UiO : University of Oslo

FYS3240-4240
Data acquisition & control

Essential Electronics & Microcontrollers

Spring 2019 – Lecture #4

Reading: RWI Chapter 2 page 39-57 (and Ch 4 for C-programming)

Bekkeng, 28.12.2018

Essential Electronics

Topics you should know

- Duty-cycle
- Port
- High-impedance state (outputs)
- Pull-up and pull-down
- Relay driver
- Counters and Timers

Pulse-width modulation (PWM)

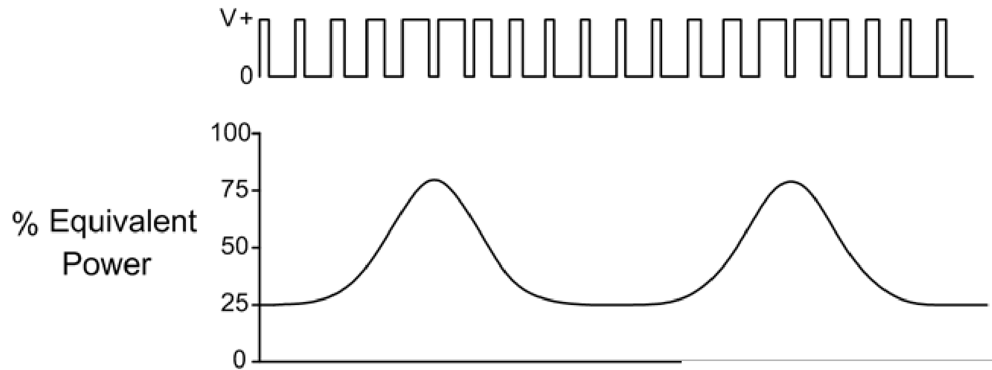


Figure 2-35. Pulse-width modulation

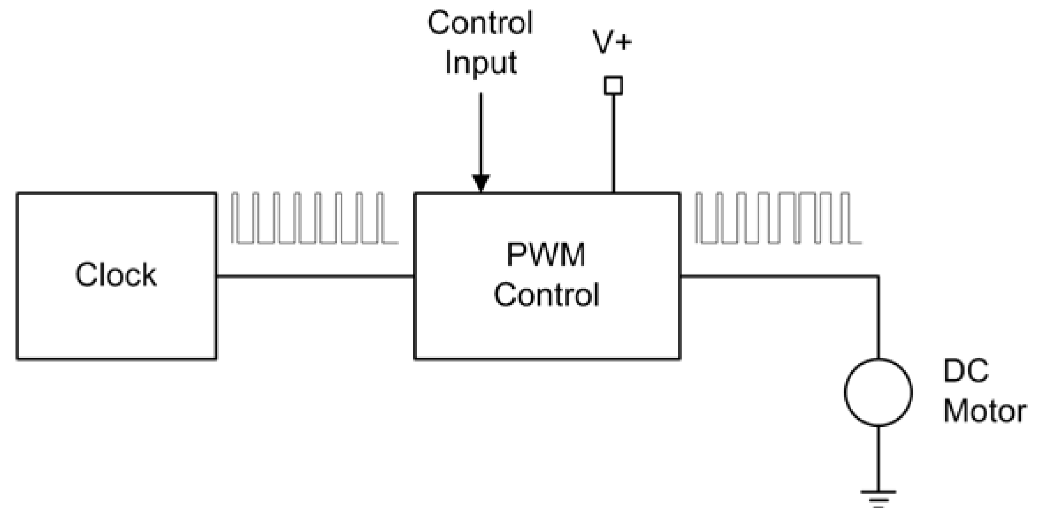
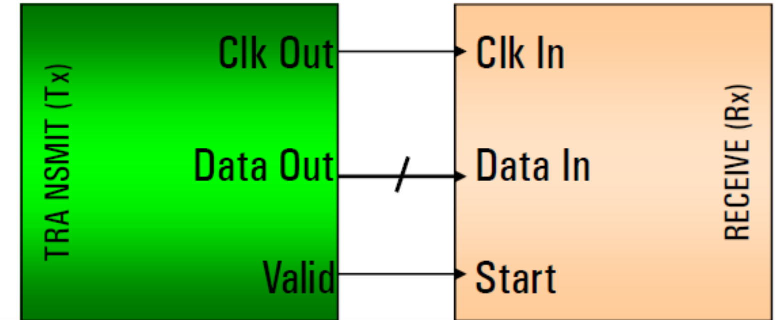
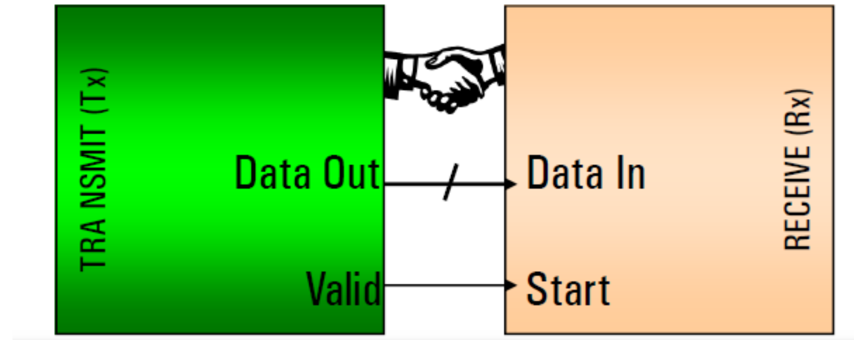


