Fixed Point Library Based on ISO/IEC Standard DTR 18037 for Atmel AVR Microcontrollers

Wilfried Elmenreich, Maximilian Rosenblattl, Andreas Wolf

Overview

- Floating Point and Fixed Point
- ISO/IEC Standard DTR 18037
- Library Overview
- Evaluation Results
- Conclusion and Future Work

Embedded 8-Bit Microcontrollers

- On-chip SRAM (typically 128 Byte … 4096 Byte)
- On-chip Flash (typically 8 KB…128 KB)
- 8 MHz on-chip oscillator (max external clocking 16…20MHz)
- No floating point unit!

➔ We need embedded software that is **small** and efficient in **memory** and **time** consumption

Floating Point and Fixed Point

- Representation for non-integer numbers
  for example 0.1, 10.5, 3.141592654
- Floating point representation: sign * mantissa * $2^{exponent}$
  - Length (IEEE 754) 32 bit (single precision)
  - 64 bit (double precision)
- Fixed point: sign * mantissa * 2^{-const}
  - Precision = 2^{-const}
  - const=0 => Integer
  - const=8 and 16 bit data type: -128.000 … +127,996

Fixed Point Applications

- Digital signal processing
  - Increase throughput due to higher performance on non FPU architectures
- Desktop application programm GnuCash
  - To avoid unpredictable rounding errors
- Embedded projects on small microcontrollers
  - Speed and memory concerns

ISO/IEC Standard DTR 18037

- Extensions for the programming language C to support embedded processors
- Standardizes Fixed-Point representations and type names
- Specifies also features like selectable precision levels and saturation behavior
- Implementation guideline for C compilers with fixed point support

ISO/IEC Standard DTR 18037

- Specified data types:

ISO/IEC Definition	
`signed short _Fract`	s.7
`signed _Fract`	s.15
`signed long _Fract`	s.23
`signed short _Accum`	s4.7
`signed _Accum`	s4.15
`signed long _Accum`	s4.23

- But what if there is no compiler with such fixed point support for your target architecture?

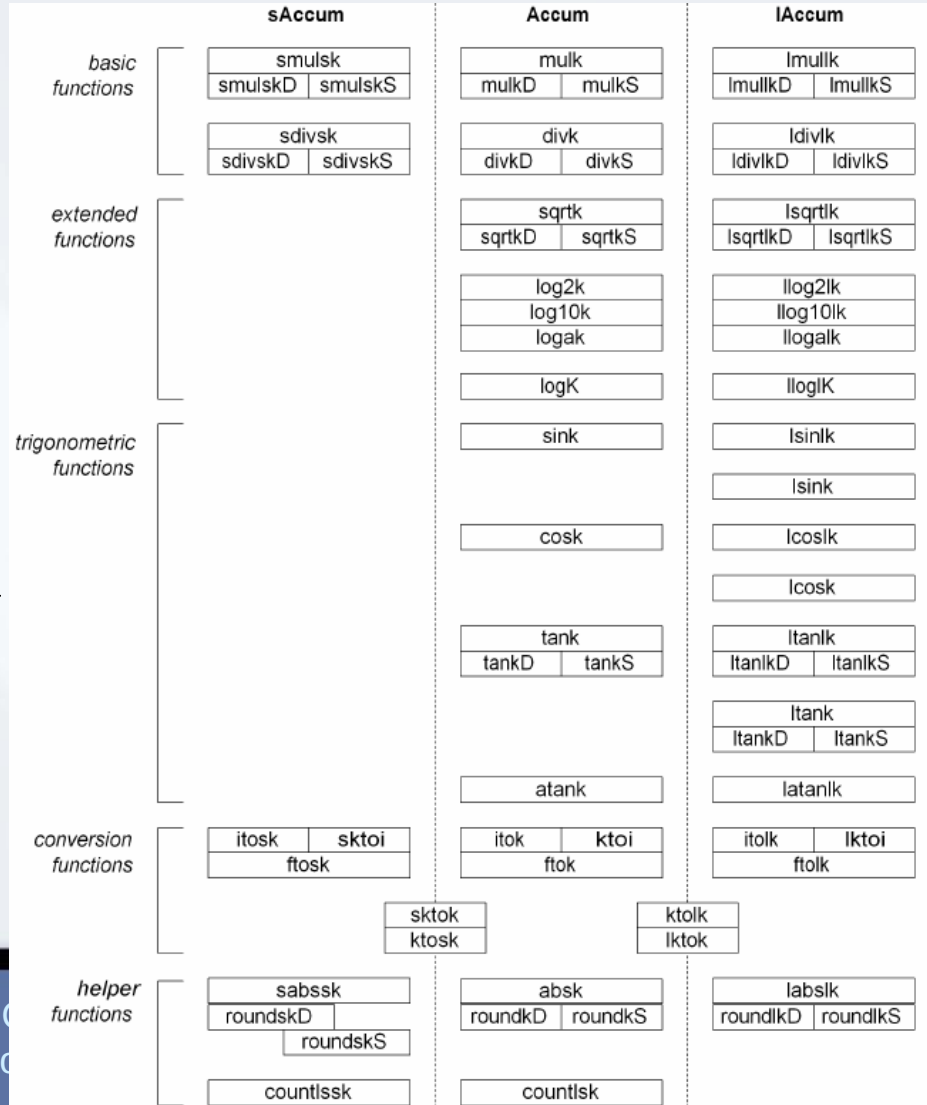
Our solution: provide a library „avrfix“

