**MCS0_all_channels**	3785917	3618109	167808	9.03	95.568	0.022
11	**MCS1_all_channels**	3785917	3619688	166229	9.09	95.610	0.042
12	**MCS2_all_channels**	3785917	3621214	164703	9.34	95.650	0.040
13	**MCS3_all_channels**	3785917	3621388	164529	8.92	95.655	0.005
14	**MCS4_all_channels**	3785917	3621702	164215	8.89	95.663	0.008
15	**MCS012(alu_test)**	3785917	3621733	164184	16.69	95.664	0.001
16	**MCS34(alu_test)**	3785917	3621753	164164	13.16	95.665	0.001
17	**ARU_test**	3785917	3621753	164164	5.89	95.665	0.000
18	**MCS0_awr_ch0**	3785917	3623104	162813	11.26	95.700	0.035
19	**MCS0_awr_ch1**	3785917	3624816	161101	11.24	95.745	0.045
20	**MCS0_awr_ch2**	3785917	3626782	159135	11.14	95.797	0.052
21	**MCS0_awr_ch3**	3785917	3628571	157346	12.59	95.844	0.047
22	**MCS0_awr_ch4**	3785917	3629994	155923	12.59	95.882	0.038
23	**MCS0_awr_ch5**	3785917	3631297	154620	12.56	95.916	0.034
24	**MCS0_awr_ch6**	3785917	3632704	153213	11.24	95.954	0.038
25	**MCS0_awr_ch7**	3785917	3633982	151935	12.24	95.987	0.033
26	**MCS0_TIM_ard**	3785917	3637740	148177	12.24	96.087	0.100
27	**MCS0_ATOM_ard**	3785917	3637964	147953	12.64	96.100	0.013
28	**MCS1_awr_ch0**	3785917	3639917	146000	11.81	96.150	0.050
29	**MCS1_awr_ch1**	3785917	3641394	144523	11.78	96.191	0.041
30	**MCS1_awr_ch2**	3785917	3643391	142526	11.86	96.237	0.046
31	**MCS1_awr_ch3**	3785917	3644995	140922	11.87	96.285	0.048
32	**MCS1_awr_ch4**	3785917	3646365	139552	11.86	96.321	0.036
33	**MCS1_awr_ch5**	3785917	3647860	138057	11.88	96.358	0.037

Table 5.2: Summary table of GTM tests coverages (Part 2).

SLT application		**Toggle coverage [%]**					
		Total ports	Toggled ports	Not toggled Ports	Single coverage	Incremental coverage	Delta
34	**MCS1__awr__ch6**	3785917	3649064	136853	11.87	96.390	0.032
35	**MCS1__awr__ch7**	3785917	3649719	136198	11.86	96.438	0.048
36	**MCS1__TIM__ard**	3785917	3654201	131716	12.28	96.528	0.090
37	**MCS1__ATOM__ard**	3785917	3654315	131602	12.56	96.533	0.005
38	**MCS2__awr__ch0**	3785917	3655459	130458	11.88	96.555	0.022
39	**MCS2__awr__ch1**	3785917	3657366	128551	11.84	96.605	0.050
40	**MCS2__awr__ch2**	3785917	3659377	126540	11.86	96.658	0.053
41	**MCS2__awr__ch3**	3785917	3660671	125246	11.85	96.692	0.034
42	**MCS2__awr__ch4**	3785917	3662262	123655	11.86	96.734	0.042
43	**MCS2__awr__ch5**	3785917	3663787	122130	11.85	96.775	0.041
44	**MCS2__awr__ch6**	3785917	3664925	120992	11.86	96.805	0.030
45	**MCS2__awr__ch7**	3785917	3666210	119707	11.85	96.839	0.034
46	**MCS2__TIM__ard**	3785917	3670167	115750	12.27	96.943	0.104

5.1.1 Incremental toggled gates results

From the provided Table 5.2, it has been also possible to extract the number of gates specifically stimulated by each individual test developed, delineating a sort of trend in the contribution given to circuit stress in terms of the quantity of toggled elements.
This information has been represented in the graph depicted in Figure 5.1.

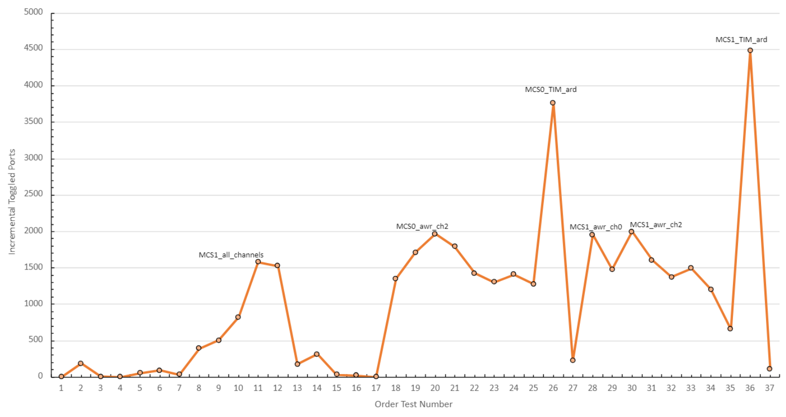


Figure 5.1: Stress gates graph.