Figure 2-36. PWM DC motor control

Serial I/O

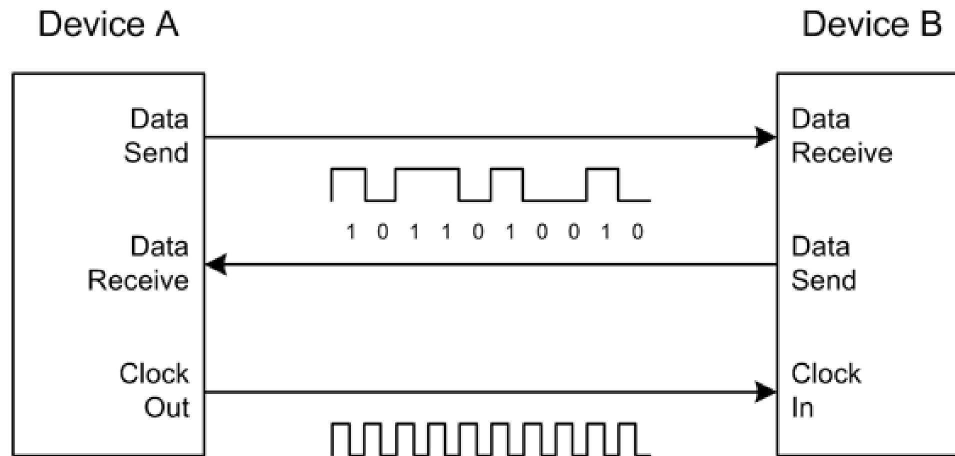
- **SynchronousTransfer**
  - **Data sent at constant rate using a shared periodic clock**
- **AsynchronousTransfer (i.e., Handshaking)**
  - **Data sent upon request using handshake signals**
  - Tx and Rx still have internal clocks, they just don't share them

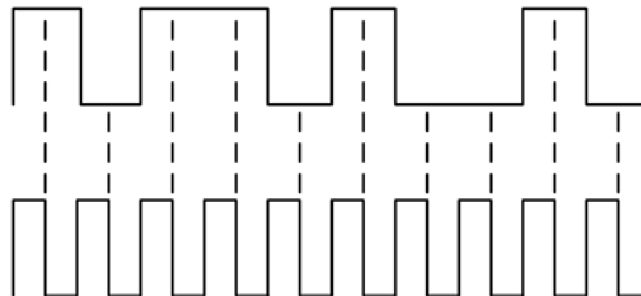




Asynchronous transfer means that the information is not sent in predefined time slots. Data transfer can start at any given time and it is the task of the receiver to detect when a message starts and ends.

