

White Paper

The Impact of the \$1 32-bit Microcontroller

Simon Glass

Paper Presented to Austronics, Melbourne 2006

Bluewater Systems Limited
PO Box 13 889
Christchurch
New Zealand

Tel. +64 3 377 9127
Fax. +64 3 377 9135
Aus. 1800 148 751
solutions@bluewatersys.com

www.bluewatersys.com

Simon Glass

Simon Glass has been using ARM technology since 1987 and has worked as a software engineer in embedded electronics. He has held several positions at the chip design company ARM PLC in Cambridge, UK and Austin, Texas. Returning to his home country of New Zealand, he set up Bluewater Systems in 1996 to promote the benefits of ARM technology. The company is now devoted to designing and implementing ARM-based embedded systems. Simon Glass holds a B.Sc (Hons 1) in Computer Science from Canterbury University, Christchurch, New Zealand.

Table of Contents

Simon Glass.....	2
Rise of the Microcontroller.....	3
The push for 8-bit.....	3
Limitations of 8-bit.....	4
The \$1 ARM Micro.....	4
Architectural Innovations.....	5
What do you get for \$1?.....	6
What can a 32-bit \$1 micro do?.....	8
Productivity.....	8
Impact of the \$1 32-bit microcontroller.....	10
Applications.....	12
Conclusion.....	13

Rise of the Microcontroller

Only a few brave souls would argue that the microcontroller has become a major force in the modern world, its impact felt in transportation, the home and in government.

For example, in the UK, micro-controlled trains ferry passengers around London's docklands and Stansted Airport. Washing machines, dishwashers and microwaves automate mundane tasks. The police state continues apace, with speed cameras and license plate recognition systems automatically issuing fines and collecting revenue for the government without any human intervention.

In the consumer market, the microcontroller has enabled gadgets such as cellphones, music players and digital cameras. Computing power has dramatically increased in these devices. A modern smart phone has the computing power of a 2000 PC ignoring the signal processing technology, and significantly more if this is taken into account. Instead of requiring dozens of watts of power, these devices survive on milliwatts with battery life of a week or more. ARM introduced a single chip dual CPU approach years before AMD and Intel, suggesting that the innovation in microprocessor design is no longer where it once was.

And yet, of the roughly 8 billion microcontrollers shipped in 2005, almost half are what the author calls crappy little 8-bit micros. Not for these are the glamorous image processing and music playing systems of the modern age. Legions of dedicated programmers shoe-horn their code into tiny memories, fiddle with fragmented address spaces and resort to assembler code early and often. This is the real digital divide: between the haves, with their 32-bit leading edge technology and development tools, and the have nots, largely making do with the microcomputer industry's cast-offs from the 1980s.

How did the nasty little 8051 come to dominate the microcontroller world? Does it offer an unbeatable combination of features combined with low power consumption, high performance and great development tools? Not, really, no. But it is cheap!

Readers may complain that this analysis is not entirely fair. There have been several attempts at designing a low cost 8 or 16-bit microcontroller from the ground up. But most micros in common use hail from the 1980s: 8051, Z80, 6809 to name a few. In this paper the author will argue that 8/16-bit micros are past their use-by date.

The push for 8-bit

It is tempting to conclude that the wide use of 8-bit microcontrollers in low-end applications is simply a cost-based decision. But that is too simple a view. Many engineers look at the need to watch a few switches and control a few LEDs and cannot bear to use anything other than an 8-bit micro. 'Anything else is overkill!' they shriek, shaking their heads. It's like commuting to work in a F-18 fighter jet.

Many 8-bit micros are not even that cheap. It is not uncommon to pay \$5 for something with the right peripheral set and enough I/O pins to do the job. No, something larger is at work here. It seems that many engineers are wedded to the idea of an appropriate and proportional response to a problem. In military terms, this might be like meeting infantry with infantry, rather than sending in the artillery.

To my mind this attitude is madness. Are you trying to win the war or battle on for years towards stalemate? Most engineers would agree they are under pressure to complete their project as soon as possible and with the minimum of fuss. Yet a 16 or 32-bit micro is definitely 'overkill' for the task?

Limitations of 8-bit

Let's briefly review some of the limitations of 8-bit micros.

