The New MCU On The Block: AVR

The ***AVR*** is the only new 8-bit MCU architecture the world has seen in the past 10 years. One could always ask “why?”, as the question is quite relevant, but the answer is not given here. One explanation can be eliminated, though; that MCU architectures widely used today are good enough in most cases. As a matter of fact, they are *not*, and they will definitely not be good enough in the future, considering the increasing demand for higher instruction throughput and lower power. Over the past decade, semiconductor technology has improved vastly. Despite this and the fact that there has been and always will be a constant need for higher performance, 8-bit MCU technology is for some reason lagging behind. Expensive and slow architectures that were developed 15 - 20 years ago dominate the market. With the ***AVR***, the state of the art technology of today is combined with all the advantages of different architectures into one single MCU design. The design goal was clear; to bring MCU customers an architecture that is better than anything else available.

The ***AVR*** architecture exceeds other 8-bit MCUs in the following areas:

- Highest Performance
- Lowest Power Consumption
- Most Compact Assembly Code
- Most Compact C Code
- In-System Programmable Flash Program Memory
- In-System Programmable EEPROM Data Memory
- Best price/performance ratio

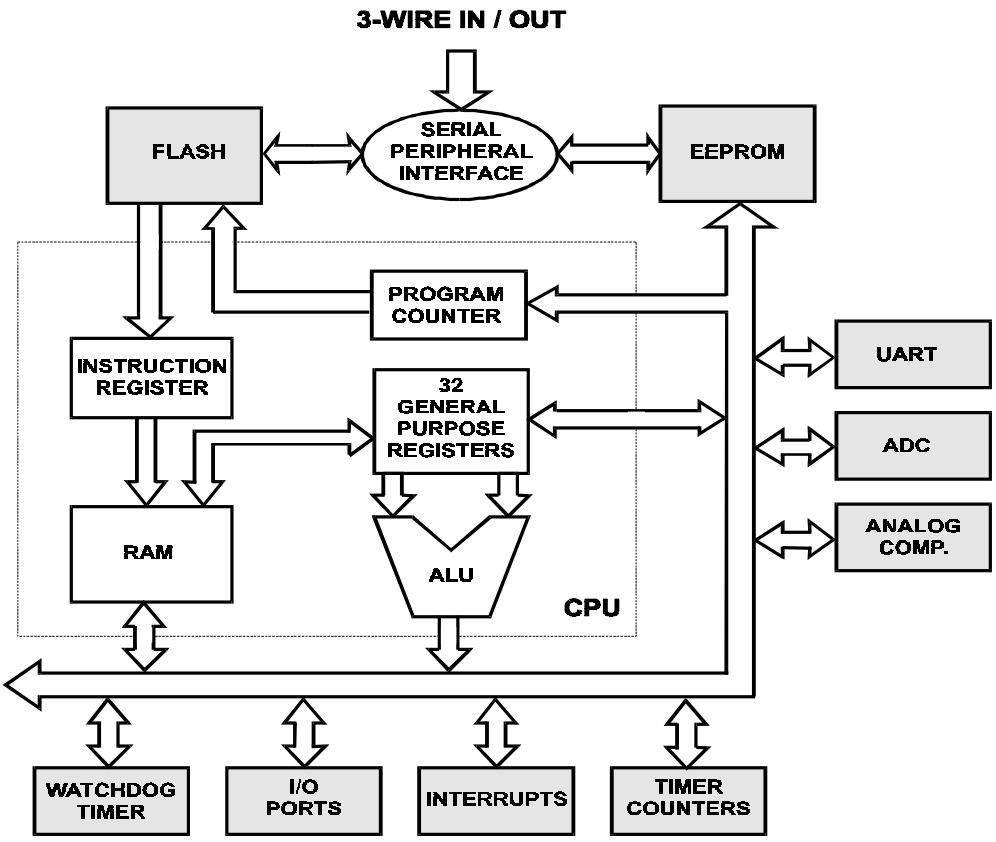
The Architecture in General

Figure 1 shows a general block diagram of ***AVR*** Enhanced RISC Architecture. The design is an enhanced Harvard architecture with separate buses and address spaces for Program and Data memories. With 32 general purpose working registers, true single cycle execution and downloadable Flash and EEPROM memory, this MCU concept offers denser code, higher throughput, and more flexible programming solutions than any other 8-bit MCU available.

