Advantages and Pitfalls of Moving from an 8 bit System to 32 bit Architectures.

David Kerr-Munslow[1].

1: Cortus S.A., Le Génésis, 97 rue de Freyr, 34000 Montpellier, France

Abstract:

This paper explores the considerations of designers of embedded systems when they come to choosing the bit width of the embedded CPU architecture, especially in the domain of System on Chip designs. Two typical architectures are compared and contrasted, one 8 bit and the other 32 bits, the 8051 and the Cortus APS3.

Embedded systems are a designed with a number of constraints not found in other computer systems. In addition they are also expected to give real-time responses, often with limited resources.

Received ideas are explored and evaluated in the light of benchmarks and concrete examples. Attention is paid to modern implementation and programming methodologies.

Issues such as power consumption, code density, suitability for real time systems, ease of software development are discussed.

Keywords: CPU Architecture, bus width, SoC, microcontroller, processor, CPU

1. Introduction

In this paper we explore the challenges and misconceptions involved in processor architecture selection for embedded systems. We concentrate on the design choices currently in front of system designers. The key choice is frequently the bus width of the architecture. There are a number of advantages and a few pitfalls associated with choosing a 32 bit architecture over an 8 bit CPU.

Two representative processors were chosen, the 8051 which remains a very popular microcontroller architecture, and is popular in SoC designs and the Cortus APS3 which is a member of a new generation of 32 bit processor architectures. The APS3 is particularly appropriate as it was specifically designed to correspond to the needs of embedded systems being designed at the moment.

We explore the implications of bus width notably:

- Ease of Programming
- Code Density
- Performance
- Real Time Considerations
- Core Size
- Power Consumption
- Porting

These analyses and measurements lead to the conclusion that questions the assumption that there is still a niche for the 8-bit microprocessor core. This is especially evident in the SoC arena, but also for the microcontroller and FPGA sectors.

2. Ease of Programming

The most significant development cost of most systems nowadays is that of software development [3]. Therefore it is necessary to pay attention to the implications of the architecture choice on software development costs.

Most program development nowadays is done in high level languages, and currently in the embedded world that generally means C and sometimes C++. Most young engineers have graduated with experience of C and C++ from their University courses, with perhaps only a minimal exposure to some assembly language programming. Increasingly the knowledge and practice of low level interaction with hardware and assembly language programming is becoming a specialised skill.

Features of C

The C programming language was initially conceived for developing system software, for mini computers [4]. There are certain assumptions made about the underlying hardware by the C language. These are:

- Byte addressable memory
- Integers and pointers of the same size
- Single address space
- Stack and heap

C/C++ compilers are available for the vast majority of processors, though sometimes not without concession to certain architectural features.

Impact of Architecture

The architecture assumptions implicit in the C language have an impact upon the ease of implementation for a specific architecture.

Most implementations of the C language rely on the presence of a stack, and use this stack not only for parameter passing and return address storage, but also for automatic variables. This use of the stack gives lexical scope to automatic variables, at least implicitly.

Processors that have a limited stack space must implement another mechanism for the short term storage of local variables. For instance the internal stack of the 8051 is 128 bytes, and is used for subroutine and interrupt returns. Stack overflow is unsignalled and can result in hard to reproduce crashes when nested interrupts exceed the stack depth, this is something to which the author can attest!

The popular C compiler for the 8051, SDCC, stores local variables in general RAM, and lexical scope collisions become an issue. This makes re-entrant functions difficult to implement, and the use of library functions within an interrupt handler potentially treacherous. Recursion is clearly not possible.

Pointer arithmetic

The assumption that memory is a linear array of storage cells, permits, and even encourages, the manipulation of data structures using explicit pointer arithmetic. In embedded systems where speed is more prized over elegance this is even more the case. This is exacerbated by the treatment of arrays and pointers as aspects of the same language feature by C.

In 8 bit CPUs the addresses are often 16 bit values, however the natural integer size is 8 bits. This issue is often palliated by special index registers and addressing modes in the CPU. However it is difficult for a compiler to take full advantage of these resources.

Address Space

The size of the address space is a critical issue. Most 8 bit CPUs can directly access 64Kbytes of address space. Modern embedded systems often have easily over 64 Kbytes of program code, and require complex bank switching schemes. This makes interrupts and subroutine access complex, and potentially error prone. Convoluted schemes must be created to ensure that library routines are always available in the address space.

