UNIVERSITY OF TWENTE.

**Faculty of Electrical Engineering,
Mathematics & Computer Science**

Area-Optimized RISC-V-Based Control System for 22nm FDSOI Analog and Mixed-Signal Test Chips

Timon Kruiper

M.Sc. Thesis

January 2023

Committee:

Chair:	dr.ir. S.H. Gerez
Member:	dr.ir. M.S. Oude Alink
Member:	S. Yadav MSc
Additional member:	dr.ir. M. Ottavi

Computer Architecture
for Embedded Systems

Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Area-Optimized RISC-V-Based Control System for 22nm FDSOI Analog and Mixed-Signal Test Chips

Timon Kruiper, Computer Architecture for Embedded System, University of Twente, Enschede

***Abstract*—This paper presents the work on a RISC-V control system labeled UTRISCY. This design is an area-optimized, modular and reusable system designed for on-chip control to accelerate the testing by the ICD group on analog and mixed-signal CMOS designs. The system consists of several building blocks that communicate using standard bus interfaces, and can be configured to use one or two memory components depending on the performance needs. A comprehensive search for open-source RISC-V cores has been done, and the open-source SCR1 core is shown to be the most suitable for use in this system. A standard JTAG interface allows a debug probe connected to a host PC to program the memories of the UTRISCY control system, while also providing general communication capabilities. A voltage-controlled oscillator, the calibration of which is performed by UTRISCY, has been used as a typical use case. To verify the system, a comprehensive verification setup is provided that simulates the complete system, including the emulation of a connected debug probe using the OpenOCD debugger. Finally, the several components of the UTRISCY control system have been synthesized and placed-and-routed for the Globalfoundries' 22nm FDSOI technology (22FDX®), which results in an area usage of 57283 µm^2 including six bondpads, and two 4KB memories, with a maximum operating frequency of 156MHZ at 0.8V. It has been shown that this area can be reduced to 36710 µm^2 by using the smallest configuration of the SCR1 core and only including a single 2KB memory.**

I. Introduction

Today's process nodes are becoming smaller and smaller. This is beneficial for digital designs, since it allows for more transistors to be placed on the same die area. However, traditional analog/RF designs often do not benefit from this scaling, and the designs may even get worse due to the reduced supply headroom [1]. Besides that, smaller process nodes also make it harder to match two or more transistors to, for example, get equal bias currents, because the smaller manufacturing processes causes on-chip variations to increase [2]. Digital techniques can come to the rescue to compensate or even cancel these variations. This can be done by, for example, using off-chip or on-chip calibrations using mixed-signal feedback and digital signal processing.

As of today the Integrated Circuit Design (ICD) group of the University of Twente has a general solution for doing the calibrations off-chip. **Fig. 1** shows how these off-chip calibrations are typically set up. The device-under-test (DUT) is the analog and mixed-signal CMOS design being researched by the ICD group. Examples of these devices are analog-to-digital converters (ADCs), filters, oscillators etc. The one or more outputs of the DUT are typically analog signals, which are analyzed using specialized measurement equipment off-chip. To be able to change various parameters of the DUT, the UTCONTROL component is used. It is a simple shift-register-based design, that receives off-chip data serially using a SPI-like interface, and outputs it in parallel using double buffers to ensure that the intermediate state of the shift-register is not visible to the DUT [3]. A host PC can use the data of the measurement equipment to change the parameters of the DUT and transfers them onto the chip using UTCONTROL. This allows the analog designer to calibrate and characterize the DUT using a calibration algorithm that runs on the host PC.

For a DUT with a few parameters, such as the one shown in **Fig. 1**, UTCONTROL provides a viable solution to calibrate the parameters of the device. However, when a more complex design must be tested or calibrated, there are several problems/shortcomings that arise when UTCONTROL is used. The problems are listed below:

- The current version of UTCONTROL does not provide a way to inspect the digital state of the system. The interface is uni-directional, and for more complex system it is necessary to be able to inspect this state.
- The update rate of the parameters of the DUT is limited by the throughput provided by the interface of UTCONTROL. To control more complex systems, it is required to update the parameters at a higher rate than is currently possible with UTCONTROL.

Both of these problems could be solved by modifying UTCONTROL to use a bidirectional high-speed interface, instead of the slow serial interface that is currently

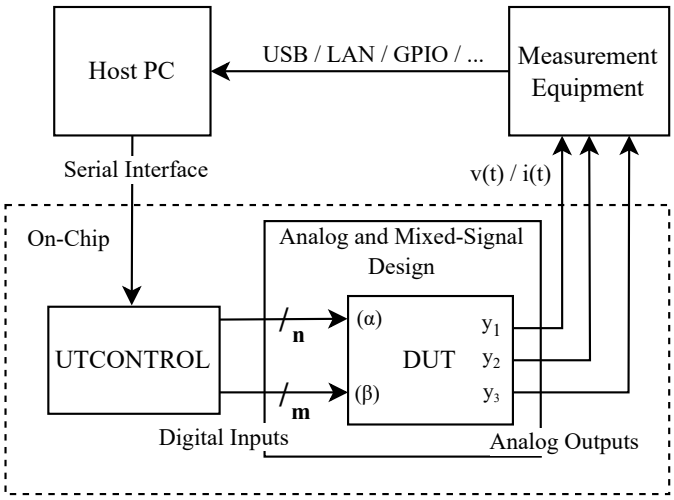

Fig. 1. Typical test setup of a research chip from the ICD group. The device-under-test (DUT) in this test setup is an analog and mixed-signal design being researched by the ICD group. The component used to control the parameters of the DUT is UTCONTROL. More information about the exact operation of UTCONTROL can be found on the ICDWiki [3]. The outputs of the DUT are analyzed off-chip using measurement equipment, which is then fed to the host PC, forming a control-loop.

used, however that comes with its own problems and challenges. Instead, this paper solves the shortcomings of UTCONTROL, by moving the control system on-chip. This means that test data could be generated on-chip, therefore skipping the need to use the slower interface to the outside world. Additionally, the analysis of the analog output signal of the DUT, could also be done on-chip using mixed-signal feedback and digital signal processing. In general, such a system provides the ICD group with a lot of flexibility. One important aspect is that the on-chip control system cannot use a lot of area on chip, because in general the analog and mixed-signal designs occupy a large chip area. Furthermore, more chip area means more expensive chips, so it is important for the on-chip control system to be area-optimized.

The on-chip control system should be configurable, and the configurability can be designed in a few ways. The first options is to make a configurable digital state-machine, designed for some use-cases specified by the ICD group. This has the advantage of resulting in a very small design, however it is also limited by the use-cases specified. For the most amount of configurability a processor is very useful. This has the following advantages: it allows the tester/analog design engineer to write the test-code in C/C++, instead of having to design their own system in a hardware description language (HDL). Because of being able to write in C/C++, the test schemes can be more elaborate, and more importantly can be changed after the chip has been taped-out. Additionally, this processor could be used for other purposes, such as decoding Bluetooth packets on-chip if the analog design is something like a wireless transceiver. The processor will be based on the Reduced Instruction Set Computer - Five (RISC-V) Instruction Set Architecture (ISA), because the use of the RISC-V ISA requires no fees. This ISA was introduced by the University of Berkeley in 2016 [4], and since then a lot of open-source implementations of RISC-V core in hardware description languages (HDLs) have been designed. Because of that, this project will determine which one of these open-source RISC-V cores is most suitable for the control system.

A. System requirements

Based on the needs by the ICD group as explained before, this paper presents the work on an on-chip RISC-V control system. For the design of the system the following requirements are used:

- Minimize the required area. The target area is 250 µm by 250 µm (62500 µm²) for the system implemented in the Globalfoundries' 22nm FDSOI technology (22FDX®).
- Minimize the number of bondpads needed for the interface between the control system and the host PC.
- Modular and easily configurable to support small single component to complex multi-component analog and mixed-signal designs.
- Support digital signal processing (DSP) applications.
- Easy to use and integrate for the analog design engineer.

To guide the design process and to verify that the RISC-V control system works according to the requirements defined above, an example application is used. The high level overview of this system can be seen in **Fig. 2**. The DUT in the example system is a voltage-controlled oscillator (VCO). This is an analog oscillator design that can change its oscillating frequency by changing two parameters (α and β). α and β must be calibrated to make the output of the VCO (y_{vco}) oscillate at a specific frequency. Instead of using UTCONTROL to do this calibration, the proposed on-chip RISC-V control system is used. The calibration can be done in two ways, either completely on-chip or using an external spectrum analyzer to measure the frequency of the VCO. These are the two use-cases shown in **Fig. 2**:

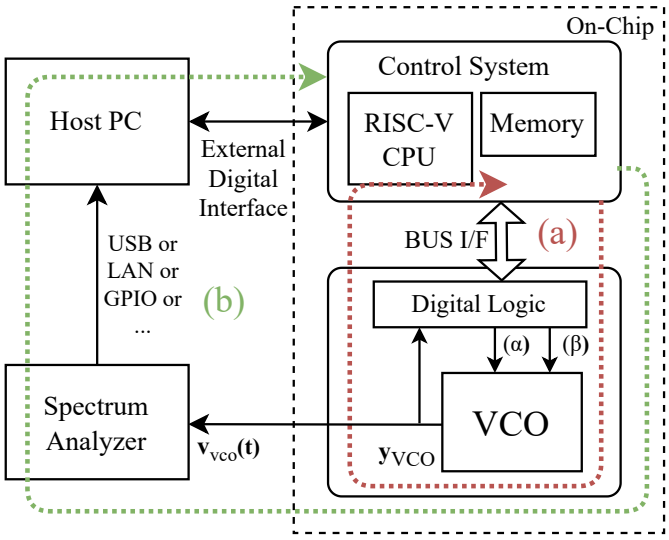

Fig. 2. High level overview of the system used for verification of the RISC-V control system. The on-chip analog component is a voltage-controlled oscillator (VCO), of which the parameters are controlled by the RISC-V control system. Two ways are used to determine the output frequency of the VCO: (a) uses an on-chip counter connected to the VCO, and (b) determines the frequency by using a off-chip spectrum analyzer.

(a) The frequency of the VCO is fully calibrated on-chip. This is accomplished by using mixed-signal feedback, using an added on-chip counter, which increments its value on each oscillation of the VCO. The RISC-V control system can readout this value periodically and determine the frequency of the VCO accordingly. Based on this frequency, the parameters (α and β) are calibrated.

(b) For more complex systems it might not be possible to use a simple counter to measure the frequency, so for that reason an off-chip spectrum analyzer must be used to determine the frequency of the VCO. This means the RISC-V control system must use the external digital interface to communicate with the host PC to determine the frequency. Based on this frequency, the parameters (α and β) are calibrated.

