**TRISTAN**

**Together for RISc-V Technology and ApplicatioNs**



**Architecture Description and Design Specifications**

**Document Number** D2.1 for WI2.5.1

**Primary Author(s)** Tiberio Fanti, Luca Lingardo, Sammy Ipenza

**Document Date** 2023-09-15

**Document Version / Status** Ver. 0.3, Preliminary;

**Distribution Level** Restricted - CONSORTIUM CONFIDENTIAL

**Reference DoA** TBD

------------------------------

**Project Coordinator** Patrick Pype, NXP Semiconductors, patrick.pype@nxp.com

**Project Website** www.tristan-project.eu

**JU Grant Agreement Number** 101095947

|  |  |
| --- | --- |
|  | TRISTAN has received funding from the Key Digital Technologies Joint Undertaking (KDT JU) under grant agreement nr. 101095947. The KDT JU receives support from the European Union’s Horizon Europe’s research and innovation programme and Austria, Belgium, Bulgaria, Croatia, Cyprus, Czechia, Germany, Denmark, Estonia, Greece, Spain, Finland, France, Hungary, Ireland, Israel, Iceland, Italy, Lithuania, Luxembourg, Latvia, Malta, Netherlands, Norway, Poland, Portugal, Romania, Sweden, Slovenia, Slovakia, Turkey |

Table of Contents

1. Introduction 3

1.1. General Information 3

1.2. Purpose and Scope 4

1.3. Acronyms and Definitions 4

1.4. Document History 5

2. Architecture 6

2.1. Place in the System 6

2.2. Block Diagram 7

2.2.1. Hardware 7

2.2.2. Software 12

2.3. Interfaces 15

2.3.1. Instruction fetch interface / Load-store interface 15

2.3.2. Interrupt interface 16

2.3.3. Extension interface 17

2.3.4. Firmware interface 18

2.4. Sub-Modules 19

2.4.1. CV32E40X core 19

2.4.2. Coprocessor – ID Stage 20

2.4.3. Coprocessor – EX Stage 20

2.4.4. Coprocessor – FIFOs 21

2.5. Clocking Strategy 21

2.6. Reset Strategy 21

2.7. Power Management Strategy 21

2.8. Debugging Strategy 21

3. Design Specifications 23

3.1. Pin-list 23

3.2. Register Map 25

3.3. Functional Description 25

3.3.1. <sub-module A> 26

3.3.2. <sub-module B> 26

3.3.3. … 26

4. References 27

# Introduction

* 1. General Information

The described module is a co-processing unit meant for DSP operations. It is meant to be used in conjunction with an OpenHW Group CV32E40X RISC-V high performance core which offers an interface for extending it with a coprocessing unit. This eXtension Interface (XIF) eases the Instruction Set Architecture (ISA) extension to support dedicated DSP operation for performance enhancement. The core is coupled thanks to this interface with a coprocessor in order to perform accelerated DSP filtering operations. Alongside with the according firmware the system is able to demodulate and decode ISO 14443 TypeA 106kBd card response. Future work will further extend the firmware to fully support different baud-rates as well as other standards. The actual challenge is performing all the tasks supported in an efficient way, providing an architecture that operates at a feasible clock rate for real-time DSP applications and that ensures a proper trade-off between gate count and power consumption. The further ambition is to provide this module as an off-the-shell block to be integrated into a more complex NFC Software Defined Radio (SDR) modem.

* 1. Purpose and Scope

Purpose of this document is to provide guidance to the HW Concept and Design development phases of the project. Main requirements specifications have been collected in a separate document which makes part of TRISTAN WP1 first deliverable.

The document refers to a Consortium Confidential deliverable. As such, its distribution is meant to be limited to the owners’ team and, for information, to all TRISTAN Consortium partners.