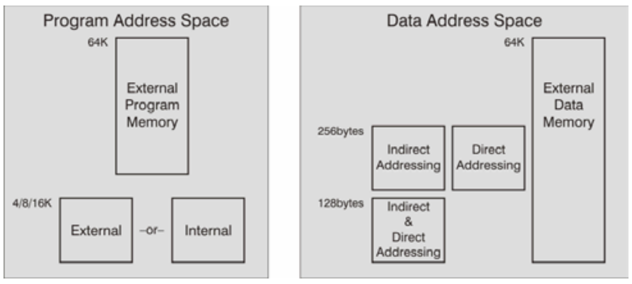


Figure 1: 8051 Address Spaces

Figure 1 shows the various address spaces of the 8051, certain spaces are only accessible either by direct addressing modes, or by indirect addressing modes [2, 1-6]. This aspect of many 8 bit CPUs with multiple address spaces, for program, for data, for I/O requiring different access techniques, frequently requires an extension of the C language is to manipulate data in these spaces.

Here there is a clear advantage of 32 bit architectures which naturally address 4 Gbytes.

3. Code Density

Code density is the measure of how much memory is required to store the program to perform a task.

Importance of Code Density

Code density influences the following factors:

- Consumption of the memories
- Execution speed
- On-chip vs off-chip memory placement

Clearly power consumption of memories is proportional to their size. Equally the number of address bits used influences the routing and bus capacitance and therefore the drive strength required.

The number of fetches that are required directly influences the execution speed, the fewer memory accesses required to read the program into the CPU the quicker and more efficiently the program can be executed.

One key aspect in SoC design is whether the memories are placed on-chip or off chip. On-chip memory tends to be more limited in size, off-chip memory is bigger, however I/O pins must be dedicated to accessing this memory. This can significantly increase the cost of packaging.

There is also an impact on power consumption, I/O buffers required to drive external pins consume considerably more power than the internal buffers required to access on-chip memories.

A Practical Example: FreeRTOS

FreeRTOS is an open source real time operating system. It has the advantage of being small, lightweight and available freely as source code.

Ports are available for a number of processor architectures

The Core FreeRTOS kernel was compiled for the 8051 and for the Cortus APS3.

Architecture	.text
8051 (SDCC)	26007 bytes
APS3 (GCC)	11084 bytes

Table 1: Code size for FreeRTOS

The same options and demo code were chosen.

A further analysis was made of the resulting assembly code to determine the instruction length mix.

Architecture	8 bits	16 bits	24 bits	32 bits
8051	37%	48%	15%	—
APS3	—	56%	—	44%

Table 2: Instruction Mix for FreeRTOS

Table 1 shows that the 32 bit APS3 architecture is 2.35 times more memory efficient compared to the 8051.

Furthermore Table 2 shows the breakdown of the instruction lengths for the two processors, the APS3 has two lengths of instruction (16 and 32 bits), the 8051 has instructions 8, 16 and 24 bits long. It is interesting to note that only just over a third of the instructions of the 8 bit processor are actually 8 bits in length.

Stack Based Architectures

In terms of code density, stack based architectures offer a clear advantage compared to register based machines. These architectures however are not without their disadvantages. They require that all the operands are present on a stack (either internal, or in external memory) and a significant amount of processing can be taken up with stack manipulation.

4. Performance

Many factors influence the performance of architecture. The use of a benchmark can give a general indication of performance.

The Dhrystone benchmark is a popular synthetic benchmark for embedded systems. It produces a measure of CPU performance in DMIPS/MHz. This is a synthetic benchmark and should be treated with caution; the only way to determine how quickly a program will run is to actually run it. However it can give broadly useful indications. The performance measures for the 8051, and enhanced 8051 and the APS3 show significant performance advantages for the 32 bit architecture.

Architecture	"Performance"
8051	0.026 DMIPS/MHz
Enhanced 8051	0.296 DMIPS/MHz
APS3	0.85 DMIPS/MHz

Table 3: Performance

The significance of the measurements shown in Table 3 is that the 32 bit architecture manages to do significantly more useful work per clock cycle than the 8051 [5]. This means that more work can be done in a given unit of time (at the same clock frequency), or the same amount of work can be spread over a longer period (at a lower clock frequency).

Register Set

The register set has a significant impact upon the ability of a compiler to generate efficient code, and also the suitability of the processor for handling time critical interrupts without large quantities of overhead.

