Lesson

C-Programming etc.

(HWP I1)

C/C++ contra Java

- Header files (*.h)
- Pre processor
  - #define: constants, macros etc.
  - #ifndef
- args in main: args[0] is the command itself
- Conditions – assignments ==/=
- Pointers
  - Can point directly to memory addresses
- Memory management

C-Hints

- Declaring of variables with `volatile` keyword means that the compiler will not optimize on the variable
  Ex:
  ```
  volatile unsigned short *pIOPORT4 = (unsigned short *)0x7FE2;
  ```
  Multiple writes to `pIOPORT4` will not be discarded of the optimizer
- Use `struct` to declare registers in peripherals Ex:
  ```
  struct TimerCounter
  {
      unsigned short count;      // Current Count, offset 0x00
      unsigned short maxCount;   // Max Count, offset 0x02};
  ```

C-Hints

- Manipulation of bits
  - Set bits:
    ```
    aWord |= 1<<3 | 1<<5;// Set bit 3 and 5
    ```
  - Clear bits:
    ```
    aWord &= ~(1<<2 | 1<<7); //Clear bit 2
    and 7
    ```
  - Toggle bits:
    ```
    aWord ^= (1<<4); // Toggle bit 4
    ```
- Use constants for bit numbers
  ```
  #define MIRQ 3
  maskRegister |= 1<<MIRQ;
  ```

AVR Toolchain

- Defines for all MCU registers are available if `<avr/io.h>` is include and the correct MCU type is set in the Project properties
- Download the manual to MCU to be used in your project (http://www.atmel.com/products/)

Toolchain

- *avr-gcc* is special version of the *gcc* compiler
- Take a look at the macros defined in *avr/sfr_defs.h*
  - `_BV( 3 ) // Converts a bit-number to a byte-value`
    - Does the same as `(1<<3)`
  - `is_bit_set( PINA, PA3 )`
    - Does the same as: `PINA & (1<<PA3)`
  - `is_bit_cleared( PINA, PA6 )`
    - Does the same as: `~(PINA & (1<<PA6))`

Introduction to AVR

- Microcontroller
  – Complete systems with FLASH/IO/RAM etc.
- Pins has typically more than one function
  – Analog inputs, Digital input/outputs, Timer input/output, Interrupt inputs etc.
  – Function are chosen by setting bits in registers

AVR Introduction

- Registers for standard IO-Ports

Port D Data Register – PORTD

Bit	7	6	5	4	3	2	1	0	
	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
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	

