AN02023: xcore.ai Power Consumption Estimation

Publication Date: 2024/9/24
Document Number: XM-014234-AN v1.0.0

IN THIS DOCUMENT

1 Introduction

This application note discusses the power consumed by *XMOS xcore.ai* devices and methods for optimising for power. The datasheet contains power figures that enable system designers to size their power supply, and estimate power consumption in the field. Real power consumption depends very much on device use, and this note an attempt is made to explain the variables, and how the system designer can optimise power use.

Note: All figures in this application note are measured at room temperature in typical conditions

1.1 Power consumption

xcore.ai devices consume power from the following supplies, not all of which may be bonded out on all of the package variants:

- 0V9 supplies:
  - Core supply (*VDD*). This is the main supply of the package, and is used to run the core processing tiles, and the *xCONNECT* switch.
  - USB core supply (*USB_VDD*). This is only bonded out on some packages with USB enabled, and is used to supply the digital side of the USB PHY.
  - MIPI core supply (*MIPI_VDD*). This is only bonded out on some packages with MIPI enabled, and is used to supply the digital side of the MIPI PHY.
  - 0V9 analogue supply (*PLL_AVDD*, *PLL_AVDD2*). This supplies power to the on-chip PLL(s), which is used to provide the clock to the processor tiles, or to create an application clock
- 1V8 supplies:
  - USB analogue supply (*USB_VDD18*). This is only bonded out on package with USB enabled, and is used to supply the analogue side of the USB PHY.
  - MIPI analogue supply (*MIPI_1V8*). This is only bonded out on package with MIPI enabled, and is used to supply the analogue side of the MIPI PHY.
- 3V3 supplies:
  - USB analogue supply (*USB_VDD33*). This is only bonded out on package with USB enabled, and is used to supply the analogue side of the USB PHY.
- IO supplies (VDDIO*). These supply all IO pins; the xCORE device itself uses hardly any IO power, and normally external devices dominate the power required on this supply.

2 Power consumption estimation

The power required by the various components is expressed as two quantities:

- For components where the power depends on the operating frequency the power is measured as Watts normalised to the operating frequency in Hertz: Watts/Hz. This is equivalent to the number of Joules required to perform a single operation. Multiplying the number of Joules with the core frequency yields the Watts required for a sustained operation.
  Because the numbers are all very small, this note typically use microWatts per mega-Hertz uW/mHz (equivalent to pico-Joules or pJ which is 10^{-12} J)
- For components where the power is independent of the operating frequency it is expressed in milli-Watts, mw, 10^{-3} W.

For example, a component that consumes 4 mW statically and 200 uW per mHz per operation will, at 500 MHz, require $4 \times 10^{-3} + 200 \times 10^{-12} \times 500 \times 10^{6} = 104 \times 10^{-3} = 104$ mW.

2.1 Core supply

The core power requirements are highly variable and depend on the following factors:

- Operating frequency of each physical core
- Type of computation, utilisation, and data values operated on
- Switch operating frequency
- PLL operating frequency
- Leakage

The dynamic elements are estimated in the table below with examples for total power consumption with a single or dual tile system at 600 and 800 MHz.

Element	uW per MHz	mW at 1 x 600	mW at 2 x 800
Base physical core	50	30	2 x 40
Computation 100% Scalar	U x 220	U x 132	U x 2 x 176
Computation 100% VPU	U x 530	U x 318	U x 2 x 424
Switch power	30	18	24
PLL power	6	4	5

In this table *U* is the fraction of the time that the processor is utilised. The scalar power consumption works out as 220 pJ per instruction, which can execute a single integer or floating point multiply-accumulate. The vector power consumption works out as 16.5 pJ per byte-multiply-accumulate, or 2 pJ per bit multiply-accumulate.

The switch power can increase when the switch is heavily used; in typical applications the switch is used for relatively small amounts of traffic, compared to the throughput of the switch.

In addition to the dynamic power, the chip requires static power of approximately 5 mW (single and dual tile). This is irrespective of the frequency at which the chip runs, but is temperature dependent: leakage goes up for higher temperatures.

With these figures, we can work out how much power *xcore.ai* will use approximately in different configurations and with different usage scenarios. The tables below lists power consumption for four example conditions (idle, using half the cycles for scalar activity, using all cycles split over the scalar and the vector unit, or using all cycles on the vector unit). The first table shows power consumption in high performance modes, for a single tile at 500 Mhz (for small applications), dual tiles at 600 Mhz, and dual tiles at 800 MHz (compute intensive applications):

Clock freq and #tiles	1 x 500		2 x 600		2 x 800	
Condition	MMACS	mW	MMACS	mW	MMACS	mW
Idle		33		87		114
50% scalar	500	88	1,200	219	1,600	290
50% scalar 50% vector	8,500	221	20,400	537	27,200	714
Flat out, 100% vector	16,000	298	38,400	723	51,200	962

The 'MMACS' column is the Millions of Multiply Accumulate Operations that can be performed every second in this scenario, where we assume that the vector unit is performing byte-multiply-accumulate operations, and the scalar unit is performing floating point or long integer multiply-accumulates. We assume that the switch is disabled in the single-tile scenarios and runs at full speed in the dual-tile scenarios.