B. Research questions

To have a proper structured approach to the design of the RISC-V control system, a main research question is defined, from which several sub-questions are derived. These research questions will be answered throughout the paper, and analysis will be done to support the findings. Based on the requirements given in the previous section, the main research question is: *Which design of an on-chip RISC-V-based control system is optimized for area and meets the requirements for calibration and control of analog and mixed-signal components?*

In the time budget of the project it is infeasible to completely design a RISC-V processor core from scratch, so an open-source RISC-V core will be used, which leads to the first sub-question: *Which open-source RISC-V core is most suitable for an area-optimized design?*

To be able to inspect the state of the RISC-V-based control system, some kind of interface to communicate with the outside world is necessary. This is what the next sub-question is about: *Which interface between the RISC-V control system and the host PC minimizes the number of bondpads and enables bidirectional communication?*

A RISC-V core by itself is not very useful, and needs at least some memory to execute instructions and to store its results. Connecting these memories and analog peripherals to the RISC-V core can be achieved using different architectures and bus protocols, with each of them having its own advantages and disadvantages. This leads to the next sub-question: *Which on-chip bus-protocol and memory architecture are most suitable for the RISC-V control system, such that it is easy for the analog design engineer to interact with the system?*

C. Main contributions

This work mainly focuses on the practical aspect of the design of the RISC-V control system, since the goal of the project is to design and implement a control system that the analog and mixed-signal designers of the ICD group can actually use. To summarize, the main contributions of this work are listed below:

- A reusable and modular design of the RISC-V control system, which is easily extendable and provides an easy-to-use interface for the analog and mixed-signal design engineers.
- A complete verification setup to simulate the RISC-V control system, including simulation of a debug probe using the OpenOCD debugger.
- A fully placed and routed RISC-V control system using a custom designed digital EDA flow for Globalfoundries' 22nm FDSOI technology, including clock tree synthesis, and timing verification.

D. Paper structure

First, **Section II** will show some of related work available in research and open-source communities. **Section III** then focuses on the RISC-V core of the con-

trol system, and contains a study of different open-source RISC-V cores. It also explains the various design decisions made for the different configurations of the RISC-V core. Next, **Section V** will explain how the RISC-V core is integrated with the memories and the rest of the control system, and show how the different components fit together to form a modular and reusable control system. In addition to that, **Section VI** explains how the complete RISC-V control system is tested and validated, and finally **Section VII** shows the results of the implemented design on Globalfoundries' 22FDX® platform.

II. Related work

This section presents some existing RISC-V control systems and the the possibility of reusing (parts) of these designs is discussed. Two different type of systems are looked into, systems designed by research groups, and open-source implementations of RISC-V control systems.

The PULP (Parallel Ultra-Low-Power) team, which is a collaboration between the ETH Zürich and University of Bologna, has contributed a lot of research to the different RISC-V cores and systems. Their focus is mainly on energy efficient multicore IoT devices [5], however there have also been efforts from the PULP team to build smaller systems using their RISC-V cores. Once such example is the PULPissimo system, which consists of a single RISC-V core with a complex memory system. The effective area reported in [6] is 1.22 mm^2, which uses the same technology as the work in this paper. This area is too big to be considered for this work, as the target of this design is to fit in 250 µm by 250 µm, so it is not possible to reuse the complete system. Another smaller RISC-V system from the PULP team is the PULPino system (2016) [7]. This system uses a custom debug protocol for its communication, because at the time when this system was designed, the *RISC-V Debug Specification* [8] was not standardized yet. New systems benefit from using this standard, since it means that debug hardware outside the chip does not need to be redesigned.

Recently, Google started investing in open-source silicon, where there is now the OpenMPW (Open Multi Project Wafer) project that allows for anybody to design their own chip, as long as the design is completely open-source [9]. To do this, Skywater released their 130nm PDK to the public, and the Efabless company provides the tooling to make this happen. For the submission it is required to use the provided design template, which is called the *Caravel* design, that also contains a small RISC-V management core [10]. This design however, is very specific to this project, and contains all kind of functionality such as a flash controller and logic analyzer that are not needed for the work in this paper. Additionally, there are open-source RISC-V designs that make use of the OpenMPW to submit their projects, one of them is the RISCDUINO project [11]. Similar to the other options, this project is also pretty sophisticated and includes lots of functionality to the system. For the design in this paper, it is important to start with a small area-optimized design, and build on top of that. So, for that reason it is not possible to reuse these projects.

III. RISC-V Core

In this section, the design decisions related to the RISC-V core in the control system are discussed. It starts by looking at the different requirements needed for the RISC-V core. Based on these requirements, four different RISC-V cores are selected, and different aspects of these cores are compared to each other. From this comparison, the most suitable RISC-V core for the control system is selected. In the last subsection, the different configuration options of this core are discussed.

A. Requirements

The RISC-V specification is divided into two different volumes, *Volume I: Unprivileged ISA* [12], and *Volume II: Privileged Architecture* [13]. *Volume I* specifies the encoding of the different instruction sets and its extensions, and *Volume II* covers the different privilege modes that a RISC-V core can implement. First, the requirements based on the ISA specification (*Volume I*) are discussed.

The RISC-V ISA is built as a modular design, where one base integer instruction set is combined with one or more extensions. The base set provides the minimum number of instructions needed to implement a processor, such as load/store, branching and arithmetic instructions. **Table I** shows an overview of the different base instruction sets and the available extensions. One of the requirements for the control system is to use as little area as possible. For this reason, the RV32E base variant is especially useful, since it reduces the number of registers from 32 to 16. However, at this point in time, the specification for RV32E is not yet ratified. Although it is not very likely to change much, to make sure the control system is future-proof, the decision is made to use the RV32I variant as the base integer instruction set. In addition to the base instruction set, extensions can be

added to provide additional functionality. One extension which in particular is useful for an area-optimized device is the extension for compressed instructions (C). This extension adds the ability to encode common instructions as 16 bits, instead of 32 bits. According to [14], on average, 60% of the RISC-V instructions in an embedded program can be replaced with compressed instructions, resulting in a 30% static code-size reduction. This significantly reduces the size needed for the instruction memory, thus reducing the area of the control system. Additionally, one of the requirements is to support digital signal processing (DSP) applications. According to [15], the extension for integer multiplication and division (M) improves the performance by 1.5-1.7x, and also reduces the code-size by 3%-6%, on a test-suite specifically designed for embedded systems. Based on this claim, and the fact the DSP algorithms use a lot of multiply operations, the RISC-V core should also support the M-extension to have the benefits of the improved performance, despite the fact that it will increase the area of the core. **Section III-C** contains some analysis on this trade-off. To summarize, the RISC-V core used in the control system should support the RV32IMC specifications.

As previously mentioned, the second part of the RISC-V specification is *Volume II: Privileged Architecture*. This part of the specification covers the 3 different privilege modes that a RISC-V core can implement, the machine-mode (M), supervisor-mode (S) and user-mode (U). These different modes provide different memory protections and different views to the underlying hardware. Every RISC-V core must implement the M-mode, which provides unrestricted access to all the hardware. The U-mode can be added to provide memory protection, which allows the system to be protected from (malicious) application code. In addition to that, the S-mode implements the typical features needed to support a full-fledged operating system. For the RISC-V core in the control system, the application code can always be trusted, and there is no need for operating-system-like functionality, so the RISC-V core should only implement the machine-mode (M). The machine-mode part of the specification already specifies support for at least a single interrupt line, so there is no need to define an extra requirement for interrupt support. However, the system should support more than one peripheral, so it would be an additional benefit if the RISC-V core included support for handling more than one interrupt line. This way, there is no need to design custom logic to merge the multiple interrupt lines, however it is not a strict requirement to have this.

Another requirement is the ability to inspect the state of the RISC-V core. While during simulation all possible states of the core can be inspected, when the design is taped-out this is no longer possible. Because of that, is it important for the RISC-V core to include a debug system. The designers of the RISC-V architecture were aware of this, and specifically designed the *RISC-V External Debug Support* specification for this [8]. The specification defines a protocol which allows an external debugger to i.e. inspect the state, set breakpoints, or access memory and peripherals while the core is held in a debug state. The physical interface used to transfer this protocol is not specified, and it is up to the designer to select the appropriate interface. **Section IV** will go into more detail about the physical interface. So, for the selection of the RISC-V core it is important that the core supports the *RISC-V External Debug Support* specification.

Additionally, is it important for the RISV-V core to be silicon proven. That ensures that the RTL is designed

TABLE I
RISC-V BASE INSTRUCTION SET ARCHITECTURE AND EXTENSIONS [12]

Name	Description	Version	Status	Count
	Base			
RV32I	Base Integer Instruction Set, 32-bit	2.1	Ratified	40
RV32E	Base Integer Instruction Set (embedded), 32-bit, 16 registers	1.9	Draft	40
RV64I	Base Integer Instruction Set, 64-bit	2.1	Ratified	15
	Extension			
M	Standard Extension for Integer Multiplication and Division	2.0	Ratified	8
A	Standard Extension for Atomic Instructions	2.1	Ratified	11
F	Standard Extension for Single-Precision Floating-Point	2.2	Ratified	26
Zicsr	Control and Status Register (CSR)	2.0	Ratified	6
C	Standard Extension for Compressed Instructions	2.0	Ratified	40
B	Standard Extension for Bit Manipulation	0.0	Draft	43
V	Standard Extension for Vector Operations	0.7	Draft	187

with ASICs in mind, such that, for example, clock and reset synchronizers are added to the design. Finally, it is important that the RISC-V core is extensively verified by the authors, such that there will be confidence that the core also correctly handles all the corner cases of the RISC-V ISA. This is checked by consulting the documentation of the core. To summarize, the RISC-V core should:

- support at least the RV32IMC specification;
- support only the machine-mode operation;
- support the *RISC-V External Debug* specification;
- be silicon proven; and
- be extensively verified.

B. Comparison of different RISC-V cores

To select the most suitable RISC-V core for the control system, an extensive search for open-source RISC-V cores is performed. The main source is the list provided by the RISC-V Foundation itself. They maintain a list of both open-source and industrial RISC-V cores on their website, which contains 182 number of different cores [20]. Additionally, the code sharing platform Github is also used as a source, by searching for cores on their search engine [21]. Work-in-progress cores are not added to this list. To reduce this list of RISC-V cores, the requirements listed in the previous section are used. RISC-V cores that do not meet the requirements are removed. This removes a lot of cores, as they do not implement the debug features, or they support more privileges modes, which removes all the bigger operating-system-like cores. From this reduction, only four different RISC-V cores are left that meet all the requirements.

