Programming

Networks and Embedded Software

Introduction

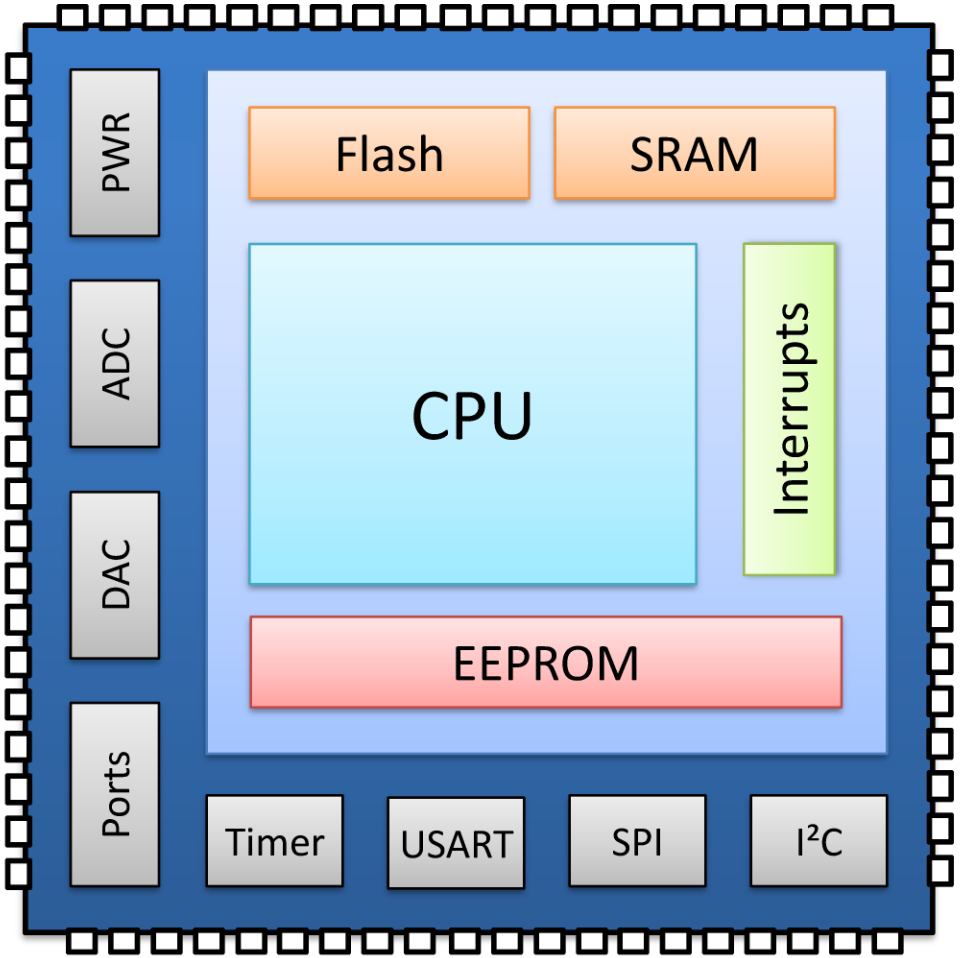
by Wolfgang Neff

ATxmega128A1 (1)

- Atmel AVR 8-bit Microcontroller
  - Introduced 2009
  - Harvard RISC architecture
  - 32 registers (8 bit)
  - 138 machine code instructions
  - 100 multiplexed and configurable pins
  - Flash and SRAM integrated on chip
  - Wide range of peripheral modules

