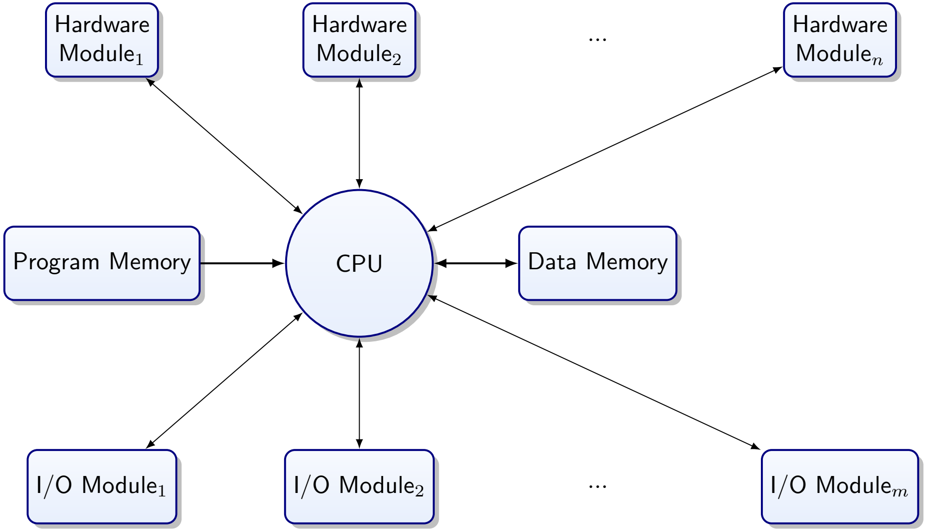
Memory Mapped I/O

Klaus-Peter Zauner

COMP2215: Computer Systems II

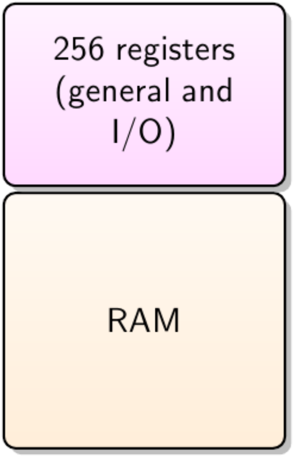
μC = CPU + Memory + Hardware Modules

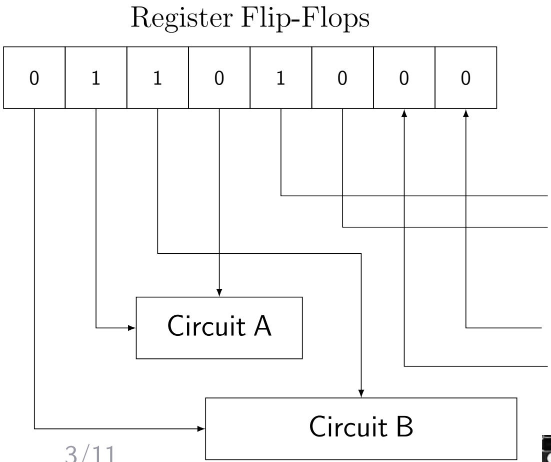


How is Software connected to Hardware?

Control registers

Control registers are sets of flip flops which are not only connected for reading and writing: their inputs or outputs are wired into other circuits.





How does the CPU interact with the control registers?

Two methods:

I/O Instructions

- Instruction set of processor has special I/O commands
- I/O commands control I/O port registers

Memory Mapped I/O

- I/O registers have addresses in reserved memory space
- Memory access

In some CPUs a mix of both methods is used.

How does the CPU interact with the control registers?

Two methods:

I/O Instructions

- Instruction set of processor has special I/O commands
- I/O commands control I/O port registers

Memory Mapped I/O

- I/O registers have addresses in reserved memory space
- Memory access

In some CPUs a mix of both methods is used.