Synchronous serial data communication

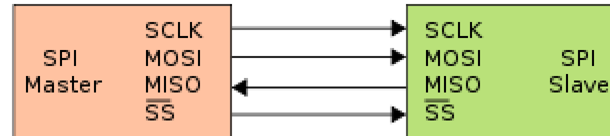




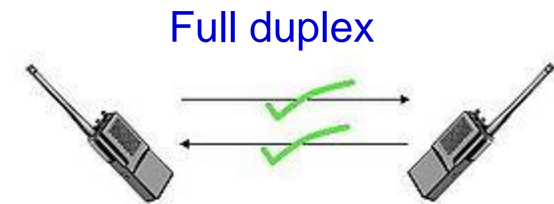
Data is valid on the falling edge of the clock signal.

Figure 2-37 RWI

SPI



Full duplex



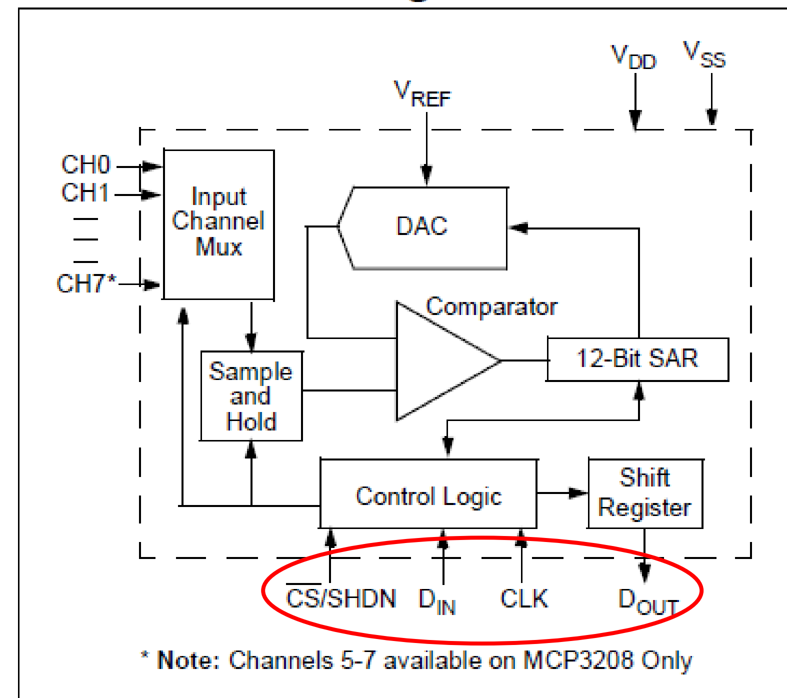
- SPI = Serial Peripheral Interface
- Serial data link (bus) standard that operates in full duplex mode
- Devices communicate in master/slave mode where the master (only one master) device initiates the data frame. Multiple slave devices are allowed with individual slave select (chip select) lines.
- Sometimes SPI is called a "four-wire" serial bus, contrasting with three-, two-, and one-wire serial buses.
  - Serial Clock (output from master)
  - Serial Data In
  - Serial Data Out
  - nCS
- Bit rate usually in the MHz range
- Short distance communication
  - Longer cables means lower speed (because of cable capacitance etc.)

Notes:

- CS = Chip Select (an enable signal)
- An n before a signal name, such as nCS, means that the signal is active low
- A bar over a signal name, such as $\overline{CS}$, means that the signal is active low

ADCs with SPI interface

- Many ADCs have an SPI interface
- Example: MCP3204
  - *4-Channel,12-Bit A/D Converters with SPI Interface*



