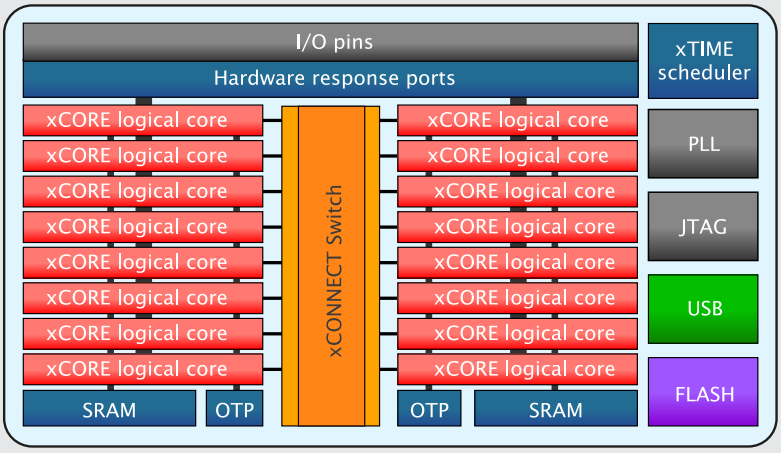
xCORE

xCORE ARCHITECTURE

There is a limit to how much any of us can do alone. When faced with complex problems, and tight timescales, we break problems down into smaller tasks, then we distribute them to capable individuals, and we try to leave them to get the job done. When this is accepted wisdom in the workplace, why do we tolerate computer systems that operate alone, that must be interrupted to respond to urgent requirements, that exhibit huge inefficiency and/or deliver results late? Because, until now, there has been no choice.



xCORE-200 XUF216

xCORE multicore microcontrollers are unique. Like traditional MCUs they are programmed in C/C++, but unlike traditional products each device contains a team of multiple processor cores that can execute several tasks concurrently and independently. Each core is incredibly flexible - capable of control, DSP and high performance GPI/O processing. Each core is also predictable, exhibiting completely reliable timing characteristics. Finally, each core is connected, tightly coupled with other members of the team.

Put simply, xCORE multicore microcontrollers let you deliver smarter solutions quicker, using technology that is as simple as it is flexible, predictable and scalable.

THE ANSWER IS SIMPLE, MULTICORE MICROCONTROLLERS

Today's embedded applications must process multiple concurrent tasks quickly, minimize jitter and handle combinations of fast, complex interfaces. It must provide easy integration of a wide range of components and be scalable to large and small systems.

Traditional real-time systems are interrupt-driven, relying on an RTOS to schedule tasks and handle communication. These systems face many challenges; such as handling demanding I/O streams, interrupt latency, kernel processing time, scheduling jitter, cold-cache effects and memory/resource contention. Systems built on such a foundation are difficult or impossible to verify, and the RTOS itself imposes significant processor and memory overhead.

The xCORE architecture solves these problems by removing all of the features of a traditional MCU that introduce uncertainty. Many RTOS features are integrated in hardware: for example, system events are handled using a scheduled single-cycle context switch, so applications do not suffer context-switching latency or jitter.

On xCORE, each activity has its own dedicated core and resources ensuring that, when needed, a response is immediate.

With multiple processing cores executing completely independently, tasks are guaranteed to complete within a strict timing budget. Each core can run I/O, DSP and application code. The connection between processor resource and hardware ports is so tight that response times are measured in nanoseconds - thousands of times faster than traditional microcontrollers. This hardware-like performance enables interfaces to be entirely written in software, allowing systems to be designed with the exact combination of peripheral interfaces they require.

- **Multiple processing cores**
  - Runs multiple tasks concurrently
  - Guaranteed performance
- **Immediate response time**
  - 100x faster than other MCUs
- **Flexible ports and peripherals**
  - Sophisticated port logic
  - Supports fast complex interfaces
  - Multiple peripherals in one chip
- **RTOS features in hardware**
  - Scheduler, timers, communications
  - Predictable repeatable behavior
  - Low latency
- **Integrated development tools**
  - Programmed in C and C++
  - Instrumentation/trace libraries
  - Static timing analyzer
- **Multicore extensions for C**
  - Concurrent tasks, timing, communication, I/O, memory management