First, they use 8 bits wide data paths, and thus are limited in the speed at which they can process numbers larger than 255. Many have 16-bit registers made by placing two 8-bit registers together. But in any case this makes for torturous code when dealing with any real-world numbers. C programmers can avoid this problem, although at the cost of blowing all their code space on a few careless 32-bit additions. Strange limitations may also apply, such as supporting only one parameter to a C function and being unable to get the address of some variables. In general, any C code which has been written for an 8-bit micro tends to look like like road-kill fairly quickly, as the programmer jumps through the various hoops to reduce code size and improve performance.

Second, they have limited and fragmented memory space. Code space may be architecturally limited to 64Kbytes or less, with a separately addressed data RAM, strange indexed addressing modes and obtuse limitations about what can be retrieved from where.

Third, despite their seemingly respectable clock speeds, on-chip dividers often result in very poor instruction throughput. Even ignoring the 8-bit limitation, many micros take 12 clock cycles to execute an instruction.

Fourth, despite their small silicon size, 8-bit micros are not particularly power efficient. There are some exceptions, but when you take the instruction throughput into consideration, most 8-bit micros use more power than an equivalent ARM micro doing the same task.

Fifth, 8-bit micros have poor code density. This has certainly improved recently, with C compilers becoming better at navigating the minefield of limitations and special cases in the architecture. But for many non-trivial applications, engineers resort to assembler to moderate the prodigious C compiler output. These micros are not good C targets, and companies such as Keil were initially derided for trying to produce an efficient compiler for the 8051, for example.

Sixth, tools support is often poor. ICE units are not always available, there is too much 'magic' behind the scenes in areas such as USB and serial/timer implementation, and the wide variations in architecture provide no guarantee that tools which work on one 8051 or 6809 implementation will work on another.

For all these reasons and many others, I believe the time has come to consign the 8-bit micro to history, even for the most trivial applications.

The \$1 ARM Micro

ARM is the world's leading supplier of silicon IP, accounting for around half the market. The ARM7TDMI released in 1994 set off a revolution in the cellphone market. Its excellent code density shaved a significant amount of code size for Nokia, partly due to an additional 16-bit instruction set which still provided 32-bit registers and addressing. Its efficiency allowed a clock speed of only 13MHz even with a massive increase in functionality from the previous generation, thus reducing power consumption. From 1998, using a cellphone was no more a matter of holding a heater to your ear and talking quickly before the battery gave out.

ARM's engineers were justly proud of the revolution they enabled, but several always felt they could do better. The ARM7TDMI was low power largely by accident, because of its inherent simplicity and small number of transistors. Future higher end cores such as ARM920T lead the way in low-power design with innovative cache design and power management. But they felt that the original ARM7 core could be improved upon, if designed from the ground up with all the accumulated

knowledge of 10 years in the market.

Three years ago, ARM's engineers got that opportunity. ARM approved a project to completely re-architect the ARM7TDMI. This 'microcontroller' project had the following aims:

- Small transistor count, therefore small silicon area and low cost
- Very low power consumption
- 32-bit performance and ease of use
- A microcontroller approach, with integrated timers and interrupt controller (a change of direction for the 'pure core' ARM)
- The best possible code density

Several of these aims conflicted, but last year ARM released the results of its efforts: Cortex-M3, the 'M' standing for microcontroller. This is an implementation of the new ARM v7M architecture, a variant of ARM v7. It includes a number of improvements but also some level of backwards compatibility to ARM v4T as used by ARM7TDMI and most current low-end ARM microcontrollers.

Architectural Innovations

There are a number of innovations in Cortex-M3, which taken together give it a unique place in the microcontrollers market. These are the vectored interrupt controller, processor state save/restore and the 'Thumb 2' instruction set.

The vectored interrupt controller allows the programmer to set up a table of addresses of interrupt routines along with priority information. When an interrupt occurs, the micro will set up a single 32-bit memory location with the address of the interrupt routine which needs to be executed next. This is the highest priority interrupt. To handle an interrupt, then, the micro simply needs to load the contents of this address into the PC (program counter), and continue. Once an interrupt has been handled, the routine can similar load the contents again to handle the next interrupt at the same or lower priority. This process continues until there are no more interrupts. The Cortex-M3 can switch between pending or higher priority interrupts in only 6 cycles (including switching state).

