Introduction to XS1 ports

IN THIS DOCUMENT

1 Introduction

A *port* connects an xCORE tile to one or more physical pins and as such defines the interface between hardware attached to an xCORE multicore microcontroller and software running on the xCORE device. 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 accessed using dedicated instructions.

Data is transferred between the pins and core using a FIFO that comprises a SERDES and transfer register, providing options for serialization and buffered data.

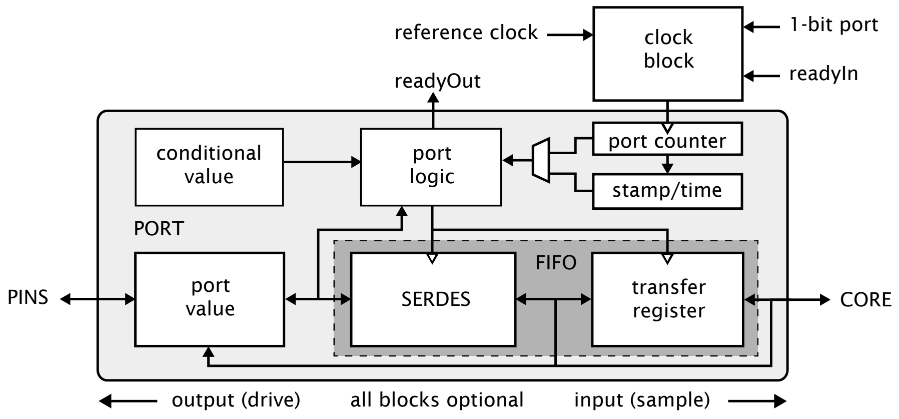

Figure 1: Port block diagram

1.1 Port counters

Each port has a counter that can be used to control the time at which data is transferred between the port value and transfer register. Port counters are 16-bit and tick once for each cycle of the clock input, incrementing on the falling edge of the signal. The counter values can be obtained at any time to find out when data was obtained, or used to delay I/O until some time in the future.

The port counter value is automatically stored in the timestamp register when data is moved into or out of the transfer register. The timestamped value is available to the application programmatically providing precise control of response times.

As the port counter counts in the application clock domain, it is only a measure of time if the input clock is a signal that has clock edges at regular intervals. If the clock is irregular the port counter is not a measure of time.

1.2 Clock blocks

Many I/O operations need data to be sampled and driven on specific edges of a clock. xCORE devices include a set of programmable clocks called *clock blocks* that can be used to govern the rate at which ports execute. Each xCORE tile has six clock blocks: XS1_CLKBLK_REF is the tile reference clock and runs at a default frequency of 100MHz; XS1_CLKBLK_1 to 5 can be set to run at different frequencies.

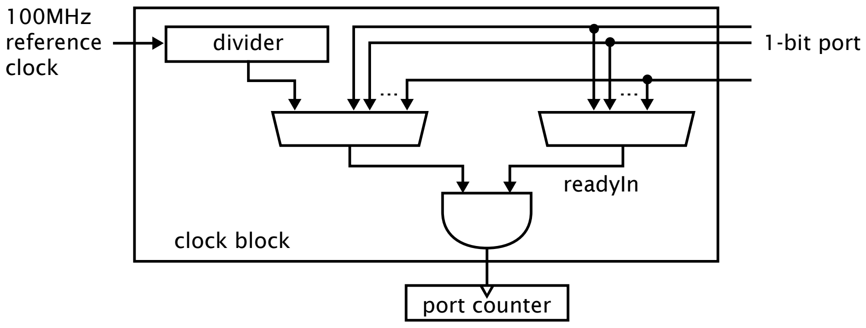

Figure 2: Clock block diagram

A clock block can use a 1-bit port as its clock source allowing external application clocks to be used to drive the input and output interfaces.

A clock block may use the reference clock as its clock source and divide the tile reference frequency using an 8-bit divider. When this is set to 0, the reference clock passes directly to the output. The falling edge of the clock is used to perform the division. Hence a setting of 1 results in an output from the clock which changes each falling edge of the input, halving the input frequency f ; and a setting of n produces an output frequency of *f/2n*.