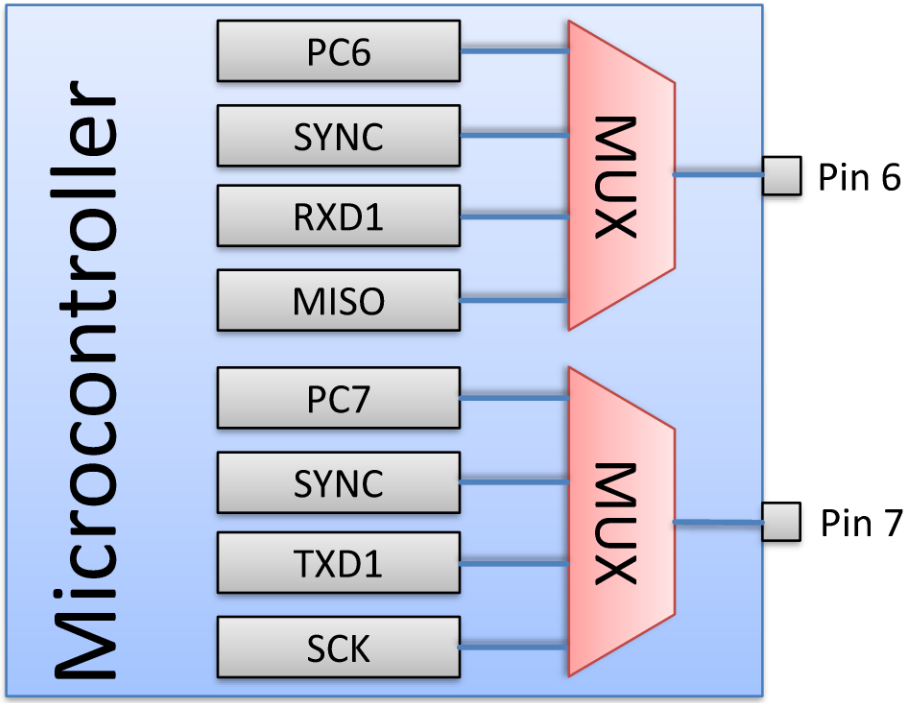
ATxmega128A1 (2)

- Block Diagram
  - Power lines
  - I/O Ports
    - A, B, C, D, E, F, …
  - Serial Ports
    - USART, SPI, I²C
  - Converter
    - ADC, DAC
  - Timer etc.



ATxmega128A1 (3)

- Alternate Port Functions
  - Pins are multiplexed (consult datasheet for details)



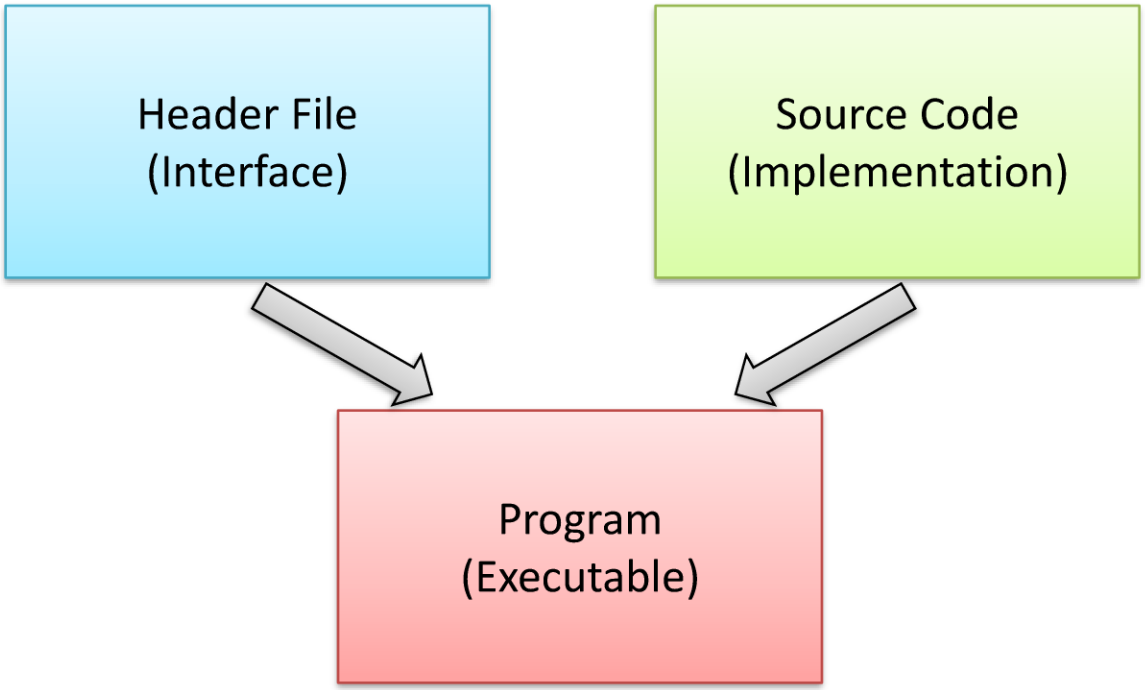
Port	INT	USARTC1	SPIC
PC0	SYNC		
PC1	SYNC		
PC2	SYNC		
PC3	SYNC		
PC4	SYNC		SS
PC5	SYNC	XCK1	MOSI
PC6	SYNC	RXD1	MISO
PC7	SYNC	TXD1	SCK