The register set can be characterised by the following features:

- General Purpose
- Accumulator
- Width (8, 16 or 32 bits)
- Number of Registers

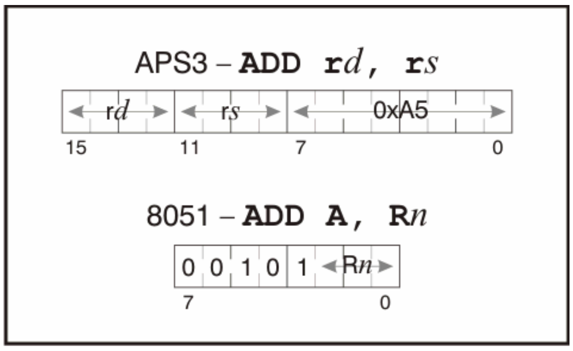


Figure 2: Register Specification, Add Instruction

Eight bit architectures generally have 8 bit instructions, with one or more extension bytes as necessary. The small instructions limit the number of registers that can be specified and most instructions operate upon an implicit register, an accumulator. Processors with wider bus widths often have larger instruction sizes, either 16 or 32 bits long. Figure 2 shows a typical instruction from the APS3 and the 8051, an add instruction, for comparison. It can be seen that the larger instruction width permits more bits to be used to specify the registers to operate on, for example the APS3 uses two four bit fields to specify which of the sixteen registers are to be used.

The ability of the ALU to operate on a wider range of architectures reduces the requirement to move data from memory to specific registers

An accumulator architecture reduces the size of the instructions, as at least one of the registers is implicit in the instruction. However this reduces flexibility. This is very important for an 8 processor as its instruction size is naturally a multiple of 8 bits.

Stack Based Architectures

Stack based architectures are processors for which the majority of the operations operate upon a stack, taking the operands from the stack and placing the result back on the stack. In this approach the operands and destination of all operations are implicit, allowing a very compact coding of the instructions. This architecture suffers from the same disadvantage of requiring significant manipulation of data to ensure that it is in the “right place” for the desired operations. In addition this architecture can result in many memory accesses, which can be inefficient in terms of time and power.

Peripheral Access

The large address space of 32 bit architectures can influence the design of peripheral registers. Each peripheral register can be designed for ease of access, rather than efficiency of address space usage.

The abundant address space permits functionality to be grouped into logical operational sets, rather than mixed to ensure optimal packing.

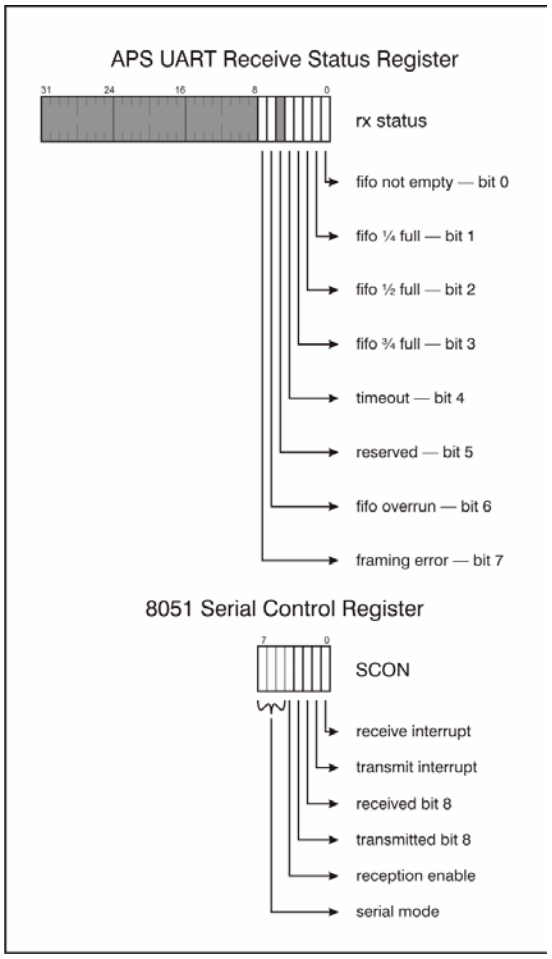


Figure 3: Peripheral Register Comparison

The figure shows two peripheral registers, the Serial Control (SCON) register from the 8051 and the Receive Status register used with the ASP3. The SCON register controls several aspects of the UARTof the 8051, the receive status (interrupt status) is buried amongst transmitter status bits, control bits and data bits. Significant bit manipulation is required to determine the status of the receiver.

The APS Receive Status register contains just the bits concerning the receiver status, the “nothing to report” status of the receiver is indicated by an all zero register.

