USING RISC-V TO DEFINE SOCS IN SOFTWARE

WHITEPAPER

XMOS

©2022 XMOS Ltd.

IN THIS DOCUMENT

- Introduction: XMOS processors
- Adoption of RIS-V
- Real-time guarantees
- Hardware vs software

XMOS has moved to a RISC-V compatible instruction set for its latest processor that will be available for sampling in 2023. In this whitepaper we discuss how this processor and its RISC-V compatible architecture can be used.

INTRODUCTION: XMOS PROCESSORS

XMOS prides itself on designing and selling processors that can fully implement the digital requirements of modern embedded systems. Specifically:

- XMOS xcore® processors are highly flexible like an FPGA and can target complex and highly varied designs. Unlike an FPGA, this is all expressed in software.
- XMOS processors are high performance like a DSP or an AI accelerator. Unlike DSP and AI accelerators, XMOS processors offer a single, unified processor, programmed through a single flow, where data does not need to be moved to take advantage of optimisations.
- XMOS processors are scalable (much like high performance computers), enabling designers to gradually extend their design without going back to square one. This allows them to easily create spin-off products with extra capabilities.
- XMOS processors are affordable, easy to use, and low energy, much like a microcontroller.

In that capacity, XMOS processors can often absorb elements of the design that would traditionally require different classes of components. Where traditionally one may use a microcontroller to control the design, and a DSP to do the signal processing, and maybe a CPLD to interface to a complex digital interface, XMOS processors can do these three tasks in a single device using a single, unified software-based workflow.

©2022 XMOS Ltd.

ADOPTION OF RISC-V

For its latest generation of xcore, XMOS has adopted compatibility with the RISC-V architecture. The centrepiece of all XMOS processors is that the system is programmed using a collection of hardware threads. Individual parts of the system are programmed using one or more threads each, and all threads are composited together into a single system that implements the full embedded system.

With the adoption of RISC-V, these threads become RISC-V HARTs; which stands for "HARdware Threads". Each HART can be seen as its own individual execution engine that is running in parallel with all the other HARTs.

HARTs enable system integrators to customise the xcore processor for their particular solution. For example, in the case of an industrial robot, a few HARTs may be dedicated to implementing a highly efficient PWM generator. Another designer can use the same chip in consumer electronics where these HARTs are being used to interface and mix various audio signals.

Because both the PWM and Audio interfaces are implemented in software, there is unrivalled opportunity for the system designer to customise the software and put their own unique IP at the core of their system; they do not have to work around limitations of any hardware blocks.

Each HART is a RISC-V compatible machine, with custom extensions to enable fast synchronisation and communication between HARTs and between the software and I/O signals. The core instruction set is RV32IM: a 32-bit RISC-V with 31 registers per HART, and the M extension for multiplies and divisions. All control code can use this instruction set, and indeed the toolchain is based on the LLVM RISC-V compiler, assembler, and linker. Assembly programs written for the RV32IM can run without any modification.

REAL-TIME GUARANTEES

The architecture underneath the threads guarantees that each HART issues an instruction every thread-cycle. Threads follow each other through a shared pipeline, so with eight active threads, an instruction will be issued every eight clock cycle. When clocked at 600 MHz, that guarantees an issue rate of 75 MHz for each HART. Because of this guarantee, each HART can be programmed independently, tested independently, and have its real-time properties characterised and "signed-off" independently. Based on this foundation, software can be composited together simply by running different real-time functions on different HARTs.

This means that a collection of RISC-V HARTs replaces a large swath of logic that would normally require dedicated hardware, for example, a USB MAC. Whereas a hardware USB MAC would have a fixed number of endpoints with fixed functionality and fixed buffering, the software-based MAC will only have the buffering and endpoints that the endpoint requires.

RISC-V HARTs use a programming model that is known as concurrent real-time programming. We show an example of this in Figure 1 that shows how a single xcore processor can execute

©2022 XMOS Ltd.

eight threads simultaneously; each HART can be seen as an individual processor with a guaranteed real-time execution rate. The device contains two processors, for a total of 16 threads that execute simultaneously. Each instruction that can execute will execute within a known amount of time, and hence the programmer can predict whether programs will meet timing deadlines or not.

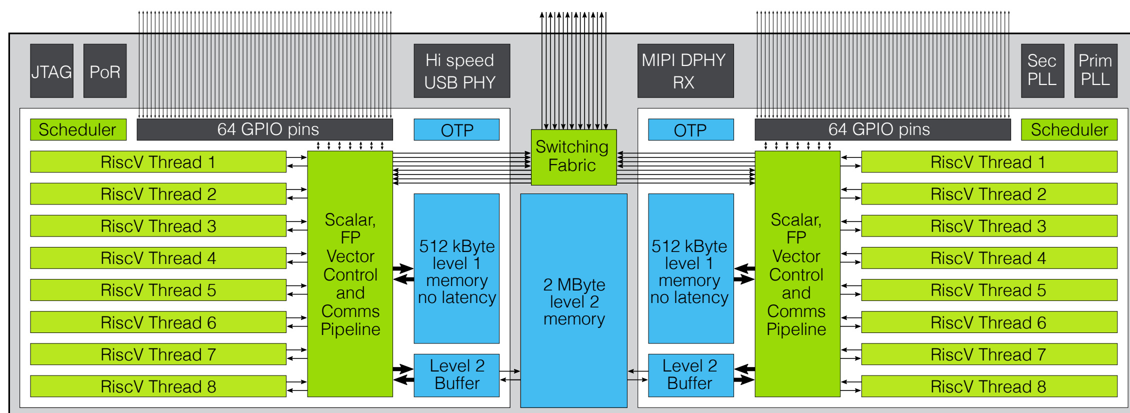


Figure 1

The beauty of this model is that timing prediction is entirely self-contained to the task. As an example, consider the PWM task mentioned before: once the program has been written to execute on, for example, a 75 MHz RISCV HART, then regardless of modifications to other threads, this PWM thread will always meet its timing.