ATxmega128A1 (4)

- Peripheral Modules
  - Peripherals are organised as modules
  - Modules have several instances
    - Eight timers, eight USARTs, four SPIs etc.
  - Modules and instances are like classes and objects
  - Example
    - PORTA.DIR
      PORTA.OUT
      PORTE.DIR
      PORTE.OUT

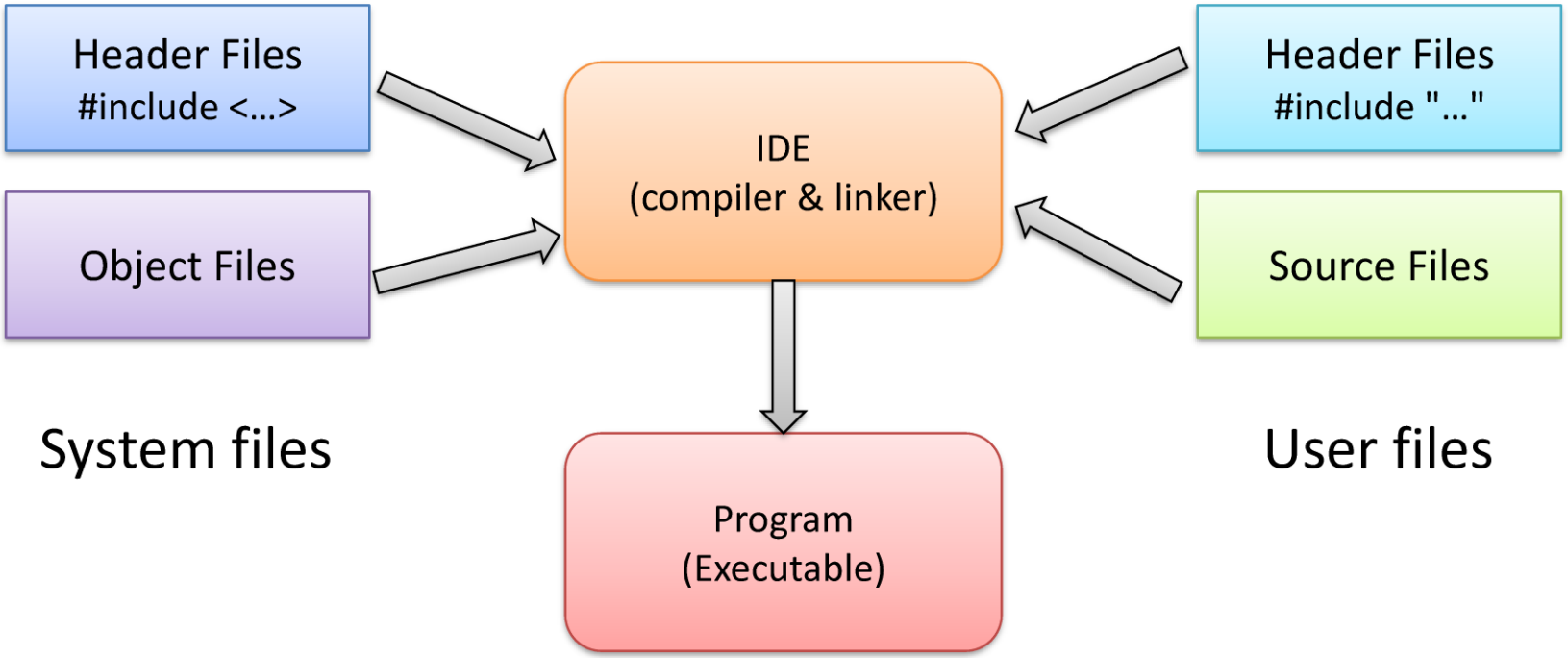
C Programming (1)