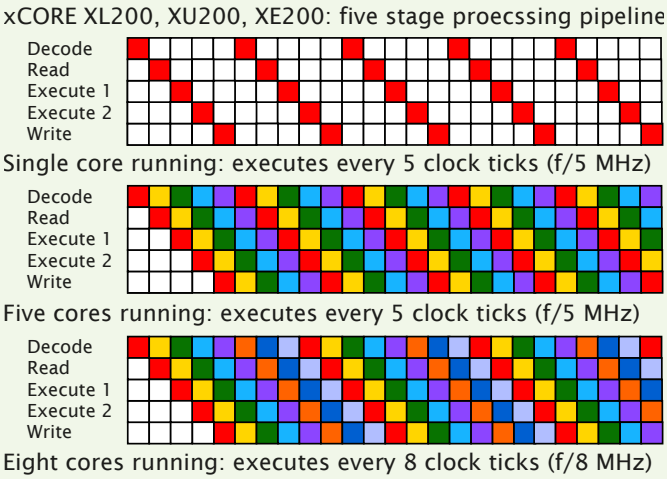
xTIME™ SCHEDULER

Each xCORE device has one or more tiles. Each tile has up to eight independent 32-bit logical cores that run in parallel without interruption from other cores.

Active cores are guaranteed a minimum level of MIPS. Cores that are idle are not scheduled to the processing resource.

All instructions complete in a single core cycle, or pause the core, waiting for an external event. On xCORE-200 each core can issue up to 2 instructions per clock cycle.

Cores are triggered by events that are managed by the xTIME scheduler. Events that occur at I/O pins are fed directly to a core by the Hardware Response ports. Events can also be generated by timers and tasks, and serviced by the scheduler, with guaranteed behavior.



EVENTS AND INTERRUPTS

Traditional devices take a number of instruction cycles to respond to an interrupt, during which time they store the state associated with the running task and then load the new state associated with the higher priority task that needs to be started.

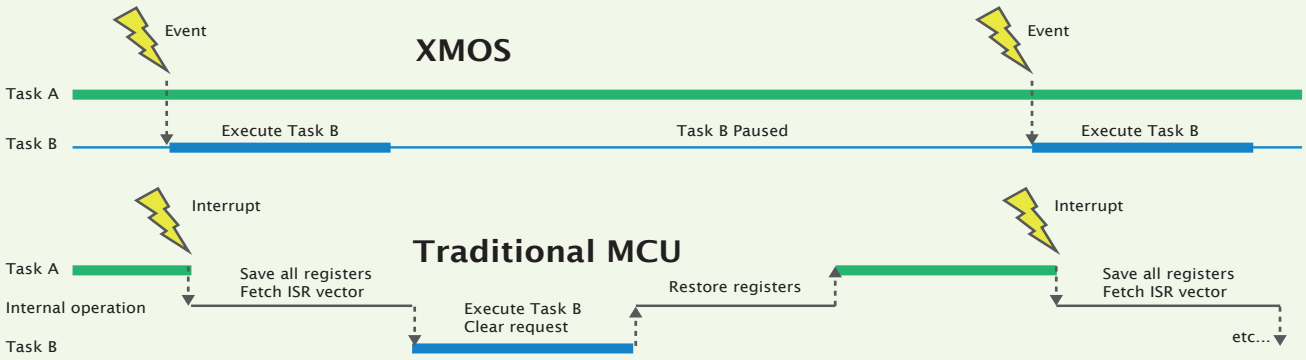
XMOS devices respond to events triggered by I/O pins, timers and tasks, rather than interrupts. Since an xCORE device can run multiple tasks in parallel there is no need for one task to interrupt another. A task can handle an event by running in parallel with other tasks and waiting for the event to happen.

The advantages of the XMOS approach include:

- Response time to events is dramatically improved (in conjunction with the multi-core xCORE architecture).
- Reasoning about worst-case execution time (WCET) is easier since code cannot be interrupted during its execution.

Independent tests show that xCORE devices respond to single events within 10ns and handle multiple asynchronous real-time events within a worst-case response time of 100ns; this is 100x times faster than conventional real-time systems and critically, does not degrade in larger, more integrated systems.

(http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6389416)



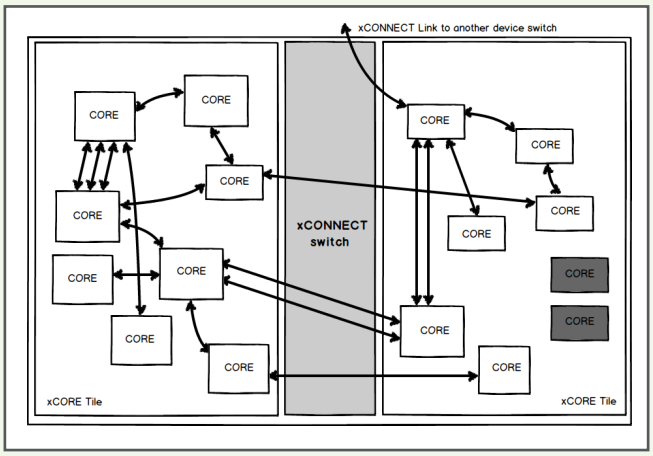
TASK PRIORITY

By default, each task in an xCORE application is placed on a different logical core. This means the task runs independently of the others and has incredibly quick event response time. In RTOS terms, each task running on its own core enjoys the highest priority scheduling by the hardware. To reduce resource requirements, lower priority tasks can share a core via more traditional multi-tasking; either by running an RTOS, or by using built-in co-operative multitasking features.

TASK COMMUNICATION

The high-speed xCONNECT network ensures that all tasks can communicate with very low overhead, whether they are on the same tile, different tiles within a chip, or across a network of xCORE devices.

When combined with the predictability of xCORE, the scalability of xCONNECT enables both large and small systems can be constructed from the same software investment.

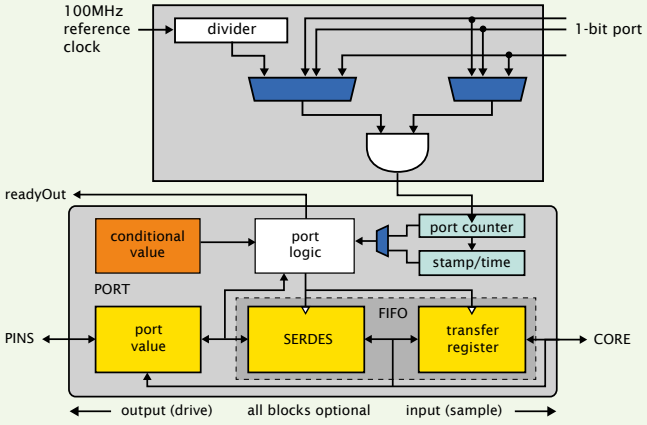


HARDWARE RESPONSE™ PORTS

The GPIO pins of the xCORE device are managed by port logic that can efficiently drive external pins high and low, and sample values.

Ports are available in different widths (1/4/8/16/32bit) depending on the device package. They are driven by clocks or timers, and data can be buffered, serialized and timestamped.

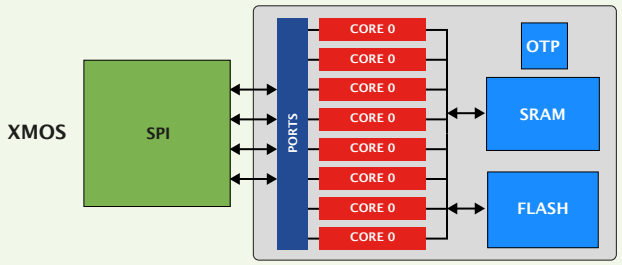
xCORE devices can communicate with fast, complex interfaces that would not be possible using standard bit-banging techniques required by other microcontrollers. As well as running real-time parallel applications, XMOS microcontrollers allow complex I/O protocols and combinations of peripherals to be implemented via the ports within a single device.

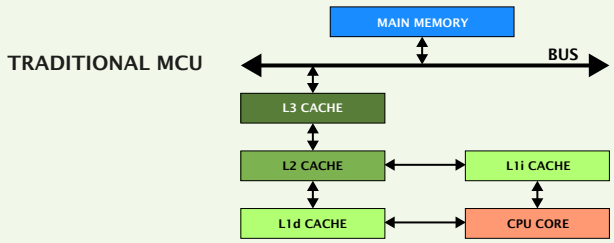


MEMORY SYSTEM

Many microcontrollers have a single complex memory system with caches to hide the latency of memory accesses. The uncertainty in the behavior of these caches contributes to the lack of predictability of the overall system. Rather than accelerate a slower memory subsystem with caches, the entire xCORE memory runs at the operating frequency of the processing pipeline and is entirely deterministic:

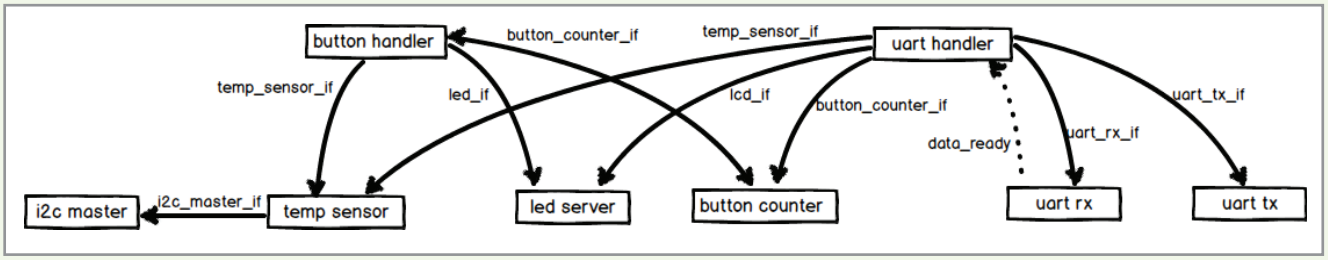
- Each tile contains local SRAM memory, which is shared between all cores on that tile for code and data.
- Each scheduled core has an allocated slot to access the memory in a single cycle;
- The xCORE memory will always respond within the allocated cycle
- Tasks communicate via inter-task communication channels over the xCONNECT switch, or shared memory.
- Tasks communicate explicitly with external FLASH/SDRAM memory using I/O ports, separate to the local tile memory; no additional memory manager is required.
- Each tile also has a block of one-time programmable memory for secure boot code and encryption keys.





PROGRAMMING MULTICORE APPLICATIONS

On xCORE devices, programs are composed of multiple tasks running in parallel. The concurrent tasks manage their own state and resources and interact by communicating with each other through xCONNECT. The ability of the xCORE architecture to run code independently on multiple cores allows tasks to be very responsive to events that occur in the system. Each task can be expressed entirely in ANSI C, and built into a system using multicore extensions. The compiler maps tasks onto the logical cores of the device under user direction in the code.



MULTICORE EXTENSIONS TO C

To help programmers access the real-time hardware features of the architecture, some easy-to-use, yet powerful, multicore language extensions for C have been added.

Software projects can mix C source files with or without the multicore extensions enabled. The xTIMEcomposer compiler automatically enables the extensions based on the file extension.

The C extensions includes features for:

- Task based parallelism
- Task communication
- Accurate timing and timestamping
- Ports and I/O
- Safe memory management

SOFTWARE DEVELOPMENT ENVIRONMENT

The hardware features are complemented by a software development environment, which makes it easy to define real-time tasks as a scalable parallel system. The xTIMEcomposer tools include fully standards-compliant C and C++ compilers plus the standard language libraries, an IDE, simulator, symbolic debugger, runtime instrumentation and trace libraries and a static code timing analyzer (XTA).

All of the components are aware of the real-time multicore nature of the programs, giving a fully integrated approach.

WHAT'S NEW IN xCORE-200

Dual issue

The XS1 architecture issues a maximum of 1 instruction per clock cycle, xCORE-200 can issue 2 instructions per clock cycle. This means that, for a given clock frequency, xCORE-200 has twice the peak instruction issue rate.

64-bit load and store

Each memory cycle can now load 64b of instruction or data, compared to 32b on the XS1 architecture.

High priority logical cores

xCORE-200 allows some cores to have a higher priority in the scheduler. Groups of high priority cores and low priority cores will be scheduled in a round robin. Determinism is not affected.

Kernel mode

A kernel mode added to enable traditional RTOS to be ported more comprehensively to an xCORE logical core.

More memory

Four times as much instruction/data memory is made available to xCORE-200 applications.

More instructions

As well as adding load and store instructions for the wider word width, there are also additional instructions to accelerate common compression, extraction, DSP and timing functions.