IO timings for xcore.ai

Publication Date: 2024/10/7
Document Number: XM-014231-AN v2.0.0

IN THIS DOCUMENT

In this document we show you how to estimate the IO timings for data input and output by the xcore.ai. Most interfaces are slow enough that the timings work out in any case. Faster interfaces need their timings analysed to ensure operation under all circumstances.

On reading this document you will see that there are a couple of circumstances where it will “just work” because the clock timings are so forgiving:

- Application clocks of 15 MHz or slower typically just work
- For data and clock travelling in the same direction (ie, source-synchronous clocking where both output by the xCORE or both input by the xCORE), frequencies of up to 60 MHz can work.

Faster clocks may also work, but an analysis is needed to show whether any programmatic delays are necessary to cover all corners. In particular, some timings depend on PVT variations (Process, Voltage, and Temperature). An analysis will ensure that an interface will work under all possible corners, rather than the corners that were tested.

1 Basics

IO refers to transferring digital data between devices. In this document, we are concerned by transferring a single bit of data between an xcore.ai device and some external device. We only consider *synchronous IO*, that is, we assume that there is a *clock*, and the data is transferred synchronous to the clock. The limitation to just use a single data-line is trivially generalised to multiple data lines, including data valid lines.

We assume that the data is transferred in a specific direction. That is, there will be two sides to the data transfer. One side will be *outputting* the data-bit by *driving* the data-net, the other side will be *inputting* the data-bit by *sampling* the data-net. The clock is generated by either side or by a third party. An example is shown in Fig. 1 where the xCORE is using an externally provided clock to sample a data signal, and the external device is generating the clock and the data. We assume that data is clocked in by the input side on the *rising edge* of the clock, and that the data is clocked out by the output side on the *falling edge* of the clock.

We use the term *application clock* to refer to the clock signal; this makes it clear that it is different from other clock signals in the system, such as the *Core clock* of the xCORE (typically 600 MHz), the *Reference clock* of the xCORE (typically 100 MHz), or the PLL input clock (typically 20-25 MHz).

There are three critical times in the analysis of the IO timings; one on the output side, and two on the input side. Every digital device will specify those times in one form or another, although terminology may differ:

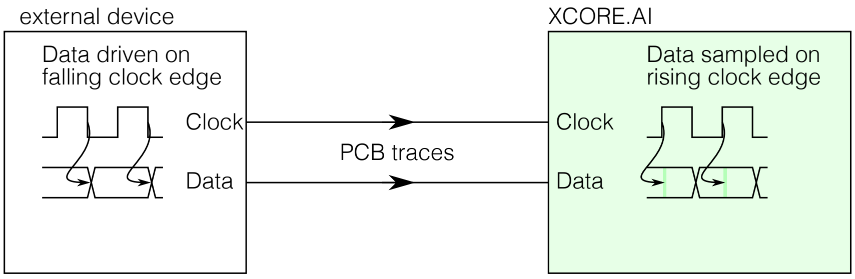
external device
Data driven on falling clock edge
Clock
Data
PCB traces
XCORE.AI
Data sampled on rising clock edge
Clock
Data

Fig. 1: Example: external device transferring data to xcore.ai

Timing parameter	Brief description
T(CLKtoDATA)	Time at which output will change
T(setup)	Time at which input must be stable
T(hold)	Time until which input must be kept stable

On the output side, T(CLKtoDATA) refers to the time between the clock edge falling and the data appearing on the data line. T(CLKtoDATA) normally has a range of possible values that depend on environmental conditions, and we use the *minimum* and *maximum* T(CLKtoDATA) to define the first possible time where the data could become valid, and the last possible time where the data will be valid. In between these times, the data may be old, new, or something undefined. Normally, T(CLKtoDATA) has a positive value, indicating that the data will be valid after the falling clock edge. A negative value indicates that the data may or will be valid prior to the falling clock edge.

On the input side, T(setup) refers to the time at which the data must be valid for it to be clocked in correctly. T(setup) is the length time before the rising application clock edge. T(hold) refers to the amount of time that the data should stay valid for, in order for it to be clocked in correctly. T(hold) is the length of time after the rising edge of the application clock. Normally setup and hold times are positive numbers. By convention, the setup-time is the time that the data has to be stable *before* the clock edge, and the hold time is the time that the data has to remain stable *after* the clock edge. We use a *negative* setup-time to indicate that the data only has to be stable *after* the clock edge; and a *negative* hold time to indicate that the data does not have to be held after the clock edge.

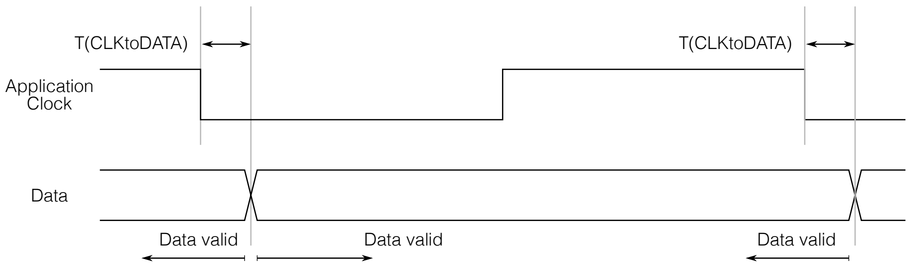

Fig. 2: Definition of clock-to-data times for xCORE outputting data.

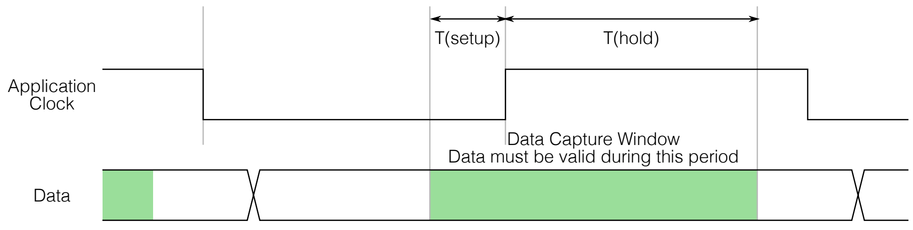

Fig. 3: Definition of setup and hold times for xCORE inputting data.