* 1. Acronyms and Definitions

|  |  |
| --- | --- |
| Acronym | Description |
| ALU | Arithmetic Logic Unit |
| CU | Control Unit |
| DSP | Digital Signal Processing |
| EX | Execute |
| FIFO | First In First Out |
| IF | Instruction Fetch |
| ID | Instruction Decode |
| ISA | Instruction Set Architecture |
| MAC | Multiply and Accumulate |
| NFC | Near Field Communication |
| OM | Output Modifier |
| SIMD | Single Instruction Multiple Data |
| SDR | Software Defined Radio |
| WB | Write Back |
| XIF | eXtension Interface |

* 1. List of Contributors

|  |  |
| --- | --- |
| Contributor (or entity) | Role |
| Tiberio Fanti | NXP-AT team leader; TRISTAN WI2.5 leader |
| Luca Lingardo | TRISTAN WI2.5.1 coordinator |
| Technical University of Graz | NXP-AT academic partner for TRISTAN |
| Technical University of Turin | NXP-AT academic partner for TRISTAN |

* 1. Document History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Author | Date | Notes | Status |
| 0.1 | All Team | 2023-09-12 | First Draft version | Draft |
| 0.2 | Tiberio Fanti | 2023-09-15 | Clean up of all unnecessary pages and guidance-notes left in from the original template. Ready to be uploaded. Requires additional data before being ready for D2.1. | Preliminary |
| 0.3 | Luca Lingardo | 2023-10-31 | Final preliminary version of the document ready. Reviews to be done. | Preliminary |

# Architecture

* 1. Hardware

The block diagram in Figure 3 shows as sub-modules an id-stage for the decoding of the instruction offloaded by the core, an ex-stage for executing the different coprocessor tasks and sending the result back to the core and finally two FIFOs. It is also possible to notice the five sub-interfaces of the extension interface XIF involved: **issue\_if** for the offloaded instruction by the core, **commit\_if** to assess whether the instruction must be committed or killed, **memory\_if** for asking a memory access through the core as intermediate, **memory\_result\_if** for collecting the data eventually coming from the memory access and finally the **result\_if** for providing the final result of the coprocessor instruction back to the core.

.*A diagram of a computer system

Description automatically generated*

Figure 3: Coprocessor high level block diagram and XIF interfaces

DSP operations are performed via digital filters which are represented by logical banks inside the coprocessor. Each bank consists of multiple elements which store filter coefficients and the current data values. When such a bank gets executed, each coefficient will be multiplied with the according data value and each intermediate result will be summed up (accumulated). This whole operation is performed inside the Multiply and Accumulate (MAC) unit. Furthermore, the coprocessor is capable to perform DSP on samples in Single Instruction Multiple Data (SIMD). Incoming samples do have an I and Q parts which are stored in the upper and lower 16 bit parts of a 32 bit word. To make use of SIMD a bank needs to be specifically configured to support SIMD. When activated the coefficients of that bank are also split into upper and lower parts so that both I and Q parts are calculated separately by multiplying the according coefficient.

The data flow for one bank execution can be seen in Figure 4 and Figure 5. The second figure contains an additional module called output modifier (OM) which as the name suggests is able to apply simple data manipulations on the result of the MAC unit.

A diagram of a blue rectangle with black text

Description automatically generated

Figure 4: Initial Processing chain with MAC arithmetic part + rounding unit + result interface

*A blue rectangle with black text

Description automatically generated*

Figure 5: New processing chain with the output modifier as last step of the processing before the result if

Starting from the bottlenecks intrinsic to the first implementation of the coprocessor RTL + firmware what came out was that some features could be ported in HW to be executed in a more efficient way (absolute computation and sum), leading to a SW with a lower number of instructions and saving also a bank in the filtering pipe of the coprocessor.

The internal structure of the output modifier module can be seen in Figure 6, showing the two operations absolute and sum of the upper and lower parts of the result word (SIMD case). Furthermore, new operations can easily be added if needed in the future.

