XMOS

AN03001: XCORE Clocked Input and Output

Publication Date: 2025/3/11
Document Number: XM-015254-AN v1.0.0

IN THIS DOCUMENT

Many data signal protocols require data to be sampled and driven on specific edges of a clock. XCORE ports can be configured to use either an internally generated clock or an externally sourced clock, and the processor can record and control on which edges each input and output operation occurs. These operations can be directly expressed in the input and output function calls that can use a *timed* parameter or yield a *timestamp*.

This document describes how to configure ports to function with clocks, and is part of a group of application notes that describes ports. It is assumed that you have read the basic use of ports:

- AN03000: XCORE Input and Output

Further details on ports can be found in

- AN03002: XCORE Port Buffering
- AN03003: XCORE Serialization and Strobing
- AN03007: XCORE Ports
- AN02039: Ports, Pins, and the XN file
- lib_xcore documentation

1 Generating a Clock Signal

The waveform diagram shown in Fig. 1 shows an example of an data bus driven by a port along with a clock signal. The rising edge of the clock signal would be used by an external device to sample the data.

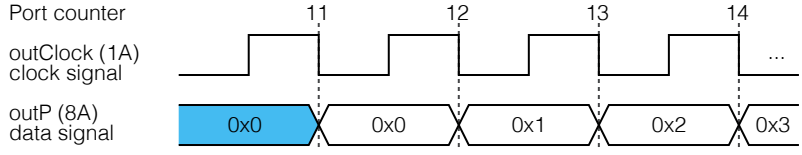

Fig. 1: Waveform diagram

An output instruction from the XCORE processor causes the port to drive output data on the next falling edge of its clock; the data is held by the port until another output is performed.

To generate the waveform shown above, port 8A (`outP`) should be configured as an 8-bit output while port 1A (`outClock`) is configured as a 1 bit clock signal as illustrated in Fig. 2.

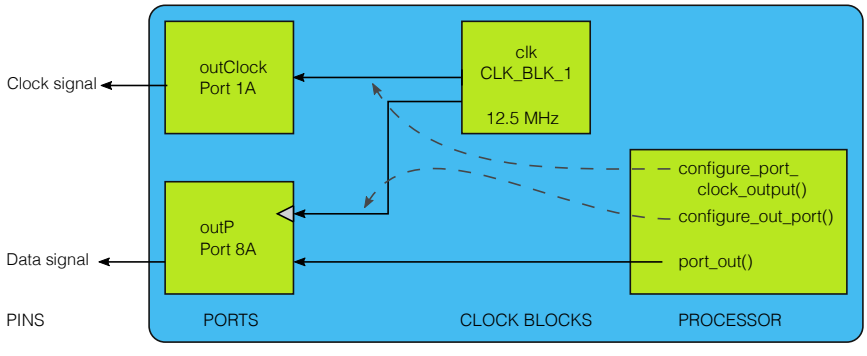

Fig. 2: Port configuration diagram

The programme below shows how to configure the port to implement the above configuration, where the port is clocked at a rate of 12.5 MHz, outputting the corresponding clock signal with its output data.

```
#include <xcore/port.h>
#include <xs1.h>

port_t outP     = XS1_PORT_8A;
port_t outClock = XS1_PORT_1A;
xclock_t clk    = XS1_CLKBLK_1;

int main(void) {
    port_enable(outP);
    port_enable(outClock);
    clock_enable(clk);

    clock_set_divide(clk, 4); // divide by 8
    port_out(outP, 0);
    port_set_clock(outP, clk);
    port_set_clock(outClock, clk);
    port_set_out_clock(outClock);
    clock_start(clk);

    for(int i=0; i<5; i++) {
        port_out(outP, i);
```

(continues on next page)

(continued from previous page)

```
    }
}
```

The declaration

```
xclock_t clk = XS1_CLKBLK_1;
```

declares a clock named `clk`, which refers to the clock block identifier `XS1_CLKBLK_1`. Variables of the type `xclock_t` refer to physical entities, and care should be taken to not use the same clock-block in two places.

The statement

```
clock_set_divide(clk, 4); // divide by 8
```

configures the clock `clk` to have a rate of 12.5 MHz. By default clock-blocks are clocked from a 100 MHz reference clock, and setting the divider to `4` divides the clock by two times that value for *100/(4x2)=12.5*.

The statements:

```
port_out(outP, 0);
port_set_clock(outP, clk);
```

configure the output port `outP` to be clocked by the clock `clk`, with an initial value of 0 driven on its pins. Note that we set the port to 0 before change the clock to the new clock. The new clock is not yet running, so we should be careful to not use that port between setting it up and starting the clock.