1.3 Strobe signals

In many cases I/O signals are accompanied by *strobing* signals. The xCORE ports can input and interpret strobe (known as *readyIn* and *readyOut*) signals generated by external sources, and ports can generate strobe signals to accompany output data.

See Section §5 for further information on strobe signals.

2 Unbuffered data transfer

When a port is used in unbuffered mode, the signalling operation of the pins is synchronized with the core instruction execution.

2.1 Clocked ports

Data input or output on a clocked unbuffered port is defined as follows:

- An *output* causes data to be driven on the next falling edge of the port's clock. The data output changes state synchronously with the port clock.
- An *input* causes data to be sampled by the port on the next rising edge of the port's clock. The data input is sampled synchronously with the port clock.

2.2 Timed ports

If an input or output port is configured to wait for a port counter, the port waits until the counter has reached the specified value and then moved the data to the transfer register.

If the port is used for an output operation and the transfer register is full, the transfer is halted until the transfer register is empty to make sure that the port time is not changed until the pending output is completed.

2.3 Timestamped ports

The value of the port counter at which data is transferred in or out of the transfer register is captured in the timestamp register.

2.4 Conditional input

A condition can be set to an input port, which causes the port to wait until the specified condition is met, and then behave like a clocked input. The conditions include:

- pins equal (`pinseq`): the value on the pins must equal the specified value
- pins not equal (`pinsneq`): the value on the pins must not equal the specified value

If a condition is set, the port compares the value from the pins with the conditional value. If the condition is met, a timestamp is set and the port becomes ready for input.

If both timing and conditions are used, the port waits for the specified value on the port counter, and then for the specified condition to be met, before data is input.

Conditions cannot be applied to output ports.

3 Buffered data transfer

Instead of clocked ports passing data directly between pins and core, ports can be buffered so that data is held in the FIFO until the core is ready to input or output data, allowing the core to execute other instructions during this time. Using buffers, a single logical core can perform I/O operations on multiple ports in parallel.

Ports used as inputs to clock blocks and strobe signals cannot be buffered.

3.1 Buffered input

The behavior of data input on a buffered port is defined as follows:

- On each rising edge, data is transferred from the FIFO to the transfer register starting with the least significant bit (nibble, byte or 16-bit entity). If the transfer register is full, data is discarded to make room for the most recently sampled value.
- An input reads the next data from the transfer register; the core blocks until the transfer register contains data.
- The time in a timed input represents the time in the future when input will start; it causes the core to discard any data in the buffer prior to performing the input.
- A conditional input moves the data into the transfer register on the rising edge of the clock when the condition is met. It then clears the condition.
- The value of the port counter when data is transferred into the transfer register is recorded in the timestamp register.
- If several buffered input ports are driven from the same clock they appear to operate as a single input port provided that the core is able to take the data from all of them during each clock cycle.

3.2 Buffered output

The behavior of data output on a buffered port is defined as follows:

- The port transfers data from the transfer register on each falling clock edge.
- The core is blocked if it tries to output while the transfer register is full. If the FIFO becomes empty the output port will continue to drive its last value.

- A timed output causes the port to wait until the specified time and then transfer data from the transfer register to the FIFO.
- The value of the port counter is recorded in the *timestamp* register when the data is moved from the transfer register.
- Conditional buffered output is not available.
- If several buffered output ports are driven from the same clock block, they appear to operate as a single output port, provided that the core is able to supply new data to all of them during each clock cycle.

4 Serialized data transfer

The SERDES can be used to serialize (output) or deserialize (input) data, reducing the number of instructions required to input or output data. The number of bits in the transfer register and the SERDES determine the width of the transfers (the transfer width) between the core and the port; this is a multiple of the port width (the number of pins).