The graph has been structured such that the number of stimulated gates is indicated on the y-axis, while the test identification number, as listed in Table 5.2, is represented on the x-axis.
The trend of the data interpolation shows some peaks corresponding to the tests targeting the MCS component, with the highest result obtained from programs aimed at the interaction between TIM and MCS.
This could be due to the high number of possible connection combinations with the different channels of the two submodules, focusing on a large area within the zone of interest. In addition, the intensive use of ARU read instructions, used in these type of programs, have played a key role in test execution.

5.1.2 Heatmaps results

In order to obtain a physical feedback on the layout of the Bernina board, it has been also possible to generate the heatmap of the entire system after the stimulation applied by the tests, showing the exact location of the gates of interest. This representation is shown in the Figure 5.2.

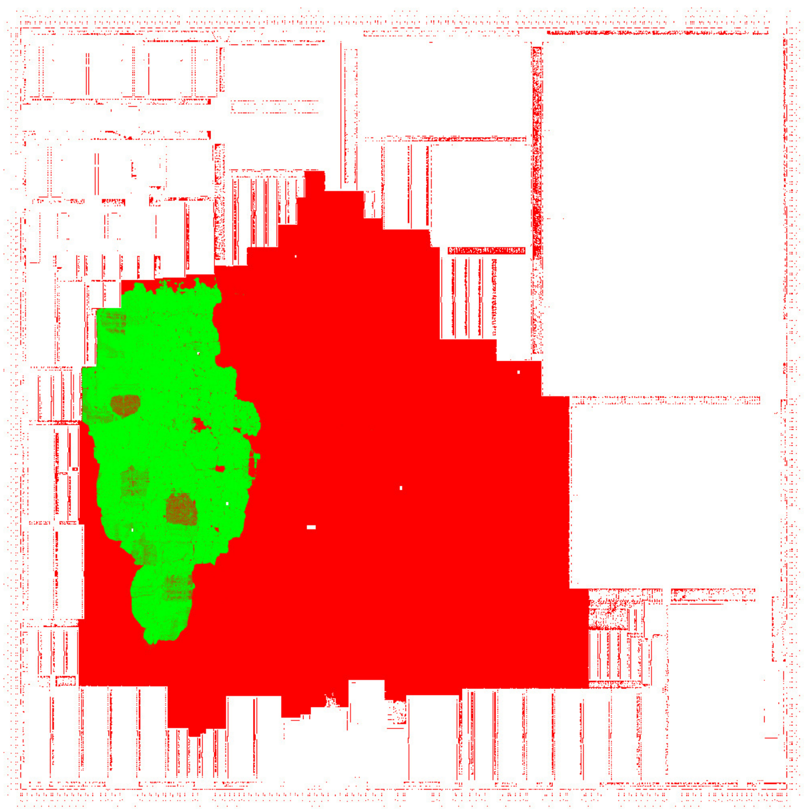


Figure 5.2: Heatmap of the SLT tests.

The green zones are that areas stressed by the functional tests listed in the Table 5.2, while the red zones are the parts of the circuit not stressed, and so not used. This representation allows to show the position of the Generic Timer Module in the microcontroller.
Finally, as last step of the research, an heatmap image representing the intersection between the SLT programs and the structural tests has been also generated. This has been done to facilitate a comparison with the initial heatmap, which only considered the structural approach. The two pictures are shown in Figures 5.3 and 5.4.

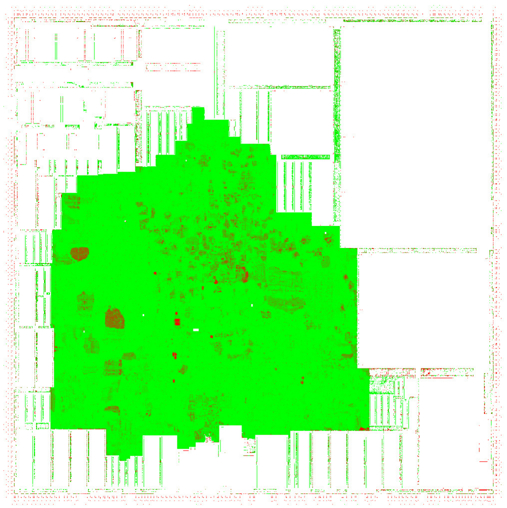


Figure 5.3: Heatmap - SLT and structural test application.