The statements:

```
port_set_clock(outClock, clk);
port_set_out_clock(outClock);
```

cause the clock signal `clk` to be driven on the pin connected to the port `outClock`, which a device connected to the XCORE can use to sample the data driven by the port `outP`. The statement:

```
clock_start(clk);
```

causes the clock block to start producing edges.

A port also has an internal 16-bit counter, which is incremented on each falling edge of its clock. This can be used to output data on specific clock edges as described later in this document.

2 Using an External Clock

When using an XCORE port as an input signal to the processor, it is often necessary to synchronise the sampling of data to an external clock.

The waveform diagram shown in Fig. 3 shows an example of a 8-bit data bus sampled by a port on the rising edge of an external clock.

Fig. 3 shows the port counter, clock signal, and example input stimuli.

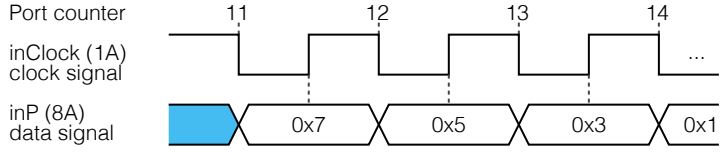

Fig. 3: Waveform diagram

An input instruction by the processor causes the port to sample data on the next rising edge of its clock. The values input are 0x7, 0x5, 0x3, 0x1 and 0x0.

The program configures the ports **`inP`** and **`inClock`** as illustrated in Fig. 4.

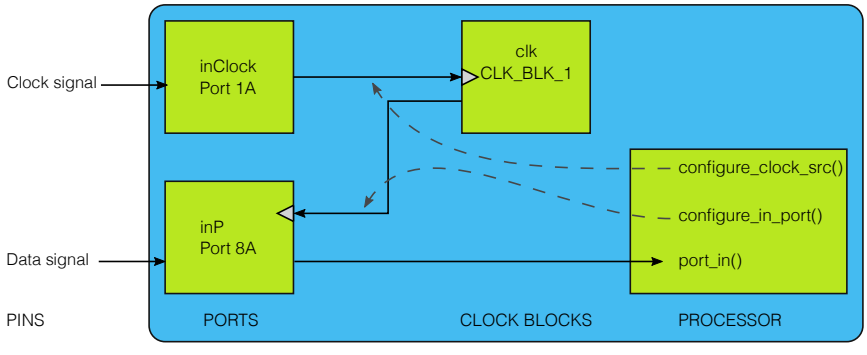

Fig. 4: Port configuration diagram

The following programme illustrates how to configures the port to synchronise the sampling of data to an external clock as shown in Fig. 4.

```
#include <xcore/port.h>
#include <xs1.h>

port_t inP     = XS1_PORT_8A;
port_t inClock = XS1_PORT_1A;
xclock_t clk   = XS1_CLKBLK_1;

int main(void) {
    int data[5];
    port_enable(inP);
    port_enable(inClock);
    clock_enable(clk);

    clock_set_source_port(clk, inClock);
    port_set_clock(inP, clk);

    clock_start(clk);
    for (int i=0; i<5; i++) {
        data[i] = port_in(inP);
    }
    for (int i=0; i<5; i++) {
        printf("%d\n", data[i]);
    }
```

(continues on next page)

(continued from previous page)

```
}
```

The program configures the ports **inP** and **inClock** as illustrated in Fig. 4.

The statement:

```
clock_set_source_port(clk, inClock);
```

configure the 1-bit input port **inClock** to provide edges for the clock **clk**. An edge occurs every time the value sampled by the port changes.

The statement:

```
port_set_clock(inP, clk);
```

configures the input port **inP** to be clocked by the clock **clk**.

3 Performing I/O on Specific Clock Edges

It is often necessary to perform an I/O operation on a port at a specific time with respect to its clock. This allows ports to directly generate some more complex waveforms.

As an example, consider the waveform diagram shown in Fig. 5.

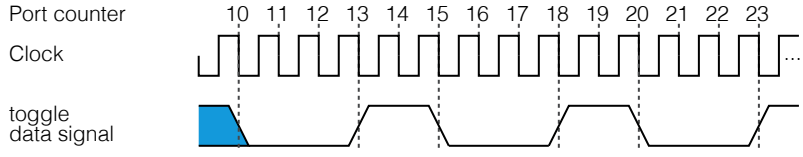

Fig. 5: Waveform diagram

In this waveform the signal on an output port is changed on the third and fifth clock cycle.

