Optimising a better solution

Architectural optimisation as a tool for low cost low power solutions. By **Asher Hazanchuk**.

The need for low cost and low power consumption drives semiconductor manufacturers towards making devices on advanced processes. But it is possible to reduce cost, size and power consumption through architectural optimisation. These non expensive optimisation methods are often overlooked and this is a missed opportunity for further cost and power reduction.

Cost and power reduction

According to IBS, the average cost of developing a chip has risen from $45million at 65nm to $150m at 22nm. This higher investment eventually pays for itself by lowering chip cost. The main reason that semiconductor companies develop chips on advanced process nodes is the promise of cost and power consumption reductions.

In general, device cost decreases in the more advanced smaller process nodes, although it takes time until the process node matures and achieves better yield performance than the previous node. For this reason, it takes time for the minimum cost point to move the last two state of the art advanced process nodes. In fig 1, Fp is the process cost optimisation factor and, in this example, is the process cost optimisation factor between a device made on a 65nm process (point A) and the same part made on a 45nm process (point B) without architecture optimisation. Meanwhile, Fa is the architecture cost optimisation factor. In fig 1, this is the cost of a device made on a 45nm process (point B) without architecture optimisation and with architecture optimisation (point D). The best way of minimising cost is to switch to a 45nm process and employ architecture optimisation.

In fig 2, the minimum power consumption is achieved through moving from a 32nm process to the 22nm node (points A and B) and through architecture optimisation at 22nm (point D).

Symmetry

In many wireless and video algorithms, application special characteristics can be exploited for architectural optimisation. For example, it is common to have finite impulse response (FIR) filter based algorithms and applications with symmetrical coefficients (fig 3).

Symmetrical FIR filters have an even number of taps and each coefficient at the same distance from the centre has the same value. It is therefore possible to do one multiplication with the sum of the two samples related to the same two coefficients in a specific time. This halves the number of multipliers and the related logic required for the FIR filter implementation.

While FIR filters with odd symmetry have an odd number of taps, each two coefficients still have the same distance from the centre and the same value; only the centre coefficient doesn't have another tap with the same value. The number of multipliers required to implement an odd symmetry FIR filter can be reduced by 2n/(n+1), where n is the number of taps. This reduction factor closes on 2 as the number of taps grows.

An additional characteristic can be exploited in half band symmetric FIR filters. Here, each second coefficient, except for the centre coefficient, is zero. Since these filters are also odd symmetric FIR filters, it is possible to reduce the number of required multipliers by up to a factor of four.

Operations with 2d symmetry are very common in video applications. In fig 4, the 5x5 2d FIR filter's vertical and horizontal symmetry means each circle with same colour represents a coefficient with the same value. It is possible to exploit 2d symmetry so the same multiplier can be used for all pixels with the same coefficient value. As shown in fig 4, there are cases of 4, 2 and 1 pixels with the same coefficients. Therefore, the potential exists for the size of a generic matrix to be reduced by up to four times.

Complex multiplication

Significant portions of wireless and communications applications are based on algorithms featuring complex numbers, such as

Fig 1: Device cost optimisation

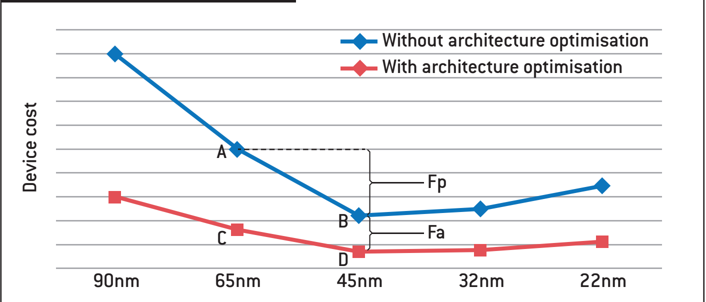


Fig 2: Device power consumption optimisation

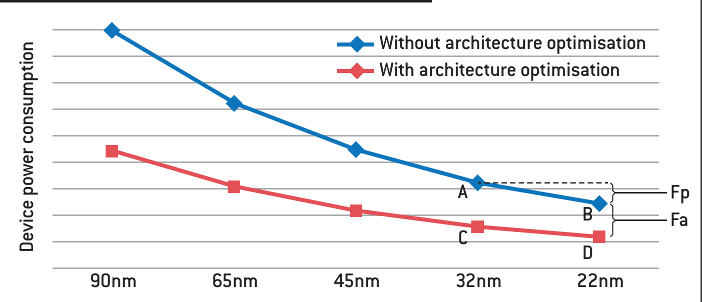
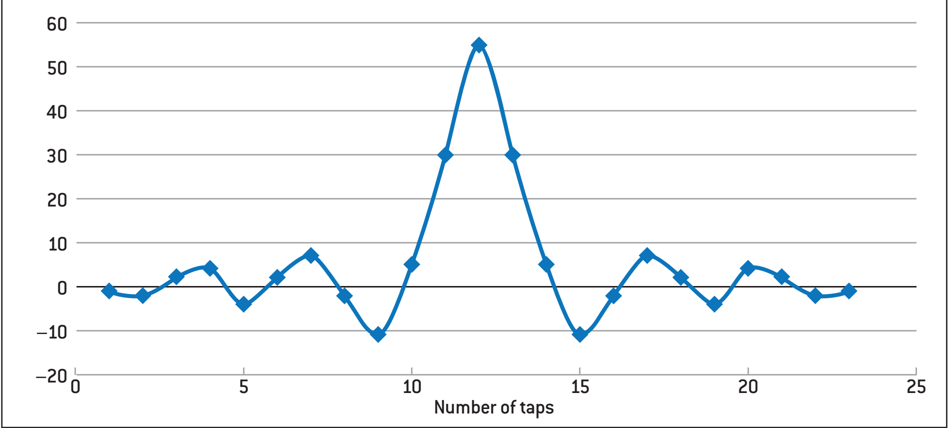