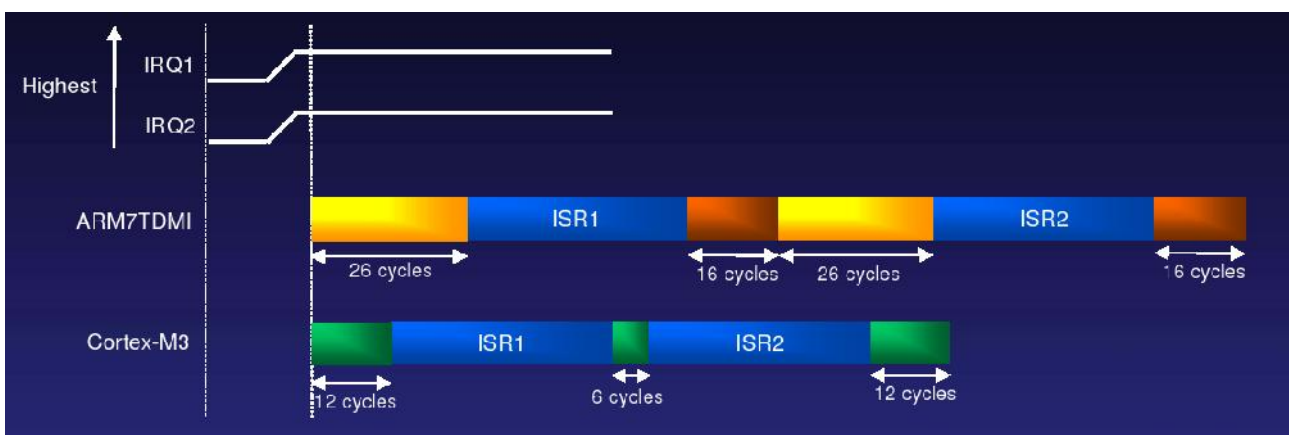


Illustration 1: The impact of the Cortex-M3 interrupt architecture. Not only is the state save/restore time shorter than ARM7TDMI, but it need not be done between the execution of two back-to-back interrupt routines. This saves significant time under heavy interrupt load.

Saving and restoring of processor state is a major overhead in simple microcontroller applications. The actual interrupt routine may only move a few bytes of data, and the cost of entry and exit from the routine may in fact take longer than the actual useful processing.

Cortex-M3 provides hardware support for saving and restoring this state, without requiring either software instructions or memory accesses. This is possible partly because the 6 processor modes provided by the full ARM v7 architecture have been reduced to two: interrupt and user.

In addition, Cortex-M3 automatically optimises back-to-back interrupt routines. Traditionally, processor state is restored on exit from one interrupt routine, then saved again before starting the next. Cortex-M3 avoids this overhead such that the state is saved and restored only once, at the beginning and end of the chain of interrupt routines. This feature has the side effect that it is no longer necessary to write interrupt routines in assembler to get the best performance.

The Thumb-2 16-bit instruction set builds on Thumb, which you may be familiar with, but has two major differences. Firstly it can 'stand alone', meaning that it supports all required processor state instructions. The original Thumb requires that interrupts and processor setup be implemented in 32-bit ARM code.

Secondly it improves performance of 16-bit code to almost reach 32-bit levels. Thumb's 16-bit code performance was 30% less than 32-bit code. This means that there is almost no execution time penalty (about 2%) for using Thumb-2 code, but a significant code space reduction (about 30%).