- Originally intended for Atmel, therefore the name
- Actual implementation in pure C without Assembler code, therefore highly portable to other architectures
- Provide various functions (including sqrt, pow, sin, …) for various data types (according to DTR18037)

ISO/IEC Definition		Implemented	
`signed short _Fract`	s.7	-	
`signed _Fract`	s.15	-	
`signed long _Fract`	s.23	-	
`signed short _Accum`	s4.7	`_sAccum`	s7.8
`signed _Accum`	s4.15	`_Accum`	s15.16
`signed long _Accum`	s4.23	`_lAccum`	s7.24

avrfix Function Overview

- Automatically only the used functions are included in the final program
- Trigonometric functions are implemented using COordinate Rotation DIgital Computer (CORDIC) algorithm



Differences of avrfix to DTR 18037

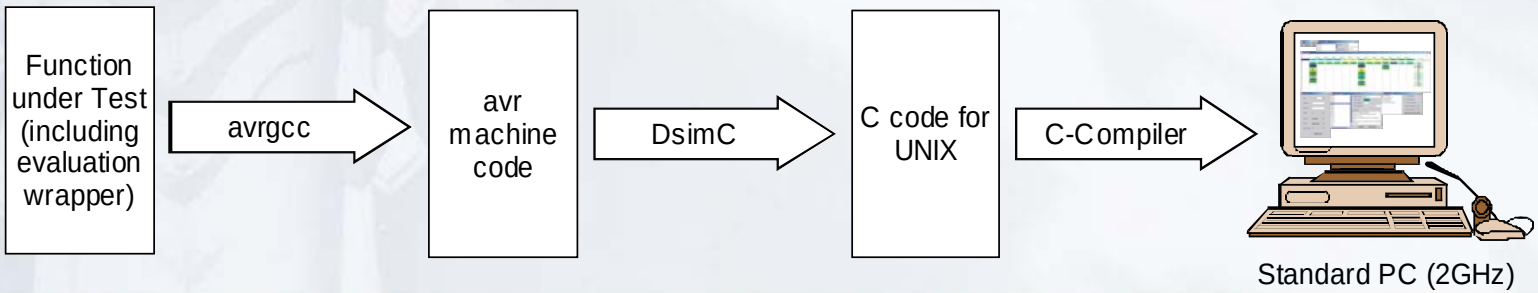
- No support for _Fract data types
  no technical reason, we just didn‘t need them
- c=mulk(a,b) instead of c=a*b
  since there‘s no operator overloading in C
- **#define** instead of **#pragma** for setting saturation and precision level
  since we did not change the compiler

Performance Evaluation

- Thorough analysis of timing behavior
- For each function, a large input data set (if possible, complete) is tested
- Results give information on average and worst-case behavior
- Used compiler avrgcc 3.3.2 with –Os option