Asynchronous serial interface

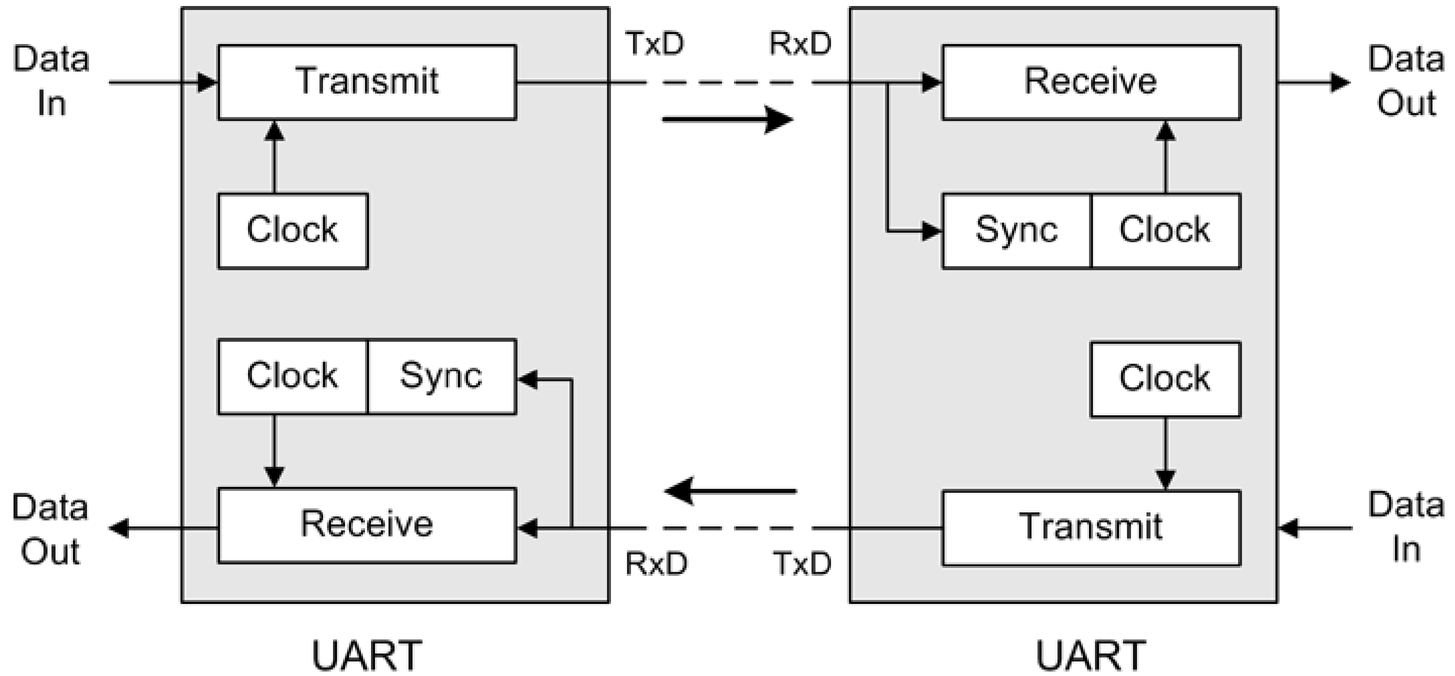
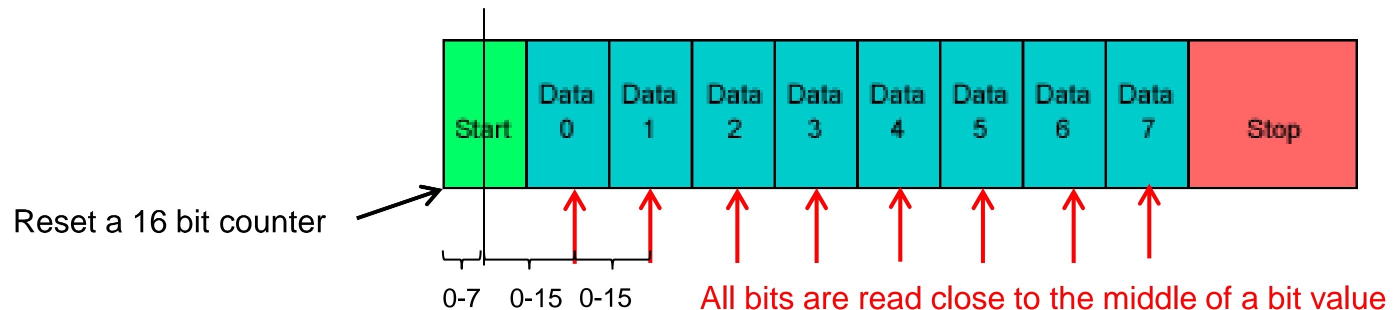