These times are summarised in Fig. 2 and Fig. 3. The important consideration is that the external device shall not change in the capture window of the xCORE (for input on the xCORE), or that the xCORE shall not change the data in the capture window of the external device (for output by the xCORE)

Note that this assumes an idealised setup - typically clock and data travel across the PCB, and signals will not abruptly change from a zero to a one but take time to rise and fall depending on the PCB characteristics. A discussion on this is outside the scope of this document. In this document we assume an ideal setup where there are no latencies incurred in PCB traces.

The ideal way to implement data transfer uses a *source-synchronous clock*. That is, the device that drives the data also drives the clock. The alternative is that the clock is driven by the device that inputs the data, or by some third party. Section 4 describes the timings of source-synchronous clocked systems; Section 5 describes the timings of systems that are not source-synchronous.

2 The xcore.ai IO circuitry

IO circuitry comprises two clock domains: logic that is running synchronous to the core clock, and IO drivers that are asynchronous. These are shown in Fig. 4.

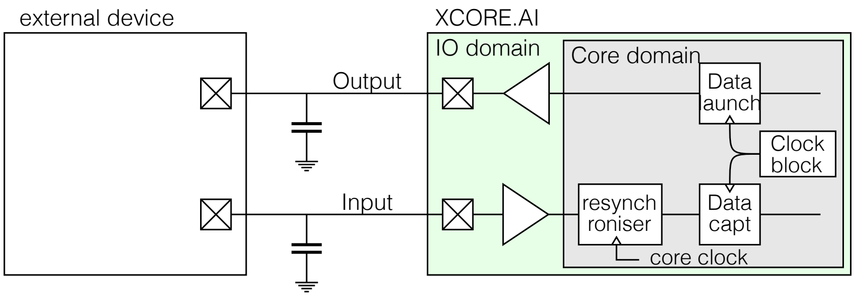

Fig. 4: xcore.ai IO elements

On the output path, data is launched on a clock-edge, it then has to traverse the IO drivers, which in turn have to drive the net, until the signal is stable. On the input side, the external device has to drive the net, it then has to traverse the input buffer, a resynchroniser which makes the signal synchronous to the core clock, until it is captured on a clock edge. There are multiple clock blocks available to deal with different application clocks.

The IO driver logic introduces a delay and uncertainty that is independent of the core clock, but that does depend on environmental factors such as the core voltage, IO voltage, temperature, and process variation ("PVT variation"). The timings depend on the pins that you use; some pins are better matched and/or faster than others. In the main body of this document we show how to calculate worst case T(setup), T(hold), and T(CLKtoDATA) for xcore.ai devices. These calculations are based on three device parameters, the input skew, output skew, and round trip time. These three device parameters are characterised for xcore.ai devices as follows:

Parameter	Description	Min	Typ	Max	Units
T(iskew)	Input skew			0.9	ns
T(oskew)	Output skew			1.2	ns
T(RTT)	Round trip time	1.8		10.3	ns

T(oskew) is defined as measuring the time difference between two signals that are launched by the xCORE at the same time (Fig. 5). *T(iskew)* is defined as the time difference between two signals entering the xCORE at the same time (Fig. 6). *T(RTT)* finally is defined as the time difference between a signal being launched by the xCORE, travelling through a loopback, and then back in (Fig. 7). Note that all three only relate to the IO domain, not to the core domain.

We will explain later how to use these parameters. The values above are worst-case values assuming an external load of 5 pF. Appendix A lists tighter timings for these parameters for specific groups of IO pins on specific tiles, and explains in which situations the minimum and maximum values may be encountered. This will enable you to make tighter designs, and guide you in using the pins that have tighter IO timings. Appendix A also gives timings for higher loads.

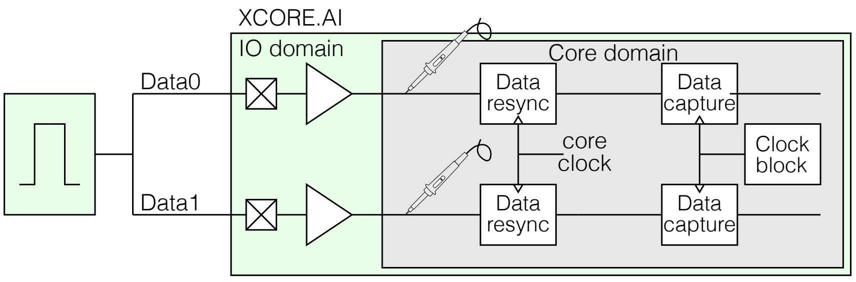

Fig. 5: Definition of *T(iskew)*

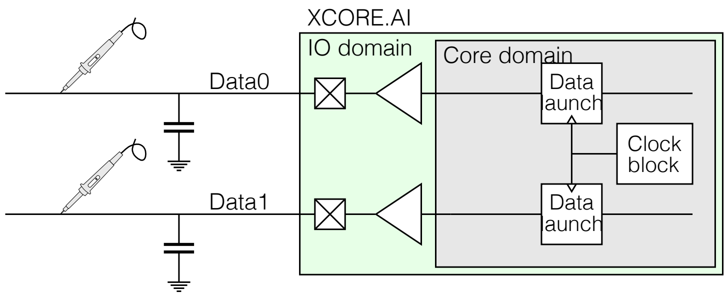

Fig. 6: Definition of *T(oskew)*

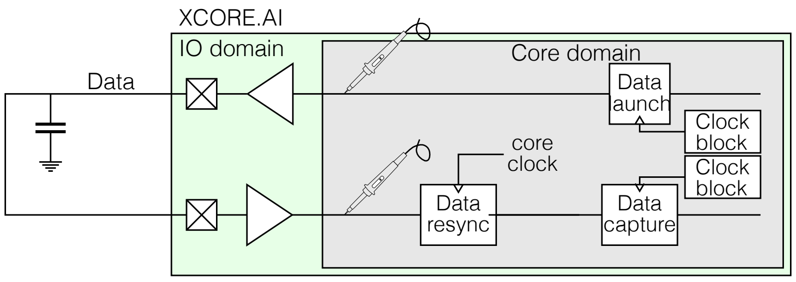

Fig. 7: Definition of *T(RTT)*

In the rest of this document we use the above worst-case values for *T(iskew)*, *T(oskew)* and *T(RTT)* to calculate example IO timings.