For a FIFO that holds at most *bpw* bits and a *w*-bit wide port, the size of the FIFO is limited to *bpw/w* elements. The most significant *w* bits of the FIFO are the value that is most recently clocked in. The lengths of the FIFO that are supported are:

Port width	Transfer width
1-bit	4, 8, 32
4-bit	8 (2 elements), 32 (8 elements)
8-bit	32 (4 elements)
16-bit	32 (2 elements)

Serialization and deserialization always shift right. If you need to start with the most significant bit you can use a single cycle bit-reverse operation for a 1-bit port or a byte reverse for an 8-bit port.

4.1 Serialized buffered input

The behavior of data input on a serialized buffered port is defined as follows:

- Port value is shifted right into the most significant bits of the SERDES on the rising edge.
- When the SERDES is full, the data is moved to the transfer register.
- If full, the transfer register is overwritten with the latest value.
- For timed input, data in the SERDES is moved to the transfer register when the port counter matches the specified time, whether full or not

- For conditional input, the entire SERDES is moved to the transfer register when the condition is matched. The matching value is in the most significant bits; the condition is cleared so the next value will contain the next *n* values.

4.2 Serialized buffered output

The behavior of data output by a serialized buffered port is defined as follows:

- The least significant bits of the SERDES are driven to the pins.
- If the SERDES is empty it is filled on the next falling edge with the transfer register value.
- While the SERDES is not empty, data is shifted right on each falling edge.
- If both the transfer register and SERDES are empty, the last value is held.
- For timed serialization, the transfer register is transferred to the SERDES when the port counter matches the specified time.
- No conditional serialized buffered output is available.

5 Strobing

The xCORE architecture provides support for strobing, where data is accompanied by a separate data valid signal. Up to two pins can be used to strobe data (readyIn and readyOut signals) providing four strobe modes; the readyIn strobe gates the clock to the port and the readyOut strobe is high anytime that the port can progress.

Implicit strobing	Data is transferred every clock cycle. Default setting.
Slave strobing	readyIn strobe is used. Data is only input or output when the input strobe is high and a clock is present.
Master strobing	readyOut strobe is used. For an output, the port signal is driven high if there is data in transfer register. For an input port, the signal is driven high if there is room in transfer register.
Bidirectional strobing	Both input and output strobes are used. The readyIn signal controls the data and the readyOut signals the state.

The behavior of strobe signals for conditional and timed input and output operations is defined below:

- During a conditional input on a buffered port, the readyOut signal is kept high while the condition does not match; any non matching input is discarded. The readyOut signal is kept high for as long as there is space in the transfer register.

- During a timed input on a buffered port, the readyOut signal is kept low until the port counter reaches the specified value; at that stage the readyOut signal is pulled high, requesting data until the transfer register is filled up.
- During a timed output on a buffered port, the readyOut signal is kept low until the port counter reaches the specified value; at that stage the readyOut signal is pulled high, outputting data until the transfer register is empty.

Ready-in strobes are attached to the clock block; so multiple ports can share a single readyIn signal through one clock block.

Ports used for readyIn and readyOut signals cannot be timed, buffered or serialized.

6 Bidirectional ports

If a port is programmed in bidirectional mode, it tri-states when an input is made, and starts to drive when an output is made.

In the case of a timed input to output, a change in tri-state or drive mode is delayed until the port counter reaches the specified value.

When a conditional input is made, the input pins are tristated the input.

No buffering is available in bidirectional mode.

7 Hardware port pin-out

Each xCORE tile exposes a combination of 1,4, 8, 16 and 32-bit ports, depending on the package and resources used internally. The total number of possible GPIO far exceeds the number of pins actually bonded out, so ports and xCONNECT Links are *multiplexed*, and there is a defined *precedence* when overlapping ports and links exist.

In cases where only a few pins from a wide port are available, these pins can be used as standard GPIO pins but the full width of the underlying port is not available.

