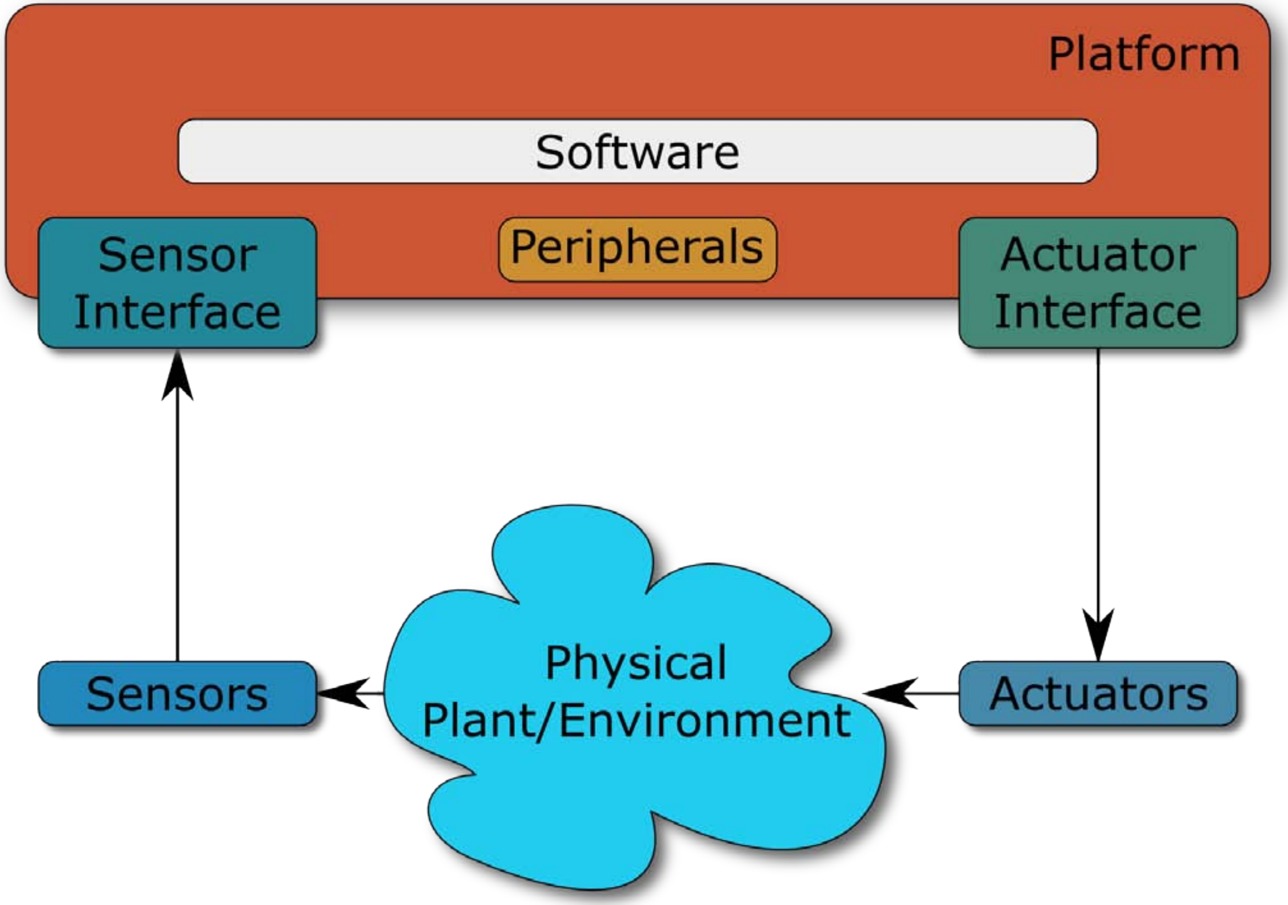
Real-Time Systems

Part 3: Embedded Hardware

Motivation


Platform
Software
Sensor Interface
Peripherals
Actuator Interface
Physical Plant/Environment
Sensors
Actuators

Content

- Embedded Processors
- Memory Architectures
- Interrupts
- Interfacing to Sensors & Actuators
- Sensors & Actuators

Embedded Processors

- Microcontroller
- DSP
- ASIC
- PLD (FPGA)

Embedded Processors – Microcontrollers

- A microcontroller is a microprocessor combined with peripheral devices
  - Computer on a chip
- Optimized for embedded applications with
  - Many interrupts
  - High I/O functionality
- Standard architectures
  - Intel 8051
  - Atmel AVR
  - ARM

Embedded Processors – RISC vs. CISC

- Reduced instruction set RISC (e.g. ARM – Acorn Risc Machine)
  - Only simple machine instructions; Compiler has to map high-level language onto simple instructions
  - All instructions take the same time
  - Used in embedded systems (Real-time hardware, smart phones, …)
- Complex instruction set CISC (e.g. x86)
  - Map complexity of common instructions directly in machine code
  - Complex instructions can consist of several simple instructions
  - Can lead to subtle timing issues
  - Used in general purpose computing

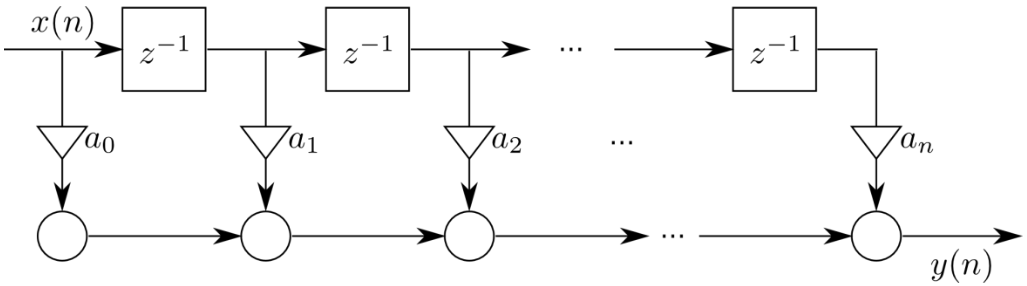
Embedded Processors – DSP

- Digital Signal Processor (DSP)
- Optimized for signal processing
  - Filtering
  - System identification
  - Frequency analysis
  - Machine learning
  - Feature extraction

Embedded Processors – DSP

- Example: FIR filter – Tapped delay line

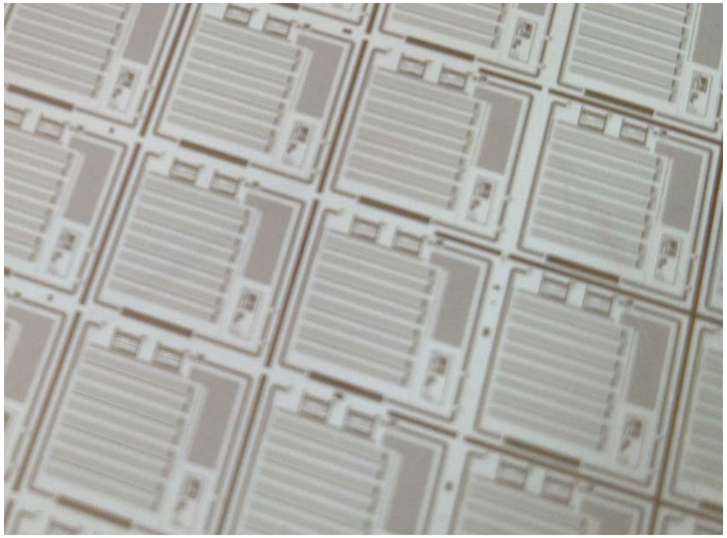
$$y(n) = \sum_{i=0}^{N-1} a_i x_{n-i}$$



- Common operation: Multiply & Accumulate (MAC)
=> DSPs implement MAC operation as single instruction

Embedded Processors – ASIC

- Application specific integrated circuit (ASIC)
- Instead of using off the shelf hardware and implementing tasks in software, the tasks can be implemented in a fully application specific hardware design
- Functionality cannot be changed after shipping
- Only economically feasible for large quantities
  - Expensive development
  - High initial production costs