Two of these open-source cores are designed by research groups, and the other two are designed by companies. The two cores designed by research groups are the Ibex [16] and the CV32E40P [17] core. Both of these cores originate from the PULP research team, but are now further developed and maintained by the non-profit parties lowRISC and the OpenHW Group respectively. Both of these cores require the open-source *PULP RISC-V Debug Module* [22] for a functional debug system, so the area numbers reported include this module. The other two cores in the comparison are the SCR1 core by Syntacore [18] and the SweRV EL2 core by Western Digital [19]. Both of these companies are part of the founding members of the RISC-V Foundation, so they are well-established in the area. To select the most suitable core for the control system, several aspects of the cores are compared to each other. **Table II** shows the comparison summarized in a table. The RTL quality is determined by manually scanning the source code and comparing the code-style used. Similarly, the quality of the documentation is also determined by comparing them to each other. For the area comparison, each core has been synthesized for the Globalfoundries' 22nm FDSOI technology with a 8-track standard cell library characterized at 0.8V using a timing constraint of 100MHz. Each core has several configuration options to tweak the performance and area requirements, but for the comparison each of the cores is configured for the RV32IMC ISA specification as defined by the requirements.

Based on the comparison shown in **Table II**, the SCR1 core is most suitable for the control system. First, the documentation and RTL quality of this core are very good. The documentation covers all the aspects of the core, and includes various diagrams that are very useful. The RTL code is also very straightforward to understand, and includes a lot of comments to help the reader understand the code better. The SweRV EL2 core has similar RTL and documentation quality, however the area is to big compared to the other cores. As an additional benefit, the SCR1 core has supports for the open-standard ARM Advanced Microcontroller Bus Architecture (AMBA). This makes the integration of the core more straightforward, compared to the OBI bus interface of the Ibex and CV32E40P cores, because a lot of open-source IP also supports the AMBA interfaces. Note that, the Ibex core is 6% smaller than the SCR1 core, however the bigger area does not out-weight the other aspects of the core.

C. Syntacore SCR1

In this section, the structure and various subsystems of the SCR1 core are explained. Additionally, the area of the core is analyzed, and from this the final configuration of the core is determined.

Fig. 3 shows a diagram of the different components in the SCR1 core. The core is divided into two major parts, the pipeline and the debug part. The pipeline is responsible for the actual execution of the RISC-V instructions, and consists of 4 main components: the instruction fetch unit (IFU), instruction decoder unit (IDU), the execution unit (EXU) and the load-store unit (LSU). This split of components is to allow the core to break-up the execution of an instruction into multiple steps, which allows the core to process multiple instructions simultaneously. This concept is called pipelining, and the number of pipeline stages of the SCR1 core is configurable from 2 till 4

TABLE II
RISC-V CORES COMPARISON

RISC-V Core	Ibex [16]	CV32E40P [17]	SCR1 [18]	SweRV EL2 [19]
Designed by	lowRISC	OpenHW Group	Syntacore	Western Digital
Pipeline stages	2	4	2-4	4
RISC-V ISA support	RV32I/E[MB]C	RV32IMC[F]	RV32I/E[MC]	RV32IMC+Zbb+Zbs
Memory interface	32-bit OBI	32-bit OBI	32-bit AXI4/AHB-Lite	64-bit AXI4/AHB-Lite
CoreMark (per MHz)	2.47	3.19	2.43	3.6
Documentation quality	–	+/-	++	+++
RTL quality	+/-	+/-	+++	+++
Area in GF22 (μm^2)	11055	16422	11791	33483
f_{max} (MHz)	188	164	152	116

stages. Additionally, the pipeline part of the SCR1 core also contains the multi port register file (MPRF) and the control status registers (CSR) to manipulate the state of the core by executing specific CSR instructions. The other part of the SCR1 core is the debug part, which is used to inspect the state of the pipeline from outside using a JTAG interface. The SCR1 core implements all the debug functionally according to the *RISC-V External Debug Support* specification, so for the exact operation the specification can be consulted [8]. But most importantly, by halting the SCR1 core, the hart debug unit (HDU) can use the LSU to access memory. Likewise, the trigger debug unit (TDU) is responsible for providing breakpoint support, such that the core will halt when a certain address is executed. The remaining components of the SCR1 core are the integrated programmable interrupt controller (IPIC) and the system control unit (SCU). The IPIC provides the core with 16 different interrupt lines, and can be programmed with an arbitrary mask to disable interrupts. Finally, the SCU is responsible for the different resets in the core. For example, only the pipeline can be reset while keeping the state of the debug part of the core. The documentation of the SCR1 core [18] provides a lot of additional information about these components.

The SCR1 core can be optimized for area, performance or power, since the core is highly configurable. Three different recommended configurations are provided: the minimum, base and max configuration. The difference is mainly in the RISC-V instruction set and the number of pipeline stages that each configuration implements. The minimum variant implements a 2 stage pipeline RV32EC core, the base a 3 stage pipeline RV32IC core and the max configuration a 4 stage pipeline RV32IMC core. Additionally, when the M-extension is added to the core, the number of cycles to execute a multiply instruction can be configured. This adds three additional configurations: base-slowM, which

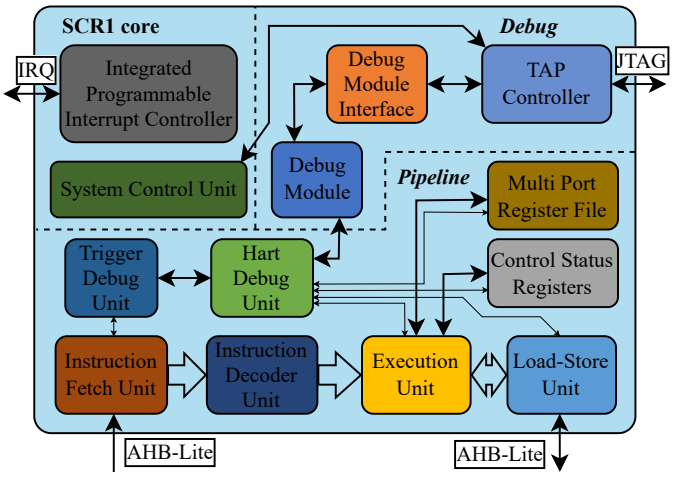

Fig. 3. Overview of SCR1 core. The core is split into two major components, the pipeline and the debug system. Arrows show the different communication channels between the various components. The core uses two different AHB-Lite interfaces to fetch instructions and access memory respectively.

uses a 32 cycle multiplier, base-3M, which uses a 3 cycle multiplier, and base-fastM, which uses a single cycle multiplier. **Fig. 4** shows the area breakdown of all these different configurations. The area previously reported in **Table II** corresponds to the max configuration, as the requirement is to support the RV32IMC ISA variant. However, based on the results shown in **Fig. 4**, this is not the optimal configuration for an area-optimized design, and instead the base-3M configuration is used. This configuration reduces the area by 10%, and provides a good trade-off between area and performance.

The SCR1 core optionally supports a tightly-coupled memory (TCM) inside the SCR1 cluster. The size of the TCM is up to 64kB, and it allows the SCR1 core to fetch instructions or load/store data with single-cycle latency. To support this, it needs a dual-port SRAM, which at this point in time is not available in the ICD group. Furthermore, there is no interface available to access this

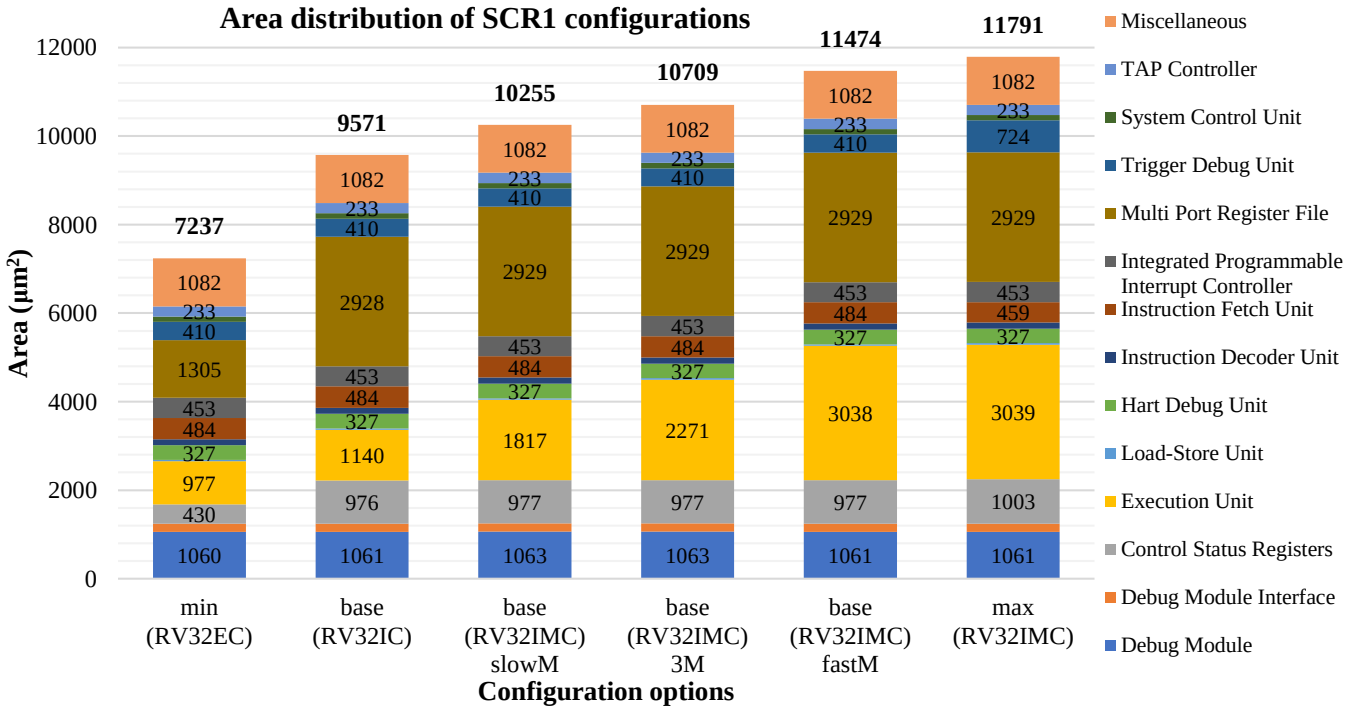

Fig. 4. Area distribution of different configurations of the SCR1 core. The minimum, base and maximum configurations are shown, as well as three modified base variants. The difference is the type of multiplier that is used, *slowM* has a 32 cycle multiplier, *3M* a 3 cycle multiplier, and *fastM* a single cycle multiplier.

memory from outside the core, and when the size of this memory is changed, it would require to redo the physical implementation of the core, which does not result in a modular and reusable system. Because of this, the optional TCM is not included in the final configuration of the core used in the control system. Additionally, the SCR1 cluster provides an instruction and data memory bridge to convert the core specific interface to either the Advanced eXtensible Interface (AXI) 4 or the Advanced High-performance Lite Bus (AHB-Lite). Both are part of the open-standard ARM Advanced Microcontroller Bus Architecture (AMBA) and provide a standard way to connect on-chip components. For an area-optimized device the AHB-Lite bus is more suitable because it uses fewer signals compared to the AXI4 bus, which will result in the various bus components using less area.