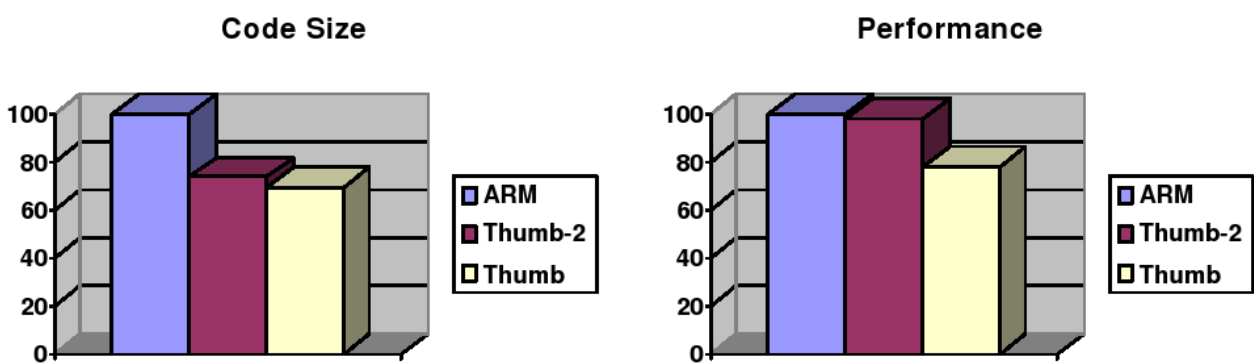




Illustration 2: When compared to Thumb, Thumb-2 offers slightly larger code size than Thumb, but without the performance overhead. In fact, performance is almost as good as ARM code

Other smaller changes are apparent in Cortex-M3, such as bit addressing, hardware divide, a single cycle multiplier, unaligned data access, optional memory protection unit and built-in sleep modes to save power.

What do you get for \$1?

A number of 8-bit microcontrollers have become available for US\$1 (or thereabouts), in particular the 8051 range. Despite their limitations these chips have been used in a wide range of applications from whiteware to motor control.

ARM-based microcontrollers have broken the \$5 barrier but have stubbornly stayed in that region since. Until now. The Cortex-M3 based Stellaris LM3S101 is US\$1 in 10k volume, a better price than a comparable 8051-based micro.

	
The Cortex-M3 based LM3S101	The Philips 8051-based LPC938

In an attempt to capture the differences between an 8051 part, a traditional ARM7TDMI part and a Cortex-M3 part, the following table has been compiled.

Feature	Philips LPC938	Stellaris LM3S101	Philips LPC2101
Digikey/Mouser US\$Price @5k	1.30	1.10	2.65
Core	8051	Cortex-M3	ARM7TDMI
Flash memory, KB	8	8	8
SRAM bytes, KB	0.25	2	2
Max clock, MHz	18	20	70
DMIPS ¹	1.37 ²	25	65
Current at max clock, mA	14	35	41
Power at max clock, mW	51.8	115.5	73.8
DMIPS/watt	26	216	880
DMIPS/MHz	0.076	1.25	0.92
GPIOs (max)	26	18	32
UARTs	1	1	2
Package	28-TSSOP	28-SOIC	LQFP48
Instruction cycle time ns	111	50	14
Clock required for 1 DMIPS, Mhz	13.1	0.8	1.07
Power required for 1 DMIPS, mW	37	4.6	1.1

The most interesting rows of this table are at the bottom. They show the clock speed and power required to achieve 1 DMIPS of performance. While the Stellaris part is not as power efficient as the Philips³, both are much more efficient than the 8051, in both dimensions. This applies even though the Philips 8051 implementation is very efficient compared to other 8051s (it can execute most instruction in 2 cycles instead of 12).

- 1 DMIPS standards for a Dhrystone MIPS, which is a measure of performance. It is obtained by running the standard Dhrystone 2.1 program, and dividing the result by 1757. This is arguably a useful benchmark for embedded CPUs. However, in the author's opinion it is somewhat biased towards 32-bit machines as some of the arithmetic is 16-bit or 32-bit. For an embedded system which only uses 8-bit numbers, the 8-bit machine would perform better than indicated. Comparison of 8-bit and 32-bit performance is a difficult subject unfortunately. However, the comparisons here are meaningful and apply particularly well for non-trivial embedded software (more than a few KB of code)
- 2 See http://www.dcd.com.pl/dcdpdf/dr8051_ds.pdf which quotes 0.153 DMIPS at 12MHz. Philips claim 6x the performance of the original 80C51 core, so we have assumed a performance of 0.153 x 18 / 12 x 6.
- 3 Luminary Micro plans additions to the range next year aimed at micro-amp power. For low power applications the ARM7TDMI-based Atmel SAM series offers sub-milliamp performance.

What can a 32-bit \$1 micro do?