3 Source-synchronous clocks

The timings of source-synchronous clocked systems depend on the *skew* between pins only. The *skew* measures the difference in timings for signals that travel in the same direction, and is up to *T(oskew)* (1.2 ns) depending on the conditions and the direction of the signal.

When designing a source-synchronous clocked system, you can use the skew times listed in this document as a start, but you should add the skew introduced by external components. All nets (clock and data) should be matched for maximum performance.

3.1 Input: Calculating setup and hold times for an external application clock

This section describes how you can compute the setup and hold time for capturing data relative to an external application clock. This is the situation where both clock and data originate outside the xcore.ai, as shown in Fig. 8. The blue arrow is the timing path of the clock, and the red arrow is the timing path of the data. The difference between the two is the skew that governs the uncertainty of where the data will be captured.

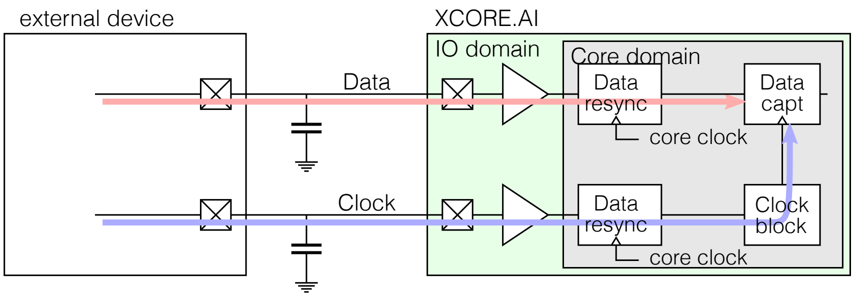

Fig. 8: Conceptual diagram of input with source-synchronous clock

The table below shows how to compute the setup and hold timings for the xcore.ai processor. The setup- and hold-time depends on the *input skew* of the xcore.ai and *core clock frequency* of the xCORE. and the table shows setup- and hold-times for a 600 MHz device, and a formula for the general case. The general formulas show that *T(iskew)* is required on both sides of the *eye* (the eye is the window during which data must be valid). An extra *T(coreclock)* is required at the end of the eye because of resynchronisation. The whole window is shifted one *T(coreclock)* to the right because the clock is delayed by one *T(coreclock)*. The signals are shown in Fig. 9.

Parameter	600 MHz	For any core-clock
T(setup)	-0.8 ns	T(iskew) - T(coreclock)
T(hold)	4.2 ns	T(iskew) + 2 x T(coreclock)

T(iskew) is at most 0.9 ns. For specific sets of pins tighter *T(iskew)* can be found in Appendix A, reducing the eye size. Note that the negative *T(setup)* indicates that the eye opens after the clock edge.

For a part that runs on a 600 MHz core clock we show the setup and hold times in the diagram below, in this example with a 50 MHz application clock. The capture point is notionally 2.55 ns after the application clock edge, with 1.75 ns uncertainty either way.

This is the *minimum* required eye, and the data may be presented earlier, for example before the clock edge. This will show a more familiar eye that straddles the clock edge.

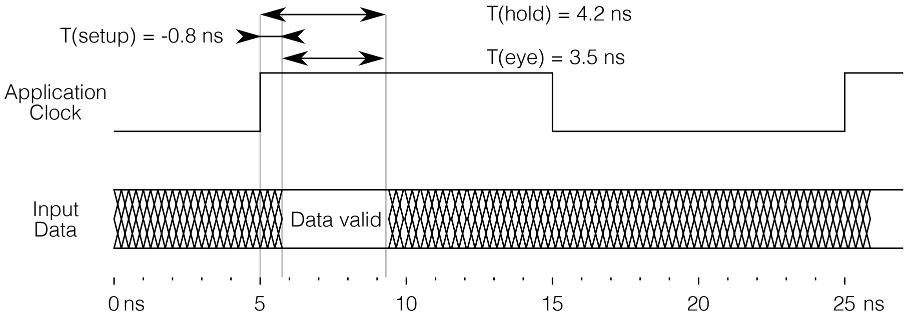

Fig. 9: Setup and hold time for a 50 MHz external clock.

We can calculate the eye by adding the setup-time and the hold-time. Given that *T(setup)* is *T(iskew)* - *T(coreclock)* and that *T(hold)* is *T(iskew)* + 2 x *T(coreclock)*, the eye is *T(iskew)* - *T(coreclock)* + *T(iskew)* + 2 x *T(coreclock)* = 2 x *T(iskew)* + *T(coreclock)* or at most 3.5 ns for an arbitrary pair of pins on a 600 MHz xcore.ai part.

Adjusting setup and hold timings

If the setup and hold times are too early or too late to reliably capture the data, then they can be adjusted in increments of a single *core-clock cycle*, ie, in steps of 1.66 ns on a 600 MHz part. This is implemented by one of two mechanisms:

- The data signal can be delayed by up to *five* core-clock cycles by setting a delay on the data port. Setting the delay to *X* on the data port, will subtract *X* core-clock cycles from the hold-time, and add *X* core-clock cycles to the setup-time. Use the `set_pad_delay` function.
- An external application clock signal can be delayed by up to *4096* core-clock cycles by setting the rise-delay and fall-delay on the clock block. Setting the rise- and fall-delays to *X*, will add *X* core-clock cycles to the hold-time, and subtract *X* core-clock cycles from the setup-time. The application clock should never be delayed by more than one application clock cycle. Use the `set_clock_fall_delay` and `set_clock_rise_delay` functions. The delay is limited to half a clock cycle.