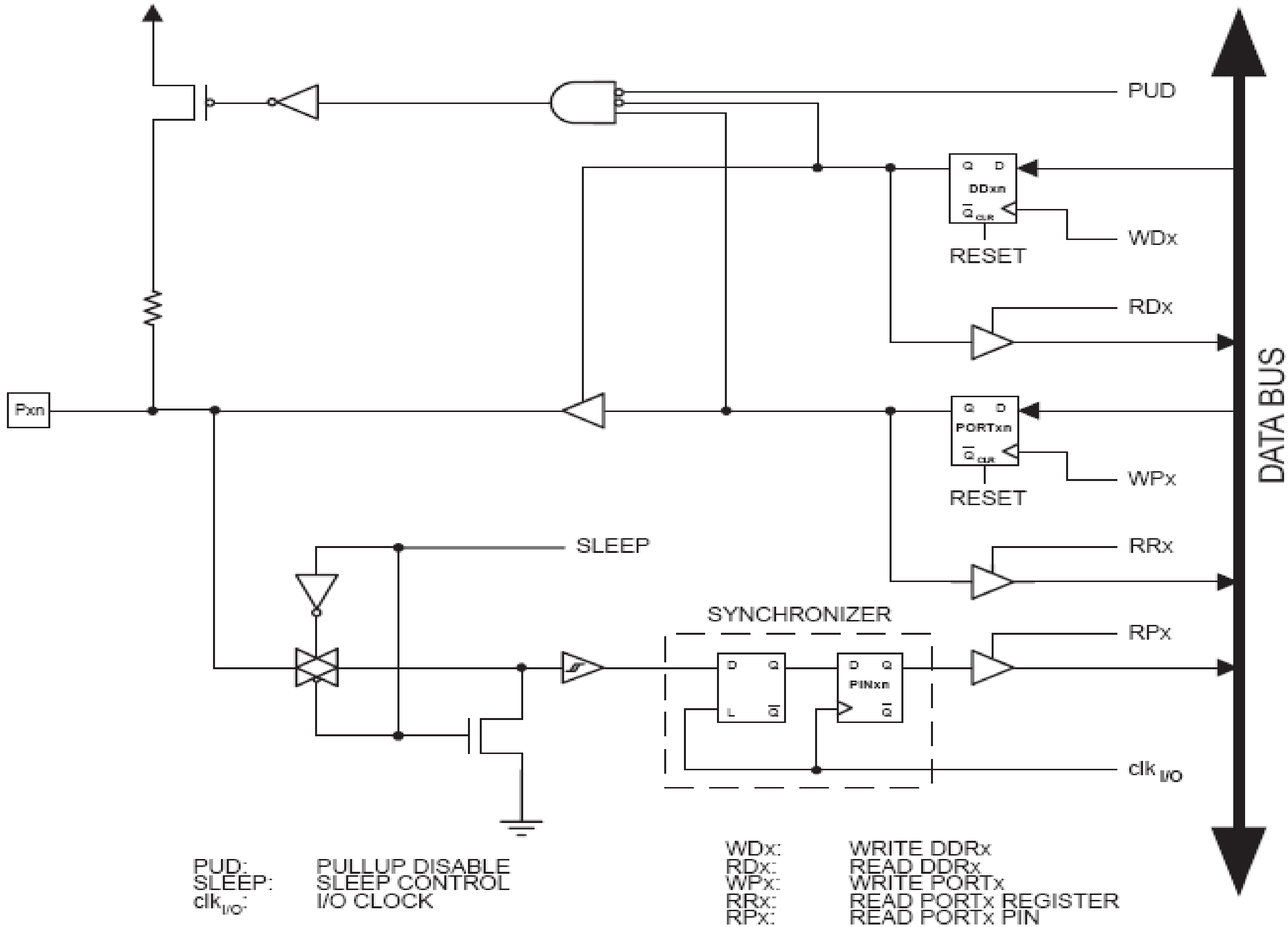
Port D Data Direction Register – DDRD

Bit	7	6	5	4	3	2	1	0	
	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
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	

Port D Input Pins Address – PIND

Bit	7	6	5	4	3	2	1	0	
	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

AVR-Introduction (General IO-pin)



AVR-Introduction

- How to setup a port

Table 24. Port Pin Configurations

DDxn	PORTxn	PUD (in SFIOR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

Introduction to AVR

- Example of alternative port functions: PORT D in a ATMEGA8515 can be used as a normal 8 bit port or used with the alternative function in the table:

Table 35. Port D Pins Alternate Functions

Port Pin	Alternate Function
PD7	$\overline{\text{RD}}$ (Read Strobe to External Memory)
PD6	$\overline{\text{WR}}$ (Write Strobe to External Memory)
PD5	OC1A (Timer/Counter1 Output Compare A Match Output)
PD4	XCK (USART External Clock Input/Output)
PD3	INT1 (External Interrupt 1 Input)
PD2	INT0 (External Interrupt 0 Input)
PD1	TXD (USART Output Pin)
PD0	RXD (USART Input Pin)

AVR programming

Register Description for I/O Ports

Port A Data Register – PORTA

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
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	

Port A Data Direction Register – DDRA

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
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	

```
// setup 2 lowest pins in port A to output

DDRA |= _BV(DDA0) | _BV(DDA1);

// set port A pin 1 to logical 1
PORTA |= _BV(PORTA1);

// read port A pin 3

uint8_t status = PINA & _BV(PINA3);
```

GNU Tools

- Macro definition
  - In source: #define ‘macro’
  - Command line: -D’macro’
- Source
  - ANSI C
- Other tools
  - make
  - Debugger
    - gbd
    - Can be used after crash

GNU Compiler

- *gcc* is the same as *cc* known from Unix
- Messages
  - Errors
    - Must be fixed to be able to compile
  - Warnings
    - Bad practice
    - Use max warning level: -Wall
- Command line options
  - -W warning level
  - -g include debug info in target
  - -o define output filename
  - -c don’t link
  - -D define macro
  - -S Generate ASM code (.s files)
- Source code
  - Use always Unix/Linux Path naming ex:
    `#include <avr/interrupt.h>`

Good practice

- Divide source modules in natural layers with separate responsibility
  - One for hardware near (hardware dependent)
  - One for high-level language interface
  - One for each separate functionality
- Leads to software that is
  - Easy to understand
  - Easy to expand
  - Generally easy to maintain
- Use defines and constants for all hardware addresses and other dependency

Software layers

Application in high level language (Java/C++/C#/VB)	`// Print voltage from the speed sensor` `System.out.println( voltageIn( card, channel));`
Language interface	`double voltageIn( short card, short channel )` …
Driver (C/C++/Assembler)	`voltageIn:     ldx io04` `               xor #$FE    mask mux bit`
Hardware interface	Some hardware
Peripherals/sensors/actuators	Analog to digital converter (A/D)

Good practice

- Create header-files that defines your hardware
- Create a memory map
- Create a IO-Map
- Create a Interrupt-Map
- Drivers demands extra good documentation
- Comment the functionality of your program
  – not the syntax of your code

Problems with embedded sw

- Bugs in the software corrupts the functionality of the complete system
  - Deadlocks
  - Memory overrun
- It is difficult to signal internal problems
  - Typical very small/simple user interfaces