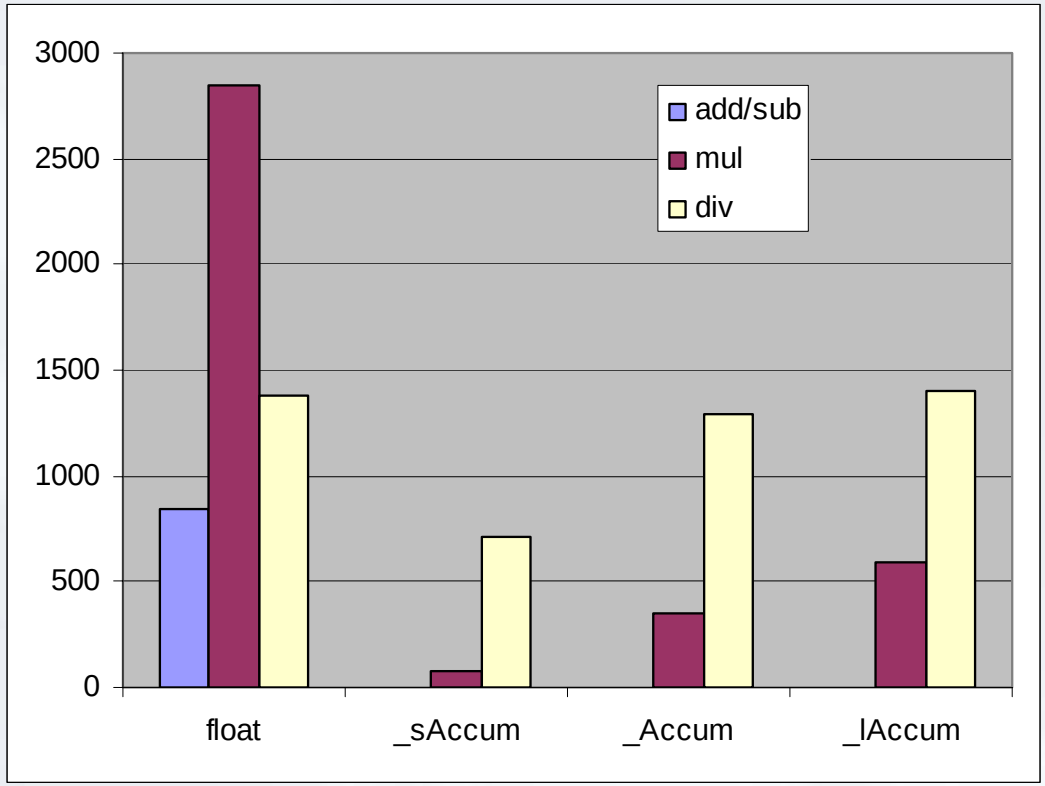
Speeding up Tests

- Exhaustive test on target architecture would take too long time
- Speed up tests by simulated AVR core
  - Java program creates simulator code out of machine code
  - Simulator runs experiments up to 30 times faster
  - I/O and timer functions need not to be simulated
  - Target hardware is simple (no caches)



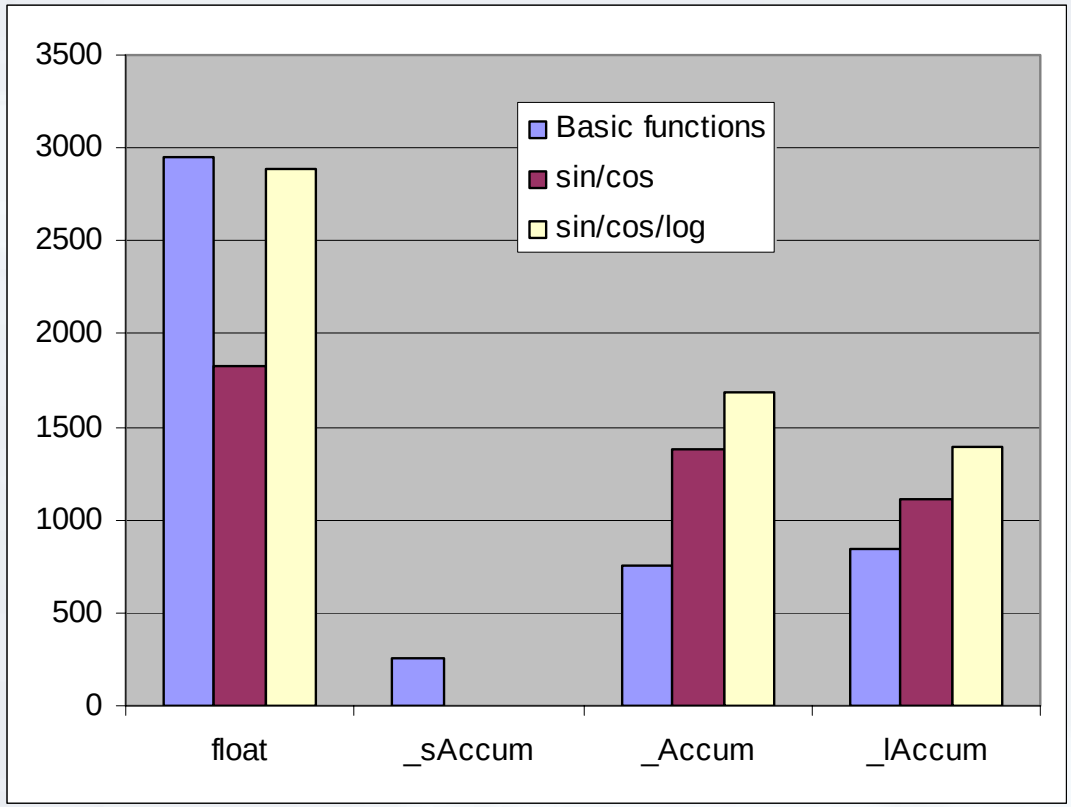
Performance Evaluation Results (WCET)

- Add/sub come almost for free
- Division of floating point is faster in the average case
- The more complex the function, the less advantage (room for optimization)



Code Size Evaluation Results

- Projects with avrfix are smaller in code
- Complex functions are not implemented for _sAccum



Outlook and Conclusion

- Open source fixed-point library: http:///sourceforge.net/projects/avrfix
- Faster for basic operations, in general smaller code
- Based on DTR 18037, enables switching to compiler version with minor modficiations
- Future work: provide a pre-compiler that supports syntax proposed in DTR 18037
- Work to do: optimize code, evaluate on new compilers

Dankeschön!

Any Questions?