IV. External Digital Interface

In this section, the external digital interface used to communicate with the control system is discussed. First, the requirements for this interface are explained, following with an description about the actual interface that is used in the system.

Based on the use-cases explained in the introduction, the external digital interface should:

- allow the host PC to debug the RISC-V core;
- provide the host PC access to the memories to program them;
- provide general bidirectional communication capabilities between the host PC and the control system; and
- minimize the number of bondpads needed.

Allowing the host PC to debug the RISC-V core is important, because there needs to be a way to verify if the code running on the RISC-V core behaves as expected. This is only possible by being able to single-step programs, set breakpoints or show the register state of the core. As explained before in **Section III-A**, the SCR1 core supports the *RISC-V External Debug Support* specification, however the physical interface that must be used is not specified. There are two different options: use a standard interface, or design/use a custom one. The ICD group already designed a custom two wire debug interface called UTwo, [23], so this could be modified to support the RISC-V debug specification. However, the problem with a custom interface is that it also requires a custom hardware/software stack outside the chip to be able to communicate with the device, which complicates the design. Alternatively, by using a standard interface, it is possible to use existing off-the-

shelf debug hardware to debug the RISC-V core. The de facto standard used for any chip with a processor on it, is the IEEE 1149.1 standard developed by the Joint Test Action Group (JTAG), also known as the JTAG interface. This interface requires a minimum of four wires to be able to use it, and the SCR1 core already contains the required logic, such as the TAP controller, to use this interface. However, four bondpads is still quite a lot. The need to use fewer bondpads was also known to the JTAG group, which developed an extension to the standard called IEEE 1149.7, or in short the compact JTAG (cJTAG) interface. Compared to the original standard, the new cJTAG standard is much more complex and supports various ways of interfacing. One of these ways is the mode where the three JTAG signals (TDI, TDO, TMS) are serialized onto a single pin. This way only two bondpads have to be used. Unfortunately, there is not a lot of adoption of this new standard yet, and only a single unfinished open-source IP was found that implements a cJTAG to JTAG bridge [24]. So for now the regular four wire JTAG interface is used to provide debugging capabilities to the system.

In the RISC-V control system, the memories are integrated on-chip, so to be able to program them, the host PC must be able to access these memories from outside the chip. For this reason, the host PC is the master and the control system is the slave of the communication system. Additionally, the interface should also provide general bidirectional communication capabilities between the host PC and the control system, or more specifically the firmware running on the RISC-V core. In this situation, there is only one possible way to achieve bidirectional communication, and that is using a shared component that both the host PC and the RISC-V core can access. The shared component in this system is the memory, because both the host PC and the RISC-V core must be able to access the memory. Communication from the host PC to the control system works by using the host PC to write to a memory location, and the RISC-V core read the same memory location to check if new data arrived. For communication the other way exactly the opposite happens. So, for both the programming of the memories, and the bidirectional communication, the external digital interface should provide a way for the host PC to access the memories. **Fig. 5** shows three possible ways in which this can be done. When the SCR1 core is executing instructions, it is using the LSU to access memory and peripherals. (a) shows that the debug module can take over the LSU to access the memory. In this situation the core must be halted first, because the debug module and the core itself cannot use the

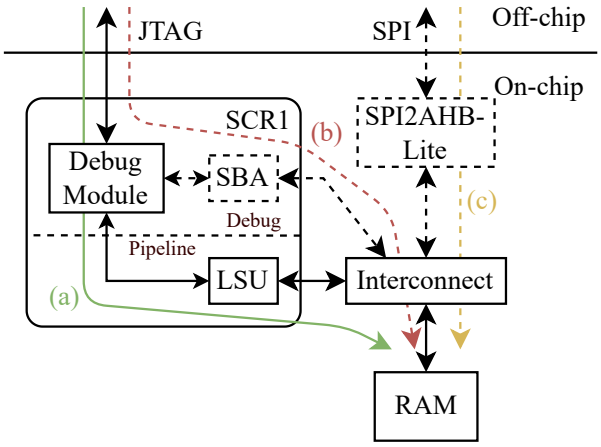

Fig. 5. Three possible ways of accessing on-chip memory. (a) uses the LSU port of the SCR1 core to access the memory. This requires the core to halt while the debug module is in control. (b) access the memory via the optional *System Bus Access* port, and (c) uses an extra SPI to AHB-Lite converter to access the memory.

LSU simultaneously. Situation (b) allows access to the memory via an extra system-bus-access (SBA) port. The SBA is an optional part of the *RISC-V External Debug Support* specification, and allows the debug module to interact with the memories without needing to halt the core. This part of the specification is not implemented for the SCR1 core. The third option (c) shows that a SPI (or any other serial interface) to AHB-Lite bus converter can also be used to provide access to the memories. However, this option requires to design of such a component from scratch.

Option (a) is used for the external digital interface, as it allows to reuse the JTAG interface. If the SPI interface would be used, more bondpads would be required, as the JTAG interface must also be provided to allow the host to have debugging capabilities. One downside of this approach is that it requires the core to halt whenever the host PC wants access to the memories. This could be solved in the future, by implementing the SBA part of the *RISC-V External Debug Support* specification. Alternatively, option (c) could be used if there was no need to inspect the state of the RISC-V core. So that means the external digital interface in the current version of the system uses the four JTAG wires.

Additionally, some more signals are required to be able to have a working system. First of all, the SCR1 core needs a core clock that drives the sequential logic of the core. For simplicity of the design this core clock will be generated off-chip. Besides the core clock signal, there also needs to be a way to set the initial state of the logic, which is why a power-up reset signal is needed. Note

that this signal resets all the digital logic, including the debug state of the core, so if only the SCR1 core needs a reset trigger, to for example re-run a test, the internal reset line can be triggered using JTAG commands. To summarize, the external digital interface for the current design of the system needs 8 bondpads, 4 wires for the JTAG interface, the core clock input, the power-up reset signal, and power and ground to supply the chip. Power and ground may be shared with other digital logic part of the analog and mixed signal design. In the future, this can be reduced to 6 bondpads when a cJTAG bridge is available.

V. System description

The RISC-V core by itself does not provide a lot of functionality yet, and requires additional components, such as memories and peripherals to actually function as a complete control system. The main idea is to have separate components/building blocks that communicate with each other via standard bus interfaces. This way the physical implementation of each component can be done separately, which makes the system modular and reusable.

Fig. 6 shows an overview of the UTRISCY system. The main component is the UTRISCY core component, which wraps the SCR1 core, and provides several standard AHB-Lite bus interfaces. AHB-Lite is used as the SCR1 core uses this interface for its instruction and data busses. The core component provides the memory-mapped input-output (MMIO) AHB-Lite slave interface for analog peripherals to connect to. Via this interface the RISC-V core can communicate with analog peripherals. Additionally, a direct memory access (DMA) AHB-Lite master interface is provided which allows analog peripherals to transfer data directly to the memories. For example, this could be used by an ADC to directly transfer the samples to the memory. Additionally, there are two separate AHB-Lite slave interfaces for the instruction and data memories. **Section V-A** explains why two instead of one. The component responsible for wrapping the static random-access memory (SRAM) is the UTRISCY memory component, which will be explained in more detail in **Section V-B**. Furthermore, is it up to the analog designer to choose whether to interface directly with the UTRISCY core component, or to use the provided UTRISCY AHB-Lite to APB bridge. In an analog and mixed-signal design the analog component could be placed far from the digital logic, and by not tying the core clock to the clock for the analog peripherals, the maximum operating frequency of the core is not limited by the analog peripherals.

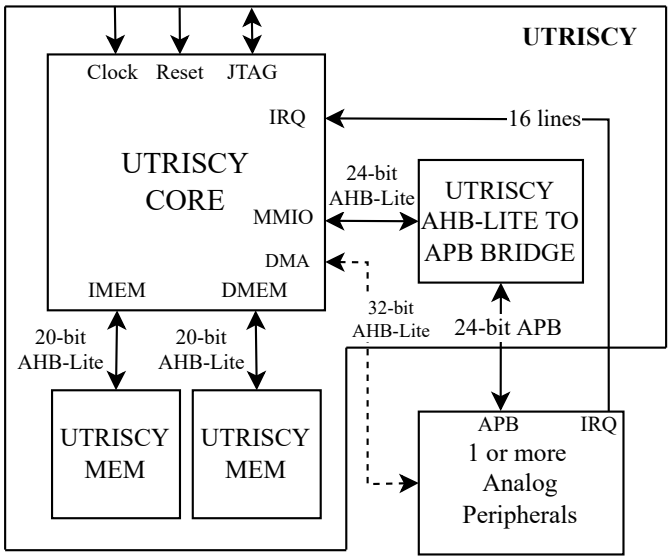

Fig. 6. Overview of UTRISCY system. The system consists of 4 different components, UTRISCY core, UTRISCY memory, UTRISCY AHB to APB bridge, and the UTRISCY analog component. The communication between the different components happens through standard bus interfaces part of ARM Advanced Microcontroller Bus Architecture (AMBA).

The UTRISCY AHB-Lite to APB bridge provides this separation. The design decisions regarding each subcomponent are explained in more detail in the next subsections.