32 General Purpose Working Registers

In the late 70s and early 80s when several of today’s widely spread architectures were developed, the gate count per silicon area unit was significantly lower than today. Especially the number of data registers, which are costly in terms of silicon area, had to be kept as low as possible. As a result of this most MCUs developed in this period has only one general purpose working register - the accumulator. This concept has for some reason survived for 20 years, and is still found in the majority of microcontrollers available. Considering the shrinkage of silicon structures that has taken place over the past two decades, more than one accumulator can now be implemented without penalties. When the ***AVR*** was designed, Atmel equipped the MCU with the generous count of 32 accumulators. They are called “General Purpose Working Registers” and the block of 32 is referenced as the “Register File”. This very important feature eliminates the frequent transfer of data in both directions between the Data memory and the accumulator. In an average program for an accumulator-based MCU, this data transfer occupies a significant amount of the total code. Data handled at the moment can reside in a 32 byte register file where the CPU has direct access to all of them at any time. As a result, a significant amount of unnecessary code can be removed in both C and Assembly programs.

Figure 1: ***AVR*** Architecture



Single Cycle Execution

Not only will an ***AVR*** program contain fewer instructions than programs for other microcontrollers, but the time to execute each instruction is only a fraction of what other MCUs require. A direct connection between the register file and the ALU enables true single cycle execution. In addition, all instructions are 16 bits long, hence the MCU does not have to decode the first byte in the instruction to decide how many more to fetch. The instruction fetch and execute is also pipelined, which means that during execution of one instruction, the next one is fetched from the Program memory. Additionally, conventional MCUs use an internal clock divider when instructions are fetched and executed. With the ***AVR***, however, most instructions take exactly *one single Crystal Oscillator period*. The resulting throughput increase is 400 - 1400% compared to conventional MCUs. The combination of single cycle instruction execution and the lack of an internal clock divider yields an 8-bit MCU with unparalleled performance, and gives the user the ability to utilize the tremendous instruction bandwidth to its fullest, or tradeoff between clock frequency and power consumption.

Many 8-bit MCU users see their performance requirements growing beyond the capabilities of the architecture they have lived with for years. A solution commonly considered is switching to a 16-bit version of the same old architecture that is currently imposing limitations to creative product improvements. In this case, the penalty for a performance increase is larger code and hence more memory plus a more expensive MCU. By changing to the ***AVR***, you can significantly increase your applications' performance, use less program memory, eliminate external non-volatile memory devices, reduce power consumption and lower your total cost.

A Simple Comparison

On the next few pages you'll see an example of how the ***AVR*** MCU compares with one of the currently popular 8-bit architectures, the Microchip PIC. It is not meant to be an exhaustive analysis but a straightforward architectural comparison, to illustrate the features and benefits of the ***AVR*** MCU.

ATMEL AVR vs. MICROCHIP PIC FACT SHEET

Introduction to the AVR MCU Families

FEATURE	COMMENTS	AVR MCU	MICROCHIP PIC
Complete family	• Family divided into two families, depending on Program Memory size: • ClassicAVR: 1KB to 8KB, of which 6 are in production • MegaAVR: 16KB to 128KB, of which the first is in production • All devices are in-system programmable using the supply voltage, down to 2.7V.	ClassicAVR: AT90S1200, AT90S2313, AT90S2323, AT90S2343, AT90S4414, AT90S8515 MegaAVR: ATMEGA103	

