AN02039: Ports, Pins, and the XN file

Publication Date: 2025/2/25
Document Number: XM-015276-AN v1.0.1

IN THIS DOCUMENT

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

In this document we describe the mechanism that links ports to pins in detail, and explain how to specify the desired pin-out of the device and how to select an appropriate device for your problem.

This document ties together the tools documentation (on the nature of XN files), the datasheets (that documents pins), and the port programming guides.

1 Introduction

The two important terms to distinguish are:

Pins

Pins are electrical connections on the XCORE package, Pins can typically be low (0V) or high (VDDIO)

Ports

A port is a part of the XCORE that provides an abstraction of a *pin*. A port maps electrical signals to logical levels, and provides elements for, for example, serialising data. A 0 value in the port corresponds to a low signal on the pin.

These two terms are described in more detail below, followed by a description of the mechanism that links ports to pins, and then how to specify the desired pin-out of the device.

2 Pins

Each tile on an XCORE device can support up to 64 I/O pins, so up to 128 I/O pins on a standard dual-tile device. The number of I/O pins available on a specific device depends on the package. For example, on xcore.ai devices the number of I/O pins is as follows

- QF60 package - 34 I/O pins
- TQ128 package - 78 I/O pins
- FB265 package - 128 I/O pins

Depending on the package, these pins may have different VDDIO voltages when driven high.

The electrical characteristics of the pins are defined in the device's datasheet, but in summary the main characteristics that can be changed under software control include:

- Direction (input or output)
- As Output - Drive strength (e.g. 2, 4, 8, 12 mA) - Slew rate control
- As Input - Schmitt trigger enable - Pull-up enable - Pull-down enable

If a pin is not configured it operates as an input with a weak pull-down resistor enabled.

These modes are set by the software with the details described in the programming guides.

Pins are identified by a label of the form `XnDmm` where `n` is the tile number, and `mm` is the pin number.

For example, X0D12 is pin 12 on tile 0, and X1D24 is pin 24 on tile 1.

This numbering scheme is consistent across all devices in the product family. However, not all pins are available on all packages and the datasheet for the individual device provides a full list of the pins available along with the physical location of these pins on the package.

3 Ports

A port is a logical abstraction of an I/O pin, or a group of I/O pins. Ports provide the interface between the XCORE processor logic and the outside world.

3.1 Basic operation

At the basic level ports provide a mechanism to input or output logical values on the device pins, and they can be configured to represent a range of different types of signals.

XCORE ports can be configured to either 1, 4, 8, 16 or 32 bits wide, and can operate as inputs, outputs, or bidirectional signals.

Each port is identified by its width in bits and a letter, with the letter distinguishing between ports of the same width, as is shown in Table 1.

Table 1: Available Ports on an XCORE Device

Port Size	Label Example	Description	Available Ports
1-bit	1A, 1B, ..., 1P	Single-bit digital I/O	16
4-bit	4A, 4B, ..., 4E	4-bit parallel I/O	6
8-bit	8A, 8B, ..., 8D	8-bit parallel I/O	4
16-bit	16A, 16B	16-bit parallel I/O	2
32-bit	32A	32-bit parallel I/O	1

Within each port the individual bits are labelled from 0 to the width of the port. For example:

- Port 4A has bits 0, 1, 2, 3 which are identified as 4A0, 4A1, 4A2, 4A3
- Port 8B has bits 0, 1, ..., 6, 7 which are identified as 8B0, 8B1, ..., 8B6, 8B7

Architecturally, ports are typically referred to by a symbolic name and are labelled as `XS1_PORT_xy` where `xy` is the port identifier from the table above (e.g. *XS1_PORT_8B* for the second 8-bit port). These names can be used anywhere in C programs and Assembly programs provided you include `xs1.h`.

3.2 Advanced Port Operations

XCORE ports are, however, much more powerful than simple digital I/O pins. Each XCORE port can operate as a small state machine that provides deterministic, hardware timed, parallel processing of signals, ensuring that the signals are processed in real-time with low latency.