7.1 Port precedence

The mapping of ports to pins is shown in Figure 3. The table lists for each pin which ports and links *can be connected* to it. Links and ports on the left hand side of the table have precedence over ports on the right hand side of the table. Each port is identified by its width (the first number 1, 4, 8, 16, or 32) and a letter that distinguishes multiple ports of the same width (A-P). The bits of the port are identified with a superscripted digit 0-31. Links are identified by means of a single letter identifier A-D. The wires of a link are identified by means of a superscripted digit 0-4.

The port or xCONNECT Link that is actually connected to a pin is determined by the program running on the xCORE device. For each xCORE tile, software can enable ports and links as required:

- If a link is enabled, then this link has access to the pins; the pins of any underlying ports are disabled.
- If a port is enabled then it overrules any ports with higher widths that share its pins.

For example, suppose that software on tile *n*, enables link A in 5-wire mode and ports 32A, 4C, and 8B. In that case:

- XnD01 - XnD10 will be connected to link A.
- XnD14, XnD15, XnD20, XnD21 will be connected to port 4C.
- XnD16 - XnD19 will be connected to bits 2 to 5 of port 8B.
- XnD49 - XnD70 will be connected to bits 0 to 19 of port 32A.

Generally, the system designer will be ensure that there is no overlap, but the precedence has been designed so that, if required, portions of the wider ports can still be used when overlapping narrower ports are used.

Where wide ports are fully wired out but a narrower port is used by the same pin (for example P4B and P32A: bits 22 to 25) the bits on the wider port must be masked after an input operation.

7.2 Banks

Figure 2 is divided in six parts which are six *banks*. Different packaging options export different numbers of banks. The first few banks have a selection of 1, 4, and 8 bit ports, and a link each. Banks further down incorporate port 32. On small packages the 32-bit port is not available.

8 Port identifiers

Each port is, architecturally, represented with a *bpw*-bit identifier, called a resource identifier. The least significant byte of a port-resource-identifier is *0* (identifying this as a port as opposed to for example a channel or timer), the next bytes identifies the port and the width of the port. A full list of ports is given in Figure 3. Note that in almost all cases one can use `XS1_PORT_1E` rather than `0x10600` since the include file `xs1.h` contains all mappings.

9 Port and clock configuration

Ports must be named by binding them to an xC identifier, and declared as buffered or serialized, before they can be used in the application code for I/O. Functions for configuring ports are provided in the `xs1.h` header file - see Figure 5.

You can configure clock blocks to either generate a clock based on a divided reference clock or use an input pin as a clock. When you have configured the clock, you must explicitly start it in the application. If you start the clock after all ports linked to the clock are configured, all the ports will have the same port counter values. Functions for configuring clocks are provided in the `xs1.h` header file - see Figure 6.

Ports have some modes, such as inverting, pull-up and pad delay, that must be set before the port is configured. Functions for configuring the advanced port modes are provided in the `xs1.h` header file - see Figure 7.

Port operations can be controlled by a set of predicate functions, such as equal and not-equal. The predicate functions are provided in the `xs1.h` header file - see Figure 8.