A. UTRISCY core

Fig. 7 shows the architecture of the UTRISCY core subsystem. It consists of two components, the SCR1 core and an AHB-Lite Interconnect. The interconnect is the open-source *AHB-Lite Multi-layer Interconnect Switch* provided by RoaLogic [25]. The IP is fully parameterized, and allows multiple masters to connect to multiple slaves with configurable address and data bus widths. The IP contains logic for round-robin arbitration when multiple masters try to access the same slave simultaneously, and optionally allow the user to specify priority for a given master port. When two or more master ports access different slaves, the IP allows these separate channels to transfer data simultaneously. Additionally, an AHB-Lite slave port has a base address and mask to specify which address range it belongs to. The IP then, for a certain transaction from a master port, routes the transaction to the correct slave. If it is outside any address range the IP will generate a bus error. The interconnect is used in the UTRISCY core because there are several different interfaces that have to be interconnected. First, the SCR1 core has two different AHB-Lite master ports, the IFU and LSU, that need

to access the memories. For the memory access there are two different slave ports, the IMEM and DMEM ports. The next paragraph will explain why two different memory interfaces are supported. Additionally, a third master port is added to support applications where direct access from outside the core to the memories is needed. This is the direct-memory-access (DMA) port. Lastly, to connect the SCR1 core to the analog peripherals, there is an AHB-Lite slave port, which is called the memory-mapped input/output (MMIO) port.

To explain why the design supports two different memory ports, the architecture of the SCR1 core is discussed. As explained in **Section III-C**, the SCR1 core is implemented as a Harvard architecture, and also contains three pipeline stages. This means that the core will fetch instructions via the IFU and access data via the LSU simultaneously. Connecting these two buses to a single-port SRAM will result in bus contention and thus performance degradation when the core uses the LSU, because the IFU is continuously fetching instructions from the memory while it is executing. To quantify how worse the performance will be when only a single memory is used, a fixed-point FIR filter benchmark [26] with 20 coefficients and 10 samples has been run. This resulted in the benchmark running 23.7% faster when two separate memories are used. Additionally, if only a single memory would have been supported, using the DMA port would reduce the performance of the core even more, since there will be three master ports accessing a single memory at the same time. So, to give the designers the most flexibility, the UTRISCY core component supports two separate memories, but if the area-overhead is too big, it is still possible to run the system using a single memory.

B. UTRISCY Memory

The memory compiler used to generate the SRAM for the Globalfoundries' 22nm technology is the *High Density Single-Port SRAM, Ultra-Low-Power Periphery (S1DU) Compiler* provided by Synopsys [27]. The compiler is highly configurable, and allows to select the size and type of the SRAM to be generated. Two types are possible, a pipeline version that includes registers at the output of the SRAM, and a flow-through version which does not have these output registers. Additionally, the compiler has a configuration option to select how many SRAM cells per bitline are generated. For this system, the smallest variant of the C256 density options is used, which has a memory size of 4160 bytes. To convert the AHB-Lite interface to the single-port SRAM, the open-source *AHB-Lite Memory* IP provided by RoaLogic [28] is used. This IP is modified to be compatible with the SRAM generated by the Globalfoundries' 22nm memory compiler, and is parameterizable for the size and type of the specific SRAM that is used. Additionally, the IP is modified to generate a bus error when a transaction requests an address outside the range of the size of the SRAM. The bus error causes the RISC-V core to execute an exception handler, such that the firmware is aware that the core tried to execute or access memory outside the SRAM. Previously, it would silently truncate the address bits of the transaction to the size of the memory, and return the data at that location.

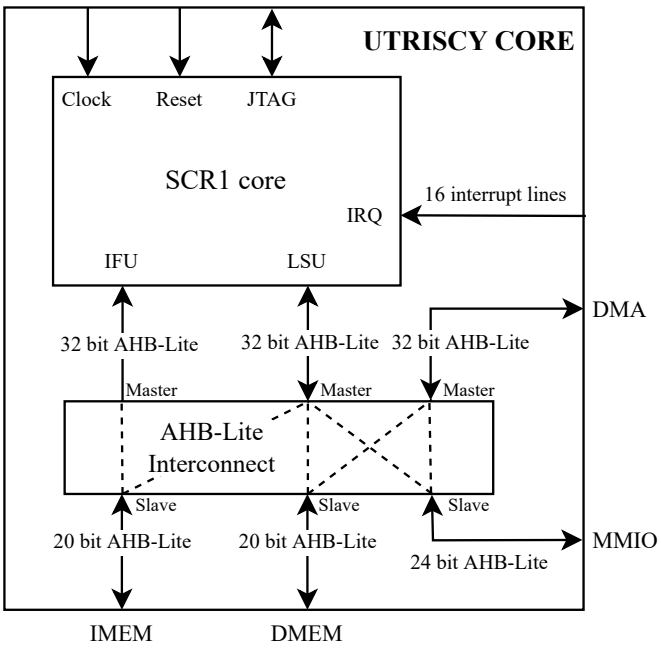

Fig. 7. Overview of the UTRISCY core component. The component wraps the SCR1 core and also contains an AHB-Lite Interconnect to connect the IFU and LSU of the SCR1 core to the IMEM, DMEM, MMIO and DMA port.

C. UTRISCY AHB-Lite to APB bridge

This component is responsible for converting the AHB-Lite bus into the simpler Advanced Peripheral Bus (APB). By using this component, the AHB-Lite clock, which is the core clock, is split from the clock that is used for the analog peripherals. This has the advantage that the clock to the analog peripherals can run at a lower clock speed, compared to the operating frequency of UTRISCY core. Additionally, the APB bus is also simpler compared to the AHB bus, which means that the analog peripherals are easier to implement for the analog designer. **Fig. 8** shows the architecture of the UTRISCY AHB-Lite to APB bridge. The component consists of

three sub-components, an AHB-Lite interconnect, an AHB-Lite clock divider and the actual AHB-Lite to APB bridge. The AHB-Lite interconnect is the same IP as the interconnect used in the UTRISCY core component, and is used as a simple simple bus divider to split the incoming AHB-Lite bus into 2 separate AHB-Lite buses. One of the buses goes to the AHB-Lite clock divider and the other to the actual bridge. The IP for the AHB-Lite to APB bridge is the open-source *AHB-Lite APB4 Bridge* by RoaLogic [29]. It supports fully asynchronous clocks on both sides of the bridge, so it includes logic for the proper clock domain crossing.

To generate the slower APB clock from the AHB-Lite clock, the AHB-Lite clock divider component is used. **Fig. 8** also shows the architecture of the AHB-Lite clock divider in more detail. This component consists of a clock divider and a clock mux to select the appropriate clock. The clock divider is a 7 bit counter incrementing its value every core clock cycle. This results in a 2^n clock divider. These 7 different clocks are muxed together using special clock multiplexer cells from the Globalfoundries' 22nm standard cell library. A clock gate is added to make sure that the clock can be disabled while switching to a different clock. Configuring the clock multiplexer and clock gate is done by writing to the control register, which is connected to the AHB-Lite bus. The highest bit of the control register controls the clock gate and the lowest 3 bits select which clock is used. The reset value of the control register has the highest bit set to one, and the rest of the bits are zero. This means that after reset the clock gate is enabled, and the selected clock is not divided. To make sure there are no glitches while switching the clock it is important to first disable the clock gate, then switch the clock, then enable the clock gate again. This is not enforced in the hardware, so the software is responsible for correctly doing this.

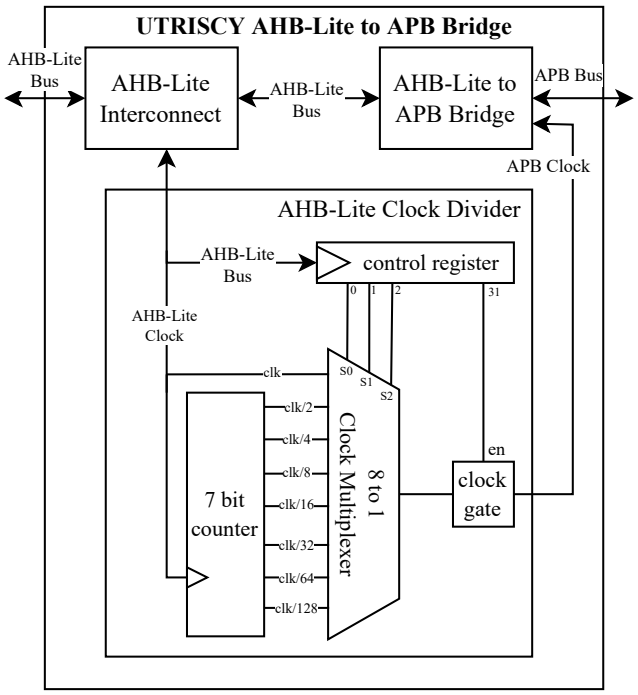

Fig. 8. AHB-Lite to APB bridge architecture for the RISC-V control system. The component consists of an AHB-Lite interconnect, which splits the AHB-Lite bus into two buses. One connects to the AHB-Lite clock divider and the other to the APB bridge. The AHB-Lite clock divider divides the core clock by a configurable number of cycles. The clock output of the AHB-Lite clock divider is used as a clock for the APB bus. This way the APB clock can be changed at run-time.

D. Analog peripherals

The idea is to have a separate analog peripheral per function. That way the peripheral can be designed once, and reused by other designs in the ICD group. For example, a trimming peripheral, that is responsible for trimming certain currents on chip could be designed. For certain designs it might be better to have a single peripheral that is completely custom designed for the needs of the analog and mixed-signal design. For the example design in this paper, a specific VCO peripheral is designed to show the functionality of the complete system. **Fig. 9** shows a detailed diagram of the VCO peripheral. The left side shows how the analog designer could design the digital logic that is needed to interface with the APB bus. The design contains four registers that can be read and written by the RISC-V core via the APB bus. The capacitance register contains the value for the capacitance of a capacitor in the VCO design, the current register contains the value for the current source, the counter register can only be read via the APB interface, and contains the value of the counter in the VCO. The last register is the interrupt register which can be used to trigger an interrupt on the IRQ line, by comparing the interrupt value to the value in the counter register. The output of the registers are fed to the analog VCO design. Changing these values will change the oscillation frequency of the VCO. Using the APB interface the VCO can be calibrated.