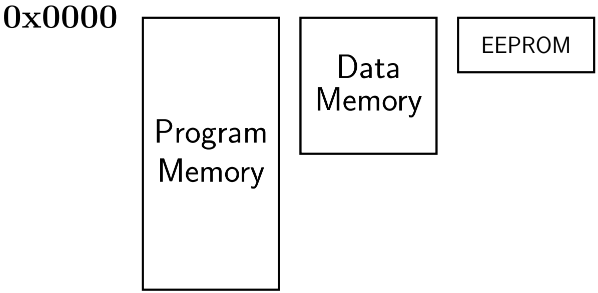
Special Instructions *vs.* Memory mapping

- Size of address space
- Convenience of access
  - different addressing modes
- Size of instruction set
  - Instruction bits are precious

Special Instructions *vs.* Memory mapping

- Size of address space
- Convenience of access
  - different addressing modes
- Size of instruction set
  - Instruction bits are precious
- Cache complications!
  - Input registers will change without instructions from the CPU

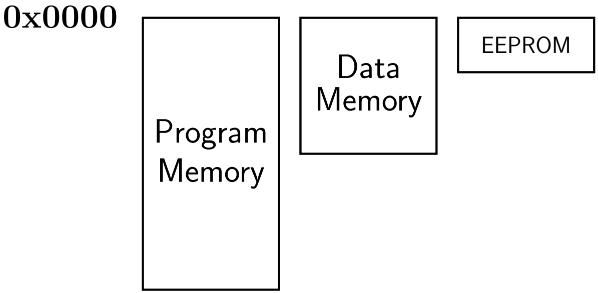
AT90USB1286: Address Spaces



- Program Memory: 0x00000 → 0x1FFFF
- EEPROM: 0x0000 → 0x0FFF

The C programming language assumes a single address space (von Neumann architecture). Workaround: the intended address space is indicated to the linker by a specific big offset.

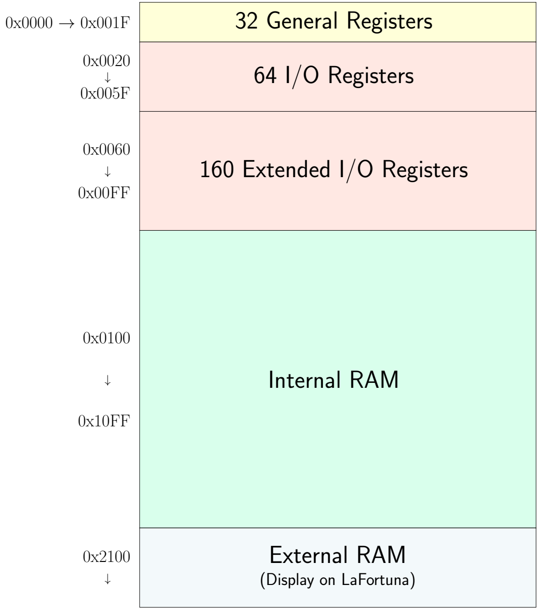
AT90USB1286: Address Spaces



- Program Memory: 0x00000 $\rightarrow$ 0x1FFFF
- EEPROM: 0x0000 $\rightarrow$ 0x0FFF

The C programming language assumes a single address space (von Neumann architecture). Workaround: the intended address space is indicated to the linker by a specific big offset.

AT90USB1286: Data Memory Map

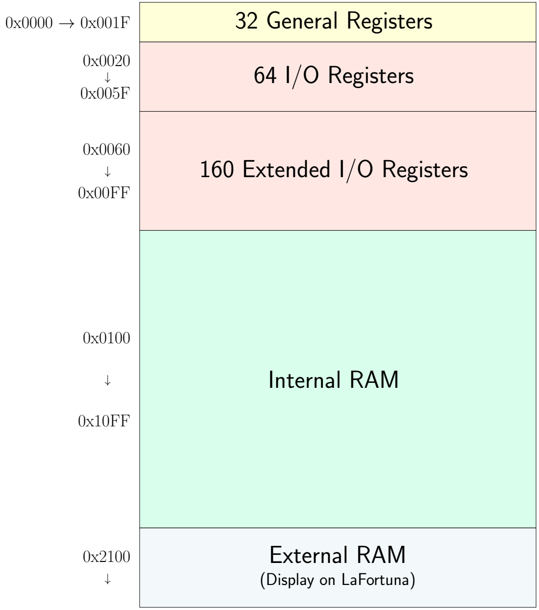


All hardware modules on the microcontroller are configured by writing to I/O registers.

All communication to and most communication from these modules is facilitated by reading and writing I/O registers.