To do this we need to use the port counter capability built into the port. The port counter is a 16-bit counter that is incremented on the falling edge of the clock. On intermediate edges for which no value is provided, the port continues to drive its pins with the data previously output.

The example function below drives a pin high on the third clock period and low on the fifth.

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

void doToggle(port_t toggle) {
    int count;
    port_out(toggle, 0);
    count = port_get_trigger_time(toggle);    // timestamped output
    for(int i = 0; i < 20; i++) {
        count += 3;
        port_set_trigger_time(toggle, count);
        port_out(toggle, 1);                  // timed output
        count += 2;
        port_set_trigger_time(toggle, count);
        port_out(toggle, 0);                  // timed output
    }
}
```

The statement

```
count = port_get_trigger_time(toggle);
```

obtains the *timestamp* of the output. The preceding call outputted the value 0 to the port `toggle`, and this function reads the value of the port-counter into the variable `count` the value of the port counter when the output data is driven on the pins. The program then increments count by a value of 3 and performs a *timed output* by calling the following two functions:

```
port_set_trigger_time(toggle, count);
port_out(toggle, 1);
```

The first call instructs the port to wait with its next output until its counter equals the value `count+3` (advancing three clock periods) and to then perform its output. The last three statements delay the next output by two clock periods. Fig. 5 shows the port counter, clock signal and data driven by the port.

4 Case Study: LCD Screen Driver

LCD screens are found in many embedded systems. The principal method of driving most screens is the same, although the specific details vary from screen to screen. Fig. 6 illustrates the operation of an LCD screen, including the waveform requirements for transmitting a single frame of video.

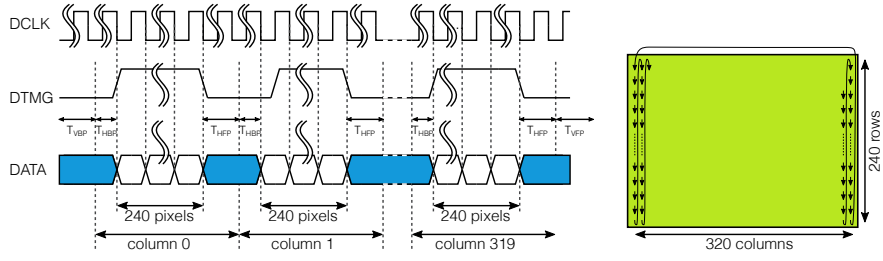

Fig. 6: LCD Screen Driver Example

The screen has a resolution of 320x240 pixels. It requires pixel data to be provided in column order with each value driven on a specific edge of a clock. The signals are as follows:

- DCLK is a clock signal generated by the driver, which must be configured within the range of 4.85 MHz to 7.00 MHz. The value chosen determines the screen refresh rate.
- DTMG is a data valid signal which must be driven high whenever data is transmitted.
- DATA carries 18-bit RGB pixel data to the screen.

The specification requires that pixel values for each column are driven on consecutive cycles with a 55 cycle delay between each column and a 4235 cycle delay between each frame.

LCD screens are usually driven by dedicated hardware components due to their clocking requirements. Implementing an LCD screen driver in C is easy due to the clock synchronisation supported by the XMOS architecture. The required port configuration is illustrated in Fig. 7.

The ports `DATA` and `DTMG` are both clocked by an internally generated clock, which is made visible on the port `DCLK`. The program below defines a function that configures the ports in this way.

```
#include <xcore/port.h>
#include <xcore/clock.h>
#include <xcore/channel.h>
#include <xs1.h>

port_t   DCLK = XS1_PORT_1A;
port_t   DTMG = XS1_PORT_1B;
port_t   DATA = XS1_PORT_32A;
xclock_t clk  = XS1_CLKBLK_1;

void lcdInit(void) {
    port_enable(DCLK);
    port_enable(DTMG);
    port_enable(DATA);
    clock_enable(clk);

    clock_set_divide(clk, 8); // divide by 16 = 100/16 = 6.25 MHz
    port_set_clock(DATA, clk);
    port_set_clock(DTMG, clk);
    port_set_clock(DCLK, clk);
    port_set_out_clock(DCLK);
    clock_start(clk);
}
```

The clock rate specified is 6.25 MHz. The time required to transmit a frame is 4235 + 320 * (20 + 240 + 35) = 98,635 clock ticks, giving a frame rate of 6,250,000/98,635 = 63 Hz.

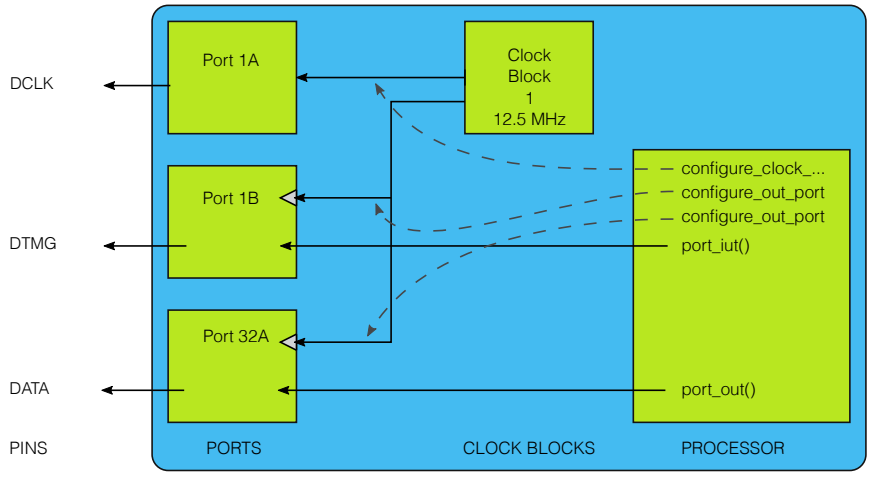

Fig. 7: Port configuration diagram

The function below outputs a sequence of pixel values to the LCD screen on the clock edges required by the specification.

```
void lcdDrive(chanend_t c) {
    unsigned x, time;
    port_out(DTMG, 0);
    time = port_get_trigger_time(DTMG);
    while (1) {
        time += 4235;
        for (int cols=0; cols<320; cols++) {
            time +=20;
            x = chan_in_word(c);
            port_set_trigger_time(DTMG, time);
            port_out(DTMG, 1);
            port_set_trigger_time(DATA, time);
            port_out(DATA, x);   // pixel 1
            for (int rows=1; rows<240; rows++) {
                x = chan_in_word(c);
                port_out(DATA, x);  // pixels 1..239
            }
            port_set_trigger_time(DTMG, time+240);
            port_out(DTMG, 0);
            time += 35;
        }
    }
}
```

A stream of data is input from a channel end. The body of the `while` loop transmits a single frame and the body of the outer `for` transmits each column. The program instructs the port `DTMG` to start driving its pin high when it starts outputting a column of data and to stop driving afterwards.

An alternate solution is to configure the port `DATA` to generate a ready-out strobe signal on `DTMG` (see AN03003: XCORE Port Serialisation and Deserialisation) and to remove the two outputs to `DTMG` by the processor in the source code.

5 Summary of Clocked Port Behavior

The semantics for inputs and outputs on clocked (unbuffered) ports are summarised as follows.

Output to a port

- An *output* to a port causes data to be driven on the next falling edge of the clock. The output blocks until the subsequent rising edge.
- A *timed output* to a port causes data to be driven by the port when its counter equals the specified time. The output blocks until the next rising edge after this time.
- The data driven on one edge continues to be driven on subsequent edges for which no new output data is provided.

Input from a port

- An *input* from a port causes data to be sampled by the port on the next rising edge of its clock. The input blocks until this time.
- A *timed input* from a port causes data to be sampled by the port when its counter equals the specified time. The input blocks until this time.
- A *conditional input* from a port causes data to be sampled by the port on each rising edge until the sampled data satisfies the condition. The input blocks until this time, taking the most recent data sampled.

SELECT_RES constructs

Input ports can be used in *SELECT_RES* constructs, a mechanism that allows a thread to deal with multiple ports (and other resources) simultaneously. SELECT_RES is documented in the lib_xcore documentation, but here we note how SELECT_RES waits for any one of the ports to become ready and complete the corresponding case:

- For an *input*, the port is ready at most once per period of its clock.
- For a *timed input*, the port is ready only when its counter equals the specified time.
- For a *conditional input*, the port is ready only when the data sampled satisfies the condition.
- For a *timed conditional input*, the port is ready only when its counter is equal or greater than the specified time and the value sampled satisfies the condition.

For a timestamped operation that records the value t, the next possible time that the thread can input or output is $t + 1$.

Caution: On XCORE devices, all ports can be buffered. The resulting semantics, which extend those given above, are discussed in AN03002: XCORE Port Buffering.