VI. Verification

This section explains how the design of the RISC-V control system is verified. First, the testbench is explained, including the use of the OpenOCD debugger. The testbench is simulated using the Questasim simulator from Siemens EDA (previously Mentor Graphics). It is used for the initial RTL simulation, but also for the post-synthesis and post-layout simulations. This section also

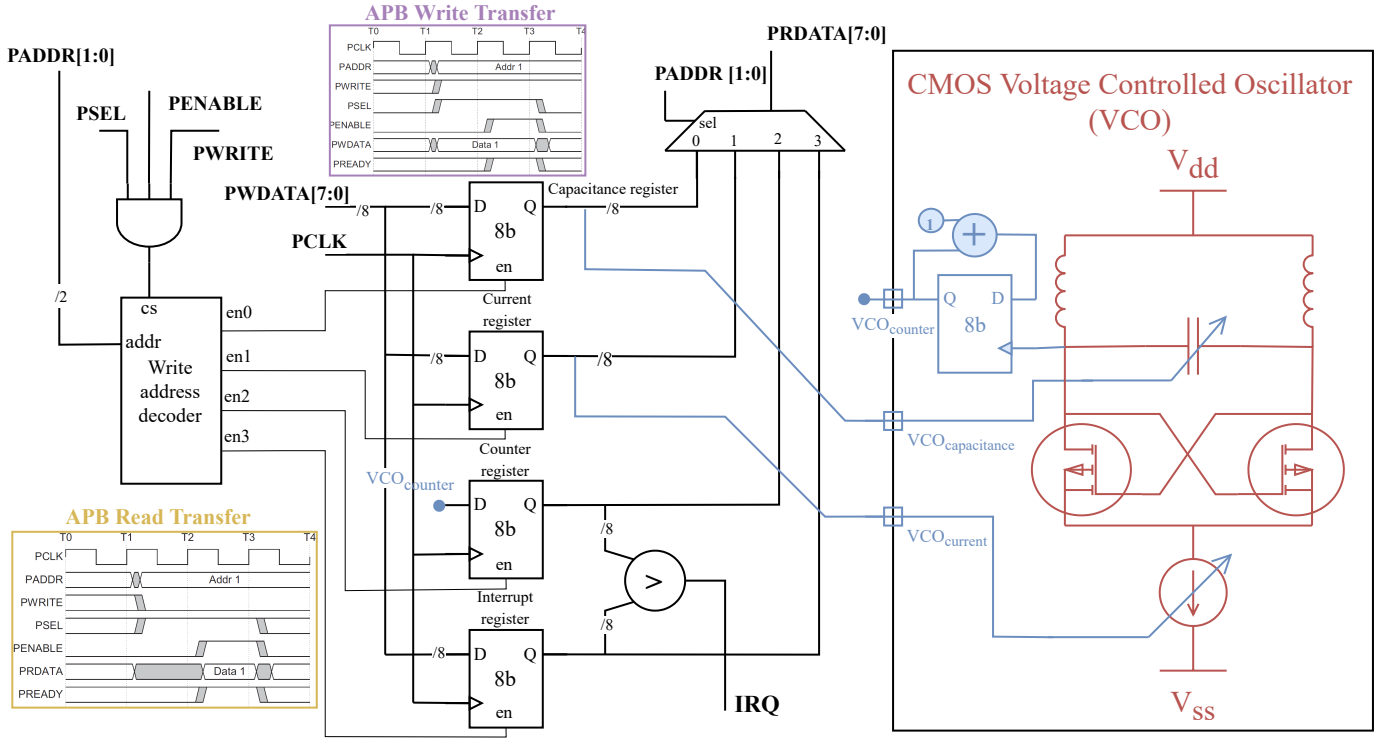

Fig. 9. Detailed overview of the UTRISCY VCO analog peripheral. The design consists of two parts: the left side shows the digital logic needed to interface with the APB interface. It contains four register that can be read-out using the APB bus. The right side shows the analog VCO design. Two parameters of the analog VCO can be changed, the capacitance value and the current delivered by the current source. Changing these values will change the oscillating frequency of the VCO. The output of the VCO is connected to the clock input of a flip-flop, thus incrementing its value on each oscillation of the VCO. This value is fed to the counter register. The waveforms show how the APB protocol works [30].

explains the functionality of the verification scenarios that are designed for this system.

The verification setup consists of a SystemVerilog testbench in which multiple components are instantiated, **Fig. 10** shows an overview of the verification setup. The VCO Control Chip is an example module that consists of the UTRISCY component, in the configuration as shown in **Fig. 6**, which include two separate memories and the UTRISCY AHB to APB bridge. From the use-case defined in the introduction of this paper, the VCO peripheral is added, which is responsible for controlling the simulated VCO. The output of the simulated VCO is routed to the VCO peripheral, where a counter is used to measure the frequency. This corresponds to use-case *a*. The output of the VCO is also routed to the host PC to simulate use-case *b*, where a spectrum analyzer is used to measure the VCOs frequency. This verification setup contains a simple simulated spectrum analyzer that measures the frequency of the VCO and sends it to the OpenOCD debugger via the TCL RPC interface [31]. Next, to be able to run the core, a simulated clock and reset generator is instantiated in the testbench. This will make sure the reset line is properly toggled, and will provide the clock for the UTRISCY component. The next paragraph will explain how the simulated JTAG master is able to control the UTRISCY core component.

To verify the system, one possibility is to design a JTAG master in SystemVerilog that implements the *RISC-V External Debug Support* specification, and manually send commands this way. However, there is no such open-source IP available, and it also does not represent a typical verification setup, because in real-life some physical debug probe will be responsible for controlling the device. **Fig. 10** shows the solution that is used in this system. The verification setup implements a simulated JTAG Master that uses the SystemVerilog Direct Programming Interface (DPI) to talk to a C file. The DPI interface allows SystemVerilog to call into C code that is attached to the simulation, and in that way the C code is able to manipulate the JTAG signals in the SystemVerilog simulation. The sim_jtag.c file then sets up a server which listens for connections using the Remote Bitbang protocol [32]. This is a protocol defined by the open-source debugger OpenOCD, and allows the

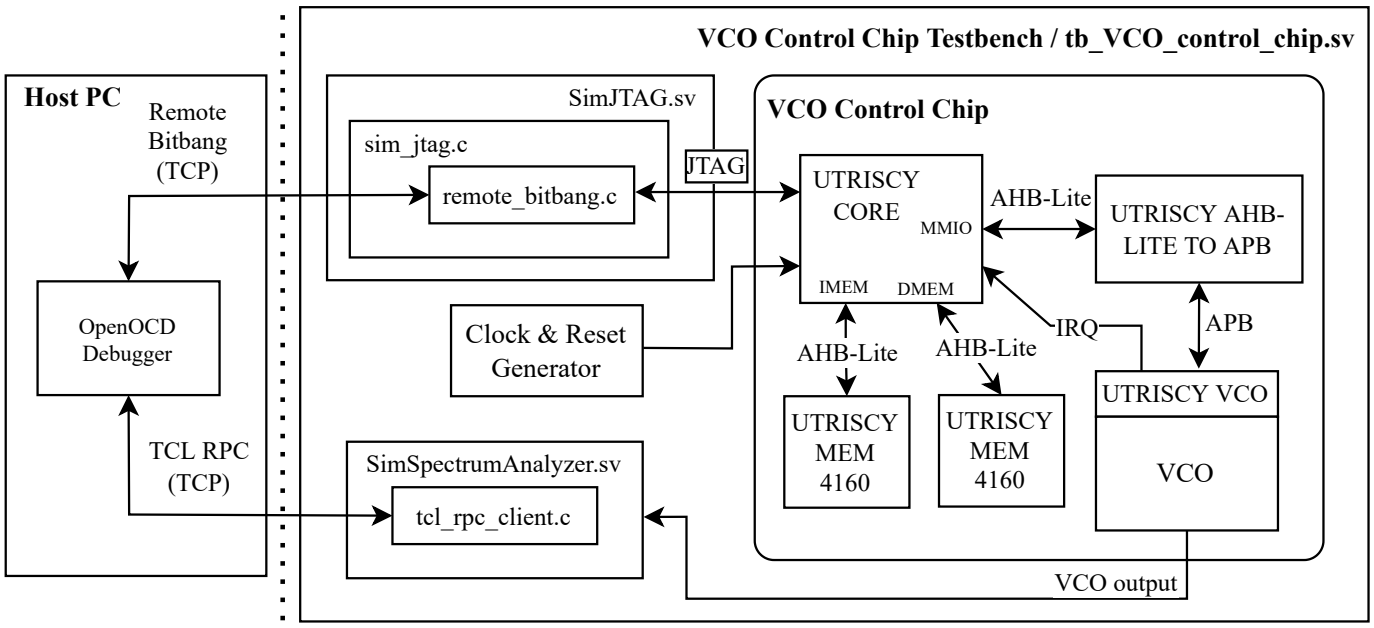

Fig. 10. Overview of the verification setup of the UTRISCY system. It is based on the use-cases explained in the introduction of the paper, and allows the complete system to be verified by using a simulated debug probe using the OpenOCD software running on the host PC. To simulate the spectrum analyzer for the second use-case, a simple design in SystemVerilog has been made. This sends the measured frequency to the OpenOCD debugger via the TCL RPC connection, which OpenOCD then sends back to the control system. A simple behaviour model of a VCO is used to simulate the behaviour of a VCO.

debugger to connect to a JTAG probe over TCP. The OpenOCD debugger implements the *RISC-V External Debug Support* specification and knows how to talk to the RISC-V Core. In this way the OpenOCD debugger is able to control the simulated RISC-V Control chip, representing a real-world verification-setup. This verification setup allows the design to be tested at all stages of the design process. First of all, it is used to test the initial behavioural RTL simulation. But after synthesis and layout the same test is used to make sure the design still behaves the same. Once the chip is taped-out the exact same setup can be used with a physical Segger J-Link Debug probe, instead of the OpenOCD debugger.

A script is designed that automatically runs all the required programs for the verification setup. This starts with the compilation of the test program, for which a standard RISC-V GCC toolchain is used [33]. The verification setup downloads the toolchain from SiFive, however any other RISC-V GCC toolchain should work the same. A Makefile is added to the verification setup that runs the necessary steps for compilation. After that Questasim is launched, and the OpenOCD debugger is attached to the running simulation. The debugger is then instructed to upload a test program to the memory of the UTRISCY component, to reset the core, and to start the execution of the test program.

The test program consists of three different parts. The first part is a general test that verifies for example that the UTRISCY memory component correctly fires a bus error when a transaction outside its memory size is done. The two other parts of the test program are the two different verification scenarios that calibrate the VCO. The first scenario calibrates the VCO completely on-chip, and the second scenarios uses the simulated spectrum analyzer to know the frequency of the VCO and also calibrate the VCO that way. The VCO is calibrate using a simple control loop, where the target frequency is programmed, and the RISC-V control system executes a proportional integral (PI) control loop to change the parameters of the VCO. **Fig. 11** shows the different waveforms that make up the test program, including the output of the OpenOCD debugger on the host PC.

VII. Results

This section shows the results of the UTRISCY system implemented in Globalfoundries' 22nm FDSOI technology. It will start by explaining the tools that have been used to achieve this, following with the the synthesis results. For the initial physical implementation of the design, the VCO control chip as shown in **Fig. 10** has been implemented. There was no time left to do the physical implementation of each component separately. The results, such as the floorplan and the final area usage are discussed at the end.