Ports can be configured to perform more complex operations such as:

- Serialisation and deserialisation of data
- Clocking data in and out
- Reading and writing data in a single clock cycle
- Strobing data
- Buffering data
- Triggering events when data is available

These advanced port operations can be used to implement a wide range of interface protocols, and they operate without the need for core processor resoureces.

These advanced port operations are described in more detail in the port application notes and the tools programming guide.

4 Linking Ports to Pins

The reason to separate ports and pins is that multiple ports (and possibly other signals) may map onto a single pin. For example, on xcore.ai, pin `X0D31` is connected to port 4F (bit 3), port 8C (bit 5), port 16B (bit 5), and the LPPDR interface (DQ3). The `0` in `X0D31` means that all ports it is connected to are on Tile 0, the `31` is just a label that makes each pin unique.

For each XMOS product family, the mapping between pins on the one hand and ports and other functions on the other hand is the same for every member of the family. That is, `X0D31` will have the same mapping on all packages. However, not all packages may make `X0D31` available as a physically accessable pin. The largest package brings out all pins, the smallest package brings out only a small subset of pins.

5 Which Ports to Use

We now look at how to select the appropriate port for a particular task. The design decisions of which port to use will depend on the both the nature of the signal and the application requirements.

The following guidelines can be used to help select the appropriate type of port.

Within a tile, all 1-bit ports are interchangeable, all 4-bit ports are interchangeable, all 8-bit ports are interchangeable, and both 16-bit ports are interchangeable. That means, if you need a 1-bit port you can pick any of them; there is no preference for a particular port. The choice whether to pick a 1-bit or 16-bit port depends on the signal that the port carries:

- Clock Signals that you need to clock data in and out must be on a 1-bit port, and so must strobe (data-valid) signals.
- Data signals of serial protocols are typically on 1-bit ports as that enables you to let the port logic to do the serialisation and deserialisation.
- Data signals on an N-bit bus should be on an N-bit port, as that enables you to input and output data to the bus in one synchronised operation, and/or to serialise wider data onto a narrower bus.
- Slow signals (LEDs, buttons, reset signals etc.) can be on any port. However, you need to make sure that all signals on a port are either all driven (outputs) or all sampled (inputs). Ports cannot do a bit of both.

- Signal groups that belong to one interface should be using ports on the same tile.

For example, if an Ethernet MII Rx signal has a clock, 4-bit data, data-valid, and error signal then you would use one 4-bit port and three 1-bit ports; all on the same tile.

If you also need an MII Tx signal with a similar set-up, you need another 4-bit port and three more 1-bit ports all on one tile. There may be value to keep the Rx and Tx part on the same tile too - that is an application decision.

6 The port multiplexer

From Table 1 we see that there are a total of 136 signals on one tile. That is too many signals since each tile has only 64 GPIO pins available. To resolve this, some of the ports are *multiplexed* (muxed) onto the same pin.

Each tile has its own multiplexer. For example, `XnD18` is connected to bit 2 of port 4D, bit 4 of port 8B, and bit 12 of port 16A. These are visualised for the QF60A/B package in Fig. 1 where we have drawn the multiplexers for `X0D07` (tile 0) and `X1D18` (tile 1).