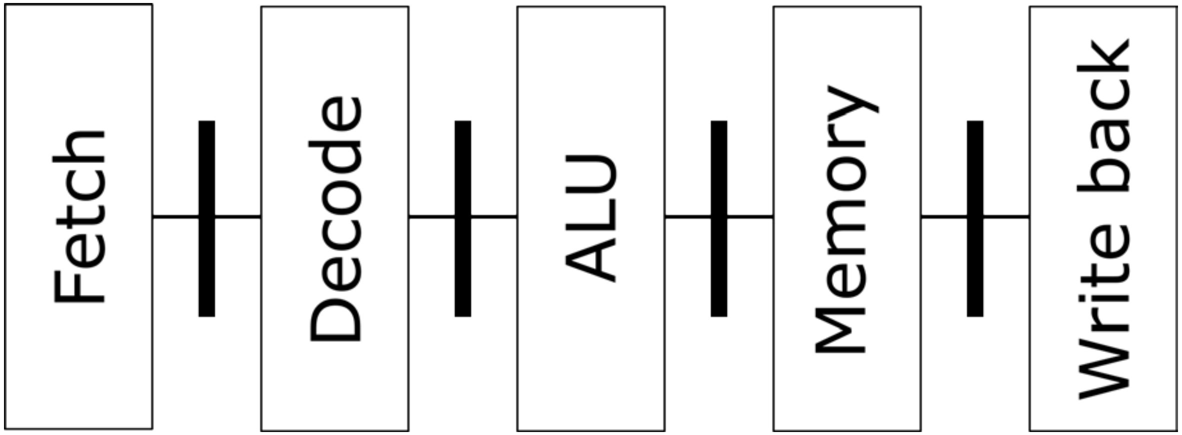
Embedded Processors – PLD

- Programmable Logic Device (PLD)
  - Electronic circuit that can be reconfigured to implement logic functions

- Field programmable gate array (FPGA) class of PLD that is programmable „in field“
  - Bridge the gap between standard hardware (microcontroller, DSP) and fully application specific hardware (ASIC)
  - Used for fast implementation of logic functions
  - Programmable through VHDL, Verilog
  - Used for low piece numbers (otherwise implementation as ASIC)

Wikimedia Commons

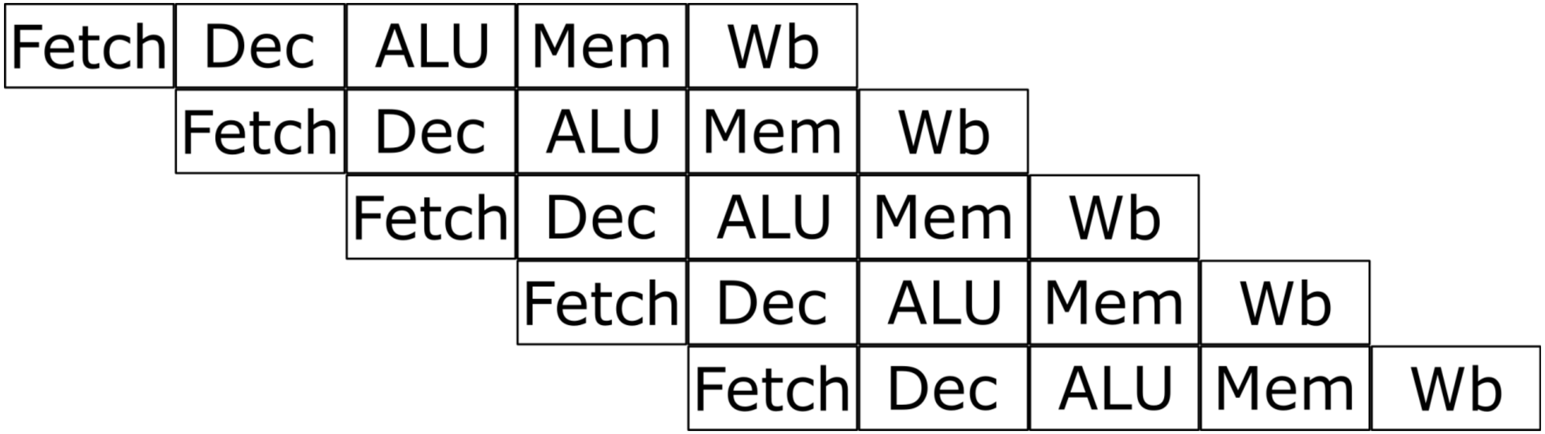
Embedded Processors - Pipelining



- Machine instructions are executed in several stages (Example: 5 stages)

1. Fetch instruction from memory
2. Decode the instruction
3. Execute the instruction on the arithmetic Logic Unit
4. Memory access (load/store)
5. Write result back into register

Embedded Processors - Pipelining



Fetch	Dec	ALU	Mem	Wb				
	Fetch	Dec	ALU	Mem	Wb			
		Fetch	Dec	ALU	Mem	Wb		
			Fetch	Dec	ALU	Mem	Wb	
				Fetch	Dec	ALU	Mem	Wb

- What happens when an operation needs data from previous command?
  - Resource hazard
  - Data hazard
  - Control hazard

Embedded Processors – Pipeline hazards

- Speculative execution:
  - Most likely branch is loaded into pipeline (branch prediction)
    Prediction works based on previous decisions
  - Undo any side effects (flush pipeline) in case of wrong estimation
- Pipelining makes timing analysis extremely difficult
  This needs to be taken into account for hard real-time constraints.
  (e.g. ARM processors have a 3-stage pipeline)
- Elaborate pipelines (superpipelining) not suitable

Embedded Processors – Pipelining

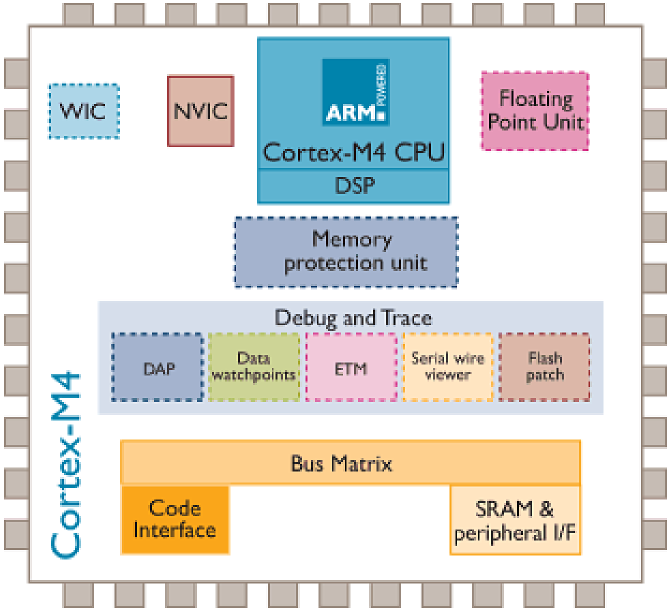
```
const unsigned aSize = 32768;
int data[aSize];

for(unsigned c = 0;
    c < aSize; ++c)
 data[c] = std::rand() % 256;

for(unsigned i = 0;
    i < 100000; ++i)
{
  // Primary loop
  for(unsigned c = 0;
      c < aSize; ++c)
  {
    if (data[c] >= 128)
      sum += data[c];
}}
```

- Code runs in 11.54 sec
- With sorted data, code runs in 1.93 sec

http://stackoverflow.com/questions/11227809/why-is-processing-a-sorted-array-faster-than-an-unsorted-array

Embedded Processors – FPU

- Not all microprocessors include a floating point unit (FPU)
- Especially in microcontrollers often ommitted for simplicity
  - Soft float emulates floating point operations by using several ALU instructions
  - Implementation in fixed point algebra



www.nxp.com

Embedded Processors – Caches

- Advanced processors use caches to store data that is often accessed to improve access times
- Different Caching strategies can improve average access times
- If data is requested that is not in cache, data has to be retrieved from the original storage
  => cache miss
- Caches make timing issues difficult to analyse

Embedded Processors – Summary

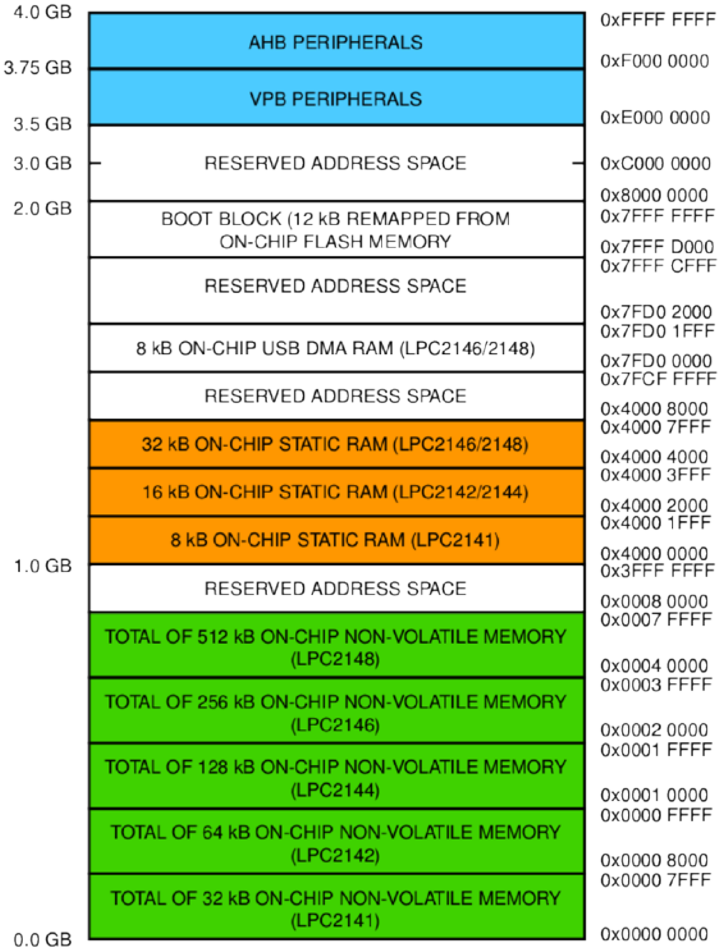
- Hard real-time constraints call for hardware where it is possible to „count cycles“
- Non-deterministic behavior is unwanted even for mechanisms that increase the overall throughput
- Resource requirements (e.g. memory, IO, processing time) need to be known when the hardware is chosen