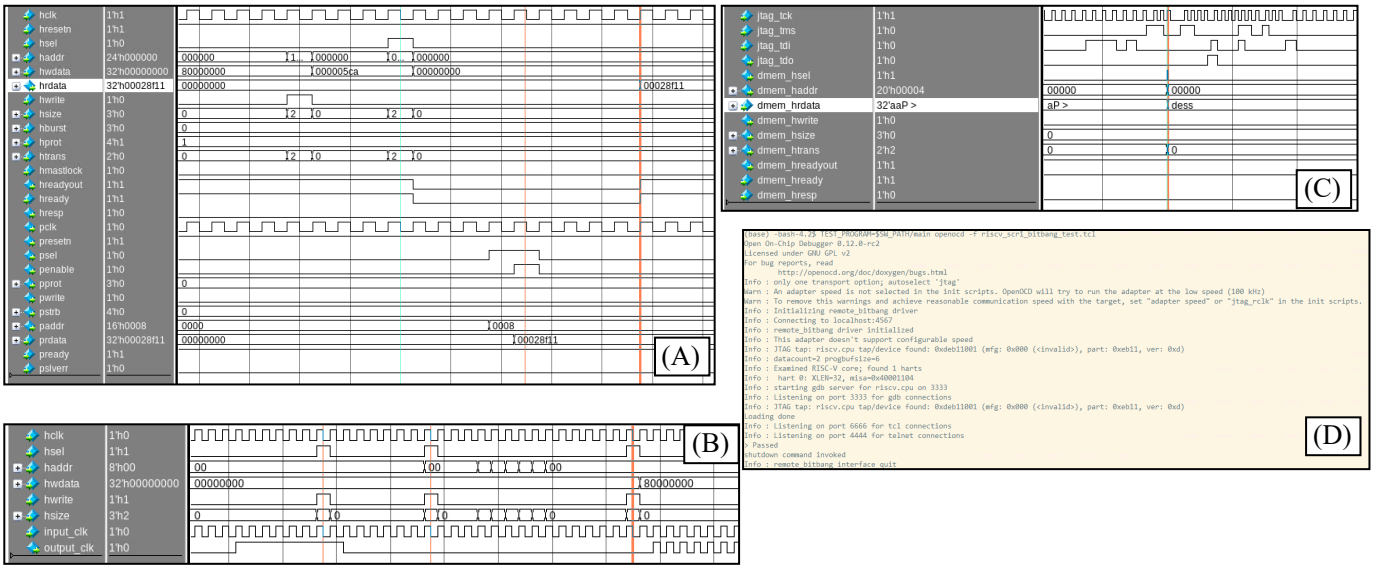

Fig. 11. Various waveforms of the verification setup. (a) shows the simulation of the AHB-Lite to APB bridge. (b) shows the AHB-Lite clock divider, and it can be seen that the frequency of the output clock is changed. (c) shows the JTAG probe reading memory to achieve the bidirectional communication. (d) shows a screenshot of the OpenOCD communication.

TABLE III
SYNTHESIS AREA USAGE PER COMPONENT

Component	Subcomponent	Non-flattened area (µm²)	Flattened area (µm²)	Difference (%)
UTRISCY core		12518	10371	17.1
	SCR1 core	10702		
	AHB-Lite Interconnect	1814		
UTRISCY bridge		1179	882	25.2
	AHB-Lite to APB bridge	647		
	AHB-Lite Interconnect	350		
	AHB-Lite clock divider	181		
UTRISCY mem 4160 F		7981	7880	1.3
	IN22FDX S1DU 4160 SRAM	7794	7794	0.0
UTRISCY VCO		487	464	4.6
VCO control chip		30144	27478	8.8
VCO control chip excluding memories		14183	11717	17.4

The digital EDA flow that is designed for the system, uses the Synopsys DC compiler for the synthesis of the RTL to the standard cells of the Globalfoundries' 22nm FDSOI technology. For the physical design of the chip the tools from Cadence are used. Specifically, Innovus is used for the placement and routing, including clock-tree synthesis.

A. Synthesis

To optimize the design for the smallest possible area, the 8-track standard cells are used. Compared to, for example, 12-track, the 8-track standard cells use less area, but that also makes them slower. Additionally, a relaxed timing constraint of 100MHz for the core clock is used. This constraint is a good trade-off between area and performance, as it does not increase the area of the design. The constraint for the JTAG clock is 12 times slower than the core clock, because that is the requirement for the SCR1 core. **Table III** shows the area usage of the different components of the UTRISCY control system. The area for both the non-flattened and flattened components are shown. This shows that flattening the design saves 17% for the UTRISCY core, and 25% for the UTRISCY bridge. The savings for UTRISCY memory components are not that big, as most of the area of the UTRISCY memory is the SRAM macro. In

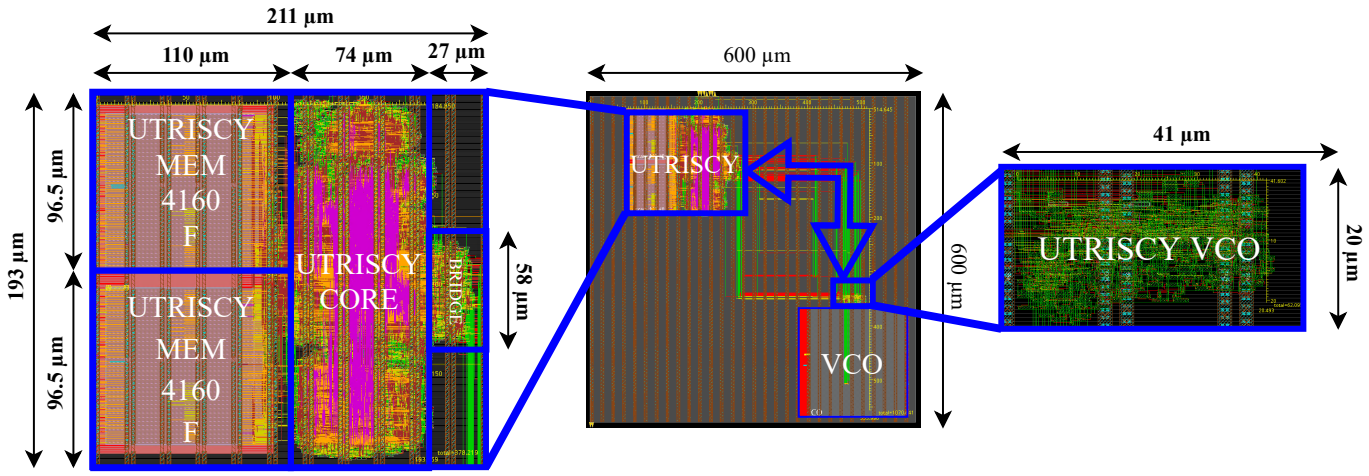

Fig. 12. Overview of the layout of the VCO control chip. The complete layout is shown in the middle, and the UTRISCY part is shown in more detail.

this design, two 4160 byte memories are used, which contribute to 47% of the total area. As can be seen in the table, the total synthesis area of the flattened VCO control chip is 27478 µm^2.

B. Placement-and-routing

The flattened VCO control chip netlist is used for the final implementation of the design on Globalfoundries' 22nm FDSOI technology. This netlist does have separate modules for each component, such that they can also be placed as separate components. To start the design of the VCO control chip, a floorplan of 600 µm by 600 µm was used. This was done to allow the UTRISCY VCO component to be placed 'far' away from the UTRISCY core component. For the VCO a simple block macro was used to be able to place it in the design. For each component the utilization was set to 70%, as the placer needs some margin to be able to place all the standard cells. **Fig. 12** shows the final layout of the VCO control chip. The UTRISCY core, two times UTRISCY mem 4160 and the UTRISCY bridge have been placed together in the left corner, and occupy an area of 211 µm by 193 µm (40723 µm^2). The UTRISCY core component occupies 74 µm by 193 µm (14282 µm^2). This is 37% bigger than the area as reported by the synthesis tool. The UTRISCY bridge needs 58 µm by 27 µm (1566 µm^2), and the UTRISCY VCO component has an area of 41 µm by 20 µm. The UTRISCY memory (4160 bytes) needs 10615 µm^2. For the implementation of the design three different process corners have been used: min (0.72V, -40C), typical (0.8V, 25C), and max (0.88V, 125C). The setup timing slack at the typical corner is 3594 ps, which means the design can run at 156MHz at 0.8V. To solve the hold violations the max corner has been used, so at all these three corners there are no hold violations in the circuit. The area of an IO cell for GF22 is 2760 µm^2, and this design requires 6 bondpads, excluding the power and ground as they can be shared with other digital logic, so that adds another 16560 µm^2 to the design, so the total required area for the design is 57283 µm^2.

VIII. Discussion

The results show that the currently implemented design requires 57283 µm^2 including bondpads for an implementation in Globalfoundries' 22nm FDSOI technology. This is just within the 62500 µm^2 and is consider to be quite big after a discussion with analog design engineers at the ICD group during a chiptalk.

During this design several trade-offs were made that actually increased the area of the design. For example, the requirement of being able to debug the software running on the RISC-V core, made it such that the JTAG interface had to be used, instead of using something like SPI, UART or I2C for the bidirectional communication, and no cJTAG to JTAG bridge was available at the time. This means the total number of bondpads could have been four, two for the serial interface, and a clock and reset, which would have saved 5520 µm^2. Additionally, the implemented design consists of two 4160 bytes memories, where one small 2KB memory could have been used instead, but because of the performance trade-off the two memories were included. One extra point where area could have been saved was the configuration of the SCR1 core, as the configuration used in this design was not the smallest, again because of the requirement for digital signal processing for example. To give an idea

what the area would have been if the performance trade-offs were not made, the following configuration of the system is used: smallest SCR1 configuration (RV32EC), one 2KB memory, including the JTAG interface. The JTAG interface is included in this configuration, as it requires more work to use a different interface, but changing the configuration of the SCR1 core, and memory size would only require to redo the physical implementation. The synthesis area of UTRISCY core would be 9051 μm^2, flattening that would probably lower the area by 10%, so 8151 μm^2. The area of UTRISCY bridge and UTRISCY VCO peripheral would be the same. The UTRISCY memory 2KB would be something like 5500 μm^2. This would make the total synthesis area of the UTRISCY control system around 15500 μm^2. Placement and routing would add another 30%, making it 20150 μm^2. With the six bondpads added, this would end up with an area of 36710 μm^2, which is significantly lower that what is presented in this work.