Despite its 16-bit instruction set Cortex-M3 has an internal 32-bit data path and is to all intents and purposes a 32-bit machine. All registers are 32-bits wide as is the ALU (Arithmetic and Logic Unit).

Being a RISC machine, Cortex-M3 uses a load/store architecture. This means that instructions can either load data, save data or store data. However, there are no 'complex' instructions as on CISC machines with complex addressing modes that simultaneously load and process data. This means that RISC machine instructions are less powerful, but since they take fewer cycles to execute (generally one), the instruction throughput (as measured by Dhrystone MIPS for example) is higher overall. Cortex-M3 blurs this distinction slightly with its new atomic bit addressing instructions.

The multiplier can also deal with 32-bit multiplication and offers single cycle 16x16 multiplication.

Dealing with 32-bit data is less expensive in terms of code space. On an 8051 it can take $32 \cdot 42^4$ bytes of code to add two 32-bit numbers in RAM. On Cortex-M3 it takes 8 bytes (2 loads, add, store) assuming that the values are not used elsewhere or contiguous in memory, and less otherwise.

Calling a function on an 8-bit micro is extremely expensive (in time and code space) particularly when there are arguments to pass. These must be copied from their current location and stored onto the stack, then loaded from the stack within the called function. On Cortex-M3 the first four arguments are passed in registers, and a function call is a single instruction (BL or 'branch and link') involving no memory accesses. Provided you pass four arguments or less around, you may split your code up as you please with very little performance or code size penalty.

Productivity

Many tools vendors despair of trying to encourage Australasian companies to invest in decent tools. We have lost count of the number of fairly substantial companies who are using free or very cheap tools, despite paying market rates for their staff. Investing in good tools (when you can afford it) is a sure-fire way to reduce development time and cut risk. Of course, staff must know how to use these tools.

That said, Cortex-M3 has a number of productivity-enhancing benefits:

- Excellent tools are available at a reasonable price. For example, the RealView Microcontroller Kit is around AU\$6000, a little over a month's salary. This includes ARM's own compiler, the best in the industry, as used by Nokia, Symbian, ST Microelectronics, Texas Instrument, Bosch and many others.

4 Testing performed using the sdcc compiler



Illustration 3: ARM's RealView tools provide a level of functionality, reliability and performance not generally available with 8-bit micros

- There is no need to program in assembler. Even interrupt and exception handlers can be written directly in C

```
void irq_uart (void)
{
    unsigned int status, pass_counter = AMBA_ISR_PASS_LIMIT;
    uart_info *port = (uart_info *)UART0_BASE;

    status = readb (port->membase + UART0_IIR);
    if (status)
    {
        do
        {
            if (status & (UART0_IIR_RTIS | UART0_IIR_RIS))
                p1010_rx_chars(port);
            if (status & UART0_IIR_MIS)
                p1010_modem_status(port);
            if (status & UART0_IIR_TIS)
                p1010_tx_chars(port);
            if (pass_counter-- == 0)
                break;
            status = readb(port->membase + UART0_IIR);
        } while (status & (UART0_IIR_RTIS | UART0_IIR_RIS |
            UART0_IIR_TIS));
    }
}

```

Illustration 4: C code for an interrupt route - no special syntax or assembly code is required

```
irq_uart PROC
[]
    PUSH    {r4-r10,lr}
    LDR     r6,IL1,132I
    MOVS   r5,#5
    LDR     r0,[r6,#0]
    LDRB   r4,[r0,#0xc]
    CMP    r4,#0
    BEQ   IL1,124I
    LDR     r7,IL1,128I
    MOV    r8,#2
    MOV    r9,#3
IL1,82I
    TST    r4,#0xa
    BEQ   IL1,94I
    MOV    r0,r6
    BL    p1010_rx_chars
IL1,94I
    LSL   r0,r4,#31
    BEQ   IL1,102I
    STR   r8,[r7,#0] ; fred
IL1,102I
    LSL   r0,r4,#29
    BPL  IL1,110I

```

- Full 32-bit numbers are supported, reducing the contortions required to work with values larger than 256 or 65536. In particular this means that porting standard algorithms and open source code into Cortex-M3 is relatively painless.