	⇐ highest		Precedence			lowest ⇒
Pin	link	1-bit ports	4-bit ports	8-bit ports	16-bit ports	32-bit port
XnD00		$1A$				
XnD01	A^4 out	$1B$				
XnD02	A^3 out		$4A^0$	$8A^0$	$16A^0$	$32A^{20}$
XnD03	A^2 out		$4A^1$	$8A^1$	$16A^1$	$32A^{21}$
XnD04	A^1 out		$4B^0$	$8A^2$	$16A^2$	$32A^{22}$
XnD05	A^0 out		$4B^1$	$8A^3$	$16A^3$	$32A^{23}$
XnD06	A^0 in		$4B^2$	$8A^4$	$16A^4$	$32A^{24}$
XnD07	A^1 in		$4B^3$	$8A^5$	$16A^5$	$32A^{25}$
XnD08	A^2 in		$4A^2$	$8A^6$	$16A^6$	$32A^{26}$
XnD09	A^3 in		$4A^3$	$8A^7$	$16A^7$	$32A^{27}$
XnD10	A^4 in	$1C$				
XnD11		$1D$				
XnD12		$1E$				
XnD13	B^4 out	$1F$				
XnD14	B^3 out		$4C^0$	$8B^0$	$16A^8$	$32A^{28}$
XnD15	B^2 out		$4C^1$	$8B^1$	$16A^9$	$32A^{29}$
XnD16	B^1 out		$4D^0$	$8B^2$	$16A^{10}$	
XnD17	B^0 out		$4D^1$	$8B^3$	$16A^{11}$	
XnD18	B^0 in		$4D^2$	$8B^4$	$16A^{12}$	
XnD19	B^1 in		$4D^3$	$8B^5$	$16A^{13}$	
XnD20	B^2 in		$4C^2$	$8B^6$	$16A^{14}$	$32A^{30}$
XnD21	B^3 in		$4C^3$	$8B^7$	$16A^{15}$	$32A^{31}$
XnD22	B^4 in	$1G$				
XnD23		$1H$				
XnD24		$1I$				
XnD25		$1J$				
XnD26			$4E^0$	$8C^0$	$16B^0$	
XnD27			$4E^1$	$8C^1$	$16B^1$	
XnD28			$4F^0$	$8C^2$	$16B^2$	
XnD29			$4F^1$	$8C^3$	$16B^3$	
XnD30			$4F^2$	$8C^4$	$16B^4$	
XnD31			$4F^3$	$8C^5$	$16B^5$	
XnD32			$4E^2$	$8C^6$	$16B^6$	
XnD33			$4E^3$	$8C^7$	$16B^7$	
XnD34		$1K$				
XnD35		$1L$				
XnD36		$1M$		$8D^0$	$16B^8$	
XnD37		$1N$		$8D^1$	$16B^9$	
XnD38		$1O$		$8D^2$	$16B^{10}$	
XnD39		$1P$		$8D^3$	$16B^{11}$	
XnD40				$8D^4$	$16B^{12}$	
XnD41				$8D^5$	$16B^{13}$	
XnD42				$8D^6$	$16B^{14}$	
XnD43				$8D^7$	$16B^{15}$	
XnD49	C^4 out					$32A^0$
XnD50	C^3 out					$32A^1$
XnD51	C^2 out					$32A^2$
XnD52	C^1 out					$32A^3$
XnD53	C^0 out					$32A^4$
XnD54	C^0 in					$32A^5$
XnD55	C^1 in					$32A^6$
XnD56	C^2 in					$32A^7$
XnD57	C^3 in					$32A^8$
XnD58	C^4 in					$32A^9$
XnD61	D^4 out					$32A^{10}$
XnD62	D^3 out					$32A^{11}$
XnD63	D^2 out					$32A^{12}$
XnD64	D^1 out					$32A^{13}$
XnD65	D^0 out					$32A^{14}$
XnD66	D^0 in					$32A^{15}$
XnD67	D^1 in					$32A^{16}$
XnD68	D^2 in					$32A^{17}$
XnD69	D^3 in					$32A^{18}$
XnD70	D^4 in					$32A^{19}$

Figure 3: Available links and ports for each pin in order of decreasing precedence

Port name	Resource identifier
XS1_PORT_32A	0x200000
XS1_PORT_16A	0x100000
XS1_PORT_16B	0x100100
XS1_PORT_8A	0x80000
XS1_PORT_8B	0x80100
XS1_PORT_8C	0x80200
XS1_PORT_8D	0x80300
XS1_PORT_4A	0x40000
XS1_PORT_4B	0x40100
XS1_PORT_4C	0x40200
XS1_PORT_4D	0x40300
XS1_PORT_4E	0x40400
XS1_PORT_4F	0x40500
XS1_PORT_1A	0x10200
XS1_PORT_1B	0x10000
XS1_PORT_1C	0x10100
XS1_PORT_1D	0x10300
XS1_PORT_1E	0x10600
XS1_PORT_1F	0x10400
XS1_PORT_1G	0x10500
XS1_PORT_1H	0x10700
XS1_PORT_1I	0x10a00
XS1_PORT_1J	0x10800
XS1_PORT_1K	0x10900
XS1_PORT_1L	0x10b00
XS1_PORT_1M	0x10c00
XS1_PORT_1N	0x10d00
XS1_PORT_1O	0x10e00
XS1_PORT_1P	0x10f00