- Code file organization



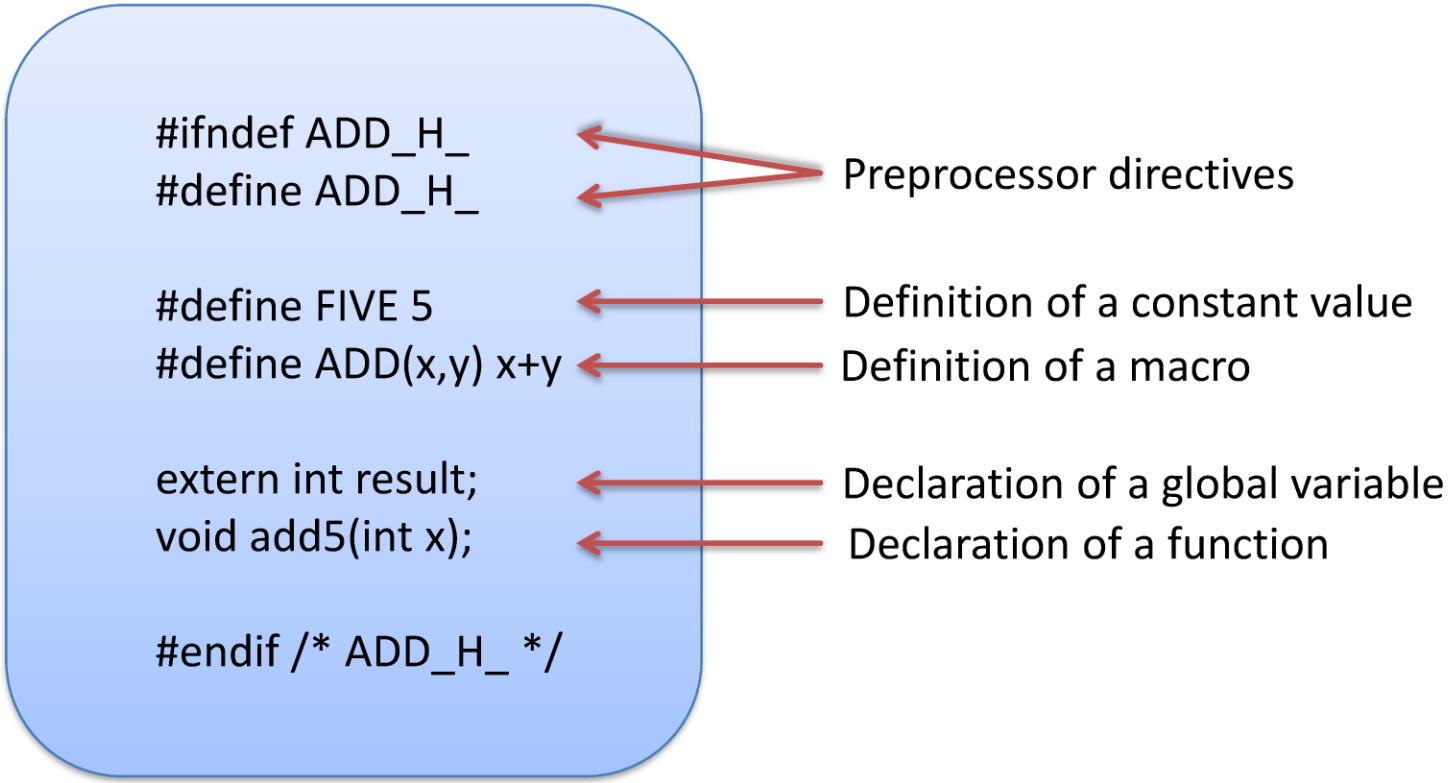
C Programming (2)

