



# Simply symbiotic

Has the definitive example of embedded programmable logic arrived? By **Philip Ling**.

**T**he idea of burying programmable logic in an asic or a standard device isn't new, it has been around long before the current (and converse) vogue for embedding standard logic into a programmable fabric took hold.

Various attempts have been made, though many failed to meet expectations. The problem seems to stem from the fact that as soon as the logic becomes smothered by a larger and more important piece of silicon, it loses some sparkle.

Very high performance discrete fpgas sit at the upper end of the price scale largely because it takes a lot of silicon area to be that good. Shrink wrapping that into a device that has been optimised for cost and space, is contrary to getting a high level of performance or flexibility.

For this reason, the embedded logic variant needs to be a little more focussed than the discrete version. Nevertheless, the potential benefits of embedded programmable logic are clear. For example, embedded logic could be used to alter the operation of the device, or in another configuration used to provide some on chip 'glue logic'.

Examples of larger, more complex programmable logic based devices used to accelerate single functions are not hard to find either. It is an application that epitomises the strength of an fpga – the ability to dedicate large amounts of gates to accelerating a single function using multiple and parallel arrays of hardware. Here, there are two paths to take; that of

parallel arrays of configurable logic, and parallel arrays of processing engines. There are early success and failure stories for both.

The problem is, it isn't absolutely clear why any of these examples succeed or fail, at least from a technology point of view. Ultimately it comes down to market acceptance, which is why the latest example will be watched with interest.

## Best of breed

Fabless startup Stretch Incorporated ([www.stretchinc.com](http://www.stretchinc.com)) is using a combination of processor core and programmable logic. Rather than force the two to co-operate, Stretch has grafted them together, making them almost interdependent.

Unexpectedly, it hasn't opted for the industry's 'standard' embedded core (ARM), or for a high performance core

with added hooks for customised instructions (MIPS). Instead, it has chosen a core specifically designed to be extensible, Xtensa from Tensilica. An array of arithmetic units and multiplier units have been grafted within the core, which Stretch refers to as the Instruction Set Extension Fabric. The arithmetic/multiplier units can be configured (in software) to create various instruction lengths, from a single bit up to (theoretically) three 128bit words. Although Stretch isn't divulging many details of what, exactly, the array is comprised, it describes it as being closer to a cpld than to an fpga.

To complement the fabric, proprietary instructions have been added to the Xtensa core to implement wide load/store operations between the fabric and the processor. In addition to these specific instructions which are used

Figure 1: Standard C colour conversion

```
#include "rgb2ycbcr.h"

void
rgb2ycbcr(char r, char g, char b, char *y, char *cb, char *cr)
{
    *y = ( 77*r + 150*g + 29*b      ) >> 8;
    *cb = (-43*r - 85*g + 128*b + 32768) >> 8;
    *cr = (128*r - 107*g - 21*b + 32768) >> 8;
}

This function would be called using this:

char RGB[3 * NP], YCbCr[3 * NP];
int i;
...
/* loop over RGB data, converting 1 pixel at a time */
for (i = 0; i < 3 * NP; i += 3) {
    rgb2ycbcr(RGB[i], RGB[i+1], RGB[i+2], &YCbCr[i], &YCbCr[i+1], &YCbCr[i+2]);
}
```



Figure 2: Stretch approach to colour conversion

```
#include <stretch.h>

/* Extension instruction converting 5 pixels */
/* SE_FUNC tells scc to map rgb2ycbcr to a single instruction */
SE_FUNC void rgb2ycbcr(WR A, WR *B)
{
    se_sint<8> r[5], g[5], b[5];
    se_sint<8> y[5], cb[5], cr[5];
    int i, j;

    /* unpack A to RGB data, does not use any ISEF logic */
    for (i = 0; i < 5; i++) {
        j = i * 3 * 8;
        r[i] = A(j+7, j);
        g[i] = A(j+15, j+8);
        b[i] = A(j+23, j+16);
    }

    /* converting 5 pixels */
    for (i = 0; i < 5; i++) {
        y[i] = ( 77*r[i] + 150*g[i] + 29*b[i] ) >> 8;
        cb[i] = (-43*r[i] - 85*g[i] + 128*b[i] + 32768) >> 8;
        cr[i] = (128*r[i] - 107*g[i] - 21*b[i] + 32768) >> 8;
    }

    /* pack YCbCr to B */
    *B = (cr[4],cb[4],y[4],cr[3],cb[3],y[3],cr[2],cb[2],y[2],cr[1],cb[1],y[1],cr[0],cb[0],y[0]);
}
```

specifically to invoke the array, an optimised compiler allows the engineer to create further instructions, which are transformed in to hardware and executed in the fabric, as the program is executed.

As a result, the fabric is limited to performing compute functions that could be carried out by the core, albeit a lot slower, as opposed to being truly flexible logic as, say, an fpga would be. But in doing this, Stretch has overcome the penalties of embedding programmable logic - that of space and interconnect complexity - in favour of increased system performance. The result is that it brings parallelism to an otherwise sequential process.

It has achieved this without the need for a very long instruction word (vliw), which have proven to be difficult to fill

consistently. Neither is it limited to the 'single instruction multiple data', or simd, technique, which has migrated out of complex architectures down to some risc processors.

The advantage of implementing instructions in hardware this way is it can remove iterative load/execute/store cycles on the same data. Modern processors use cache to alleviate off chip loads and stores, while ideally data would only be fetched once, executed upon and then stored. Thanks to the configurability of the fabric, Stretch's ceo and co-founder, Gary Banta, believes it invariably achieves this.

#### Intensive care

Banta describes it as being targeted at 'compute intensive applications, where a lot of maths has to be done on long data

words, or huge amounts of maths has to be performed on short data words'. Typically this includes: image processing or manipulation in consumer audio/video equipment; CAT scan and ultrasound medical electronics; as well as compute intensive parts of the ubiquitous telecomms/wireless/networking and military markets. As an example, a security and medical scanning application, which took six general purpose processors and six fpgas, was implemented using a single general purpose processor and one Stretch S5000. It resulted in an 80% price reduction in the bill of materials and an increased performance from four to six frames/sec.

A further advantage of Stretch's approach is that it doesn't require any hardware description coding. Everything is compiled from a standard C program using its optimised tool flow, all that changes is the programmer needs to write the code with the fabric in mind. Banta claims the number of rules the programmer has to learn are few, taking less than a day to appreciate.

By way of example, two alternative blocks of code are shown (figure 1 and figure 2), using colour conversion from RGB to YCbCr to illustrate the point. Figure 1 shows a standard approach, where one pixel is converted at a time. Figure 2 shows how this should be coded for the S5000 in a way that will invoke the fabric at compilation to convert five pixels in parallel.

As can be seen, the function written for the fabric is larger than the standard approach, but the result is a 50x improvement in performance.

Further benchmarks have already been performed and validated by EEMBC, which put the S5000 ahead of Texas Instruments' TMS320C6, and the FastMATH device from Intrinsity - which runs at 2GHz - for its Telemark benchmark (go to [www.eembc.hot-desk.com](http://www.eembc.hot-desk.com) for more details). It looks as though Stretch has managed to achieve what companies including BOPS and Adaptive Silicon couldn't, match innovation with performance, but without sacrificing ease of use. **NE**