Setting the data-delay or the application-clock rise- and fall-delay brings the eye forwards and backwards; it does not affect the size of the eye.

An example is shown in Fig. 10 for an external clock, where the data is delayed by one core-clock. As it is delayed in the chip, it needs to be presented on the pads one core-clock-cycles, or 1.66 ns, earlier to meet the setup-time. The delay also reduces the hold-time by 1.66 ns, effectively bringing the eye forward by 1.66 ns.

It is tempting to think that any device that meets this eye will meet the required setup- and hold-times. However, given that the eye-window can only be moved in steps of a single core-clock cycle, it is not possible to meet all setup- and hold-time combinations that sum to the minimum eye opening. However, any device that provides an eye-opening that is at least one core clock cycle longer than the minimum eye, can always have its setup and hold times met.

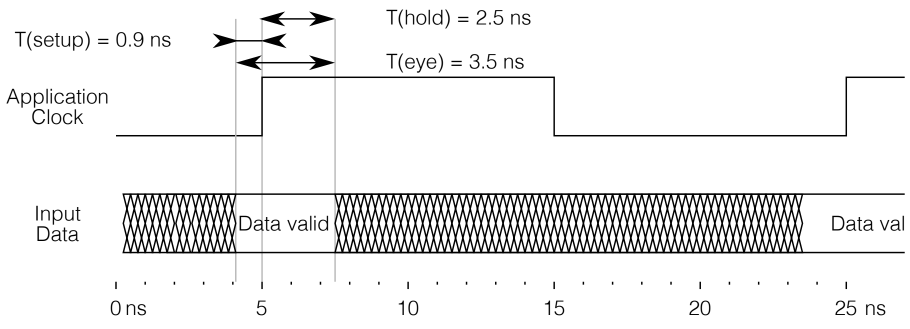

Fig. 10: Setup and hold times for data given an external clock, with a delay of one core-clock cycle in the data.

Maximum clock rates

The application clock should never be faster than half the core-clock-frequency. For example, if the xCORE is running at 100 MHz (rather than the normal 600 Mhz), then the application clock is limited to 50 MHz.

The application clock is also limited by the speed at which signals can be supplied to the xCORE. This is limited by the drive strength, the analogue characteristics of the pad, and inductances, capacitances, and resistance of the PCB traces. In addition, the data has to be stable during the eye.

For example, for a specific driver and PCB design, you may have calculated that 12.5 ns is sufficient for the data-signal to stabilise after an external clock. In that case the minimum application clock period is *3.5 ns+12.5 ns*, or *66 MHz* (assuming a 600 MHz part, and assuming that the eye can be lined up with the application clock). This is visualised in Fig. 11.

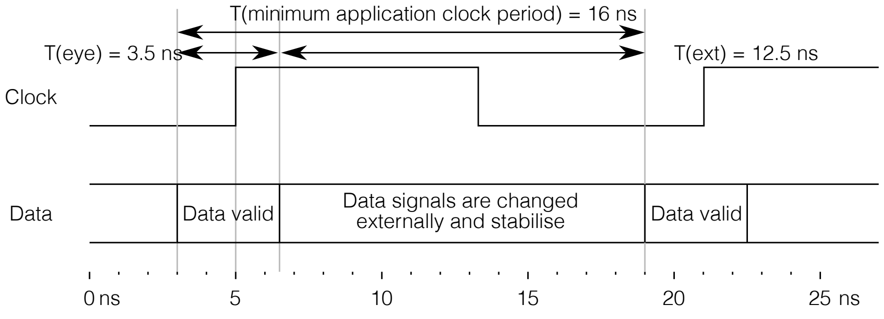
T(minimum application clock period) = 16 ns
T(eye) = 3.5 ns
T(ext) = 12.5 ns
Clock
Data
Data valid
Data signals are changed externally and stabilise
Data valid
0 ns
5
10
15
20
25 ns

Fig. 11: Establishing the minimum external application clock period.

Input window summary

Setup and hold times for an external clock for various data delays are shown below:

Core freq	Data delay	T(setup)	T(hold)
600 MHz	2	2.5 ns	0.9 ns
	1	0.9 ns	2.5 ns
	0	-0.8 ns	4.2 ns

Assuming that the data is input on a port **p**, the second line can be achieved by calling `set_pad_delay(p, 1)`.

3.2 Output: Calculating clock-to-data times for an internal application clock

The output timings of an xCORE are straightforward if an internal clock is used. This is source-synchronous clocking where both data and clock will leave the synchronous core at the same time, as shown in Fig. 12. The blue arrow is the timing path of the clock, and the red arrow is the timing path of the data. The difference between the two is the skew that governs the uncertainty of where the data will be launched.

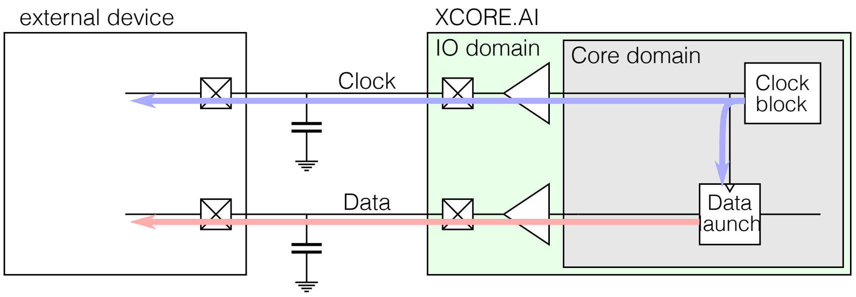

Fig. 12: Conceptual diagram of output with source-synchronous clock

The only uncertainty is the output skew. The maximum output skew on an arbitrary pair of pins on an xcore.ai is 1.2 ns, hence the Clock-to-data time is between -1.2 ns and +1.2 ns. That is, the data may precede the clock by as much as 1.2 ns; and the data will be stable 1.2 ns after the falling clock edge, as shown in Fig. 13. Better timings can be achieved by picking a tile and bank with a lower output skew, as listed in Appendix A: the IO is grouped in banks, with better timings within a bank of IO pins.