HARDWARE VS SOFTWARE

There are many why a software implementation is preferable over a hardware interface:

1. Part selection. In a traditional approach, the designer will select a part with the appropriate interfaces; any interfaces that come up at a later stage would require a different part to be selected; possibly from a different family or supplier, or additional components on the bill-of-materials.

   With a software implementation, part selection only involves selecting the size of the part required, with a trade-off between capability and price.

2. Adaptability. Hardware implementations have fixed and therefore limited functionality. In a software implementation, functionality can be adapted to allow for novel or evolving aspects to be included. This includes dealing with idiosyncrasies of specific devices, or indeed custom extensions to protocols.

3. Optimisation. Hardware-defined implementations have a fixed and prescribed functionality, with choices made by the hardware designer typically years previously.

©2022 XMOS Ltd.

A software-defined implementation can be optimised to the problem at hand. For example, a software defined interface may be optimised to minimise latency, minimising buffering and jitter in the data.

The third reason is remarkably important as many standard blocks dealing with interfaces like UARTs, I2S, MII, USB, MIPI, and other standards must have built-in buffers to decouple the interface, and those buffers are, in many cases, both unwanted and expensive as they add delay to the algorithm and area to the silicon.

Flexible hardware can enable greater product differentiation, allowing developers and product managers to deliver distinguishable value to their customers. Concurrent real-time programming allows very tight timing requirements to be guaranteed, with embedded processors such as the XMOS xcore providing silicon and development tools required to take advantage of this programming method.

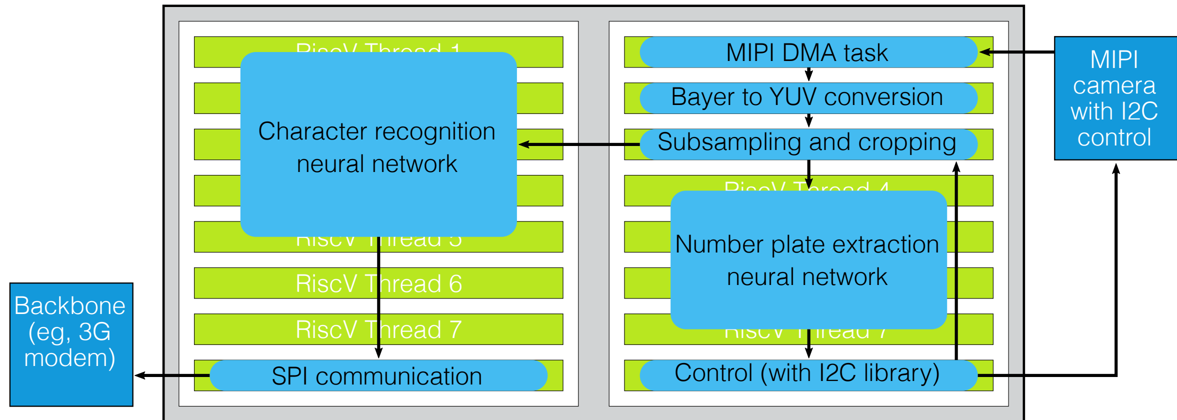
AN EXAMPLE: NUMBER PLATE RECOGNITION

As a worked example, let's look at how something like number plate recognition works on xcore. Number plate recognition typically comprises a camera, an image processing pipeline, two or more neural networks, and an interface to a back-bone.

- The MIPI DMA task is a very short C function that places lines of the image into memory.
- The debayering and subsampling tasks are also written in C, but call library functions on-the-fly to perform the DSP necessary to perform the colour conversion and image subsampling.
- The two neural network tasks are both compiled down directly from their learned models. Given the trained model, the model is quantised, and then optimised for xcore. The optimised model is then compiled to code that invokes the kernels that are optimised for xcore and compiled using the standard LLVM C-compiler. The neural networks are compiled to parallelise onto 4 and 5 threads respectively.
- The Control and SPI tasks are written in C.

As these are all software decisions, the system designer may choose to shuffle resources between the tasks; enabling trade-offs to be made between the speeds and performances of all of the components. It may seem wasteful to dedicate a hardware-thread to be a MIPI DMA engine; but it saves having dedicated DMA engines that may be unused or inappropriate. It is also important to note that this DMA engine is fully programmable, and as such can interpret CSI-2 headers in the process of performing the DMA operation.

©2022 XMOS Ltd.



It is worth noting that the whole flow is in software, and in the end the LLVM RISCV C-compiler is used to create a single image. It is this single flow that enables quick re-designs and trade-offs between the various components.

For further information or to discuss your requirements, please visit xmos.ai or contact sales@xmos.com.

©2022 XMOS Ltd.

Copyright © 2022, All Rights Reserved.

XMOS Ltd. Is the owner or licensee of this design, code, or Information (collectively, the “Information”) and is providing it to you “AS IS” with no warranty of any kind, express or implied and shall have no liability in relation to its use. XMOS Ltd. Makes no representation that the Information, or any particular implementation thereof, is or will be free from any claims of infringement and again, shall have no liability in relation to any such claims.