- Code generation



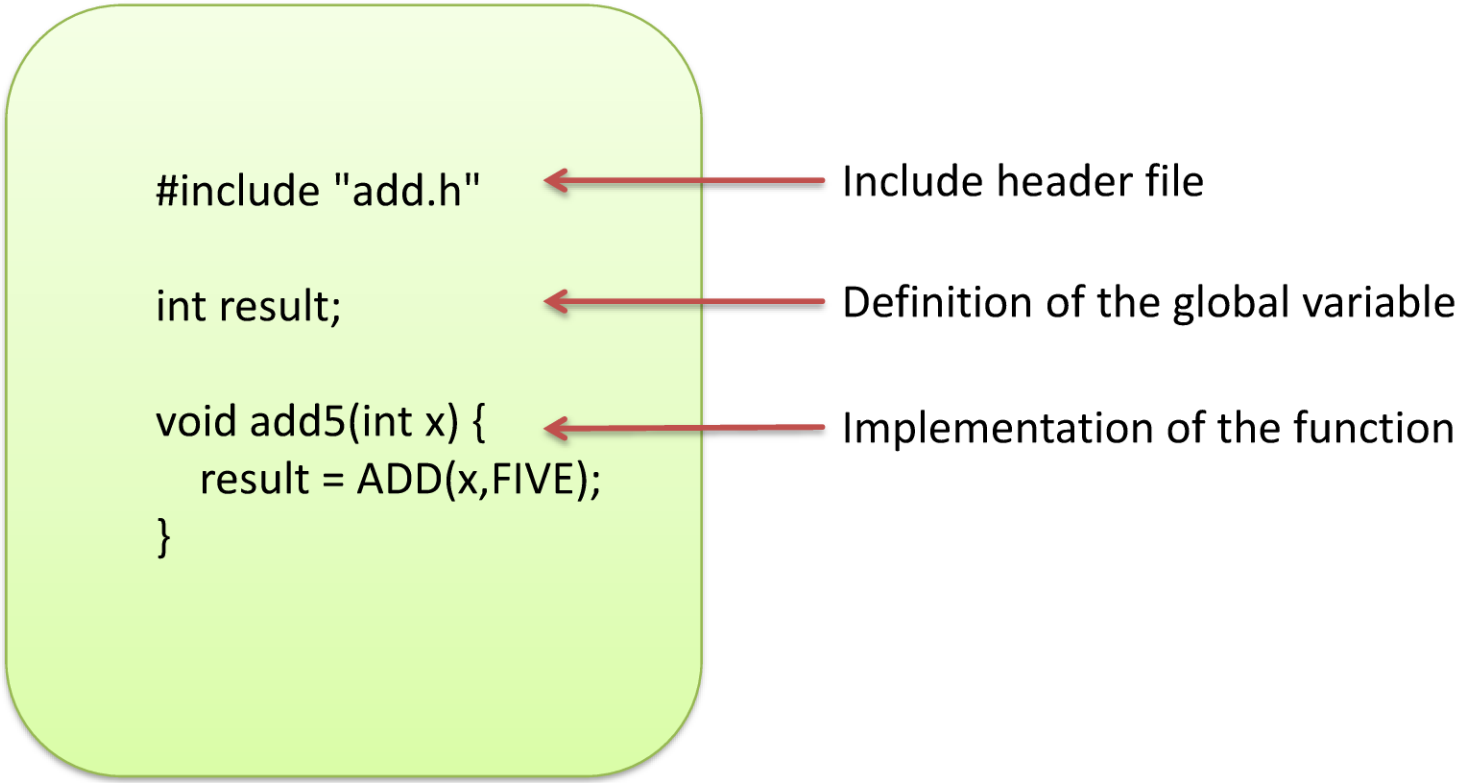
C Programming (3)

- Header file



Programming C (4)

- Source file



Programming C (5)

- Main file

```
#include <avr/io.h>
#include "add.h"

int main(void) {

   add5(4);

   while (1) {
      /* do something */
   }

}
```

Include a system header file

Include a user header file

Main program

Call function declared in add.h
(result now in global variable)

Infinite main loop
(embedded systems, only)

No good idea!