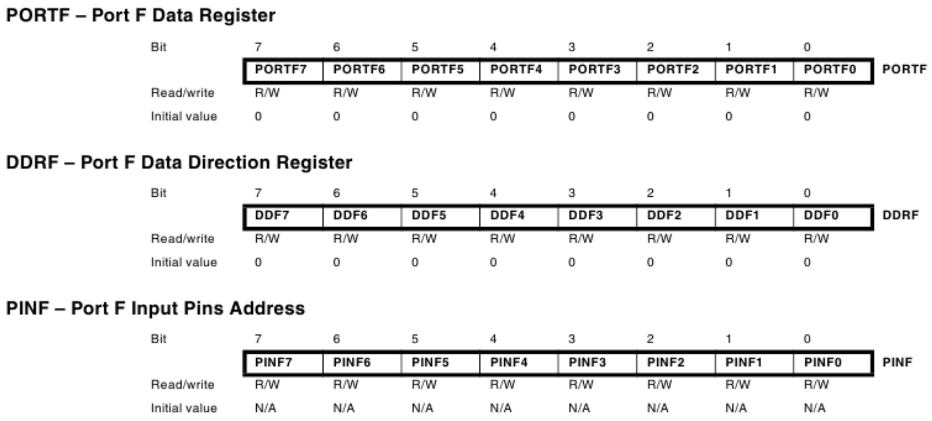
Note that even the general purpose registers are mapped into memory address space.

AT90USB1286: Data Memory Map



All hardware modules on the microcontroller are configured by writing to I/O registers.
All communication to and most communication from these modules is facilitated by reading and writing I/O registers.

Note that even the general purpose registers are mapped into memory address space.



PORTF – Port F Data Register

Bit	7	6	5	4	3	2	1	0	
	PORTF7	PORTF6	PORTF5	PORTF4	PORTF3	PORTF2	PORTF1	PORTF0	PORTF
Read/write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

DDRF – Port F Data Direction Register

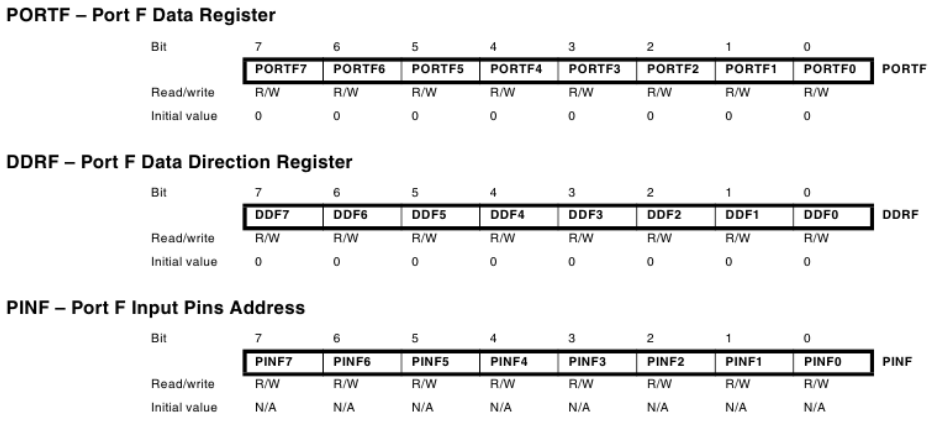
Bit	7	6	5	4	3	2	1	0	
	DDF7	DDF6	DDF5	DDF4	DDF3	DDF2	DDF1	DDF0	DDRF
Read/write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

PINF – Port F Input Pins Address

Bit	7	6	5	4	3	2	1	0	
	PINF7	PINF6	PINF5	PINF4	PINF3	PINF2	PINF1	PINF0	PINF
Read/write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

DS, p. 91

- PORTF is mapped to a unique memory address
- DDRF is mapped to a unique memory address
- PINF is mapped to a unique memory address



PORTF – Port F Data Register

Bit	7	6	5	4	3	2	1	0	
	PORTF7	PORTF6	PORTF5	PORTF4	PORTF3	PORTF2	PORTF1	PORTF0	PORTF
Read/write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