This approach simplifies driver and most particularly interrupt handler writing. The interrupt handler can quickly determine the exact cause of an interrupt and react appropriately – quickly returning control to the interrupted routines, or returning to the low power sleep mode.

The driver software can be written more simply and with fewer potential bugs as different functionalities are carefully separated. There is no risk that the receiver software will overwrite control bits of the transmitter hardware.

5. Real Time Considerations

Embedded systems are often real-time systems too. The key constraint of a real-time system is that it react in time to an event, usually an external stimulus or the expiry of a timer, and that it do this in a predictable manner.

Interrupts are often the mechanism used to provide this reactivity. The key aspect of an interrupt is the ability to stop one task and execute the interrupt code in a timely and predictable manner, and then return to the interrupted routine, or back to sleep.

There are a number of influences on the ability of a system to be timely and predictable.

- Register Set Size
- Instruction Latency
- Interrupt Efficiency
- Pipeline depth

Register Set Size

The size of the register set has a direct influence on the context switch time. The process state must be saved between context switches, this state includes all process accessible registers.

A seemingly interesting feature of the SPARC architecture is register windows, allowing subroutine link simply by incrementing or decrementing a register window pointer. This is elegant until the (necessarily limited) register set runs out of windows, when the register set must be copied to memory. This scheme also has a drawback when performing a context switch when again the entire register set must be copied to memory.

Instruction Latency

Instruction latency influences the predictability of the interrupts. Few architectures permit instructions to be interrupted in the middle of execution. Therefore interrupts must wait until an instruction completes before being acted upon. This latency increases with instruction complexity, increasing the uncertainty of when the interrupt will be recognised.

Interrupt Efficiency

Interrupt efficiency can be crucial; the implementation of the interrupt mechanism can take several forms:

- Vectored interrupts
- Shared interrupts
- Fixed routine addresses

In a processor with vectored interrupts a table of the addresses of the interrupt handlers is present. Each interrupting device is assigned a vector number and when an interrupt is signalled the processor accesses the vector table and jumps directly to the interrupt routine, this is largely the most efficient method (it is how the vast majority of modern 32 bit architectures behave).

In shared interrupts there are only one or two interrupts and the interrupt handler must poll all of the potentially interrupting devices to determine the source of the interrupt. This polling is quite inefficient.

In some processors the interrupt handlers are at fixed addresses and when an interrupt occurs the processor jumps to a fixed address corresponding to the interrupt this is more efficient than the vectored approach. However this scheme has the drawback that if there are multiple interrupts then there must be a fixed spacing between the handlers, and either this spacing will be too great and (valuable) address space will be wasted, or there will not be enough space and jumps will have to be inserted, creating convoluted code (that is difficult to write in anything other than assembly language).

The 8051 uses the latter approach, the APS3 the former approach.

Pipelines

Deep pipelines considerably improve the frequency at which a CPU can be clocked. However they require flushing and refilling when there is a change of flow, for example a branch or jump – and especially an interrupt. The delay in re-filling the pipeline can be mitigated by prediction for branches and jumps, however by their nature this is not possible for interrupts. This means that there can be considerable latency in starting to execute the interrupt code in addition to recognising the interrupt.

6. Core Size

The size of an IP core is a major consideration in the design of an SoC. It influences directly the production cost, the size of die is directly proportional to the cost of the SoC. The cost of silicon is calculated in mm^2.

Architecture	Gate Count
8051	9000 [5]
APS3	9500 [6]
Cortex-M0	12000 [6]
ARM7 TDMI	36000 [6]
Cambridge Consultants XAP5a (16 btis)	18000 [6]
Tensilica Diamond Standard 106Micro	20000 [6]

Table 4: Gate Counts for Selected Processors

Table 4 gives a summary of gate counts for a selection of processors. It can be seen that there is more than a factor of 4 in size between processors targeting the same application. What should be noticed is the similarity in size between certain 32 bits processor architectures and the 8051.

Leakage Current

The key parameter for battery life for a device that is mainly in an idle or sleep state is the leakage current. This is directly proportional to the silicon area of the circuit, which is related to the gate count.

Increasingly FPGAs are becoming used not only for system prototyping but also commercial implementation. Clearly the cost of an FPGA is proportional to its size; it is also a factor in the feasibility of prototyping the system. Here the processor core size is a key cost parameter.