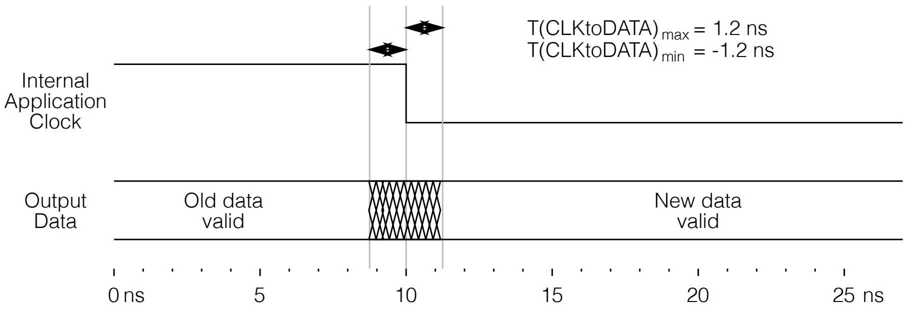

Fig. 13: T(CLKtoDATA) on an internally generated, source-synchronous clock.

If open-drain outputs are used, then an extra 5 ns skew should be accounted for, in addition to the extra times required by the external resistor to pull the signal high.

4 Non source-synchronous clocks

For non source-synchronous clocks, the timings are governed by the *round trip time* through an output pin and an input pin. We define the *round trip time* as the sum of time taken by a signal to traverse the output path of the xcore.ai, and the time taken by a signal to traverse the input path of the xcore.ai. Electrically, this path will never be taken, but logically it will be taken as one signal (eg, the clock) will travel out whereas the causally dependent signal (eg, the data) will travel in. The *round trip time* is difficult to control tightly. Systems that need to work under a set of environmental conditions need to be closed so that they work with both the best case round trip time T(RTTmin), and worst case round trip time, T(RTTmax).

When designing the system, you can use the round-trip times listed in this document as a start, but you should add the time required to drive the input and output net; taking into account the load on the net.

4.1 Input: Calculating setup and hold times for an internal application clock

When data is input on an internal application clock, we have to contend with a round-trip time out of the xCORE and back in, as shown in Fig. 14. The blue arrow shows the timing path of the clock causing the data to be driven by the device; the red arrow shows the timing path of the clock sampling the data on the xCORE. The capture window has to be aligned so that it captures the data late enough to allow for the round trip delay; and the window has to be large enough to deal with uncertainty of the minimum and the maximum round trip times.

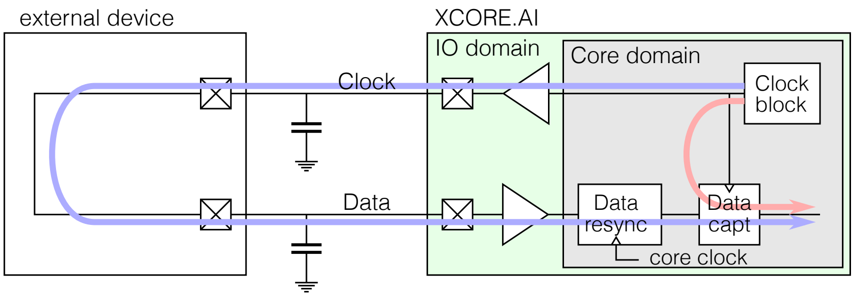

Fig. 14: Round trip of clock and data

The minimum and maximum round trip times for xcore.ai are 1.8 and 10.3 ns. (See Appendix A for specific banks that have tighter round trip times.) In addition we have a delay of between 4 and 5 core clocks for resynchronising the data and capturing the data. For a 600 MHz part this brings us to a minimum round trip time of *1.8 + 4 x 1.667 = 8.5 ns*, and a maximum round trip time of *10.3 + 5 x 1.667 = 18.6 ns* . That is an eye of *18.6 - 8.5 = 10.1 ns*, that closes 8.5 ns before the rising clock edge.

There are three ways to improve on these setup and hold times:

1. Use a pair of pins inside a specific bank (up to 2.5 ns better)
2. Use a part with a higher core clock rate of 800 MHz (0.4 ns better)
3. Increase drive strength and/or reduce the capacitance of the net (0.2 ns better)

The general formulas for setup and hold timings are given in the table below. The setup time is governed by the maximum round trip time, *T(RTTmax)*, plus the time for travelling through the resynchronisers assuming the signal arrives just after the core-clock edge. The hold time is governed by the minimum round trip time, *T(RTTmin)*, plus the time for travelling through the synchronisers assuming the signal arrives just before the core-clock edge. The hold time is negative because the data can be released before the clock edge:

Parameter	600 MHz	For any core-clock
T(setup)	18.6 ns	T(RTTmax) + 5 x T(coreclock)
T(hold)	-8.5 ns	-T(RTTmin) - 4 x T(coreclock)

The resulting eye limits the application clock to 99 MHz; at which point the data valid window occupies the whole clock cycle, and there is no longer any time for the external device to modify the data. In practice, the external device will have a clock-to-data window, that will reduce the eye and the maximum frequency.