DDRF – Port F Data Direction Register

Bit	7	6	5	4	3	2	1	0	
	DDF7	DDF6	DDF5	DDF4	DDF3	DDF2	DDF1	DDF0	DDRF
Read/write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	0	0	0	0	0	0	0	0	

PINF – Port F Input Pins Address

Bit	7	6	5	4	3	2	1	0	
	PINF7	PINF6	PINF5	PINF4	PINF3	PINF2	PINF1	PINF0	PINF
Read/write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

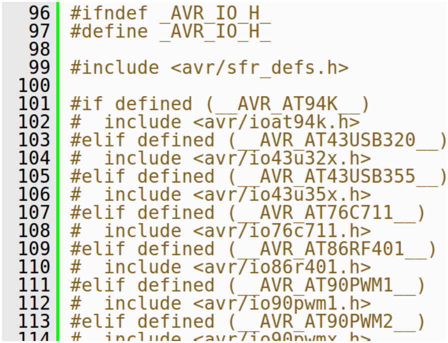
DS, p. 91

`io.h` will include—over several indirections—a set of C-preprocessor `#define` statements that define constant labels (e.g., DDRF) for the correct address of that register in the target chip.

Manipulating register bits

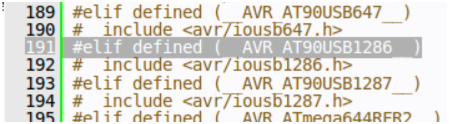
- 8-bit registers $\longrightarrow$ unsigned $\Rightarrow$ Type: `uint8_t`
  - `stdint.h`: `typedef unsigned char uint8_t`
- Setting a bit (=1):
  - `DDRB |= _BV(PB7)`
- Clearing a bit (=0):
  - `DDRB &= ~_BV(PB7)`

avr-libc: avr/io.h



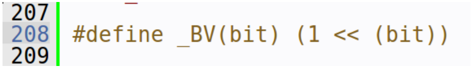
```
#ifndef _AVR_IO_H_
#define _AVR_IO_H_

#include <avr/sfr_defs.h>

#if defined (__AVR_AT94K__)
#  include <avr/ioat94k.h>
#elif defined (__AVR_AT43USB320__)
#  include <avr/io43u32x.h>
#elif defined (__AVR_AT43USB355__)
#  include <avr/io43u35x.h>
#elif defined (__AVR_AT76C711__)
#  include <avr/io76c711.h>
#elif defined (__AVR_AT86RF401__)
#  include <avr/io86r401.h>
#elif defined (__AVR_AT90PWM1__)
#  include <avr/io90pwm1.h>
#elif defined (__AVR_AT90PWM2__)
#  include <avr/io90pwmx.h>
```

⋮



```
#elif defined (__AVR_AT90USB647__)
#  include <avr/iousb647.h>
#elif defined (__AVR_AT90USB1286__)
#  include <avr/iousb1286.h>
#elif defined (__AVR_AT90USB1287__)
#  include <avr/iousb1287.h>
#elif defined (__AVR_ATmega644RFR2__)
```

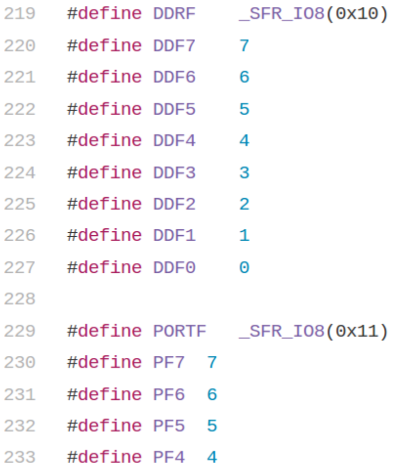
avr/io.h; see also notes from avr/sfr_defs.h and avr/portpins.h.



```
#define _BV(bit) (1 << (bit))
```

avr/sfr_defs.h

avr/iousb1286.h → avr/iousbxx6_7.h



```
#define DDRF   _SFR_IO8(0x10)
#define DDF7   7
#define DDF6   6
#define DDF5   5
#define DDF4   4
#define DDF3   3
#define DDF2   2
#define DDF1   1
#define DDF0   0

#define PORTF  _SFR_IO8(0x11)
#define PF7  7
#define PF6  6
#define PF5  5
#define PF4  4
```