7. Power Consumption

Power consumption is a primordial consideration in battery operated devices but it also influences power dissipation strategies. The key advances in mobile technology are mostly due to the reductions in energy requirements of electronic systems.

CMOS technology mainly consumes power on clock transitions, with a static leakage current also contributing a small part (significantly more in technologies smaller than 130 µm).

Power consumption is therefore a function of the number of gates that switch and the frequency at which they switch.

Architecture	Power Consumption
8051 core	34µW/MHz
APS3	24µW/MHz

Table 5: Power Consumption

(8051 figures taken from the published Dolphin Integration estimates)

Table 5 shows the power consumed per mega Hertz of clock frequency. These figures should be taken into consideration with the figures in Table 3 which shows that the 32 bit processor gives considerably "more bang for your buck" than the 8051. The APS3 outperforms the 8051 by 46 times in terms of DMIPS per µW.

To be clear, taking into account the meaninglessness of benchmarks, that for the same application in equivalent systems that the battery life of a system with the APS3 would be **46** times that of the 8051 implementation.

This figure is just considering the raw processing power of the processors. Other features will also improve the power consumption. The improved code density, as shown in Table 1, will require fewer opcode fetches, and the register to register architecture will require fewer memory access, both of these factors will reduce power consumption.

The leakage current is directly proportional to the silicon area. Table 4 shows the gate count for various processors, the relative leakage currents can be inferred from these figures.

32 bit Busses

Driving bus lines requires the bus capacitance to be charged, and discharged, therefore the more bus activity that is required the more power that will be consumed. Driving 32 lines will consume more power than 8, so clearly an 8 bit CPU has an advantage. However if constants larger than 8 bits are required, then multiple accesses are required and not only do the data lines need to be drive twice, but also the address lines.

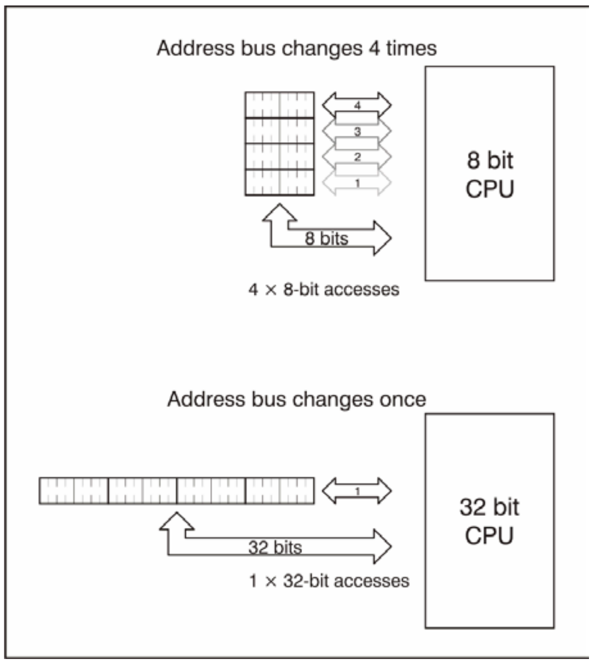


Figure 4: 8 bit bus vs 32 bit bus

In this case the 32 bit processor will require only one access to the memory (which can also be switched into low power mode more quickly) and therefore consume less power globally.

Pipelines and Speculative Execution

The more sophisticated modern CPUs have deep and complicated pipelines, which dramatically improve the instruction throughput and optimise silicon usage. This also increases the clock rate at which the CPU can be run. Speculative execution is often coupled with this to mitigate the impact of branches and jumps. This reduces the number of pipeline stalls and improves benchmark results. However a considerable drawback is that operations are performed and then the result is thrown away – the power used to perform these unused operations is wasted.

8. Porting

In many situations there is an established code base. This can be in the form of libraries external to the project or software already developed for predecessor implementations.

External libraries may be available as high level source code or coded in assembly. Existing source code is too frequently implemented in poorly documented assembly language, making re-implementation in C a tedious proposition.