Memory Comparisons

Program Memory	• The important factor is memory cost per byte • In actual applications, the AVR MCU requires fewer bytes to implement the same program • Memory organization is irrelevant; the cost depends on silicon area	AT90S1200 = 1024 bytes AT90S2313 = 2048 bytes AT90S2323 = 2048 bytes AT90S2343 = 2048 bytes AT90S4414 = 4096 bytes AT90S8515 = 8192 bytes ATMEGA103 = 128K bytes	PIC16C54 = 768 bytes PIC16C56 = 1536 bytes PIC16C58A = 3072 bytes PIC16C620 = 896 bytes PIC16C621 = 1792 bytes PIC16C622 = 3584 bytes PIC16CXXX 0.5KB – 14KB PIC17CXXX: 4 KB – 32KB
Data Memory (SRAM, EEPROM and on-chip registers).	• General Purpose Registers are also Data Memory • Total volatile memory is SRAM size and on-chip registers. • All AVR devices have 32 registers • On-chip EEPROM memory sizes shown. • All AVR Controllers have Data EEPROM • AVR can run from program memory while writing EEPROM	AT90S1200 = 32 bytes AT90S2313 = 160 bytes AT90S2323 = 160 bytes AT90S2343 = 160 bytes AT90S4414 = 288 bytes AT90S8515 = 544 bytes ATMEGA103 = 4128 bytes AT90S1200 = 64 bytes AT90S2313 = 128 bytes AT90S2323 = 128 bytes AT90S2343 = 128 bytes AT90S4414 = 256 bytes AT90S8515 = 512 bytes ATMEGA103 = 4096	PIC16C54 = 25 bytes PIC16C56 = 25 bytes PIC16C58A = 72 bytes PIC16C620 = 80 bytes PIC16C621 = 80 bytes PIC16C622 = 128 bytes PIC16CXXX = 0 bytes PIC16FXXX = 64 bytes PIC17CXXX = 0 bytes

FEATURE	COMMENTS	AVR MCU	MICROCHIP PIC

Code Efficiencies, Instruction Throughput and Interrupt Sources

Number of clocks per instruction (average) or throughput of number of clock normalized	• Single cycle execution is important • Analysis performed by the AVR Core development team when tuning the instruction set • Figures determined by analyzing 20-30 programs, obtained from actual applications • Microchip's 4.84 clocks/instruction assumes that all instructions are used in equal amounts. This is not the case in real applications	One Instruction Cycle is one cycle of the external oscillator AVR does not divide the XTAL clock	One Instruction cycle is four cycles of the external oscillator Microchip PIC divides the XTAL Clock by 4

Assembly code efficiency (in number of bytes)	• High efficiency equals low memory cost
	• High Efficiency through:
	  • More instructions
	  • 32 registers (accumulators)
	  • Linear memory maps with no paging
	• Analysis performed by the AVR Core development team when tuning the instruction set
	• Figures determined by analyzing 20-30 programs, obtained from actual applications
	• We compare byte by byte
		AVR = 1.0	PIC15C5X = 1.6

FEATURE	COMMENTS	AVR MCU	MICROCHIP PIC
AVR single clock cycle execution	• One AVR instruction cycle is one XTAL clock cycle • One PIC instruction cycle is four XTAL clock cycles • Allows throughput to be traded for lower frequency • Lower operating frequency equals lower power consumption	All AVR devices share the same instruction set (the AT90S1200 does not have instructions related to SRAM). Some instruction cycle examples: AT90S1200: 89 instructions 1 cycle: 55 instructions 1\2 cycles: 25 instructions 2 cycles: 5 instructions 3 cycles: 2 instructions 4 cycles: 2 instructions AT90S2313: 120 instructions 1 cycle: 57 instructions 1\2 cycles: 25 instructions 2 cycles: 30 instructions 3 cycles: 6 instructions 4 cycles: 2 instructions	PIC16C5X: 33 instructions 4 cycles: 26 instructions 4\8 cycles: 4 instructions 8 cycles: 3 instructions PIC16C62X: 35 instructions 4 cycles: 26 instructions 4\8 cycles: 4 instructions 8 cycles: 5 instructions
Interrupt Sources		AT90S1200 = 3 AT90S2313 = 10 AT90S2323 = 10 AT90S2343 = 10 AT90S4414/8515 = 13 ATMEGA103 = 23	PIC16C5X = None PIC16C62X =4

Architecture Comparison

Instruction Set	• More instructions • More powerful instructions • AVR is always more code efficient • C-Code compiles very efficiently • The entire AVR Data Memory is in the same page, no paging in Data Memory required • The entire AVR Program Memory is in the same page, no paging in Program Memory required	AT90S1200: 89 instructions classicAVR: 120 instructions megaAVR: 120 instructions Data Memory Address Reach: classicAVR: 64 KB megaAVR: 8 MB No paging required	33 instr for 12-bit word MCU 15 instr for14-bit word MCU 58 instr for 16-bit word MCU Data Memory Address Reach: PIC16C5X = 512 bytes PIC16CXXX = 512 bytes PIC17CXXX = 512 bytes The Data Memory is available in 4 pages of 128 bytes