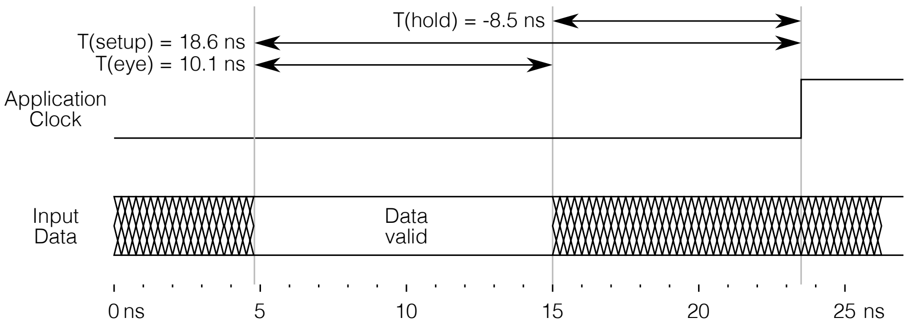

Fig. 15: Setup and hold times of an XCORE on an internal application clock

It is important to note that for clocks in excess of *1/(2 x 18.6 ns) = 26.8 MHz* the eye opening may overlap with the falling edge of the application clock. An external device with a short clock-to-data timing will modify data in the eye. In these cases, the data should either be delayed or sampled on the *falling* edge in order to move the eye away from the window where the data changes. Fig. 15 above has a clock period of 50 ns (20 MHz), if one imagines the falling edge of the clock shifting closer to the rising edge, then it will end up in the data valid window, which may cause the device that drives the data to update during the data valid window.

4.2 Output: Calculating clock-to-data times for an external application clock

When data is output on an external application clock, we have to contend with a round-trip time in to the xCORE and back out, as shown in Fig. 16. The blue arrow shows the timing path of the clock causing the data to be driven by the xCORE; the red arrow shows the timing path of the clock sampling the data inside the device. The capture window has to be aligned so that it captures the data late enough to allow for the round trip delay; and the window has to be large enough to deal with uncertainty of the minimum and the maximum round trip times.

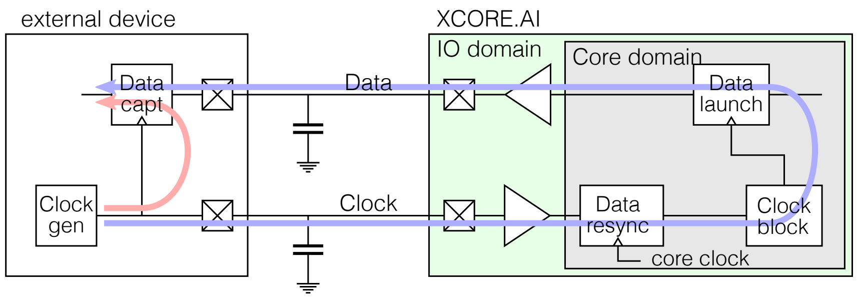

Fig. 16: Round trip of clock and data

The general formulas for setup and hold timings are as follow, and are illustrated in Fig. 17:

Parameter	600 MHz	For any core-clock
T(CLKtoDATAmax)	18.6 ns	T(RTTmax) + 5 x T(coreclock)
T(CLKtoDATAmin)	8.5 ns	T(RTTmin) + 4 x T(coreclock)

Banks with tighter timings can be found in Appendix A. One can also use a source-synchronous clock to achieve far superior timings.

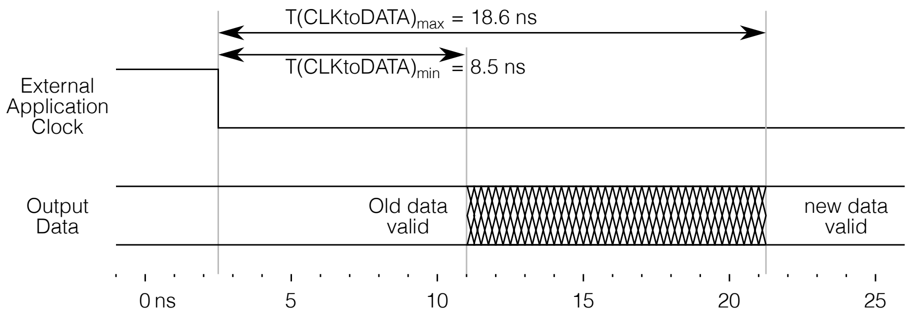
$T(CLKtoDATA)_{max}$ = 18.6 ns
$T(CLKtoDATA)_{min}$ = 8.5 ns
External Application Clock
Output Data
Old data valid
new data valid
0 ns
5
10
15
20
25

Fig. 17: T(CLKtoDATA) on externally generated clocks.

If open-drain outputs are used, then an extra 2 ns margin on the max time should be accounted for, and 3 ns should be subtracted from the min time; in addition to any time required by the external resistor to pull the signal high

5 A worked example: master I2S

Below we show the timings of a worked example: I2S. I2S uses a bit-clock (signal I2S_BCLK) to clock DAC-data out to the CODEC (signal I2S_DAC), to clock ADC-data in from the CODEC (signal I2S_ADC), and to clock out a Left-Right clock (I2S_LRCLK) that indicates whether the data transferred is a left-sample or a right sample. There are a master and a slave, and in this example the xCORE will be the master that produces I2S_BCLK and I2S_LRCLK (and I2S_DAC), and the external device will be the slave. The bit-clock is produced by the xCORE from a master-clock (signal I2S_MCLK). I2S transmits 2 x 32 bits of data for each audio-frame; a 32-bit left value and a 32-bit right value.