Fig 3: A symmetric FIR filter



I and Q channels and symbol modulation.

Complex multiplication between complex sample (a + jb) and complex coefficient (C + jD) could be implemented with four multiplications:

(a + jb)*(C + jD) = (a*C – b*D) + j(a*D + b*C)

With some algebraic manipulation, the same result can be obtained using three multiplications.

= a*C – b*D + (a*D – a*D) + j(a*D + b*C + (a*C – a*C))
= a*(C+D) – (a+b)*D + j(a*(C+D) + (b-a)*C)

Since the silicon area required to implement a multiplier is significantly larger than that required to implement adders and subtractors, this allows for size reduction.

Double data rate

Double data rate throughput optimisation is a special architectural technique designed to overcome fabric throughput issues common to fpgas. While the dsp slices in an fpga have a similar size efficiency to those in an asic made on a similar process node, they have a flexibility advantage. The problem is that, in many cases where fpga utilisation is high, the fabric's relatively lower operation frequency creates a throughput bottleneck.

The LatticeECP4 fpga has innovative throughput boosting interfaces embedded into the dsp slices. This enables the part to offer double the throughput of other fpgas. Using this feature means the LatticeECP4 can implement complex dsp functions using half the number of multipliers that would be required by other fpgas. This optimisation enables significant system cost and size reduction, as well as decreased power consumption.

In many cases, different architectural optimisation techniques can be combined to achieve higher levels of cost and power reduction.

Most FIR filters implemented in wireless or video applications are symmetric. Implementing a 64 tap symmetric FIR filter with an input data rate of 245.76Msample/s in a typical fpga would require 64 18 x 18 multipliers. The LatticeECP4 can implement the same FIR filter using 16 18 x 18 multipliers, approximately four times smaller. This provides other benefits, including an approximate halving of power consumption and the opportunity to fit the design into a smaller fpga, which further reduces cost.

Similarly, half band filters and double data rate optimisation could be implemented in other digital up and digital down converter interpolation or decimation filters. In these cases, the cost and power savings are even higher – approximately eight and four times respectively.

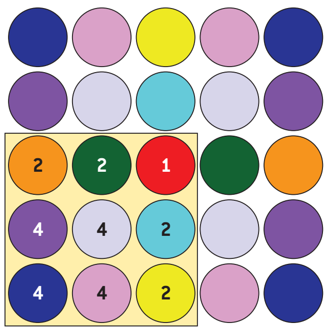
Summary

Architectural optimisation is a relatively inexpensive way to reduce silicon device cost and power consumption. There is no reason why semiconductor companies or their customers should not take advantage of these 'low hanging fruits' instead of investing in developing devices targeted at expensive advanced process nodes.

In many cases, a combination of different methods of architecture optimisation – such as data flow optimisation, algorithm characteristics optimisation or/and algebraic optimisation – can result in cost being halved and power consumption being reduced by a factor of eight.

Author profile:
Asher Hazanchuk is product planning manager with Lattice Semiconductor (www.latticesemi.com).

Fig 4: Video 2d symmetry



Even symmetry definition
Y(i) = h(1)*X(i+n) + h(2)*X(i+n-1) + Ɛ + h(n-1)*X(i+2) + h(n)*X(i+1)
Coefficients: h(1)=h(n), h(2)=h(n-1), ... {n is even}
Therefore:
Y(i) = h(1)*[X(i+n) + X(i+1)] + h(2)*[X(I+n-1) + X(I+2)] + ... + h(n)*[X(i+n/2) + X(i+n/2+1)]

Odd symmetry definition
Y(i) = h(1)*X(i+n) + h(2)*X(i+n-1) + ... + h(n-1)*X(i+2) + h(n)*X(i+1)
Coefficients: h(1)=h(n), h(2)=h(n-1), ... {n is odd}
Therefore:
Y(i) = h(1)*[X(i+n)+X(i+1)] + h(2)*[X(i+n-1)+X(i+2)] + ... + h(n-1)*[X(i +(n+1)/2 +1) + X(i +(n+1)/2 -1)] + h(n)*X(i+(n+1)/2)]