Changing the target architecture partway through a product lifecycle may be due to a number of considerations:

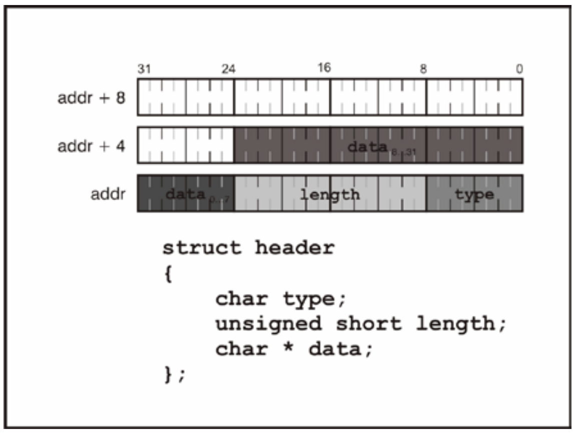
- Cost Reduction
- Replacing obsolete parts
- Adding features

In the case where porting of the code is necessary, it is clear that porting to an architecture that offers a superset of the features compared to the original target processor is considerably simpler. Removing non-standard extensions to the language is often simple though can require knowledge of the implications of the extension.

Given the increased performance of a 32 bit architecture it is possible to emulate the 8051 instruction set and key parts of the architecture for the code libraries that have not yet been implemented for the 32 bit architecture. This can be done through run time emulation (the simplest approach) or through re-compilation of the 8051 assembly language. Simple strategies using jumps tables can be remarkably effective.

Data Alignment

Moving from 8 to 32 bit architectures can cause data alignment and packing issues. This can be problematic when the architecture cannot perform aligned accesses.



```
struct header
{
    char type;
    unsigned short length;
    char * data;
};
```

Figure 5: Packed Structure Alignment

Figure shows the potential packing of a structure in memory. This packing is of no significance to 8 bit CPUs but can influence how a compiler might dispose the elements of a structure in memory for optimal speed of access, or to correspond to processor constraints. This becomes problematic when the programmer attempts to access the contents of the structure by bypassing the structure access syntax and directly access memory, byte by byte.

Some 32 bit architectures can perform unaligned access (using a two cycle bus access), for example the APS3.

Language Extensions

As we have already discussed certain features of microcontrollers require extensions to the language. It is considerably simpler moving to a more standard dialect of the C language than adapting already developed code to a more customised version of the language.

Futureproofing a Design

The goal of many projects is a successful product that is sold in significant numbers and with a long product lifetime in the market. With this goal in mind it is prudent to ensure that the design can not only be manufactured in large quantities but also over an extended time. The ability to extend the design to produce versions with extended features can also encourage sales, to beat customers or to provide an “upgrade” path. The development of a system with the ability to be extended can be a significant design objective – one technique to ensure that future development can be done with minimal effort is to ensure that all software developed is portable. That is to say: developed in a standard, “unenhanced” language targeting a uniform architecture.

9. Sixteen bit CPUs

In this paper we have not looked at 16 bit architectures. It might be considered that a 16 bit CPU could be an ideal compromise between the 8 bit CPUs and the 32 bit architectures. The problem is that 16 bit CPUs are neither fish nor fowl, that is they generally have the disadvantages of the 8 bit CPU, notably limited address spaces, and few of the advantages of the 32 bit architectures.

Decidedly few 16 bit architectures have more than a 64Kbyte address space, which offers no advantage over the 8 bit processors.

10. Conclusion

Conventional thinking favouring 8 bit microcontroller cores requires revision in the light of the processor IP that is now available in the market.

	8 bits	32 bits	Improvement
Code Density		✔	2.35 ×
Power Consumption		✔	46 ×
Core Size	✔	✔	=
Ease of Programming		✔	
Real Time		✔	
Backward Compatibility	✔		
Familiarity	✔	✔	

Table 6: Conclusions

Table 6 resumes the paper and shows that a large sector of the embedded SoC market can advantageously move to using 32 bit processors.

The increasing use of the C language removes the advantage of familiarity, especially since C compilers for 8 bit processors require non-standard extensions.

11. Acknowledgement

The author acknowledges the contribution of his colleagues to this work.

12. References

[1] Hennessy and Patterson: "Computer Architecture A Quantitative Approach",

[2] Intel: "MCS51 Microcontroller Family User's Manual" Intel, 1994

[3] http://www.softwaremetrics.com/ Articles/HardwareandSoftware.htm

[4] Denis M. Ritchie: "The Development of the C Language", Second History of Programming Languages conference (Cambridge, Mass) 1993

[5] Dolphin Integration: "http://www.dolphin.fr/flip/logic/8bit/logic_8bit.php"

[6] Tom R. Halfhill,: "Itty-Bitty 32-Bitters", Microprocessor Report, 5/11/09-01

13. Glossary

CPU:	Central Processing Unit
IP:	Intellectual Property
RAM:	Random Access Memory
SDCC:	Small Device C Compiler
SoC:	System on Chip