We assume:

- On the xCORE there are five 1-bit ports connected to the I2S_MCLK, I2S_BLCK, I2S_LRCLK, I2S_DAC and I2S_ADC signals.
- The LR-clock will operate at 192 KHz.
- The bit-clock will operate at 12.288 MHz (LR-clock x 64)
- The master-clock will operate at 24.576 MHz (LR-clock x 128)

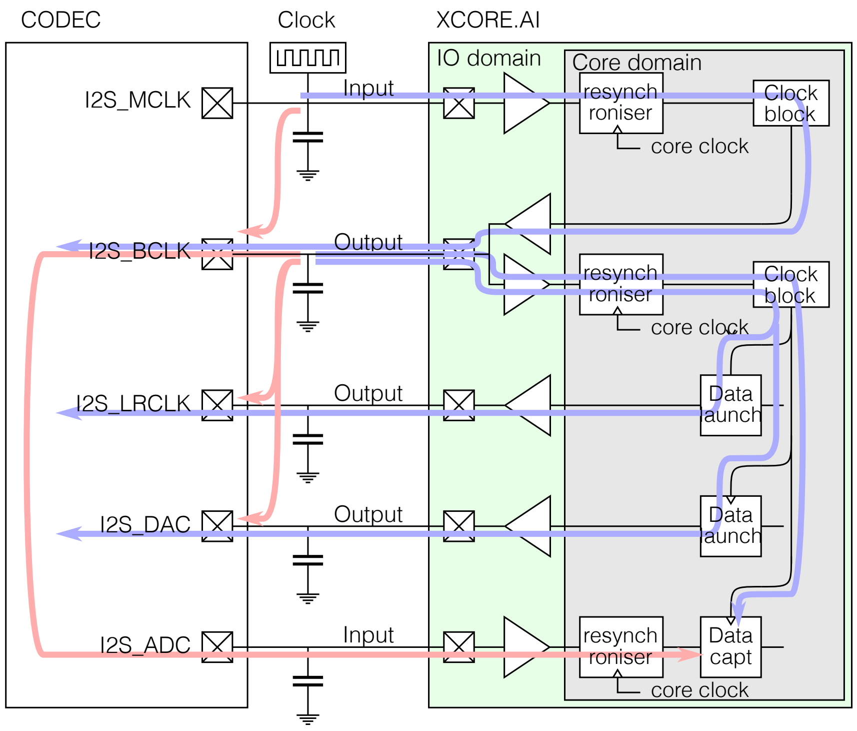

Fig. 18: I2S clock block organisation

We assume that the master-clock is sampled on a one bit port, this master clock is divided down and drives the bit-clock, which in turn drives a clock-block that is used to clock out the LRCLK and DAC data, and used to clock in the ADC data. This is shown in Fig. 18. From top to bottom note that:

- I2S_BCLK is not source-synchronous to I2S_MCLK (there is a long blue path from the Clock generator into the xCORE and back out through I2S_BCLK, and a short red path outside the xCORE).
- I2S_LRCLK is not source-synchronous with I2S_BCLK (the blue arrow is a long path into and back out of the xCORE, the red arrow is a short path near the device).
- I2S_DAC is not source-synchronous with I2S_BCLK (the blue arrow is a long path into and back out of the xCORE, the red arrow is a short path near the device).
- I2S_ADC is source-synchronous to I2S_BCLK (the blue path is the clock path into the xCORE, the red path is the data signal into the xCORE). Note that as the clock does not eminate from the DAC, there is some delay caused by the CODEC in the red path. Note that the reason it is source sychronous is because the BLCK is in this setup resampled at the pin.

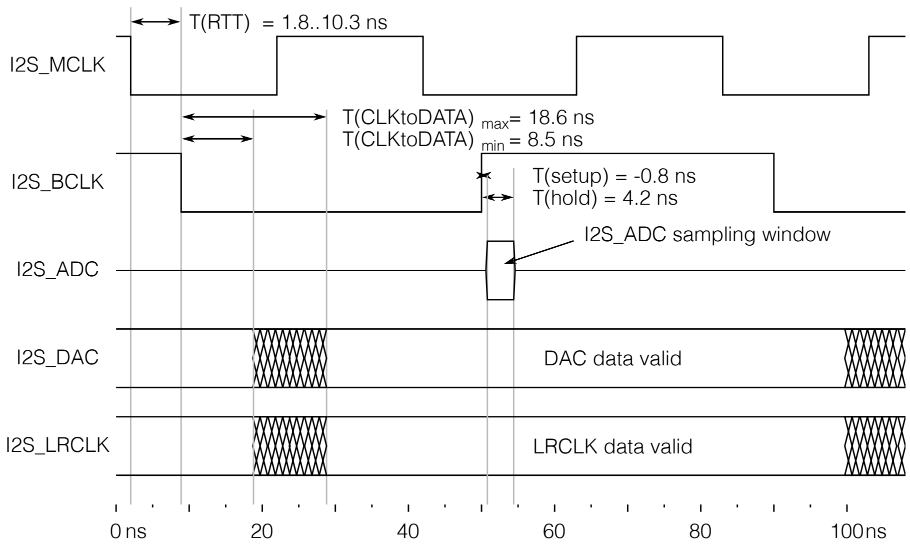

Fig. 19: I2S signal timings

Hence, I2S_BCLK is a round-trip-time behind I2S_MCLK. In addition, I2S_LRCLK and I2S_DAC will be delayed relative to I2S_BCLK (Section 4.2), and I2S_ADC will incur a small skew (Section 3.1). The time between the MCLK edge and the BCLK edge is variable amongst voltage, process, and temperature, but constant in a given environment.

The timings are shown in Fig. 19. Since the external CODEC will typically sample the I2S_DAC on the rising edge this will have a good margin (close to 20 ns); the CODEC will typically present the ADC on the falling edge which has a larger margin (around 40 ns).

Note also that as I2S_MCLK is resynchronised to XCORE.AI, I2S_BCLK will be running at 12.288 MHz, but *synchronous to the core-clock*. That is, I2S_BCLK will have a period of 81.380… ns, but this will be a sequence of clocks that are 80, 82, 82, 80, 82, 82, 80, … ns apart to, *on average*, create a clock of exactly 12.288 MHz. This introduces jitter on the BCLK; this normally does not matter, but if the CODEC is sensitive to jitter on I2S_BCLK then it can be resynchronised to the master clock using a D-type flip flop clocked by I2S_MCLK. Note that doing so will alter the setup, hold, and clock-to-data timings, because the clock edges of BCLK are shifted.