*A diagram of a diagram

Description automatically generated*

Figure 6: Output modifier block diagram

Additionally, Table 1 lists the mathematical behavior of the module.

|  |  |  |
| --- | --- | --- |
| *Output mode selection* | *!SIMD\_MODE\_I* | *SIMD\_MODE\_I* |
| *0b00 (wiring)* | *Y = X* | *Yhigh = Xlow ,Ylow = Xlow* |
| *0b01 (absolute)* | *Y = |X|* | *Yhigh = |Xhigh|, Ylow=|Xlow|* |
| *0b10 (sum)* | *NaN* | *Y=Xhigh + Xlow* |

Table 1: Behavioral table for the output modifier

Each filtering operation is executed with a dedicated custom RISC-V instruction. In the DSP demodulation algorithms, the execution of multiple filter operations is needed resulting in multiple instruction calls on the coprocessor. This introduces an additional overhead due to the need of performing handshakes between the CV32E40X core and the coprocessor as well as the latency problem of fetching the instructions from memory. A speed up can be achieved if the filtering tasks are executed in a chaining mode which means applying multiple filters sequentially with a single instruction call. Every time a filtering step is executed, the instruction, not recognized by the core, is sent to the XIF and then to the core, waiting for the processing of it and for the final result. This round-trip takes some cycles not only for the processing itself of the data but also for the handshake between the XIF and the coprocessor. Thanks to the chaining idea, instead of ‘wasting’ some clock cycles (was estimated 3 clock cycles as best effort, potentially much more in the worst case)) for each filtering step, the handshake between XIF and coprocessor is executed once for the whole processing, while the filtering steps are performed sequentially without waiting for the next instruction handshake, considering the output of the actual step as the input of the next one and so forth. The structure for applying filter chaining is shown in Figure 7. The main part is the feedback signal that transfers the output of one filter execution to the input of the next filter execution. Therefore a custom logic is needed in order to properly configure the banks involved in the chaining and properly execute the filtering on the whole amount of data before collecting the result in the result if.

A diagram of a computer

Description automatically generated with medium confidence

Figure 7: Basic schematic for the filter chaining idea

* 1. Interfaces
     1. Extension interface

This interface connects the main processing unit (CV32E40X) with the coprocessor. It is furtherly divided into six sub-interfaces: compressed, issue, commit, memory, memory result, result. The integration of the XIF into the system can be seen in figure 12.

A diagram of a computer

Description automatically generated

Figure 12: Wrapper X-core + XIF + coprocessor, showing XIF modports

The coprocessor interface consists of the modports connecting the coprocessor to the extension interface and the clk and rst signals, as can be seen in figure 13.

*A screenshot of a computer program

Description automatically generated*

Figure 13: Coprocessor interface

* 1. Sub-Modules

*In the following sub-sections there are a brief listing and description of sub-modules involved in the system.*

* + 1. Coprocessor – ID Stage

Coprocessor Instruction Decode stage (ID-stage) is in charge of decoding the instructions coming from the issue interface, signalling to the ID-stage Control Logic if an offloaded instruction is recognized and can be executed or if it is unknown and has to be rejected. Moreover, all the signals stating the properties of the decoded instruction (e.g. writeback, the number of source registers needed) are generated and sent to the ID-stage Control Logic.

The main module in this stage is certainly the decoder which is capable of properly interpret the instruction coming from the core according to the opcode, funct3 and funct7, then deliver accordingly to the execution-stage the source operands rs1, rs2 and store the destination register rd for the outcome of the instruction itself in case of write-back. Moreover, it is strictly related to the issue interface of the XIF.

* + 1. Coprocessor – EX Stage

The other essential sub-module in the coprocessor is the Execution-stage (EX-stage). It is responsible of interfacing with XIF, carrying out the proper arithmetic instruction and perform the rounding.