ARM Cortex-M3 Code	8051 Code
<pre>C code: void main (void) { long int fred, john, mary; fred = getval (); john = getval (); mary = fred + john; } armcc generates: main PROC PUSH {lr} BL getval MOV r2,r0 BL getval LDR r1, L1.36 ADD r0,r0,r2</pre>	<pre>sdcc generates: 160 ; function main 162 _main: 163 ;test32.c:20: fred = getval (); 164 ; genCall 0026 12s00r08 165 lcall _getval 0029 AA 82 166 mov r2,dpl 002B AB 83 167 mov r3,dph 002D AC F0 168 mov r4,b 002F FD 169 mov r5,a 170 ;test32.c:21: john = getval (); 171 ; genCall 0030 C0 02 172 push ar2 0032 C0 03 173 push ar3 0034 C0 04 174 push ar4 0036 C0 05 175 push ar5 0038 12s00r08 176 lcall _getval 003B AE 82 177 mov r6,dpl 003D AF 83 178 mov r7,dph 003F A8 F0 179 mov r0,b 0041 F9 180 mov r1,a 0042 D0 05 181 pop ar5 0044 D0 04 182 pop ar4 0046 D0 03 183 pop ar3 0048 D0 02 184 pop ar2 185 ;test32.c:22: mary = fred + john; 186 ; genPlus 004A EE 188 mov a,r6 004B 2A 190 add a,r2 004C FA 191 mov r2,a 004D EF 193 mov a,r7 004E 3B 195 addc a,r3 004F FB 196 mov r3,a 0050 E8 198 mov a,r0 0051 3C 200 addc a,r4 0052 FC 201 mov r4,a 0053 E9 203 mov a,r1 0054 3D 205 addc a,r5 206 ;test32.c:23: doit (mary); 207 ; genCall 0055 FD 209 mov r5,a 0056 8A 82 210 mov dpl,r2 0058 8B 83 211 mov dph,r3 005A 8C F0 212 mov b,r4</pre>

Table 1: Comparison of Cortex-M3 code and 8051 code for a simple C fragment involving 32-bit numbers

- The C/C++ compiler includes a vast number of controls aimed at embedded systems. For example, it is possible to place code and data anywhere in memory (an automatic scatter loader is added to your code by the linker) and structures can be packed to use minimal data space. Many optimisations are supported, including loop unrolling, automatic in-lining, peep-hole optimisation and automatic load/store coalescing.

Cortex-M3 represent a potential productivity revolution in low-end microcontrollers, if Australasian companies are willing to take advantage of it. This will require some investment.

Impact of the \$1 32-bit microcontroller

Now that it is possible to embed a 16/32-bit ARM chip in a product for a dollar, what changes is this likely to make in the market? In my view, these changes will be significant in a number of areas:

- Speed – faster code execution at lower clock speeds. This may simplify system design, or just provide headroom in case the requirements change during the project.
- Power – lower power consumption for the same performance. For battery- or solar-powered designs, or where equipment must operate in hot environments, this is important.
- Cost – similar to 8-bit micros. Cost is no longer a reason to use an 8-bit micro.
- Flexibility – the Cortex-M3 can perform basic DSP tasks, perform a quicksort or an MD5 checksum, or even implement an XML parser. All of these tasks would be hard for an 8-bit micro. Best of all, since programming in C is very efficient, there is little or no need to resort to assembler to cram the features into available code space or time.

- Interrupt performance – the Cortex-M3 can offer much faster interrupt performance than an 8-bit micro. This can reduce the need for hardware buffers and FIFOs, thus reducing system COS

Efficiency of 8051 vs. Cortex-M3

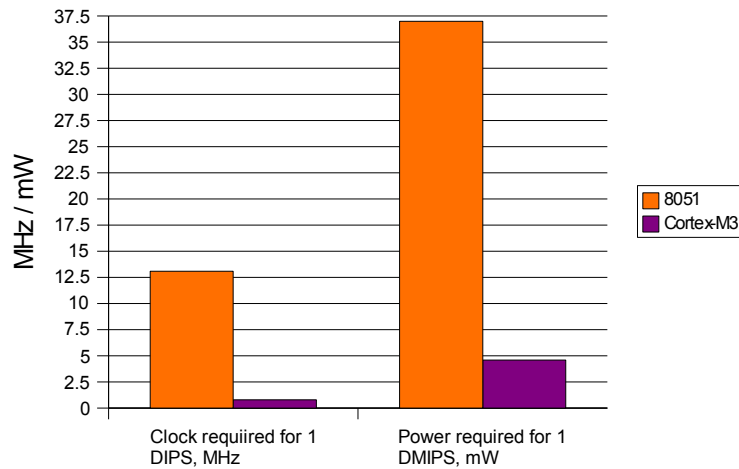


Illustration 5: Comparison of efficiency of 8051 and Cortex-M3 in terms of clock speed and power