A Appendix - Timing components

The tables below show the expected worst case minimum and maximum Round-Trip-Times for specific groups of pins and ports, and for specific capacitances and drive strengths. In the case of 4-tile devices, tile 2 behaves the same as tile 0 and tile 3 the same as tile 1. All times are given in ns (where 1 ns = 0.000,000,001 s).

The minimum round trip time is defined as the fastest possible input path plus the fastest possible output path. The maximum round trip time is defined as the slowest possible input path plus the slowest possible output path. You must sum together the capacity dependent component and the pin-dependent component. Paths can be slow or fast depending on the following parameters:

- Variation in the process will cause some devices to be slower and some devices to be faster. The values shown in the tables below assume the fastest possible silicon, and the slowest possible silicon. Typical silicon at nominal voltages will have an RTT of 0.3-3.1 ns plus 2.6ns to drive the output (1V8, 5pF, 8mA) and a skew of 0.7-0.8 ns.
- Variation in temperature will cause a device to run slower or faster. For the IO measurements, hot devices will run slowest, and cold devices will run fastest. Designs that run in a controlled temperature (for example, always hot, or always cold) will have RTTmin and RTTmax times that are closer together.
- Variation in core voltage and (to a much lesser degree) IO voltage will cause a device to run slower or faster. The minimum time is for 0.81V core and 1.62/3.0V IO, and the maximum time is for 0.99V core and 1.98/3.6V IO. Designs with tightly controlled voltages (eg, 0.9V +/- 2%) will have RTTmin and RTTmax times that are closer together.
- Some IO pins are systematically faster than others. For example, using pins X0D12..X0D23 to input data using an internal clock will result in an eye that is 2.8 ns smaller than picking random pins from X1D00..X1D71.

We assume no slew-rate control on the output, no Schmitt-Trigger on the input, and no pull-resistors applied. Extra time should be allowed for external loads, and this will add to both minimum and maximum Round-Trip-Times. Skew may also increase with an increased external load.

A.1 Example combined timings

In order to compute worst case round-trip-times, and skews for 1.8V IO, 8 mA drive strength, 5 pF load we combine the pin dependent timings with the Round trip dependent timings:

Component	RTTmin	RTTmax	Tiskew	Toskew
Pin dependent	0.2	4.5	0.9	1.2
Load dependent	1.6	5.8		
Total	**1.8**	**10.3**	**0.9**	**1.2**

These are the times we have used throughout the document, they can be refined by using the appopriate values from the tables in the rest of this appendix

A.2 Pin dependent timings

The tables below show the pin dependent round trip times and maximum input and output skew. In order to calculate a complete round trip time you must add the capacitance dependent part of the round-trip-time as listed in the next section.

Overall timings:

Pins	Ports	RTTmin	RTTmax	Tiskew	Toskew
Any IO pin		0.2	4.5	0.9	1.2

Timings per tile:

Pins	Ports	RTTmin	RTTmax	Tiskew	Toskew
X0D00..X0D71	All	0.2	2.5	0.5	0.8
X1D00..X1D71	All	0.4	4.5	0.9	1.2

Timings of pairs of banks, each pair comprising eight 1-bit ports and a 16-bit port:

Pins	Ports	RTTmin	RTTmax	Tiskew	Toskew
X0D00..X0D23	1A..H 16A	0.5	2.2	0.3	0.5
X0D24..X0D43	1I..P 16B	0.4	2.5	0.5	0.8
X1D00..X1D23	1A..H 16A	0.4	4.5	0.9	1.1
X1D24..X1D43	1I..P 16B	0.4	3.3	0.8	1.2

Timings of individual banks, each bank comprising four 1-bit ports and an 8-bit port (two 4-bit ports):

Pins	Ports	RTTmin	RTTmax	Tiskew	Toskew
X0D00..X0D11	1A..D 4A..B 8A	0.6	2.2	0.3	0.5
X0D12..X0D23	1E..H 4C..D 8B	0.5	1.8	0.3	0.4
X0D24..X0D35	1I..L 4E..F 8C	0.4	2.2	0.4	0.7
X0D36..X0D43	1M..P 8D	0.6	2.5	0.5	0.8
X1D00..X1D11	1A..D 4A..B 8A	1.1	4.5	0.9	1.1
X1D12..X1D23	1E..H 4C..D 8B	0.4	1.8	0.2	0.7
X1D24..X1D35	1I..L 4E..F 8C	0.4	1.7	0.5	0.5
X1D36..X1D43	1M..P 8D	0.6	3.3	0.8	1.2

A.3 Capacitance dependent part

These are the minimum and the maximum round-trip-times in nanoseconds of the external signal. They depend on:

- The capacitance of the net that is driven (5/10 pF, extrapolate for other values)
- The drive strength (2-12 mA)
- The IO voltage (1.8 or 3.3V nominal)

You must add the pin dependent component to these round trip times to calculate a final round trip time.

VDDIO	Load	Drive	RTTmin	RTTmax
1.8 V	5 pF	12 mA	1.6	5.7
1.8 V	5 pF	8 mA	1.6	5.8
1.8 V	5 pF	4 mA	1.7	6.2
1.8 V	5 pF	2 mA	2.0	7.2
1.8 V	10 pF	12 mA	1.7	6.1
1.8 V	10 pF	8 mA	1.8	6.3
1.8 V	10 pF	4 mA	1.9	6.8
1.8 V	10 pF	2 mA	2.4	8.4
3.3 V	5 pF	12 mA	1.5	8.4
3.3 V	5 pF	8 mA	1.6	8.7
3.3 V	5 pF	4 mA	1.7	9.0
3.3 V	5 pF	2 mA	2.3	9.9
3.3 V	10 pF	12 mA	1.7	9.0
3.3 V	10 pF	8 mA	1.8	9.4
3.3 V	10 pF	4 mA	2.1	9.9
3.3 V	10 pF	2 mA	3.0	12.1

To design an interface with a tight window, use 1.8V IO, nets with a low capacitance and set the IO driver to utilise a high drive-strength.