

XMOS Technology Whitepaper



Publication Date: 2010/04/28

Copyright © 2010 XMOS Ltd. All Rights Reserved.

1 Introduction

Designers for electronic products are challenged by requests for customized and differentiated complex products in short time frames. In this environment two factors become crucial: flexibility and simplicity. For many years ASICS or FPGAs allowed engineers to meet very specific product design requirements. However, the design process involved is complex, time-consuming and expensive. Technologies such as microcontrollers or DSPs provide simplicity, but they lack the flexibility needed to meet rapidly changing market requirements.

XMOs delivers a technology that provides the flexibility of FPGAs or ASICS with the design simplicity of processor based designs. The XMOs XCore is a processor that is programmable in a high-level language (C), yet capable of performing low level, real-time, tasks. For example, hardware interfaces can be implemented entirely in software. A system is programmed by means of a set of software components that are rapidly customized and composed. Products are hence easily differentiated. Software designed for one product can also be reappropriated in other products, speeding up development time. Source code libraries are available to implement many standard interfaces and protocols.

This document provides a brief introduction to XMOs technology:

- the XCore processor design and performance;
- the XMOs tool chain and development tools;
- software components and reference designs.

2 The XCore processor architecture

An XCore processor runs multiple real-time tasks *simultaneously* using *hardware threads*. Each task has access to a set of general purpose registers and shared memory, gets a guaranteed slice of processing power (between 50 and 100 MIPS), and executes a program using common RISC style instructions. Each task can execute straight computational code (for example DSP code), control software (including taking logic decisions and executing a state machine) or software that handles I/O operations.

I/O pins can be set and sampled in a single instruction. In its simplest form an OUT instruction drives a new value onto up to 32 I/O pins, and an IN instruction samples up to 32 I/O pins. More complex use allows data to be serialized and deserialized (up to 60 MHz), and input ports can be asked to discard any data until a condition is met. Input and output can also be done at an exact number of clock ticks in the future. I/O instructions are used to implement traditional hardware interfaces in

software. I/O can be driven with events or interrupts. The event mechanism offers a low and predictable response time, interrupts enable legacy code to be ported.

Multiple cores can be deployed and connected to each other using a build-in interconnect. This enables the designer to transparently create a system with more resources—more I/O pins, memory, or processing power. Within a single core, tasks can use shared memory to exchange data with other tasks. Tasks can also use channels to exchange data and synchronize across cores or within a core. Single cycle instructions can exchange a word of data, implicitly synchronizing the receiver with the sender. If shared memory is used, tasks can use single cycle lock-instructions to synchronize.

The processor architecture is deterministic. With t threads, each thread executes its next instruction at most t clock ticks in the future. All instructions (with the exception of divide) are single cycle, and the instruction buffer is prefetched at known places in the program, usually in parallel with program flow. Communication between threads on a core incurs no latency, and communication between cores incurs a latency that can be computed if the communication pattern is known. The communication architecture can assign links to different *networks* in order to provide Quality of Service.

3 Performance

The XCore architecture is available in 1, 2, and 4-core packages. Packages come at different speeds, up to 400 MHz (for the G4) or up to 500 MHz (for the L1). All figures below relate to a 400 MHz package. 500 MHz packages are 25% faster.

A single 400 MHz core can execute at most 400 million instructions per second. Each thread can operate at up to one quarter of the clock frequency: 100 MIPS on a 400 MHz core. When four threads execute on a core, all threads will run at 100 MIPS. When more threads execute, the MIPS rate per thread reduces, and with eight threads each thread runs at 50 MIPS: t active threads execute in *thread-cycles*, where a thread-cycle equals t 2.5-ns clock cycles.

Each core memory can serve data at up to 12.8 Gbits per second. Each thread can access at most one word of memory per instruction. Instructions are prefetched when an instruction is executed that does not access memory. If a thread has not prefetched the next instruction, then that thread will stall for one thread cycle in order to read the next word of instructions. These fetches are predicted statically and happen rarely.

A pin input- or output-operation takes a single clock cycle. A thread can program the core to wait for any number of possible events (inputs), and each thread can handle at most one event every two thread-cycles. Including a meaningful operation, each thread can handle an event every four thread cycles, so a single 400 MHz core

can handle at most 100 million events per second, enabling, for example, 100-Mbit Ethernet Mac + TCP to be implemented entirely in software.