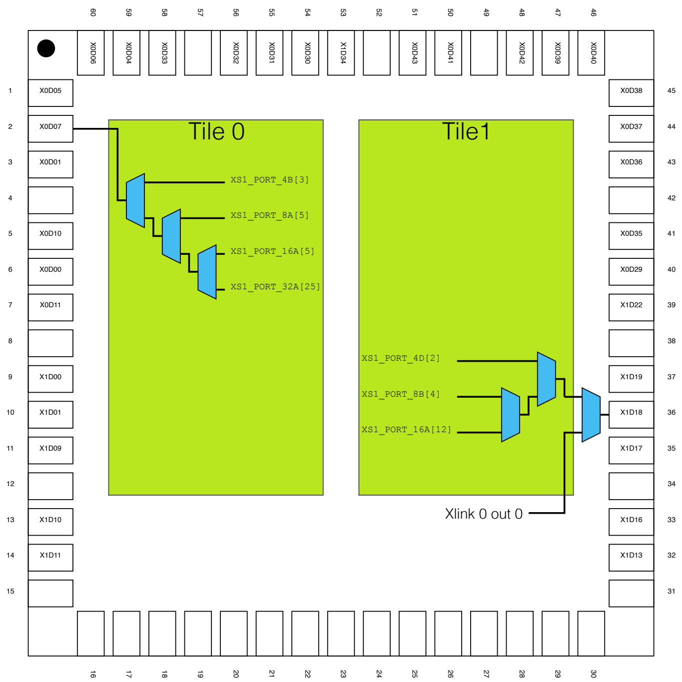

Fig. 1: Visualisation of the multiplexer in a QF60A/B package

The full structure of the multiplexed port signals on a single tile is shown in the table below:

Nr	1-bit Port	4-bit Port	8-bit Port	16-bit Port	32-bit Port	Pin Label
1	1A					XnD00
2	1B					XnD01
3		4A0	8A0	16A0	32A20	XnD02
4		4A1	8A1	16A1	32A21	XnD03

continues on next page

Table 2 – continued from previous page

Nr	1-bit Port	4-bit Port	8-bit Port	16-bit Port	32-bit Port	Pin Label
5		4B0	8A2	16A2	32A22	XnD04
6		4B1	8A3	16A3	32A23	XnD05
7		4B2	8A4	16A4	32A24	XnD06
8		4B3	8A5	16A5	32A25	XnD07
9		4A2	8A6	16A6	32A26	XnD08
10		4A3	8A7	16A7	32A27	XnD09
11	1C					XnD10
12	1D					XnD11
13	1E					XnD12
14	1F					XnD13
15		4C0	8B0	16A8	32A28	XnD14
16		4C1	8B1	16A9	32A29	XnD15
17		4D0	8B2	16A10	32A30	XnD16
18		4D1	8B3	16A11	32A31	XnD17
19		4D2	8B4	16A12		XnD18
20		4D3	8B5	16A13		XnD19
21		4C2	8B6	16A14		XnD20
22		4C3	8B7	16A15		XnD21
23	1G					XnD22
24	1H					XnD23
25	1I					XnD24
26	1J					XnD25
27		4E0	8C0	16B0		XnD26
28		4E1	8C1	16B1		XnD27
29		4F0	8C2	16B2		XnD28
30		4F1	8C3	16B3		XnD29
31		4F2	8C4	16B4		XnD30
32		4F3	8C5	16B5		XnD31
33		4E2	8C6	16B6		XnD32
34		4E3	8C7	16B7		XnD33
35	1K					XnD34
36	1L					XnD35
37	1M		8D0	16B8		XnD36
38	1N		8D1	16B9		XnD37
39	1O		8D2	16B10		XnD38
40	1P		8D3	16B11		XnD39
41			8D4	16B12		XnD40
42			8D5	16B13		XnD41
43			8D6	16B14		XnD42
44			8D7	16B15		XnD43
45					32A00	XnD49
46					32A01	XnD50
47					32A02	XnD51

continues on next page

Table 2 – continued from previous page

Nr	1-bit Port	4-bit Port	8-bit Port	16-bit Port	32-bit Port	Pin Label
48					32A03	XnD52
49					32A04	XnD53
50					32A05	XnD54
51					32A06	XnD55
52					32A07	XnD56
53					32A08	XnD57
54					32A09	XnD58
55					32A10	XnD61
56					32A11	XnD62
57					32A12	XnD63
58					32A13	XnD64
59					32A14	XnD65
60					32A15	XnD66
61					32A16	XnD67
62					32A17	XnD68
63					32A18	XnD69
64					32A19	XnD70

The eighth line tells us that bit 3 of port 4B is multiplexed on bit 5 of port 8A, bit 5 of port 16A, and bit 25 of port 32A; and it is called `XnD07`, where `n` is `0` or `1` depending on the tile that the ports are on.

You notice that the mux structure is designed so that the narrowest port takes precedence. That is, if you use port 8A to drive data, then pins 16A[0..7] are not driven out. If you also drive port 4D, then that will knock out pins 16A[10..13] too. If you do chose to drive a signal on 16A it will only show on pins XnD16, XnD17, XnD20, and XnD21. On inputting data, all data goes to all pins; but it will only make sense to sample them on the ports for which the data was intended.

In addition to the port multiplexer that governs GPIO pins there may be another second multiplexer on the edge of the chip that can multiplex XLINK signals (the communications network), LPDDR-1 (extended memory), and the application PLL onto the GPIO pins. In the example in Fig. 1 `X1D18` has an XLINK signal multiplexed on it.

All port muxes are set automatically when a port is enabled. When you enable an XLINK, LPPDR, or the application PLL, it will automatically mux the pin to those, taking precedence over any port(s) out that may be mapped to the same pin. The precise mapping depends on the device family, and it is shown in the datasheet for the particular product.

Finally, the MIPI PHY and USB PHY are muxed in to a selection of ports. The USB PHY is hard-wired to ports 8A and 8B, and ports 1E, 1F, 1H, 1I, 1J, and 1K. If you enable the USB PHY these ports should not be used by application code. You can still use ports 4A, 4B, 4C, and 4D; despite them being multiplexed with 8A and 8B, they are bypassed for the USB PHY. So six 1-bit ports are taken over by the USB PHY when enabled. Similarly, the MIPI PHY uses ports 8A, 1E, 1I, and 1O. The muxing structures are shown in the datasheet. Note that the USB and MIPI ports are only being muxed if the USB PHY and/or MIPI PHY are enabled.

7 The port map

To summarise, in order to pick the appropriate IO pins we have seen three constraints:

- Pick the right port-width for the particular IO task

- Pick the right tile to colocate IO pins (informed by the software stack)
- Pick a port that is not muxed on a pin that is already in use

If there is a choice, you may want to pick a port that is in a convenient location on the chip. For example, you will note that the QSPI pins are all located together on the top left-hand corner of the chip.

A tool that can help in solving these constraints is the *port-map*. The port-map for a package is a spreadsheet that lists the pins available on that package, the multiplex structure, pins potentially occupied by USB and MIPI etc. The portmap is available from the landing page of the particular product, and are linked below:

- xcore.ai QF60A/B port map
- xcore.ai TQ128 port map
- xcore.ai FB265 port map

An excerpt of the portmap of the QF60 packages is shown in Fig. 2. The left-most columns show the port multiplexer, then there is a list of internal pin functions, then the IO rail, the pin name, and the pin number on the particular package. Ports that may be unavailable are coloured green, yellow, and blue.

xcore.ai portmap

xcore.ai PORT MAP

Key: Port unavailable if USB enabled on that tile
Port unavailable if MIPI enabled
Port unavailable if MIPI or USB enabled

Note 1: I/O rail supply column only applies in packages where VDDIO is split into L, R and T. Please refer to the product datasheet.

	Port					Alternate Pin Function			Internal Port Options					Features	Pin name		
	1b	4b	8b	16b	32b	XMOS Links	App PLL	LPDDR	QSPI Master	SPI Master	SPI Slave	MIPI	USB	I/O rail		QF60A	BOARD SIG
Tile 0	P1A0					xlink4_rx3				MISO	SS			IOL	X0D00	6	
	P1B0								SS	SS				IOL	X0D01	3	QSPI_CS_N
		P4B0	P8A2	P16A2	P32A22				IO0			RXD2	TXD2	IOL	X0D04	59	QSPI_D0
		P4B1	P8A3	P16A3	P32A23				IO1			RXD3	TXD3	IOL	X0D05	1	QSPI_D1
		P4B2	P8A4	P16A4	P32A24				IO2			RXD4	TXD4	IOL	X0D06	60	QSPI_D2
		P4B3	P8A5	P16A5	P32A25				IO3			RXD5	TXD5	IOL	X0D07	2	QSPI_D3
	P1C0					xlink4_rx4			SCLK	SCLK	SCLK			IOL	X0D10	5	QSPI_CLK
	P1D0					xlink4_rx2				MOSI	MOSI			IOL	X0D11	7	
		P4F1	P8C3	P16B3		xlink3_tx0								IOR	X0D29	40	
		P4F2	P8C4	P16B4				DQ4						IOT	X0D30	54	
		P4F3	P8C5	P16B5				DQ3						IOT	X0D31	55	
		P4E2	P8C6	P16B6				DQ2						IOT	X0D32	56	
		P4E3	P8C7	P16B7				DQ1						IOT	X0D33	58	
	P1L0					xlink3_tx1								IOR	X0D35	41	
	P1M0		P8D0	P16B8		xlink3_tx2								IOR	X0D36	43	
	P1N0		P8D1	P16B9		xlink3_tx3								IOR	X0D37	44	
	P1O0		P8D2	P16B10		xlink3_tx4						CLK		IOR	X0D38	45	
	P1P0		P8D3	P16B11				A13						IOT	X0D39	47	
			P8D4	P16B12				A12						IOT	X0D40	46	
			P8D5	P16B13				A11						IOT	X0D41	50	
			P8D6	P16B14				A10						IOT	X0D42	48	
			P8D7	P16B15				A9						IOT	X0D43	51	

Fig. 2: Excerpt from QF60 portmap

By filling in the final column on the right we can assign each IO function a pin and port. In this case, we have assigned `QSPI_CS_N`, `QSPI_CLK`, `QSPI_D0`, `QSPI_D1`, `QSPI_D2`, and `QSPI_D3`; the four signals required for QSPI.

8 The XN-file

An XMOS `.xn` file is an XML-based description that is used by the XTC tools to define system configurations, hardware setups, and interconnections between different components.

These files are essential when developing applications as they specify the configuration of the target processor, including memory, clocks, boot mode and the port mapping. The XTC tools use the data in the `.xn` file to generate the header files required to compile and link a firmware application.

The `.xn` file is the place where we can collect all the port-data. We can assign names to ports, and the tools create a `platform.h` file that we can include in our program to use abstract port name.

For example, we can include the following lines in our `target_board.xn` file:

```
            <Port Location="XS1_PORT_1B"  Name="PORT_SQI_CS"/>
            <Port Location="XS1_PORT_1C"  Name="PORT_SQI_SCLK"/>
            <Port Location="XS1_PORT_4B"  Name="PORT_SQI_SIO"/>

            <Port Location="XS1_PORT_4C"  Name="PORT_LEDS"/>
            <Port Location="XS1_PORT_4D"  Name="PORT_BUTTONS"/>
```

By including `platform.h` into our main program we can now write:

```
#include <xcore/port.h>
#include <platform.h>

port_t leds = PORT_LEDS;
port_t buttons = PORT_BUTTONS;

int main() {
    while(1) {
        int b = port_in(buttons);
        port_out(leds, b);
    }
}
```

And use the variable `leds` to refer to `PORT_LEDS` to refer to `XS1_PORT_4C` to refer to pins `X0D14`, `X0D15`, `X0D20` and `X0D21`, to refer to balls *D4*, *D3*, *F1* and *G2* on the FB265 package.

This mechanism allows us to write code that is independent of the actual pin numbers, and allows us to easily change the the target package by changing the `.xn` file without changing the code.

A full description of XN files can be found in the XTC tools documentation.