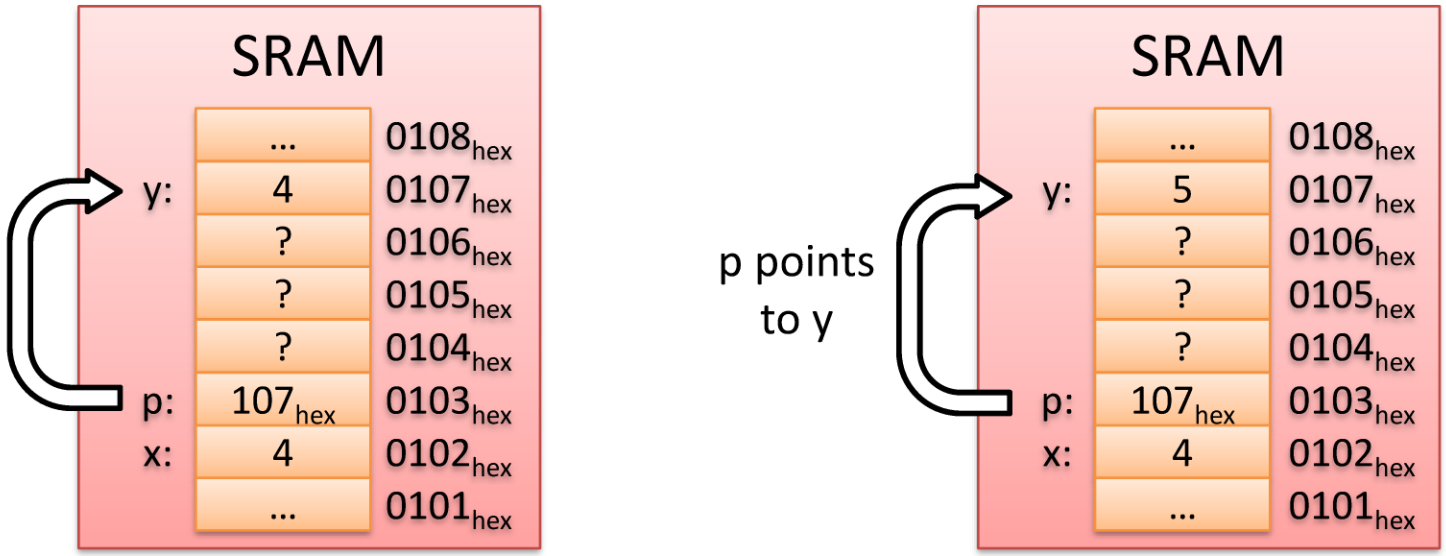
C Programming (6)

- Data types

Standard C	Alias	Embedded Systems*	Size
signed char		int8_t	8 Bits
unsigned char	char	uint8_t	8 Bits
signed short	int	int16_t	16 Bits
unsigned short		uint16_t	16 Bits
signed long		int32_t	32 Bits
unsigned long		uint32_t	32 Bits
float			32 Bits
double		(* Defined in <stdint.h>)	64 Bits

C Programming (7)

- Indirect addressing with pointers




```
int x = 4;    // immediate
int y = x;    // direct
int *p = &y;
```

```
*p = 5;    // indirect
```

C Programming (8)

- Input / Output
  - Special drivers
    - E. g. usart_put, usart_get
  - Standard input / output
    - #include <stdio.h>
    - Simple input / output
      - Characters: getchar, putchar
      - Strings: gets, puts
    - Complex input / output
      - scanf / printf

C Programming (9)

- Formatted output
  - int printf(const char *format, ...);
  - Format Specifiers
    - %d decimal signed integer
    - %x hex integer
    - %u unsigned integer
    - %c character
    - %s string
    - %p pointer

C Programming (10)

- Example
  - ```
    char s = 'A';                    // define sensor
    uint16_t d = adc_read(s);        // read sensor
    int v = d / 15;                  // convert to voltage
    ```
  - ```
    printf("Sensor %c read '%x'. This are %d V.",s,d,v);
    ```
  - Result:
    - Sensor A read 'A1'. This are 10 V.
- Drawback
  - Very expensive – storage and performance

C Programming (11)

- Volatile values
  - Not under the control of the C compiler
  - Change asynchronously
  - Optimization would be harmful
  - Example
    - volatile uint8_t i = USART.DATA;
      volatile uint8_t j = USART.DATA;
      if (i != j) { … } // has USART.DATA changed?
      Dead code is optimized away if volatile is omitted.