Figure 4: Resource identifiers for all XS1 ports

```
void configure_in_port(void port p, const clock clk);

void configure_out_port(void port p, const clock clk, unsigned initial);

void configure_port_clock_output(void port p, const clock clk);

void start_port(void port p);

void stop_port(void port p);

void configure_in_port_handshake(void port p,
                                 in port readyin,
                                 out port readyout,
                                 clock clk);

void configure_out_port_handshake(void port p,
                                  in port readyin,
                                  out port readyout,
                                  clock clk,
                                  unsigned initial);

void configure_in_port_strobed_master(void port p,
                                      out port readyout,
                                      const clock clk);

void configure_out_port_strobed_master(void port p,
                                       out port readyout,
                                       const clock clk,
                                       unsigned initial);

void configure_in_port_strobed_slave(void port p,
                                     in port readyin,
                                     clock clk);

void configure_out_port_strobed_slave(void port p,
                                      in port readyin,
                                      clock clk,
                                      unsigned initial);

void set_port_use_on(void port p);

void set_port_use_off(void port p);

void set_port_mode_data(void port p);

void set_port_mode_clock(void port p);

void set_port_mode_ready(void port p);

void set_port_master(void port p);

void set_port_slave(void port p);

void set_port_no_ready(void port p);

void set_port_strobed(void port p);

void set_port_handshake(void port p);
```

Figure 5: xs1.h port configuration functions

```
void configure_clock_src(clock clk, void port p);

void configure_clock_ref(clock clk, unsigned char divide);

void configure_clock_rate(clock clk, unsigned a, unsigned b);

void configure_clock_rate_at_least(clock clk, unsigned a, unsigned b);

void configure_clock_rate_at_most(clock clk, unsigned a, unsigned b);

void set_clock_src(clock clk, void port p);

void set_clock_ref(clock clk);

void set_clock_div(clock clk, unsigned char div);

void set_clock_rise_delay(clock clk, unsigned n);

void set_clock_fall_delay(clock clk, unsigned n);

void set_port_clock(void port p, const clock clk);

void set_port_ready_src(void port p, void port ready);

void set_clock_ready_src(clock clk, void port ready);

void set_clock_on(clock clk);

void set_clock_off(clock clk);

void start_clock(clock clk);

void stop_clock(clock clk);
```

Figure 6: xs1.h clock configuration functions

```
void set_port_drive(void port p);

void set_port_drive_low(void port p);

void set_port_pull_up(void port p);

void set_port_pull_down(void port p);

void set_port_pull_none(void port p);

void set_port_no_sample_delay(void port p);

void set_port_sample_delay(void port p);

void set_port_no_inv(void port p);

void set_port_inv(void port p);

void set_port_shift_count(buffered void port p, unsigned n);

void set_pad_delay(void port p, unsigned n);

void sync(void port p);

unsigned peek(void port p);

void clearbuf(void port p);

unsigned endin(buffered void port p);

unsigned partin(buffered void port p,
                        unsigned n);

void partout(buffered void port p,
                        unsigned n,
                        unsigned val);

unsigned partout_timed (buffered void port p,
                        unsigned n,
                        unsigned val,
                        unsigned t);

{unsigned value, unsigned timestamp} partin_timestamped(
                        buffered void port p,
                        unsigned n);

unsigned partout_timestamped(buffered void port p,
                        unsigned n,
                        unsigned val);
```

Figure 7: xs1-h advanced port configuration functions

```
void pinseq(unsigned val);

void pinsneq(unsigned val);

void pinseq_at(unsigned val, unsigned time);

void pinsneq_at(unsigned val, unsigned time);
```

Figure 8: xs1.h predicate functions