FEATURE	COMMENTS	AVR MCU	MICROCHIP PIC
Stack	• AVR Stack size limited by RAM size only • AT90S1200 uses a hardware stack, as there is no onboard RAM for a regular stack • Stack in both internal and external RAM	Limited to RAM Size for all AVRs: AT90S1200: 3 levels AT90S2313: 128 levels AT90S2323: 128 levels AT90S2343: 128 levels AT90S4414: 32720 levels AT90S8515: 32720 levels ATMEGA103: 32720 levels	PIC16C5X: 2 levels PIC16CXXX: 8 levels PIC17CXXX: 16 levels
Upward Migration & Registers	• Same architecture in all AVR devices • Same instruction set in all AVR devices • Code written for AT90S1200 will run on any device, without modifications • The register file is always the preferred storage for any variable • Adding RAM to an AT90S1200 design is not a problem - just start using the RAM as it becomes necessary	Moving to a larger AVR: Same instruction set Same architecture Same memory map Same I/O registers	Moving to a larger PIC: New instruction set New memory map
Register Space (I/O)	• AVR I/O Registers are mapped in Data Memory • Number of I/O registers limited to by the maximum data address reach • 64 I/O Registers are read and written in a single clock cycle using ‘in/out’ instructions • More than 64 I/O Registers are accessed with two cycle ‘LD/ST’ instructions	Limited to Memory Address Reach: ClassicAVR: 65504 I/O registers MegaAVR: 8388576 I/O registers No paging required	Limited to Memory Address Reach: PIC16C5X: 32-128 registers PIC16CXXX: 512 registers PIC17CXXX: 512 registers Available in 4x128 byte pages
Register Set	• The 32 General Purpose Registers are all accumulators • 32 Registers eliminate 50-90% of the regular RAM accesses required to move data to and from the accumulator	32 accumulators	1 accumulator

FEATURE	COMMENTS	AVR MCU	MICROCHIP PIC
Register Set Continued (I/O)	• Direct I/O manipulation in all AVR parts with SBI, CBI, SBIS and SBIC instructions • All AVR parts support direct peripheral manipulation • Three addresses per I/O port gives **true** Read-Modify-Write operations • Special instructions are available for faster I/O Register access • The I/O Memory is also mapped into regular data memory, and can be accessed without special instructions		
Register Set Continued (Interrupts)	• AVR Interrupt Flags are cleared by writing a ONE • Writing a ZERO to an AVR Interrupt Flag leave the flag unaltered • Efficient and elegant handling of Interrupt Flags • Any flag cleared in two cycles	Time to clear an interrupt flag: 2 clock cycles	Time to Clear an Interrupt Flag: 4 clock cycles
Program Counter	• Instruction Set handles any modification of PC • Writing PC with ICALL/IJMP at any time • PC updated with one instruction • Reading PC is never necessary, current location is far better determined at compilation time	Easy modification of PC	

FEATURE	COMMENTS	AVR MCU	MICROCHIP PIC
Read Program Memory (LPM / Lookup Tables)	• The 16-bit wide AVR Program Memory stores exactly 2 bytes in each location, • Size optimal implementation of lookup tables in flash, no padding bits wasted • Size optimal implementation of lookup tables sin EEPROM • EEPROM interface optimal for lookup tables • All devices support lookup tables in EEPROM, including AT90S1200 without LPM instruction • Any device can read constants Flash in one cycle using the LDI instruction, including the AT90S1200	Size optimal lookup tables in all devices, in EEPROM Memory Reading constants from Flash supported in all devices Size optimal lookup tables in Flash in all devices with 'LPM' Instruction	Inefficient Padding-bits waste storage space in program memory lookup tables: 12-bit instruction word MCUs: 4 padding bits for each byte stored (50% waste added) 14-bit instruction word MCUs: 6 padding bits for each byte stored (75% waste added)
C-Compiler	• AVR Instruction Set and Architecture optimized for high level language compilers, using Industry Standard ANSI C for benchmarking • AVR now has the most Code efficient 8-bit C-compiler available • Fully compliant AVR ANSI C compiler • All three software pointers X, Y and Z can be manipulated directly, as they are located in the General Purpose Register file. • Global integer modifications are handled by the C Compiler. Modifying the Return Address Stack pointer is handled like any other global integer update, by temporarily disabling the interrupts.	C-Compiler: • Support Industry Standard ANSI C • Very High Efficiency • Both local and global variables	C-Compiler: • Does not support Industry Standard ANSI C • Poor efficiency • Global variables only