- Productivity – as already mentioned we can use 'grown-up' development tools with Cortex-M3, achieving significant productivity gains.

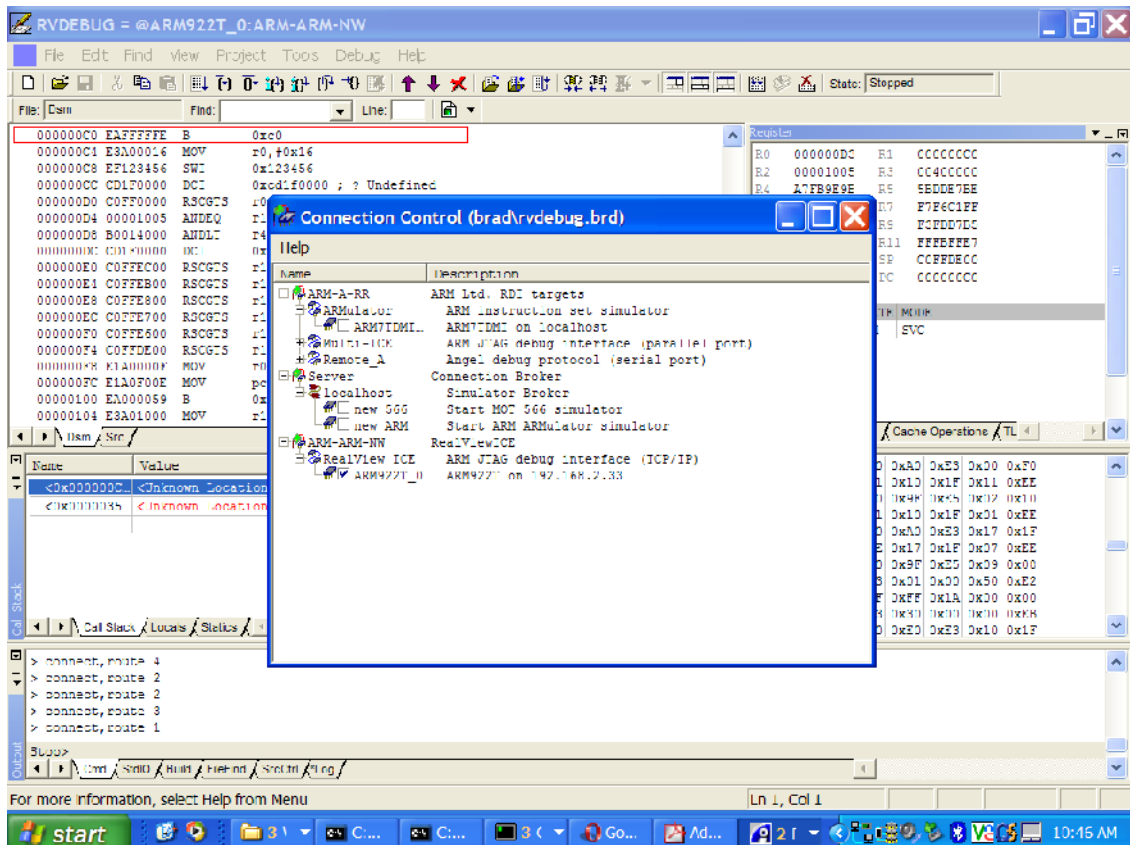


Illustration 6: The ARM tools provide a wide range of features and enable large productivity gains versus legacy 8-bit tools

Applications

What sort of things can Cortex-M3 do that an 8-bit micro cannot? This is a very broad question, but here are some examples:

- OLED dot-matrix displays seem set to find an important niche in the electronics industry, replacing the traditional 2x20 character display. Taking full advantage of these will require more processing power.