EX-stage control logic manages the commit interface, the memory interface, the memory result interface and the result interface, implementing all the handshaking and data signals needed for a correct core-coprocessor communication. It also generates the control signals for the ALU execution (e.g. instruction kill) and it manages the ones coming from it (e.g. memory accesses, final result providing) acting as a core-ALU intermediary.

Regarding the MAC general architecture, what is important to state is that there are some data registers, coefficient registers and configuration registers (better refer to them as banks) which are essential in order to perform FIR and IIR filtering. Then there is the MAC ALU, which is the core of the arithmetic processing, implementing the Multiply-Accumulate algorithm on the operands coming from the decoder and stored into the data banks, leveraging the coefficients of the several banks. When the result of the arithmetic processing is ready, it is post-processed by the rounding unit implementing the rounding scheme round-to-nearest or round-to-nearest even (modifying the RTL code respectively). All these operation are monitored thanks to a control unit (CU), that basically is a quite complex logic since there are many signals to be managed, most of them due to the handshaking with the main core. We can distinguish between memory transactions finite state machine (3-state FSM) and internal registers loading finite state machine (2-state FSM).

* + 1. Coprocessor – FIFOs

Being a producer-consumer system able to manage only one element (instruction) at a time, 1-element First-In First-Out (FIFO) is needed . Two FIFOs are present, one managing the issue part and another one the commit part and they are connected to the related XIF interfaces.

* 1. Clocking Strategy

The architecture runs on one single clock, frequency to be yet defined.

* 1. Reset Strategy

The architecture has one single reset line, active low.

* 1. Power Management Strategy

The architecture is powered by one single power domain.

* 1. Debugging Strategy

*Describe here any debugging strategy put in place for the module.*

# Design Specifications

* 1. Register Map

Access modes:

* **RW:** CSR is read-write. That is, CSR instructions (e.g. csrrw) may write any value and that value will be returned on a subsequent read (unless a side-effect causes the core to change the CSR value).
* **RO:** CSR is read-only. Writes by CSR instructions raise an illegal instruction exception.
* **R:** read fields are not affected by CSR write instructions. Such fields either return a fixed value, or a value determined by the operation of the core.
* **RW:** read/write fields store the value written by CSR writes. Subsequent reads return either the previously written value or a value determined by the operation of the core.
* **WARL:** write-any-read-legal fields store only legal values written by CSR writes. The WARL keyword can optionally be followed by a legal value (or comma separated list of legal values) enclosed in brackets. If the legal value(s) are not specified, then all possible values are considered valid. For example, a WARL (0x0) field supports only the value 0x0. Any value may be written, but all reads would return 0x0 regardless of the value being written to it. A WARL field may support more than one value. If an unsupported value is (attempted to be) written to a WARL field, the value marked with an asterix (the so-called resolution value) is written. If there is no such predefined resolution value, then the original (legal) value of the bitfield is preserved.
* **WPRI:** Software should ignore values read from these fields, and preserve the values when writing.

>ANY COPROC REGS GO HERE

* 1. Functional Description

*Describe the behavior of each functional block.*

* + 1. <sub-module A>

…

* + 1. <sub-module B>

…

* + 1. …

…

>ADD REFERENCES IF NEEDED

|  |  |
| --- | --- |
| [*www.tristan-project.eu*](http://www.tristan-project.eu) | |
| *info@tristan-project.eu* | |
|  | |
| Graphical user interface, application  Description automatically generated | *TRISTAN has received funding from the Key Digital Technologies Joint Undertaking (KDT JU) under grant agreement nr. 101095947. The KDT JU receives support from the European Union’s Horizon Europe’s research and innovation programme and Austria, Belgium, Bulgaria, Croatia, Cyprus, Czechia, Germany, Denmark, Estonia, Greece, Spain, Finland, France, Hungary, Ireland, Israel, Iceland, Italy, Lithuania, Luxembourg, Latvia, Malta, Netherlands, Norway, Poland, Portugal, Romania, Sweden, Slovenia, Slovakia, Turkey.* |