We can go to lower-power modes where we disable the PLL, and run systems directly from the oscillator. The table below shows the low-performance modes:

Clock frequency Mhz	24		12		4	
Condition	MMACS	mW	MMACS	mW	MMACS	mW
Idle		6		6		5-6
50% scalar	24	9	12	7	4	6-7
50% scalar 50% vector	408	15	204	10	68	7-8
Flat out, 100% vector	768	19	384	12	128	7-8

Although these are only running at 4-24 MHz, the computational capability of the device is still up to 768 MMACC/s at this power level.

2.2 USB supplies

The power usage of the USB PHY (if enabled by the software) depends on the type of traffic. Approximate numbers are:

- HS mode 60 mW
- FS transmit 36-95 mW
- FS receive 16 mW

The power is spread across 3V3, 1V8 and 0V9 supplies. FS transmission costs are highly dependent on the cable length and load.

2.3 MIPI supplies

The power usage of the MIPI PHY (if present on the package and enabled by the software) depends on the type of traffic. Receiving a MIPI stream takes 30-35 mW.

2.4 IO supply

The IO power consumption depends mostly on the frequency at which IO signals are toggling, and on the selected IO voltage. Each I/O line consumes *C x f x V x V* mW, where *C* is the capacitance, *f* is the frequency of the signal, and *V* is the IO supply voltage. For example, for a GPIO pin with *V=1.8*, *f=10 MHz*, and *C=10 pf*, the power consumption is 32 uW.

3 Saving power

In order to save power, one can systematically go through each of the component and reduce their power consumption.

3.1 Switch

When the switch is not used it can be clocked down to a run at a low frequency. The switch is normally used to:

- Communicate between tiles within a node.
- Communicate between nodes.
- Control the node, such as the USB PHY, MIPI PHY, LPDDR, PLL, etc. This typically requires very little bandwidth and latency is not a big issue. The switch can safely be clocked down by a factor of 10.

If none of the above are required (for example, for a single tile design that does not use USB), the switch clock divider can be set to 255, reducing the switch power consumption to a negligible value. Note that this will make writes to control registers in the switch slow.

3.2 Reducing core power consumption

Assuming that computation is required continuously, the most effective way to reduce the power consumption of the core is to reduce the clock frequency.

1. Compute the minimum clock required across all threads
2. Round this up to a multiple of 100 MHz
3. Set the core clock to this frequency.

Further savings can be made by lowering the frequency to 100 MHz or below, but care must be taken that the reference clock is always at most half the core clock frequency.

If the threads have wildly different clock requirements, it is highly beneficial to optimise the software on the thread that requires the fastest clock; which will enable the whole chip to run slower. Methods for optimisation include passing `-O3` to the compiler, or rewriting core loops in dual issue assembly code, or indeed splitting code so that all threads require similar MIPS rates.

Power consumption is data dependent. Power may be reduced by making sure that all signed fixed point values are left aligned (representing a 16 bit value for +1 and -1 by 0x00010000 and 0xFFFF0000, instead of 0x00000001 and 0xFFFFFFFF, reducing the number of bits that flip by a factor of two).

Needless to say, polling should be avoided at any cost, as it will increase utilisation to, eventually, 100%. If a process has no work to perform, it should wait for a resource, for example using an input-function on the resource or by waiting in a `SELECT` construct.

3.3 PLL supply

The PLL requirements can be minimised by setting it to the lowest whole multiple of the desired clock frequency that is greater than 260 MHz. If a very low clock frequency is required, then the PLL can be set in by-pass, taking the clock directly from the Oscillator to the core. At that point, the PLL can be set to a very low frequency (say 10 MHz), meaning it will take virtually no power. Details on this are in AN02022.

3.4 Reducing USB PHY power requirements

If the USB PHY is used, it consumes more data when transmitting data; especially when transmitting '0' data. Power consumption can be reduced by transmitting as little data as possible (for example by reducing the polling frequency of interrupt endpoints).

3.5 VDDIO supply

The *VDDIO* power requirements can be reduced by using 1.8V IO rather than 3.3V. This reduces IO power by more than 70%.

3.6 Reducing the core voltage

xcore.ai devices are characterised down to 0.855V. Supplying 0.855V instead of 0.90V will reduce power consumption by just under 10%.

Any frequency dependent part of the power consumption is a function of the voltage squared; hence reducing the voltage by 5% yields a 9.75% decrease in power requirements: $0.95^2 = 0.9025$.

The static consumption of the chip is a more complex function of the voltage, and reduces slightly faster than the square.

Reducing the voltage supply below 0.855V yields savings in accordance with the above formulas; but *xcore.ai* devices have not been characterised below 0.855 V.

4 Further reading

- The XS3 architecture manual
- XU316-1024-QF60A datasheet
- XU316-1024-QF60B datasheet
- XU316-1024-TQ128 datasheet
- XU316-1024-FB265 datasheet
- AN02022: xcore.ai Clock Frequency Control

5 Document history

5.1 AN02023 Changelog

1.0.0

- Initial public release