Figure 2-38. Asynchronous serial interface

UART

- UART = Universal Asynchronous Receiver/Transmitter.
- A UART is usually an individual integrated circuit used for serial communications over a computer or peripheral device serial port.
- UARTs are now commonly included in microcontrollers.
- A dual UART, or **DUART**, combines two UARTs into a single chip.
- Many modern ICs now come with a UART that can also communicate synchronously; these devices are called **USARTs** (universal synchronous/asynchronous receiver/transmitter).



A clock 16 times faster than the bit clock is used as input to the counter

Parallell I/O

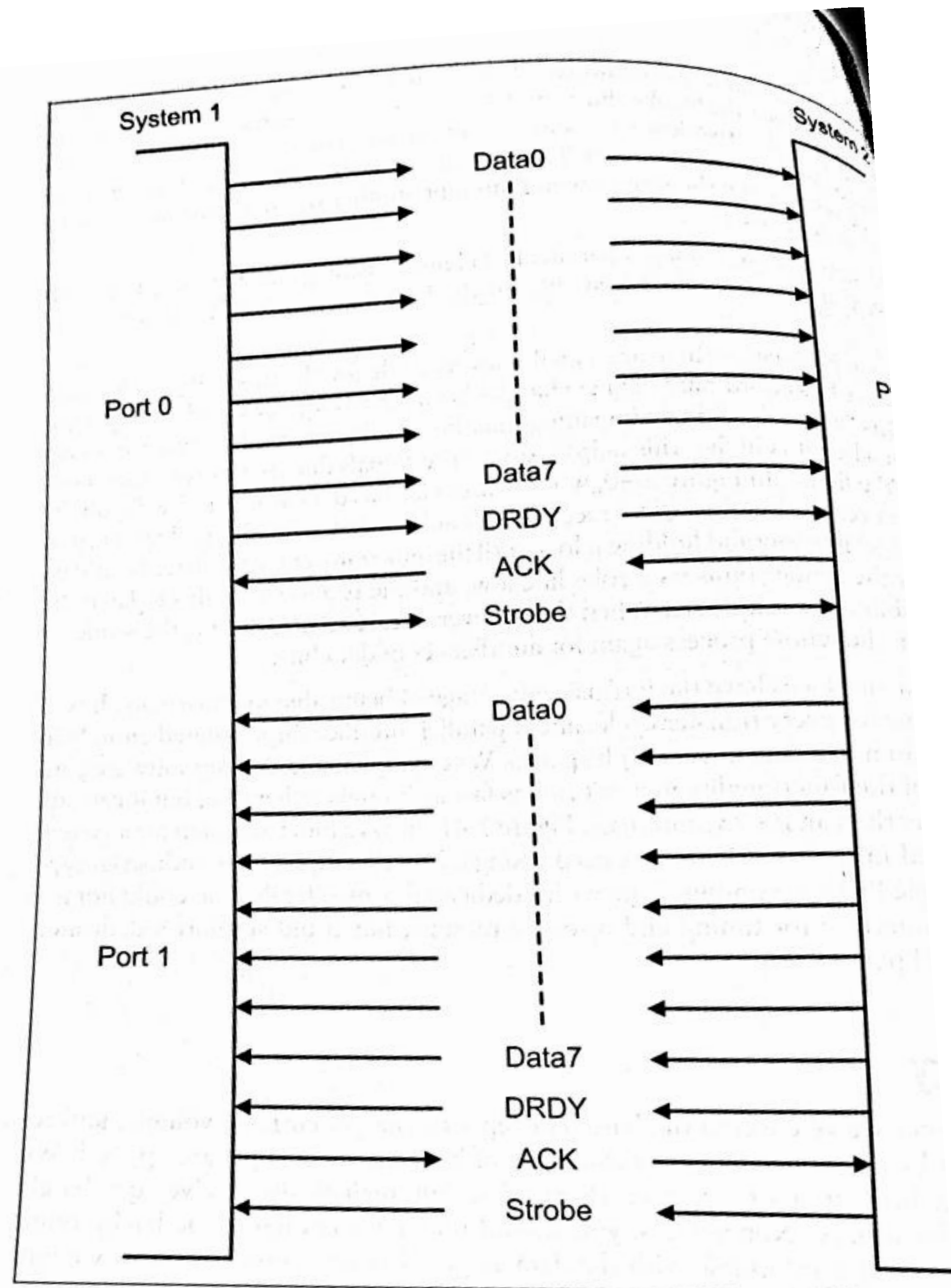


Figure 2-40. Bidirectional parallel interface

Microcontrollers

Lab: AVR Studio

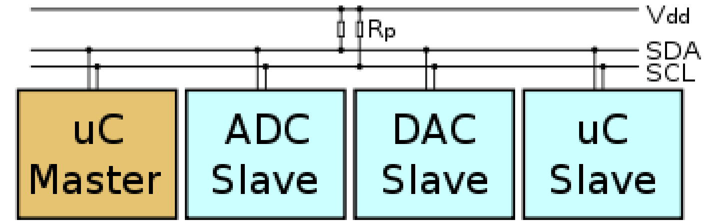
- **Microcontrollers can be programmed using Assembly or C language**
  - In FYS3240/4240 you have to program in C

- **AVR studio 5**
  - Works as editor for Assembly and C
  - Integrated C-compiler (AVR-GCC)

http://www.avrfreaks.net/

Some common uC resources

- Counters
- UART (Universal Asynchronous Receiver/Transmitter)
- A/D Converters (ADC)
  - Time-multiplexing of channels is common
  - Usually 12 or less bits per sample (8, 10, 12 bits common)
- SPI (Serial Peripheral Interface)
- I2C (at least slave function)



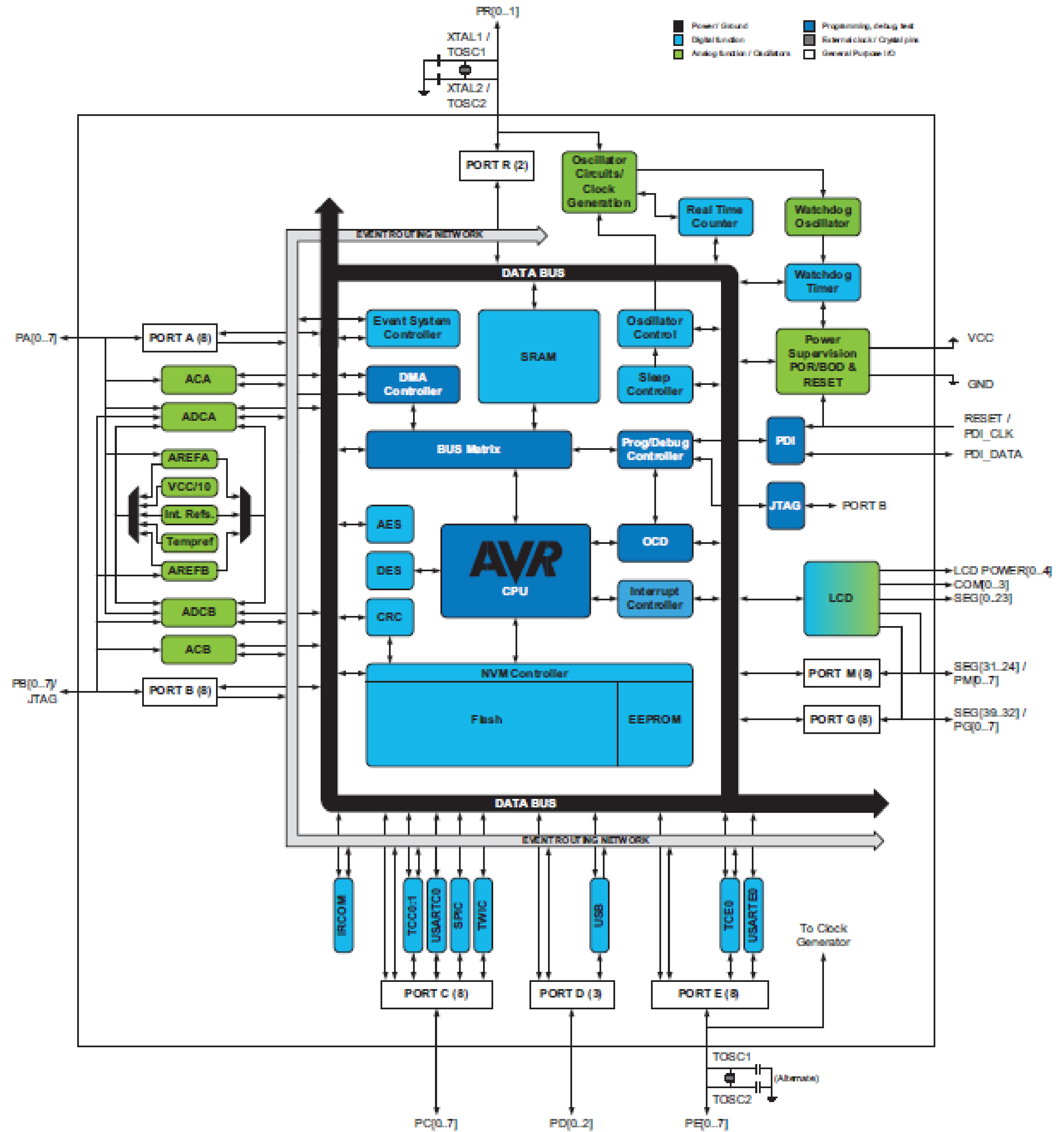
I2C

- I²C = Inter-Integrated Circuit
- Is a multi-master serial computer bus (but only one master at a time)
- Uses only two bidirectional lines
  - Data (SDA)
  - Clock (SCL)
- Speed up to 3.4 Mbit/s (high speed mode)
  - 100 kbit/s or 400 kbit/s more common?
- Practical communication distances are limited to a few meters
  - The longer the cable, the lower the speed

Example: AVR XMEGA B

- A family of low-power, high-performance, and peripheral rich CMOS 8/16-bit microcontrollers based on the AVR enhanced **RISC** (reduced instruction set) architecture
- Two-channel **DMA controller**
- Multilevel **interrupt controller**
- Up to 53 general purpose I/O lines
- 16-bit real-time counter (RTC)
- Up to three flexible 16-bit timer/counters
- Up to two **USARTs**
- **I2C** and SMBUS compatible two wire serial interface (TWI)
- One full-speed **USB 2.0** interface
- One serial peripheral interface (**SPI**)
- Up to two 8-channel, **12-bit ADCs**
- Up to four analog comparators
- Watchdog timer
- LCD controller
- Internal oscillators with PLL

Figure 2-1. Atmel AVR XMEGA B block diagram.



XMEGA

uC	I/O direction	Output	Input
XMEGA	PORTn.DIR	PORTn.OUT	PORTn.IN

PORTB.DIR = 0xFF; /* All pins in PORTB configured as output*/

Note: 0x specifies a HEX number, **0b** specifies a binary number

Example code for XMEGA

```
/* io-header for avr: */
#include <avr/io.h>

/* general gnu c */
#include <inttypes.h>

int main(void)
{
    /* use uint8_t to save space in uC */
    uint8_t value;

    /* Set Port B direction to output */
    PORTB.DIR = 0xff;

    /* Initialize value */
    value = 0xff;

    /* Run forever */
    for(;;) {
        /* Write value to Port B */
        PORTB.OUT = value;

        /* Decrement value */
        value--;
    }

}
```

Interrupt-Driven Programming

- In interrupt-driven systems software is designed such that when a registered event, such as a timer, is received, a response is fired to respond to this event.
- There are two components of any interrupt-driven system: **the interrupt** and **the interrupt handler**.
- An interrupt is a signal that is generated by hardware, which indicates an event has occurred that should halt the currently executing program.
- Interrupt handlers (also referred to as **interrupt service routines - ISRs**) are portions of code that are registered with the processor to execute once a particular interrupt has occurred. Once the processor is aware of an interrupt, it halts the currently executing process, performs a context switch to save the state of the system, and executes the interrupt handler. Once the interrupt handler code has executed, the processor returns control to the previously running program.

Interrupts

- Interrupts halt normal code execution in order to go do something more important or time sensitive
- Used Instead of polling
- Can be generated internally or externally
- Interrupts are used e.g. for:
  - RESET
  - Timers
  - Time-Critical Code
  - Hardware signaling
    - such as a switch pressed by the user

C programming

Note: in C you can only call functions that have been defined/decleared earlier

```
#include <stdio.h>

int gcd (int a, int b)
{
  while (a != b) {
    if (a > b) a -= b;
    else       b -= a;
  }
  return a;
}

void demo (int n, int m)
{
  printf("GCD(%d,%d) = %d\n", n, m, gcd(n,m));
}

int main (void)
{
  demo(7,3);
  demo(18,24);
}
```

```
#include <stdio.h>

void say_hello(void);

int main(void)
{
    say_hello();
}

void say_hello(void)
{
    printf("hello!\n");
}
```

Increment Operators

```
int i = 42;
i++;      // increment on i
// i is now 43
i--;      // decrement on i
// i is now 42
```

i+=   // in order to specify how much to increment

C programming – Pointers I

A pointer is an address in RAM

```
int* intPtr;    // declare an integer pointer variable intPtr

char* charPtr; // declares a character pointer --
               // a very common type of pointer
```

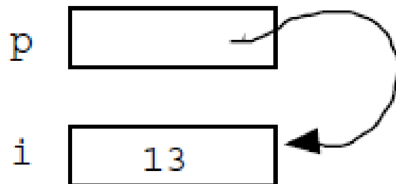
```
void foo() {
   int* p;  // p is a pointer to an integer
   int i;   // i is an integer

   p = &i;  // Set p to point to i
   *p = 13; // Change what p points to -- in this case i -- to 13

   // At this point i is 13. So is *p. In fact *p is i.
}
```

&i get the address of the variable i

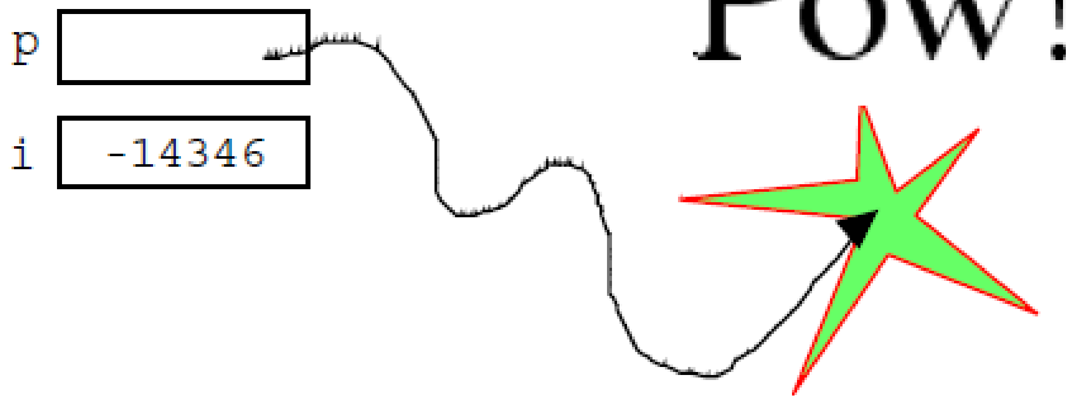
***p** get the content of the address location that the pointer points to



C programming – Pointers II

```
{
    int* p;

    *p = 13;      // NO NO NO p does not point to an int yet
                  // this just overwrites a random area in memory
}
```



C programming - Memory Management

- Allocate memory : malloc()
- Free used memory : free()

```
{
    int a[1000];

    int *b;
    b = (int*) malloc( sizeof(int) * 1000);
    assert(b != NULL);      // check that the allocation succeeded

    a[123] = 13;      // Just use good ol' [] to access elements
    b[123] = 13;      // in both arrays.

    free(b);
}
```

Look out for memory leakage!