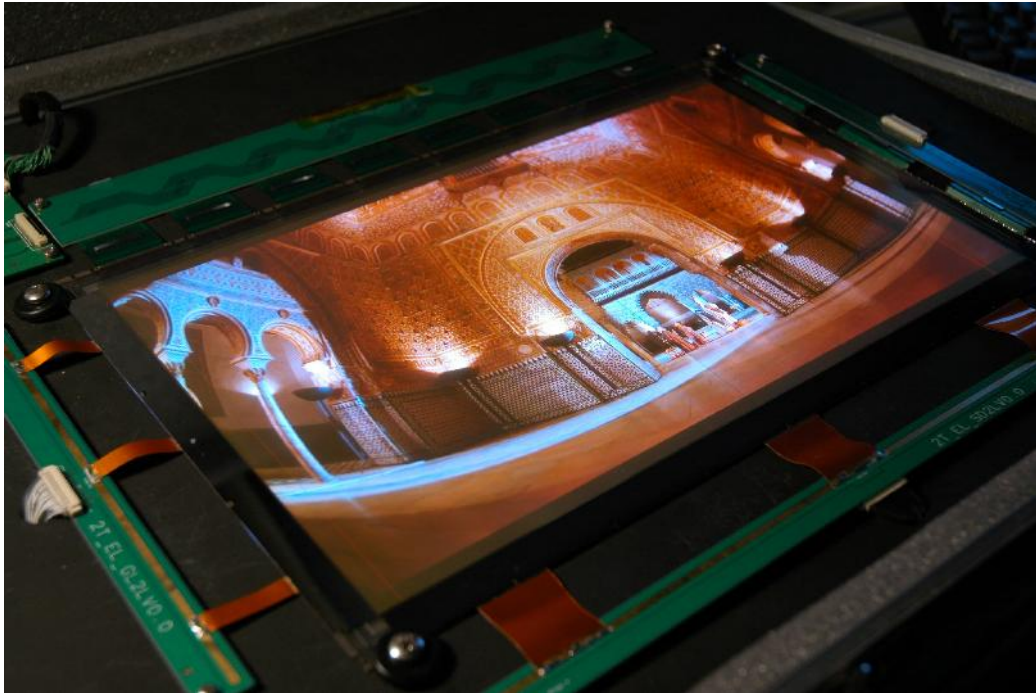


Illustration 7: A prototype 12 inch OLED display. This technology shows promise in low power portable applications

- The 'Smart Home' is a technology that has yet to deliver. But it is hard to believe that dumb stand-alone appliances will still dominate the market in 10 year's time. As robots start to enter the home they will want to know when the washing machine has finished, rather than stand there and watch it. An open extensible protocol between these appliances (perhaps XML-based) is much more likely to work than a series of 8-bit micro friendly binary interfaces.
- Wireless communications is becoming cheaper and better every year. At present, 8-bit micros rely on in-built micros within wireless modules to handle the processing, which leads to the ludicrous situation of an little 8051 telling an ARM7TDMI inside a GPRS module what to do. There are obvious economies to be made here. Cortex-M3 is capable of running full protocol stacks such as TCP/IP without enormous contortions.
- Children's toys are becoming more intelligent (or perhaps just more annoying?). Features such as basic Artificial Intelligence and audio recognition are much easier to implement on a Cortex-M3 than an 8051!



Illustration 8: Microprocessor-controlled children's toys are become more popular, and sophisticated

- In the automotive industry, we have a situation where perhaps 50 micros are embedded in a single car. This has been partly driven by the need for robustness – since the rear left ABS brake cannot fail just because the CD jams in the audio system. As these problems are solved, we may not ever see fewer micros in a car, but we will certainly see them communicating more and sharing responsibilities. There are cost savings to be made, but the trade-off is increasing system complexity. No sane engineer would take this on with an 8-bit micro.

Conclusion

The \$1 32-bit micro will enable a number of applications at lower price points than have been seen before. Perhaps more important, it will enable a level of engineer productivity which is not possible with 8-bit micros and tools. For those companies which take advantage of this change, there will be significant benefits in terms of time to market, functionality and ease of maintenance.