During the discussion with the analog design engineers from the ICD group, the question came up why there was even a need for a RISC-V control system, instead of using a simpler control system. The step from UTCONTROL (basically a small shift-register) to UTRISCY (this design) seemed quite a huge step, because for most of their design adding such as RISC-V control system to their designs would not have a lot of benefit. As explained in the introduction, using a processor has several advantages, but at the same point does it also increase the complexity for the analog design engineer as they can't just include this system in their analog simulations. Adding UTRISCY would need the analog designers to do a mixed-signal simulation, which would complicate things. Additionally, if the smallest version of this design would have been used, it would still add 36710 μm^2 to the design. Something like a system where the configurability would not be achieved by using a processor, but instead used some simpler configurable arrangement would already be much better than the current solution of UTCONTROL. One thing to look into, would be to use of the TAP controller of JTAG as an improvement over UTCONTROL, because JTAG is basically a shift-register with some additional state.

IX. Conclusion

This work has shown that the UTRISCY system can be used as a modular and reusable control system for the analog and mixed-signal test chips of the ICD group. Which answers the main question of the work: *Which design of an on-chip RISC-V-based control system is optimized for area and meets the requirements for calibration and control of analog and mixed-signal components?*. The UTRISCY design presented in this work resulted in an on-chip area of 57283 μm^2. Some trade-offs had to be made for the digital signal processing requirement, which resulted in a design that was not fully area-optimized. However, in the discussion the area for a design where performance was not needed was calculated to be 36710 μm^2.

First, the processing subsystem has been analyzed, and a study on different RISC-V cores had been done, to answer the first sub-question: *Which open-source RISC-V core is most suitable for an area-optimized design?*. This work show that the the SCR1 core is the most suitable for the RISC-V control system.

Next, the interface to the outside world has been analyzed, to answer the next sub-question: *Which interface between the RISC-V control system and the host PC minimizes the number of bondpads and enables bidirectional communication?* This work has shown that the compact JTAG interface is the optimal solution for this system. However, there is no open-source IP available for this, so for the current system the normal JTAG interface is used.

Finally, the architecture of the UTRISCY system is analyzed, and a system is designed that allows for the flexibility of the analog designer to include different components. Additionally, the AHB-Lite and APB bus interfaces were used, as the APB bus results in easy to design analog peripherals for the analog design engineers. Which answers the last sub-question: *Which on-chip bus-protocol and memory architecture are most suitable for the RISC-V control system, such that it is easy for the analog design engineer to interact with the system?*

A. Recommendations

During this work, several ideas and future optimizations were discovered, and this section will explain them.

First, the single compact JTAG to JTAG bridge open-source IP was not ready to be used in this system, and the recommendation is to start with that IP and modify it to use it in this system.

There were several part of the SCR1 core that could be improved, such as a register file optimization. The current version is flip-flop based, but generally a latch based implementation results in a smaller area. For the debug part of the SCR1 core it would be an additional

benefit if the System Bus Access port, part of the RISC-V debug specification would have been implemented, because the current version needs to halt the core for the host PC to access the memories, which is not optimal.

An additional recommendation is to implement a DMA peripheral, such that analog and mixed-signal designs could directly store results into memory, without the need of the RISC-V core to copy it to the memory. Also there is currently a latency of three clock-cycles in the AHB-lite to APB bridge, because it supports fully asynchronous clocks, however in this design the clocks are actually synchronous, so the IP could be modified to remove this extra latency.

Finally, the physical implementation of the design has to be separate into a separate layout per component. This would make it easier to change for example the memory size needed. The implemented digital EDA flow contains code that makes this almost possible, but some improvements have to be done to completely support it.

ACKNOWLEDGMENT

I would like to thank Mark and Sabih for the opportunity for this work. I learned a lot, and can now continue my career as a digital design engineer, because I was able to use the tools and learn a lot about them during this thesis. I would also like to thank Shubham for the weekly meetings that kept me sharp and motivated for this work. As a final acknowledgement, I want to thank my parents as they also kept me focused on the work.

REFERENCES

[1] S. Yan and E. Sanchez-Sinencio, "Low voltage analog circuit design techniques: A tutorial," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. 83, no. 2, pp. 179–196, 2000.

[2] J. K. Lorenz, A. Asenov, E. Baer, S. Barraud, F. Kluepfel, C. Millar, and M. Nedjalkov, "Process Variability for Devices at and beyond the 7 nm Node," *ECS Journal of Solid State Science and Technology*, vol. 7, no. 11, p. P595, 2018.

[3] ICDWiki. UTCONTROL. [Accessed 11-10-2022]. [Online]. Available: https://icejive.ewi.utwente.nl/icdwiki/doku.php?id=hdl_design:utcontrol

[4] A. Waterman, Y. Lee, D. Patterson, and K. Asanovic, "The RISC-V Instruction Set Manual," 2016.

[5] A. Pullini, D. Rossi, I. Loi, G. Tagliavini, and L. Benini, "Mr.Wolf: An Energy-Precision Scalable Parallel Ultra Low Power SoC for IoT Edge Processing," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 7, pp. 1970–1981, 2019.

[6] P. D. Schiavone, D. Rossi, A. Pullini, A. Di Mauro, F. Conti, and L. Benini, "Quentin: an Ultra-Low-Power PULPissimo SoC in 22nm FDX," in *2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, 2018, pp. 1–3.

[7] A. Traber, F. Zaruba, S. Stucki, A. Pullini, G. Haugou, E. Flamand, F. K. Gurkaynak, and L. Benini, "PULPino: A small single-core RISC-V SoC," in *3rd RISCV Workshop*, 2016.

[8] *RISC-V External Debug Support*, RISC-V Std., Rev. 0.13.2, Mar. 2019. [Online]. Available: https://riscv.org/wp-content/uploads/2019/03/riscv-debug-release.pdf

[9] G. EFabLess and Skywater. Openmpw. [Online]. Available: https://efabless.com/open_shuttle_program

[10] EFabLess. Caravel harness. [Online]. Available: https://github.com/efabless/caravel

[11] D. Annayya. Riscduino. [Online]. Available: https://github.com/dineshannayya/riscduino

[12] *The RISC-V Instruction Set Manual - Volume I: Unprivileged ISA*, RISC-V Std., Rev. 20191213, Dec. 2019. [Online]. Available: https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf

[13] *Volume II: Privileged Architecture*, RISC-V Std., Rev. 20211203, Dec. 2021. [Online]. Available: https://github.com/riscv/riscv-isa-manual/releases/download/Priv-v1.12/riscv-privileged-20211203.pdf

[14] P. Li, "Reduce Static Code Size and Improve RISC-V Compression," no. UCB/EECS-2019-107, Jun 2019. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-107.html

[15] D. Patterson, J. Bennett, P. Dabbelt, C. Garlati, and O. Shinaar. (2019, dec) Initial Evaluation of Multiple RISC ISAs using the Embench™ Benchmark Suite. What is the Cost of Simplicity? [Accessed on 29-11-2022]. [Online]. Available: https://riscv.org/wp-content/uploads/2019/12/12.10-12.50a-Code-Size-of-RISC-V-versus-ARM-using-the-Embench%E2%84%A2-0.5-Benchmark-Suite-What-is-the-Cost-of-ISA-\Simplicity.pdf

[16] lowRISC. Ibex RISC-V Core. [Online]. Available: https://github.com/lowrisc/ibex

[17] O. Group. OpenHW Group CORE-V CV32E40P RISC-V IP. [Online]. Available: https://github.com/openhwgroup/cv32e40p

[18] Syntacore. (2022, Apr.) SCR1 RISC-V Core. [Online]. Available: https://github.com/syntacore/scr1/tree/d8ba8e50ecce3bfdd47e0422a9c024384f73df60

[19] W. Digital. EL2 SweRV RISC-V CoreTM 1.4. [Online]. Available: https://github.com/chipsalliance/Cores-SweRV-EL2

[20] R.-V. International. RISC-V Exchange. [Accessed on 10-06-2022]. [Online]. Available: https://riscv.org/exchange

[21] Microsoft. Github Search RISC-V. [Accessed on 16-06-2022]. [Online]. Available: https://github.com/search?q=risc-v

[22] PULP team. RISC-V Debug Support for PULP Cores. [Online]. Available: https://github.com/pulp-platform/riscv-dbg

[23] ICDWiki. UTwo. [Accessed 2-11-2022]. [Online]. Available: https://icejive.ewi.utwente.nl/icdwiki/doku.php?id=hdl_design:utwo

[24] Stephan Nolting. (2021, Dec.) Compact-JTAG to 4-Wire JTAG Bridge. [Online]. Available: https://github.com/stnolting/cjtag_bridge

[25] Roa Logic. (2022, Feb.) AHB-Lite Multi-layer Interconnect Switch. [Online]. Available: https://github.com/RoaLogic/ahb3lite_interconnect/tree/ce8fff67a31863436baf80125662e940edd1bb6d

[26] S. Stevenson. Implementation of FIR Filtering in C (Part 2). [Accessed on 5-10-2022]. [Online]. Available: https://sestevenson.wordpress.com/implementation-of-fir-filtering-in-c-part-2/

[27] Synopsys. High Density Single-Port SRAM, Ultra-Low-Power Periphery (S1DU) Compiler. [Accessed on 28-11-2022]. [Online]. Available: https://www.synopsys.com/dw/ipdir.php?c=dwc_comp_in_gf22fdx_s1du

[28] Roa Logic. (2021, Nov.) AHB-Lite Memory. [Online]. Available: https://github.com/RoaLogic/ahb3lite_memory/tree/a749e73bf010e93d26df9356b6d4f81d36305ff4

[29] Roa Logic. (2021, Oct.) AHB-Lite APB4 Bridge. [Online]. Available: https://github.com/RoaLogic/ahb3lite_apb_bridge/tree/18d9aee1a3f5e97a7ab404067b02ca5286caf344

[30] *AMBA® APB Protocol*, ARM Std., Rev. 1.0. [Online]. Available: https://developer.arm.com/documentation/ihi0024/c

[31] OpenOCD. Tcl scripting api. [Accessed on 10-12-2022]. [Online]. Available: https://openocd.org/doc/html/Tcl-Scripting-API.html

[32] OpenOCD. remote_bitbang.txt. [Accessed on 10-12-2022]. [Online]. Available: https://github.com/openocd-org/openocd/blob/master/doc/manual/jtag/drivers/remote_bitbang.txt

[33] SiFive. SiFive GCC toolchain. [Online]. Available: https://github.com/sifive/freedom-tools/releases