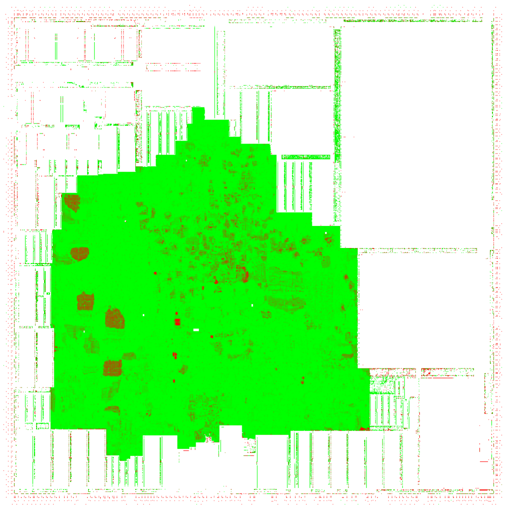


Figure 5.4: Heatmap - structural tests application.

It is possible to observe in Figure 5.4 that in the vicinity of the GTM area, some small red zones, since stressed, have been colored in green in Figure 5.3. Indeed, they could represent the modules of the Multi Channel Sequencer that have been utilized in the SLT programs.

5.2 Conclusions

After analyzing the obtained results, both for what concerns the study of the covered signals by the functional procedures and the derived toggle coverage percentages, it has been observe that there are certain areas of the timer that cannot be effectively stressed using the structural tests. Therefore, they require programs developed with an in-depth understanding of how the internal modules interact with each other. The combined use of the main timer output and input components, along with the data processing core and communication interface, managed to reach a coverage that hardly some scan-based structural tests and some simple functional procedures could gain.

Indeed, the purpose of this work has been to adopt a System Level Test approach aimed at mimicking, as closely as possible, the real-usage environment of the system, a functionality not efficiently possible for other techniques.
It should be also considered that this research can be continued by both stimulating the remaining MCS and using the ARU communication module in combination with other internal components of the GTM, in order to increase much more the overall coverage.
Furthermore, it is important to not underestimate that a significant goal has been reached. This research has successfully identified which part of a complex timer module should be taken into account to gain an efficient stress test, and how the different components should interact with each other to develop a suitable program.

Bibliography

[1] Bhunia Swarup and Tehranipoor Mark. *Hardware security*. 2019 (cit. on p. 2).

[2] Mentor Graphics Corp. *Scan Test*. 2022. URL: https://semiengineering.com/knowledge_centers/test/scan-test-2/ (cit. on p. 4).

[3] *System Level Test*. 2022. URL: https://www.teradyne.com/system-level-test/ (cit. on p. 6).

[4] Sounil Biswas and Bruce Cory. «An Industrial Study of System-Level Test». In: *IEEE Design Test of Computers* 29.1 (2012), pp. 19–27. DOI: 10.1109/MDT.2011.2178387 (cit. on p. 6).

[5] Polian Ilia et al. «Exploring the Mysteries of System-Level Test». In: (2021) (cit. on p. 6).

[6] *RM0361 Reference manual - Generic Timer Module specification revision 1.5.5.1*. STMicroelectronics. 2015 (cit. on pp. 10, 11, 23, 41, 60).

[7] *GTM-Cookbook - Overview and Application examples*. Bosch. 2014 (cit. on pp. 10, 41).

[8] Mukunda Byre Gowda, Deringer Carsten, and Karthikeyan Ramachandran. «Exploring the potential of a multi channel sequencer (MCS) in a next generation GTM-IP using virtual prototypes». In: (2017), pp. 1544–1548. DOI: 10.1109/RTEICT.2017.8256857 (cit. on p. 19).

[9] *AN5693- Application note - GTM and MCS complex PWM signal.* STMicroelectronics. 2021 (cit. on p. 23).

[10] *SPC5-STUDIO for 32-bit Power Architecture MCU's.* STMicroelectronics. 2020 (cit. on p. 32).

[11] Francesco Angione, Davide Appello, Paolo Bernardi, Andrea Calabrese, Stefano Quer, Sonza Reorda Matteo, Vincenzo Tancorre, and Roberto Ugioli. «A Toolchain to Quantify Burn-In Stress Effectiveness on Large Automotive System-on-Chips». In: *IEEE Access* (2023). DOI: `10.1109/ACCESS.2023.3316511` (cit. on pp. 32, 34, 35, 37, 51).

[12] «IEEE Standard Verilog Hardware Description Language». In: *IEEE Std 1364-2001* (2001), pp. 325–349 (cit. on p. 34).