Except for the divide instructions, all thread operations take a single thread cycle; this includes 32x32 into 64 MAC, long add, and 32-bit CRC. Instructions are required around a MAC instruction to manage data. A FIR can be implemented using three instructions per TAP, plus 10 instructions. On a single 100 MIPS thread a 4-tap FIR can execute at 4.5 MHz, a 100-tap FIR executes at 300 KHz. A 32-bank biquad filter can operate at 100 KHz on a single 100 MIPS thread. Biquad (and FIR) operations can run from registers for specific cases and a thread can execute at most 16 MBiquads per second if it just runs a single biquad filter.

On the quad core G4, processors are connected by means of a switch. Each processor can simultaneously send and receive data through the switch at up to 12.8 Gbits per second. The switch has external links (X MOS Links) that can simultaneously transmit and receive up to 6.4 Gbits per second. A single communication channel from a core can transmit into the switch at up to 3.2 Gbits/second, and transmit through a single link at up to 400 Mbts/second. Link speed can be reduced in order to cover longer distances.

4 Tool chain

The X MOS tool chain allows the designer to use C, C++, or XC to program devices, or assembly if the user wishes to have total control. ANSI C and C++ are compiled using the LLVM compiler. XC is a version of C originated by X MOS that supports concurrency and real-time I/O. Programs can be debugged by using gdb (for high-level debugging), print statements, or toggling I/O pins.

The tool-chain can generate boot images to boot from secure OTP, flash memory, secure flash memory, X MOS Links or JTAG. Typically programs are developed using JTAG or flash, whilst OTP/flash/X MOS Link are used for volume productions.

The X MOS tool chain provides both a command line interface and a graphical user interface (Eclipse) that runs on Unix, Windows, and MacOS X. It can be downloaded from the X MOS website and used for free. In addition to the compilers and assembler, the tool chain includes a simulator, waveform viewer and static timing analyzer. The combination of a compiler and static timing analyzer enables the programmer to statically close timings on modules of the design.

5 Software and Reference Designs

XCore processors can implement a variety of hardware interfaces, network protocols, control, and data processing algorithms. An exhaustive repository of software is available on the XMOs website, but example software components and reference designs include:

Ethernet	MAC, IP, TCP, web server
USB-Audio-2.0	High channel count, high sample rates
iPod dock	Digital iPod and iPhone dock ^a
USB-HID	Example mouse implementation
AVB	IEEE Audio over Ethernet
LED tiles	Controls displays over, for example, Ethernet
Class D amplifier	Based on any transport layer
UART, I2C, SPI, I2S, CAN	Standard serial interfaces
SD card	SD card interface
CMOS sensor	Camera interface
S/PDIF, ADAT	Digital audio interfaces

^aRequires MFI registration with Apple

In addition, software has been designed by a community of XMOs users:

Industrial control	Controlling industrial robots and lasers
FreeRTOS	A small standard real-time kernel
Robotics	A variety of autonomous robots

These can be downloaded from <http://www.xcore.com/>

External physical interfaces (“PHYs”) are required for many tasks, such as USB, Ethernet or CAN. These are connected to one or more pins of the XCore processor. A package must be chosen that has sufficient pins; A small package (such as the 64 LQFP) has 32 I/O pins that can, if they are all required, be bonded out using a 2-layer PCB. If more than 32 I/Os are required, a bigger package can be used. The biggest package, the 512BGA, offers 256 I/O pins, which can be bonded out using a 6-layer PCB. In addition, the L1, L2, and G4 need a 1V core power supply, a 3.3V I/O power supply (the I/O is 5V tolerant), a clock, and a reset signal.

6 Summary

XMOs technology:

- Lets you implement many hardware interfaces by means of a software task.
- Allows you to compose hard real-time software tasks into a single program.
- Provides guaranteed processing throughput and memory bandwidth to separate tasks.
- Provides an interconnection network that enables you to split large problem over a network of cores.
- Has a free tool chain that allows you to program the devices in a high-level language (for example C).

7 Further Reading

For more details, please refer to the following documents, website and videos:

- *XMOs XS1 Architecture* specifies the architecture and its instruction set, available from http://www.xmos.com/published/xs1_en
- *Programming XC on XMOs Devices* is a comprehensive guide to programming XCores using a high-level language, available from http://www.xmos.com/published/xc_en
- Datasheets are on <http://www.xmos.com/support/documentation>
- XMOs technology is presented in a collection of videos on YouTube: http://www.youtube.com/MyXMOs#p/u/4/RswWpPobA_0
- Community designs can be viewed and downloaded from <http://www.xcore.com/>

Disclaimer

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.

Copyright © 2009-10 XMOS Ltd. All Rights Reserved. XMOS and the XMOS logo are registered trademarks of XMOS Ltd in the United Kingdom and other countries, and may not be used without written permission. Company and product names mentioned in this document are the trademarks or registered trademarks of their respective owners. Where those designations appear in this document, and XMOS was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.