Memory Architectures – Memory Maps

- Example of memory architecture in NXP – LPC2148

- Single address space for
  - Flash (program memory)
  - RAM (data memory)
  - Peripherals

- Addressable memory space in a 32 bit architecture: 2^{32} Byte = 4GB



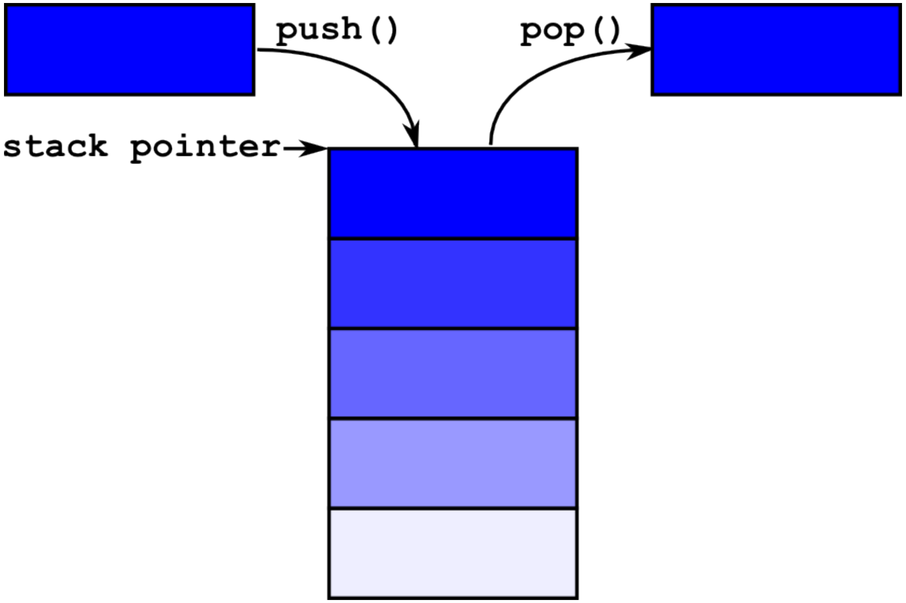
4.0 GB
3.75 GB
3.5 GB
3.0 GB
2.0 GB
1.0 GB
0.0 GB

AHB PERIPHERALS
VPB PERIPHERALS
RESERVED ADDRESS SPACE
BOOT BLOCK (12 kB REMAPPED FROM ON-CHIP FLASH MEMORY
RESERVED ADDRESS SPACE
8 kB ON-CHIP USB DMA RAM (LPC2146/2148)
RESERVED ADDRESS SPACE
32 kB ON-CHIP STATIC RAM (LPC2146/2148)
16 kB ON-CHIP STATIC RAM (LPC2142/2144)
8 kB ON-CHIP STATIC RAM (LPC2141)
RESERVED ADDRESS SPACE
TOTAL OF 512 kB ON-CHIP NON-VOLATILE MEMORY (LPC2148)
TOTAL OF 256 kB ON-CHIP NON-VOLATILE MEMORY (LPC2146)
TOTAL OF 128 kB ON-CHIP NON-VOLATILE MEMORY (LPC2144)
TOTAL OF 64 kB ON-CHIP NON-VOLATILE MEMORY (LPC2142)
TOTAL OF 32 kB ON-CHIP NON-VOLATILE MEMORY (LPC2141)

0xFFFF FFFF
0xF000 0000
0xE000 0000
0xC000 0000
0x8000 0000
0x7FFF FFFF
0x7FFF D000
0x7FFF CFFF
0x7FD0 2000
0x7FD0 1FFF
0x7FD0 0000
0x7FCF FFFF
0x4000 8000
0x4000 7FFF
0x4000 4000
0x4000 3FFF
0x4000 2000
0x4000 1FFF
0x4000 0000
0x3FFF FFFF
0x0008 0000
0x0007 FFFF
0x0004 0000
0x0003 FFFF
0x0002 0000
0x0001 FFFF
0x0001 0000
0x0000 FFFF
0x0000 8000
0x0000 7FFF
0x0000 0000

Memory Architectures - Stack

- Stack is a memory space that is organized as LIFO (Last-in-first-out Memory)
- Stack pointer (register) contains the memory address of topmost value
- A function call is executed as follows
  1. Push function parameters
  2. Push return address
  3. Push local variables
- A return
  1. Pop stack frame



Memory Architectures - Stack

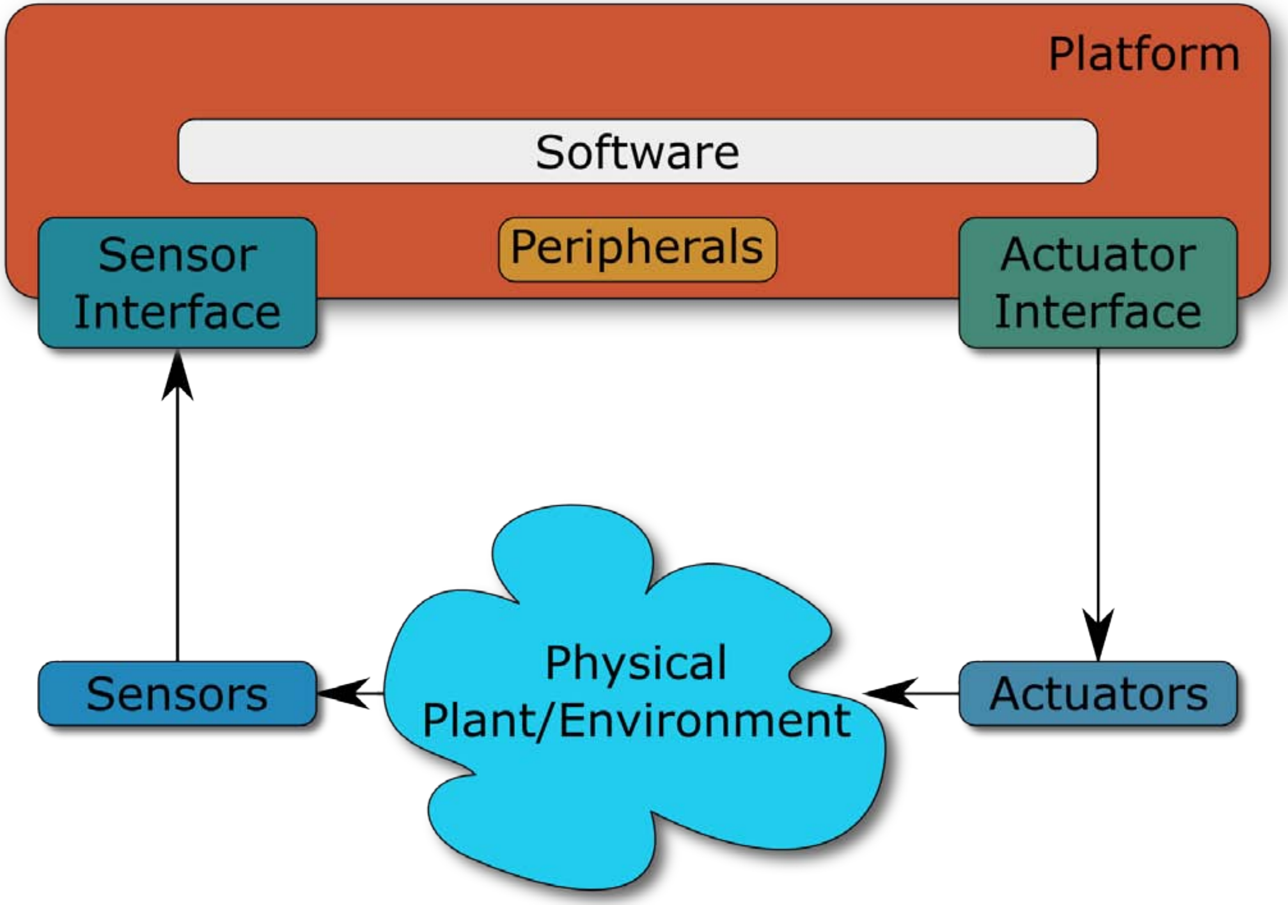
```
int *func(int a)
{
  int b;
  b = a + 15;
  return &b;
}

int main(int argv, char **argv)
{
  int *c;
  c = func(20);
  return 0;
}
```

- After returning from a function, the memory is not available anymore.
- Local variables (e.g. **`int b`**) become invalid
- !!! Returning the address of a local variable leads to undefined behavior!!!

Memory Architectures – Heap

- Dynamic memory is used when size of a data structure is unkown at compile time
- In C/C++ a **`malloc`**/**`new`** requests the operating system to allocate additional dynamic memory on the heap at run-time.
  The programmer has to take care to delete it
- In Java the garbage collector takes care of deleting unused memory
- The operating system has to take care of defragmenting the heap
- Both defragmenting and garbage collection can be very problematic in real-time systems

Motivation


Platform
Software
Sensor Interface
Peripherals
Actuator Interface
Sensors
Physical Plant/Environment
Actuators

Polling vs. Interrupt

- Mechanisms to react to changes in the environment (e.g. button pressed)

1. Polling
   continuously poll IOs for a change

- Advantages:
  - Short latencies (low number of IOs)
  - Many events do not block the normal execution of the program
- Disadvantages:
  - Most polls are unnecessary
  - High CPU usage
  - Reaction time depends on number of IOs

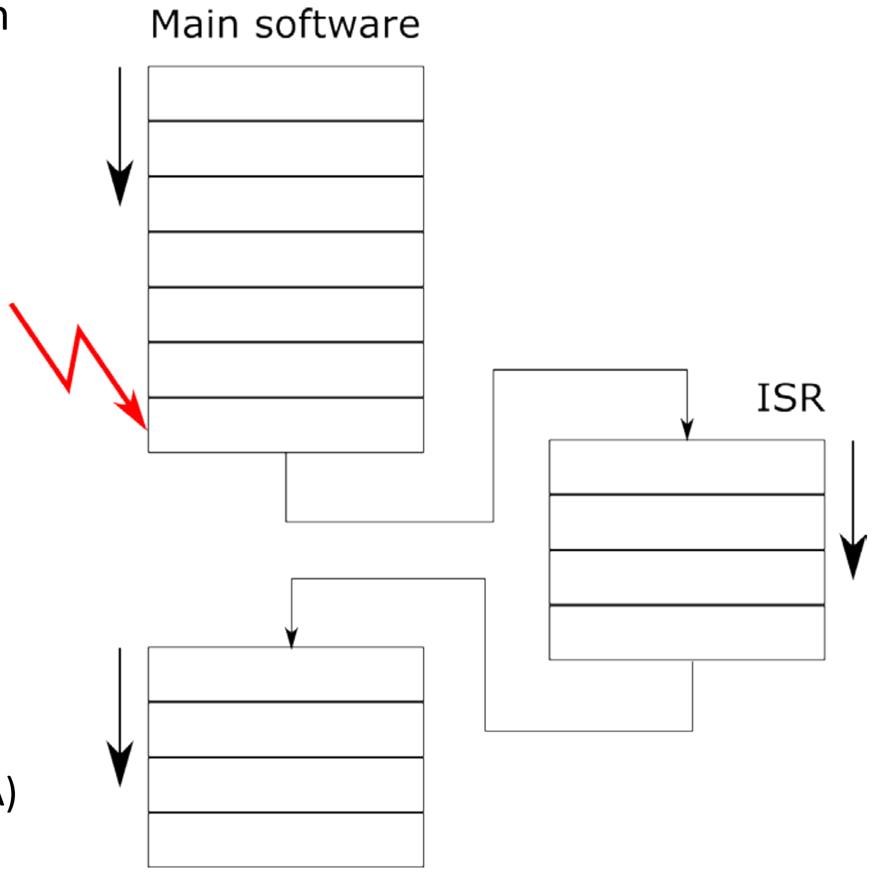
2. Interrupt
   normal execution is automatically interrupted when an event occurs

- Advantages:
  - Processor resources are only used when needed
- Disadvantages:
  - Program execution is interrupted in a non-deterministic manner

Interrupts

The processor *interrupts* the normal execution of the main software due to an event.

- **Synchronous** interrupts (traps) always happen at the same point in the code
  - Exception
  - Segmentation fault
  - Division by zero
- **Asynchronous** interrupts are caused by external (to the software) events
  - Timer overrun
  - End of IO operation (Communication, DMA)
  - External pin voltage level



Interrupt – ISR

- The ISR has to be executed without **side effects**
  (processor state should not change)
- Communication with program through global variables
- **`volatile`** key word tells compiler that variable may be changed asynchronously
- ISR can be invoked between any two machine instructions of the main software
  - Between if and assignment
  - Assignment is not always atomic
    (e.g on an 8-bit processor)

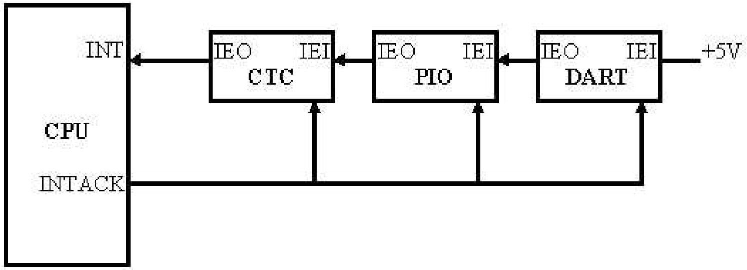
```
volatile uint32 counter = 0;

void ISR(void) __attribute__((__interrupt__))
{
  counter++;
}

int main(int argv, char **argv)
{
  …
  while(1)
  {
    if(counter > 2000)
    {
      counter = 0;
      //do stuff …
    }
  }
}
```

Interrupt – ISR

1. IRQ **triggered**
2. Check if interrupt is currently **enabled**, if yes:
   finish executing current instruction
3. **Store** content of processor registers on the stack (including the address of the current instruction – program counter)
4. Push the **stack frame** of the ISR on the stack
5. **Execute** the ISR (inform interrupt source that interrupt is being handled: *Interrupt Acknowledge* – INTA)
6. **Restore** the content of the processor register and pop the stack frame of the ISR from the stack

Interrupts – Multiple Sources

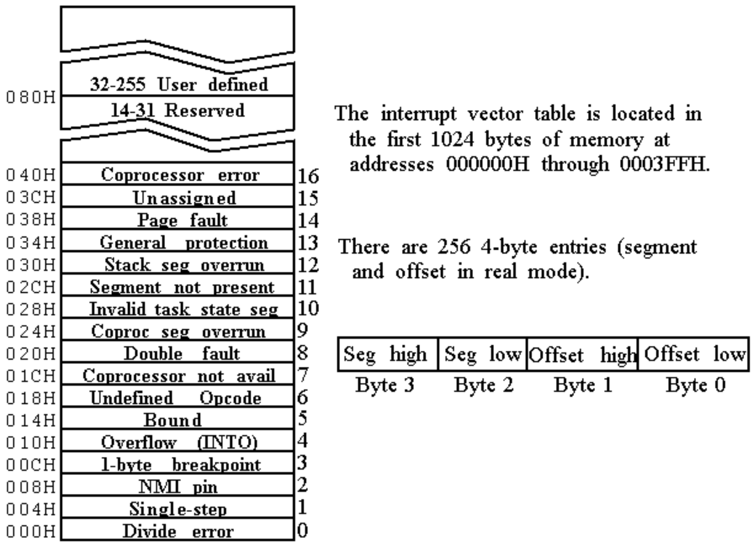
- The 6502 only has a single IRQ-Input and can store only a single ISR.
- Multiple sources (e.g. UART, Timer) share this input and are „ORed“ and therefore treated equally.
- No automatic hardware priorization
- ISR has to check the source of the interrupt (implicit priorization by order of checks)
- Hardware priorization is possible by daisy-chaining the interrupts. The unit closes to the CPU has the highest priority and disables interrupts of other sources until it gets an INTACK



Daisy-Chaining (Z80-Processor)
CTC: Counter-Timer-Circuit
PIO: Parallel In-/Out
DART: Dual Asynchronous Receiver Transmitter

Interrupts – Vectorized Interrupt Controller (VIC)

- In case of many interrupts each peripheral unit should have a dedicated ISR
- This is possible through Vectorized Interrupt Controllers (VIC)
- When the interrupt is generated, the processor receives an identifier (e.g. an 8 bit value) which is mapped to an entry in the interrupt vector table
- Each entry contains an address for the corresponding ISR
- Exceptions and interrupts use the same scheme
- Usually the interrupt controller can also handle priorization



080H	32-255 User defined	
	14-31 Reserved	
040H	Coprocessor error	16
03CH	Unassigned	15
038H	Page fault	14
034H	General protection	13
030H	Stack seg overrun	12
02CH	Segment not present	11
028H	Invalid task state seg	10
024H	Coproc seg overrun	9
020H	Double fault	8
01CH	Coprocessor not avail	7
018H	Undefined Opcode	6
014H	Bound	5
010H	Overflow (INTO)	4
00CH	1-byte breakpoint	3
008H	NMI pin	2
004H	Single-step	1
000H	Divide error	0

The interrupt vector table is located in the first 1024 bytes of memory at addresses 000000H through 0003FFH.

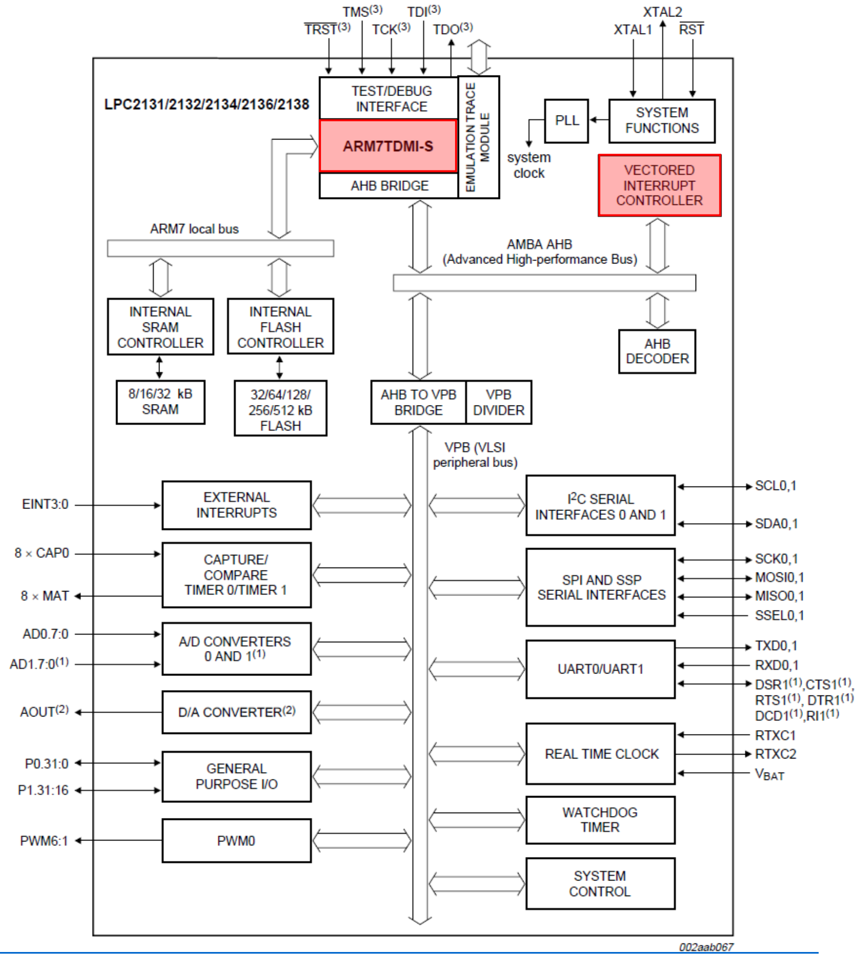
There are 256 4-byte entries (segment and offset in real mode).

Seg high	Seg low	Offset high	Offset low
Byte 3	Byte 2	Byte 1	Byte 0

- Interrupt-Vector-Table of an 8086
- 32 bit address per entry
- 0-31 internal exceptions
- Hardware interrupts can be configured to use any of the other vector entries

Interrupts – Microcontroller Example

- The ARM LPC 2138 by NXP is a 32-bit microcontroller with a real-time clock (RTC), UART, I²C bus, USB, ADC, DAC, etc.
- All peripheral units can trigger interrupts on the CPU
- All Interrupts can be prioritized
- Vectored Interrupt Controller with a table of 16



Interrupts – Example for LPC2138

- Example:
  Periodic interrupts from Timer0
  count interrupts through variable num_calls
- **1st step:** Defining the ISR

```
int volatile num_calls = 0;


void IRQ_Timer0(void)__attribute__((naked)); // attribute naked: see next slide
{
  ISR_ENTRY();          // Macro for entry sequence (store processor context)
  num_calls++;          // Count number of calls
  T0IR = 0x01;          // Clear Interrupt Flag
  VICVectAddr = 0x00;   // Update priority hardware
  ISR_EXIT();           // Macro for exit sequence (reload processor context)
}
```

Interrupts – Example for LPC2138

- `void IRQ_Timer0(void)__attribute__((naked));`

> *Gcc-Manual (http://gcc.gnu.org/onlinedocs/gcc-4.4.2/gcc.pdf) Seite 289:*
>
> *((naked))*
>
> *Use this attribute on the ARM, AVR, IP2K and SPU ports to indicate that the specified function does not need prologue/epilogue sequences generated by the compiler. It is up to the programmer to provide these sequences. The only statements that can be safely included in naked functions are asm statements that do not have operands. All other statements, including declarations of local variables, if statements, and so forth, should be avoided. Naked functions should be used to implement the body of an assembly function, while allowing the compiler to construct the requisite function declaration for the assembler.*

- `__attribute__((interrupt))`
  Instead of using the `naked` attribute and entry and exit macros, the `interrupt` attribute is available for some target platforms

Interrupts – Example for LPC2138

- **2nd step:** Programming the VIC

```
void sysInit(void)
{
  VICIntEnClear = 0xFFFFFFFF;          // clear all interrupts
  VICIntSelect = 0x00000000;           // clear all FIQ selections
  VICDefVectAddr = (uint32_t)reset;    // point unvectored IRQs to reset()
  VICVectAddr2 = (uint32_t)IRQ_Timer0; // IRQ-Timer0 (Address of ISR); Prio=2
  VICVectCntl2 = 0x20 | 0x04;          // For IRQ slot 2 set „enable slot“ Bit 5
                                       // AND activate for Timer0 interrupts
  VICIntEnable = 1 << 0x04;            // Enable Timer0 Interrupts
}
```

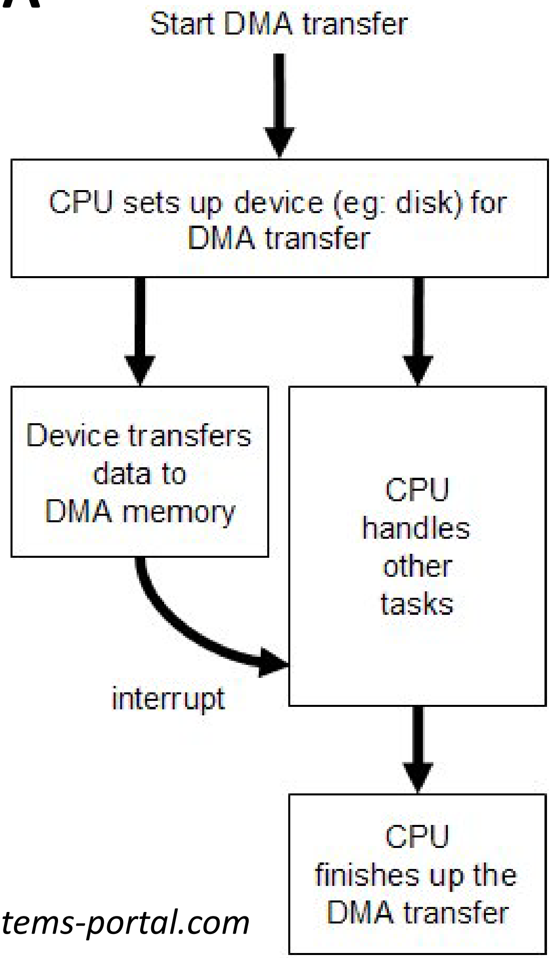
Interrupts – Example for LPC2138

- **3rd step:** Programming Timer0

```
void timerInit(void)
{
  T0TC = 0;          // Set timer value (to be incremented by clock)
  T0PR = 0;          // Set Prescaler (Scales clock frequency down from CPU clock)
  T0MR0 = 240000;    // Match-Register; Generate interrupt when timer reaches 240000
  T0MCR = 0x03;      // Generate interrupt on match and reset to zero
  T0TCR = 0x01;      // Timer control register: enable counting
  T0IR = 0x01;       // Enable Timer0 interrupts
}
```

Direct Memory Access – DMA

- Direct Memory Access (DMA)
- Instead of using the CPU to transfer data from a peripheral device into RAM, a coprocessor can be used in parallel for this task.
- CPU initiates the process and is notified (via an interrupt) when operation is done
- Leaves CPU free to do other tasks
- Important on microcontroller for fetching ADC values


Start DMA transfer
CPU sets up device (eg: disk) for DMA transfer
Device transfers data to DMA memory
CPU handles other tasks
interrupt
CPU finishes up the DMA transfer

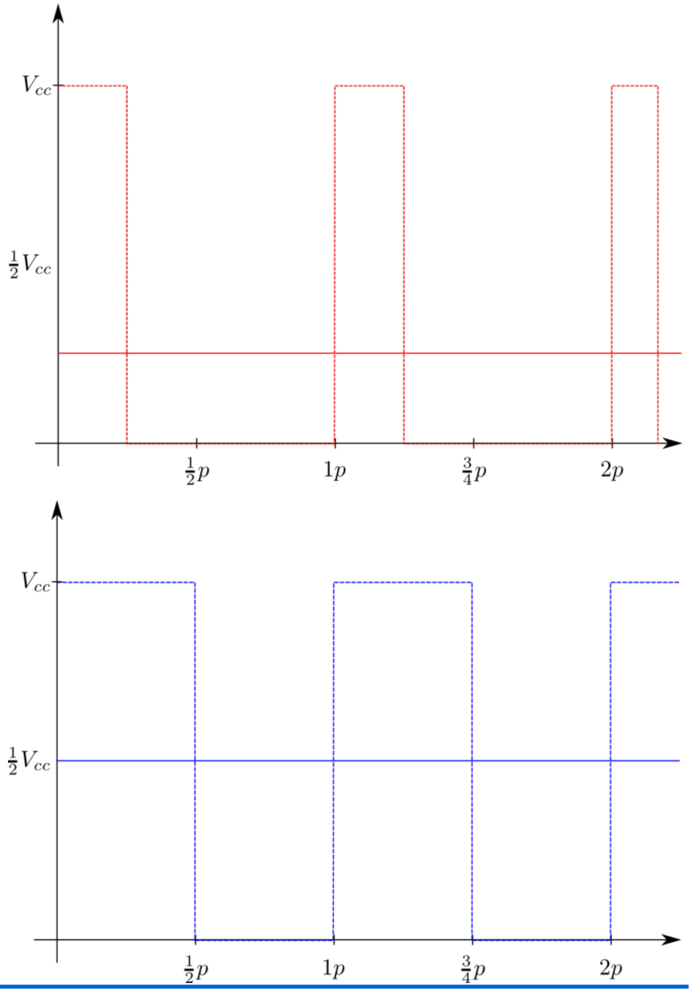
www.embedded-systems-portal.com

Interfacing to Sensors & Actuators

- PWM
- GPIO
- DAC
- ADC

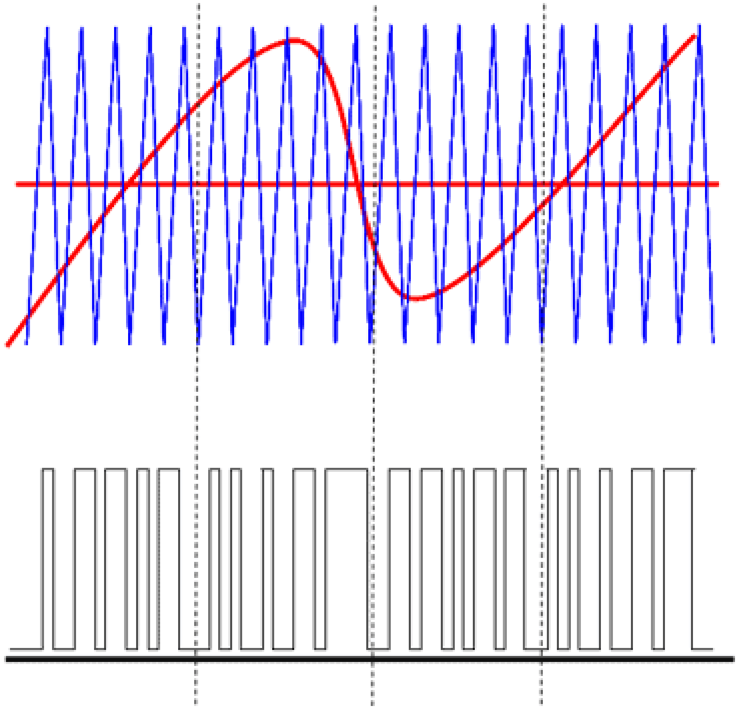
Interfacing Sensors & Actuators – PWM

- Pulse Width Modulation (PWM)
- Deliver variable amounts of power to external hardware
(e.g. LED brightness, motor driver)
- Duty cycle: proportion of cycle time where voltage is high.
- Programming through PWM interface of Timer



Interfacing Sensors & Actuators – PWM

- PWM with high frequencies (10-50kHz) to approximate signals
- Timer is configured to reload with fixed frequency (PWM frequency)
- Output Compare register is set to a value that is a percentage of the timer reload value (duty cycle)
- Output is set low when counter reaches Output Compare value



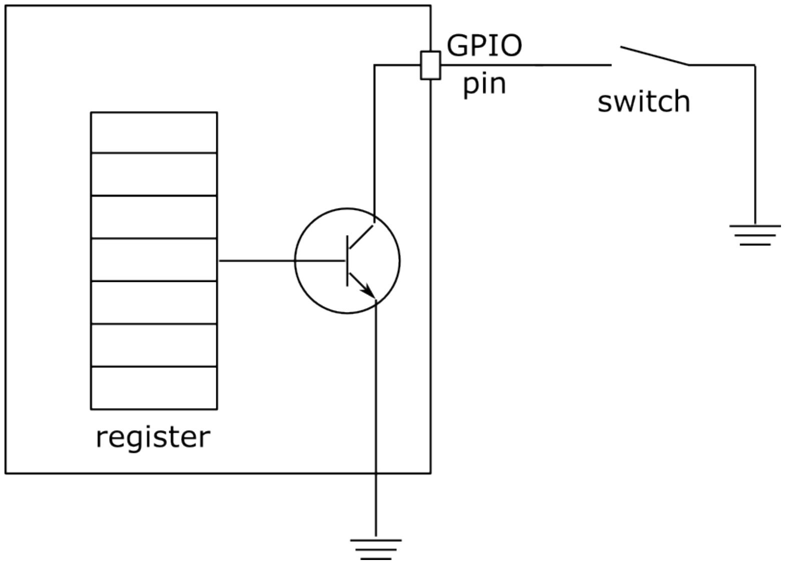
http://www.fairaudio.de

Interfacing Sensors & Actuators – Example PWM STM32F

```
TIM_OCInitTypeDef OCInit;

void pwmInit(void){
  TIM_TimeBaseInitTypeDef TimInit;
  TimInit.TIM_Period = 1439;                    // PWM frequency : 72MHz / 1440 = 50KHz
  TimInit.TIM_Prescaler = 0;
  TimInit.TIM_ClockDivision = 0;
  TimInit.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseInit(TIM2, & TimInit);
  TIM_ARRPreloadConfig(TIM2, ENABLE);
  TIM_Cmd(TIM2, ENABLE);                        // TIM2 enable counter
  OCInit.TIM_OCMode = TIM_OCMode_PWM1;          // Configure Output Compare to PWM mode
  OCInit.TIM_Pulse = 0;                         // initial duty cycle is 0
  TIM_OC3PreloadConfig(TIM2, TIM_OCPreload_Enable);
}

void setDutyCycle(uint32 dutyCycle){
  OCInit.TIM_Pulse = dutyCycle * 1440 / 100;
  TIM_OC3Init(TIM2, &OCInit);
}
```

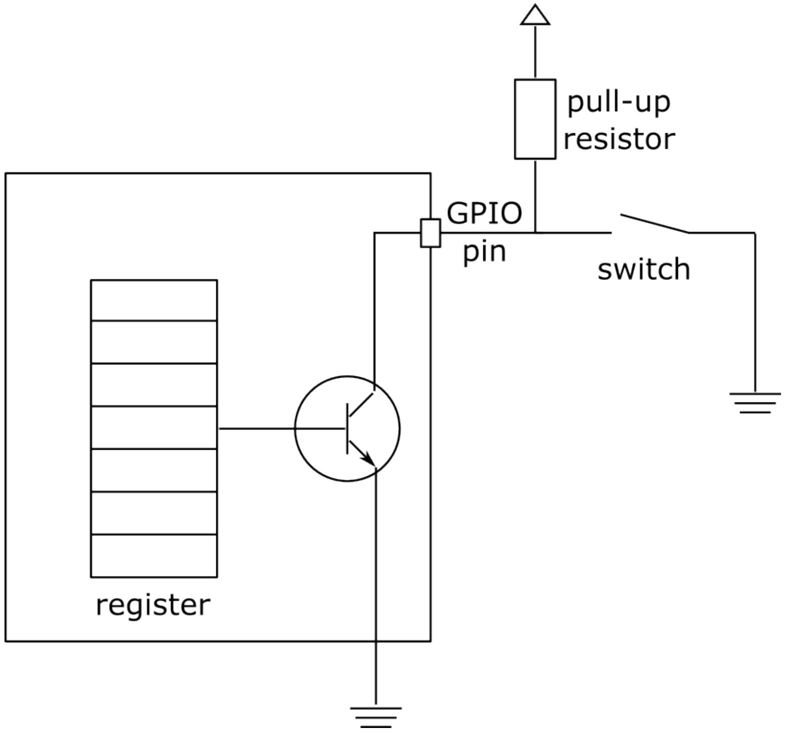
Interfacing Sensors & Actuators – GPIO

- General Purpose IO (GPIO) can be configured as digital input or output
- Digital output:
  Voltage of a pin can be set to high (VCC) or low (0V) by setting a bit in a register
- Digital input:
  State of pin (high or low) can be read by reading a bit in a register
  - What state does the pin have when switch is open?



Interfacing Sensors & Actuators – GPIO

- Pull-up resistor is needed for switch-open-state
- Most microcontrollers offer configurable pull-up and/or pull-down resistors



Interfacing Sensors & Actuators – DAC

- Digital to Analog Converter (DAC)
- Output voltage as a function of digital word *D*, scaling factor *K* and reference voltage V_{CC}

$$V_{out} = K\, V_{CC}\, D$$

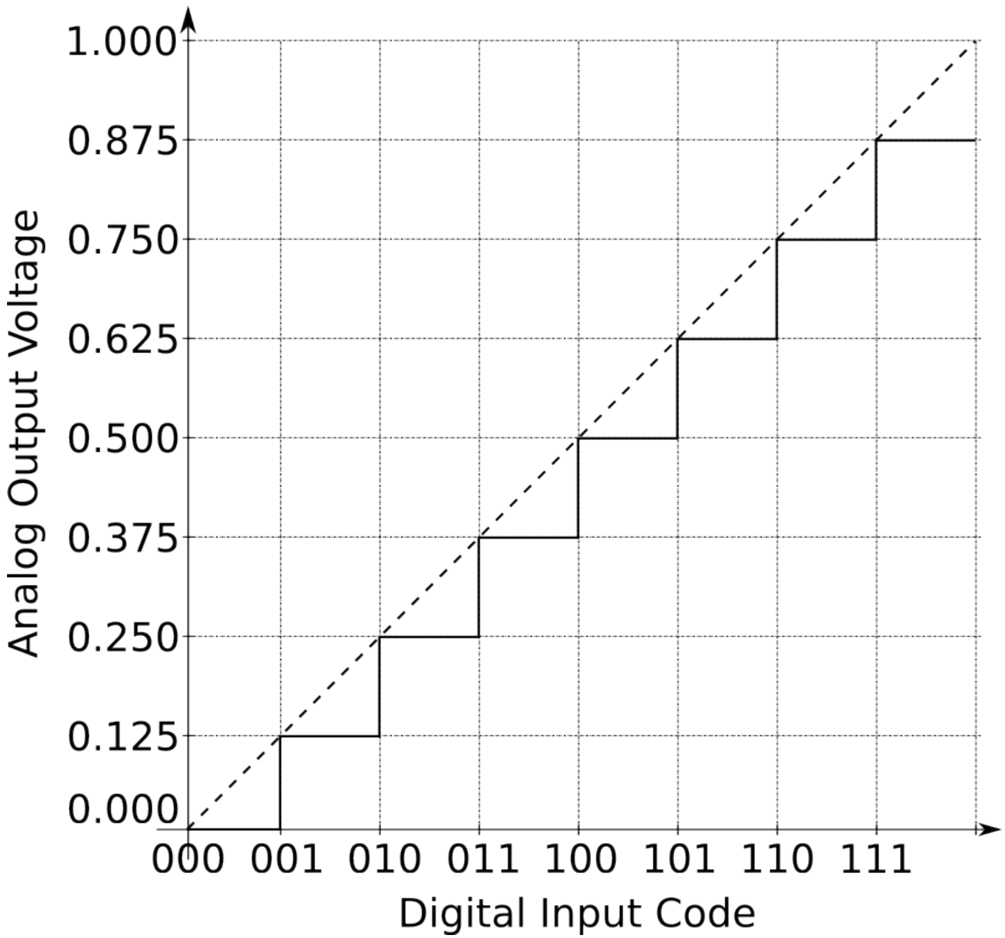
- N-bit resolution, with Least Significant Bit (LSB)

$$LSB = \frac{V_{REF}}{2^n}$$

- Full scale range (FSR) smaller than V_{CC} because of finite resolution

$$FSR = V_{CC} - LSB$$

Interfacing Sensors & Actuators – DAC



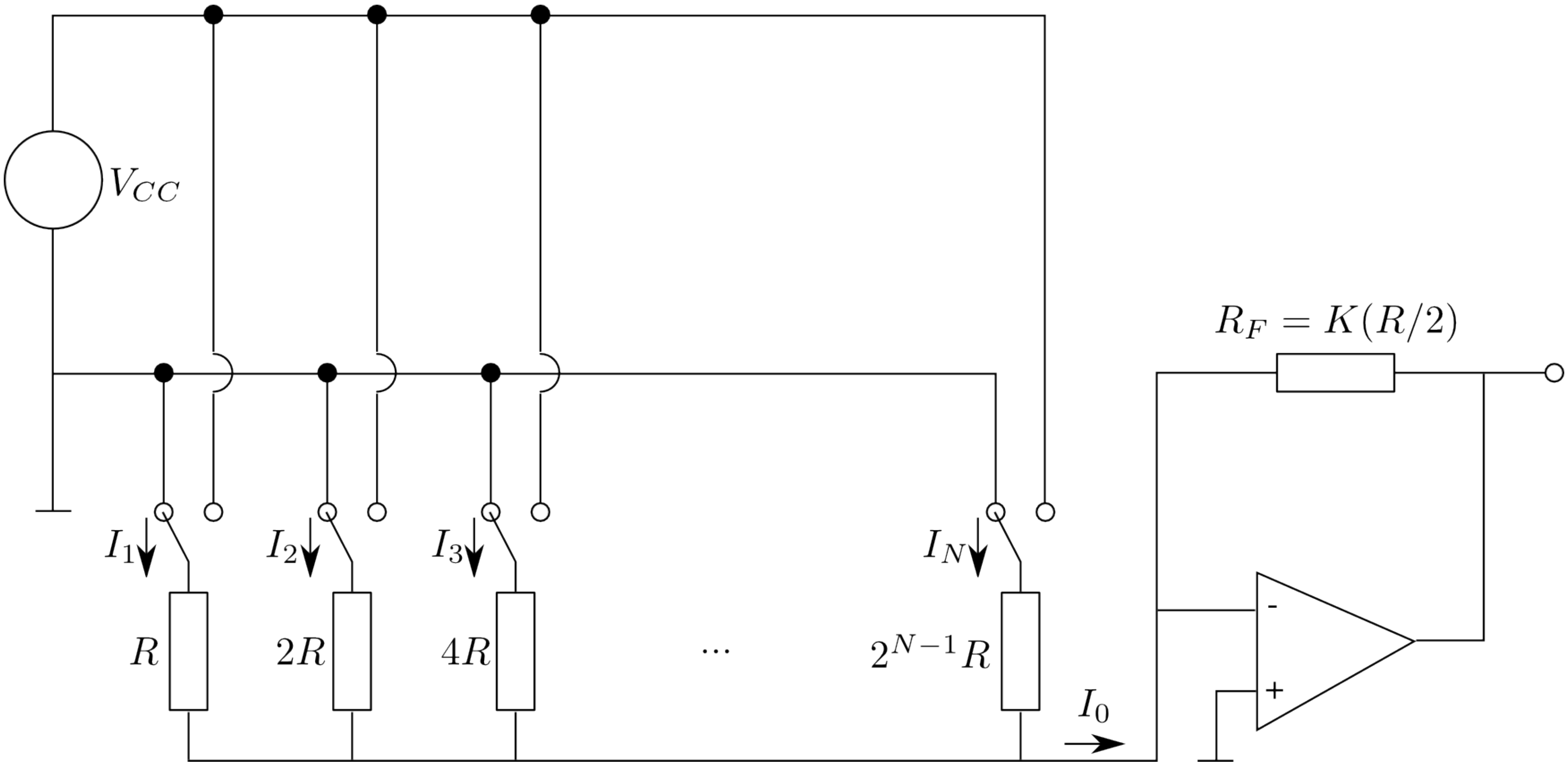
- Example of a 3-bit DAC with normalized output voltage

DAC – Different Implementations

- Parallel
  Structure with n parallel resistors and switches
  - Fast
  - Expensive for larger number of bits

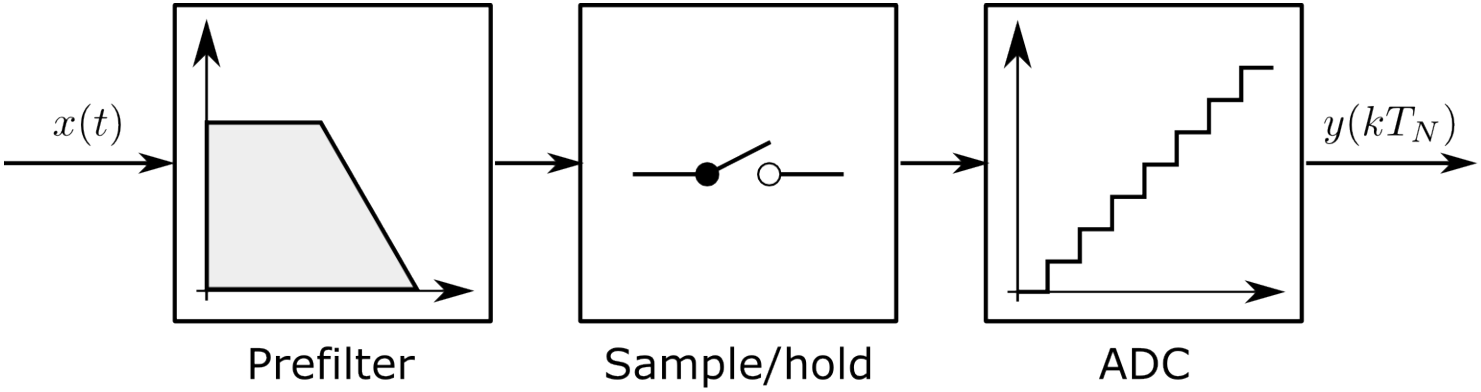
- Serial
  Single capacitor and serial approximation
  - Slow

DAC – Current Scaling



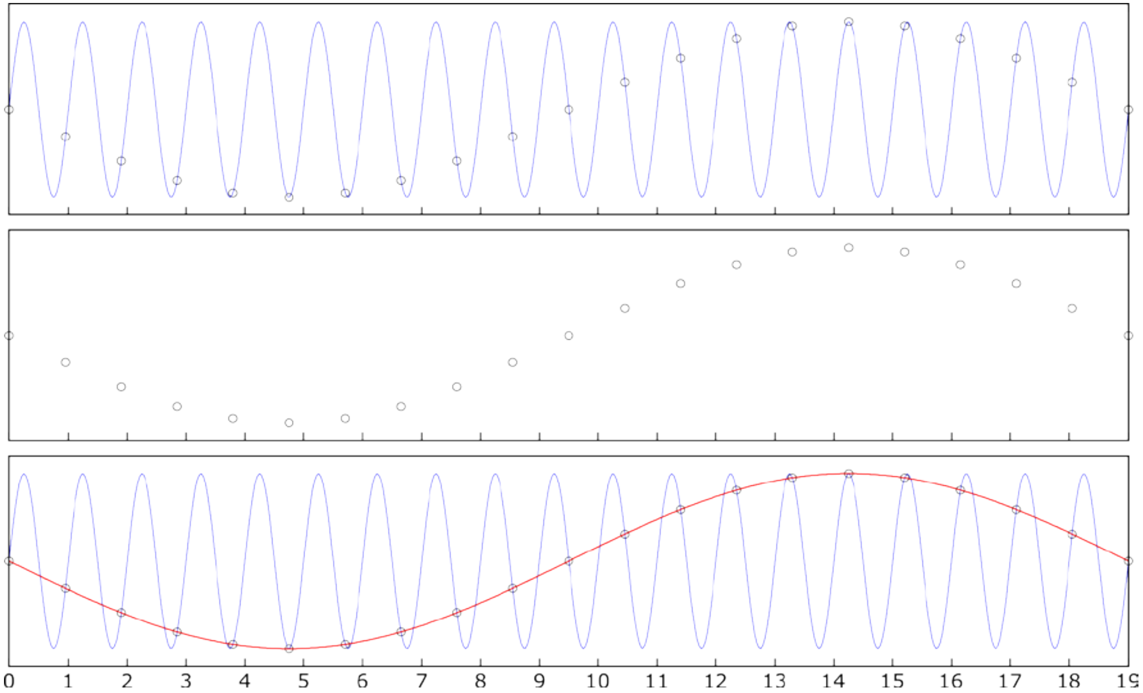
- Binary-weighted resistor DAC implementation

Interfacing Sensors & Actuators – ADC



- Analog to digital converter (ADC)
- Sample/hold circuit maintains input during *conversion time*
- Prefilter: *Antialiasing-filter* necessary
  - Avoids the aliasing of higher frequency signals
    Often implemented by the band-limiting charateristics of ADC itself

Interfacing Sensors & Actuators – Aliasing

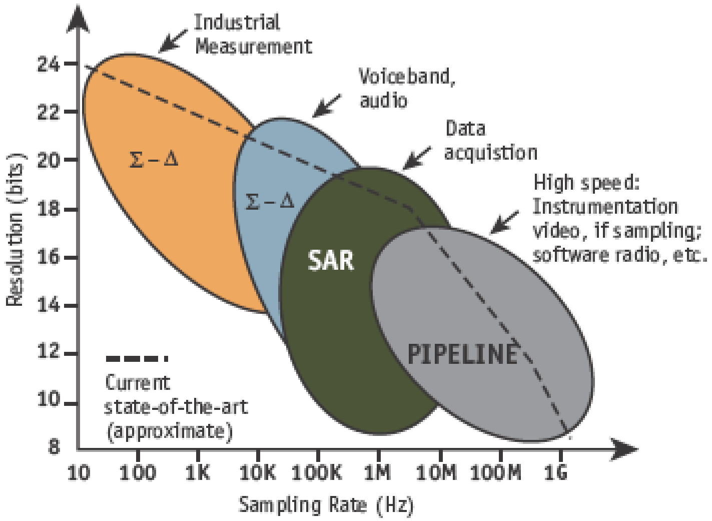


- Choice of the sampling frequency effects the capacity to extract information from a signal
- High frequencies are mirrored (aliased) onto lower frequencies

$$f_N = \frac{1}{2} f_S$$

- Antialiasing filters (low-pass filters) cut off the higher frequencies before sampling to e.g. prevent higher frequency noise from disturbing low frequency signals.

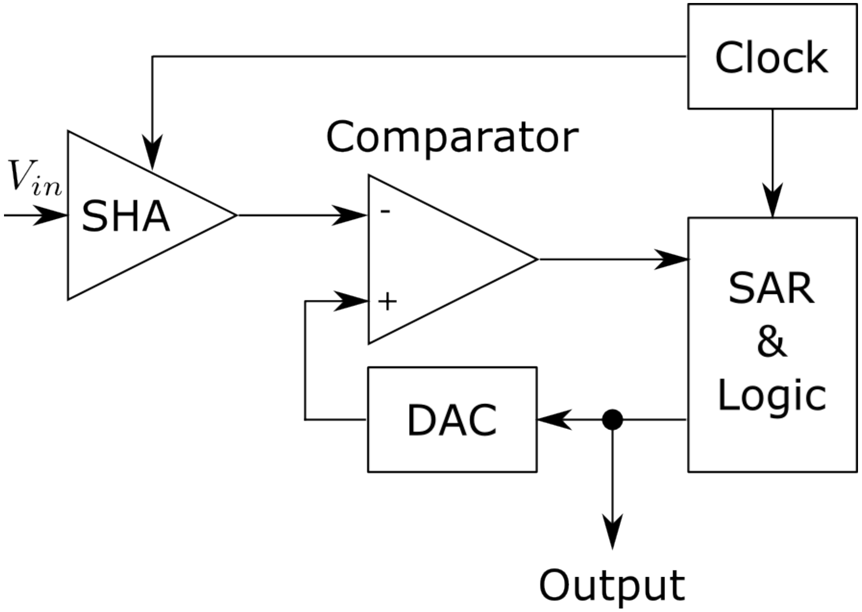
Interfacing Sensors & Actuators – ADC



- Successive approximation register (SAR)
  - Relatively slow & cheap
- