XMOS

AN03007: XCORE Ports - An Introduction

Publication Date: 2025/3/10
Document Number: XM-015272-AN v1.0.0

IN THIS DOCUMENT

The XMOS XCORE processor can interface electrical signals on a package pin to signals in the processor through a highly efficient low-latency interface called a **port**.

Ports are the interface between th physical and the logical worlds. Their structure is shown in Fig. 1. On the left side a port is connected to a physical GPIO wire that can drive a signal at some voltage, or that can sample a signal. On the right side of the diagram, a port is connected to the execution unit where those voltage levels are represented by logical values: zero, one, or multi-bit values.

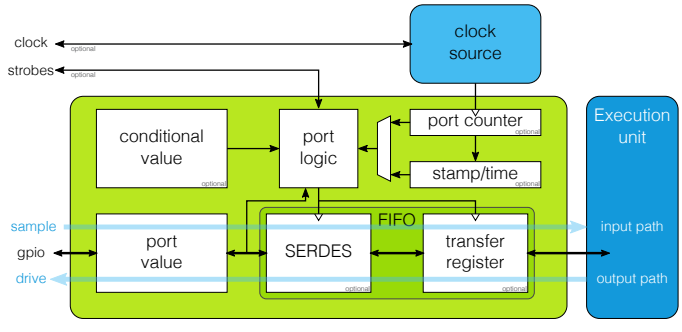

Fig. 1: Structure of a port

Between the GPIO pins and the execution unit is a set of components that can be used to serialise, deserialise, buffer, and clock data.

This document provides an overview of the capabilities of XCORE ports and the options available for system designers. It is intended to provide a starting point for understanding the capabilities of the XCORE ports, and provides links to more detailed information on the various aspects of port operation. The following application notes describe port operations in detail:

- AN03000: XCORE Input and Output
- AN03001: XCORE Clocked Input and Output
- AN03002: XCORE Port Buffering
- AN03003: XCORE Serialization and Strobing
- AN02039: Ports, Pins, and the XN file
- AN02041: Modifying the Electrical Characteristics of Pins and Ports

This document shows how these application notes relate to each other.

1 Ports and How to Programme Them

A port provides the interface between the processor and its environment. The port logic can drive its pins high or low, or it can sample the value on its pins, optionally waiting for a particular condition. Ports are not memory mapped; instead they are accessed using dedicated instructions. Each XCORE port operates autonomously as a small state machine that can provide deterministic, hardware-timed processing of signals, ensuring that the signals can be operated on in real-time with low latency.

- AN03000: XCORE Input and Output
  Ports operate independently to the instruction stream, but are directly accessible to programs running on the processor. It takes just a single instruction to, for example, output a value to a port (drive a signal) or input a value from a port (sample a signal). This is explained in detail in AN03000: XCORE Input and Output. This application note shows how to write programmes in the C language that sample buttons or drive LEDs. It also covers how input ports can be configured to wait for some condition to occur, e.g. a port is equal to a value. Using conditional inputs is much more power-efficient than the conventional approach of polling a pin in a loop, as it allows the processor to idle or perform other tasks while waiting for the condition to occur.
- AN02039: Ports, Pins, and the XN file
  To provide flexibility with small packages, the package pins are mapped onto ports through a *multiplexer*. The multiplexer also allows different uses of pins, including overlaying other physical functions such as an LPDDR interface. The structure of the multiplexer is explained in AN02039: Ports, Pins, and the XN file. That document also explains the naming structure of ports and pins, and how you can assign your own names for your design.
- AN03001: XCORE Clocked Input and Output
  An XCORE device does not have fixed hardware interfaces (such as I2C, SPI, etc.); instead, these are built using ports and software. Many interfaces rely on data to be *clocked*, meaning that one or more *data* wires are known to be stable relative to a specific transition on a *clock* wire. Clocked ports are described in AN03001: XCORE Clocked Input and Output. Clocked ports use a *clock-block* to capture an external clock or drive an internally generated clock, and we show how to connect *clock-blocks* and *ports* in that application note.
- AN03002: XCORE Port Buffering
  By default ports are tightly coupled to the instruction processing stream. The instruction processor and the pin will synchronise on input and output. This synchronisation may be too slow to implement fast interfaces (such as an Ethernet RMII) so for these we need *buffering*, *serialisation*, and *strobing*. These are three functions that the ports can perform autonomously. *Buffering* is the subject of AN03002: XCORE Port Buffering, and it introduces the notion of a decoupling register between a port and an instruction stream.
- AN03003: XCORE Serialization and Strobing
  The port itself can serialise or deserialise data (relative to a clock), and can only advance in the presence of external handshake signals. These two subjects are explained in AN03003: XCORE Serialization and Strobing.

By combining all these capabilities, ports can be configured to perform complex operations. For example, the signal shown in Fig. 2 can be deserialised and clocked in by ports functions alone. Looking at the four data wires RX_D0..RX_D3 we can see bit patterns 0b0101, 0b1101, 0b0011, 0b0000, 0b1111, 0b1100, and a port can read these in as four bytes 0xD5, 0x03, 0xCF, or it can be programmed to ignore all data up to the 0xD nibble and just input the last two bytes.

You can use these advanced port operations to implement a wide range of interface protocols, and process signals without the need for core processor resources.

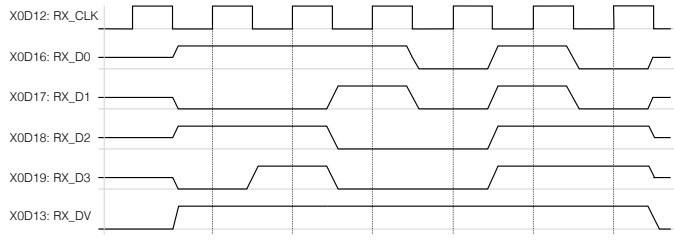

Fig. 2: Example signals that XCORE ports can input.

2 System Design Using Ports and Libraries

XCORE ports can implement interface protocols in software. To make this easier for system designer, XMOS provides a set of software libraries that implement standard interfaces such as I2C, SPI, I2S, UART, etc. These libraries can be integrated into your design to implement these interfaces quickly.

You can *instantiate* one or more of the interfaces; during the instantiation you will set some of the fixed parameters of the interface, including the ports that should be used for this interface. You can then simply call the interface functions on this interface in order to perform high level operations.

You can find a full list of the available libraries on the XMOS library website. Each library links to a landing page with documentation, source code, and application notes using the library. As the libraries are normally distributed in source code form, you can take a library and modify the code in order to implement a special requirement.

3 Example Applications Using Ports

To illustrate the use of ports, we have several example applications that show how to use ports to implement a range of functions. These application notes are available from the XMOS application note website and you can use these as starting points for your own designs. All examples below show how to use ports - they deliberately do not instantiate libraries so that they can better illustrate the fundamental port features.

The classic example in embedded programming is to make an LED flash. This is the subject matter of AN02036: Making an LED glow using a simple PDM modulator which explores various ways to make a LED flash or glow with different brightnesses.

The Universal Asynchronous Receiver/Transmitter (UART) is a simple serial interface that is used to communicate with other devices. AN03000: XCORE Input and Output ends with a tutorial example showing how to implement a simple UART interface using the XCORE port I/O instructions.

The last section of AN03001: XCORE Clocked Input and Output contains an example of how to drive an LCD screen using the XCORE ports. This example illustrates the use of clocked ports to drive a high speed parallel interface to a screen,

Many other examples are available from the XMOS website that show how to use ports to implement a wide range of interfaces. These examples can be used as a starting point for your own designs and can be modified to suit your requirements.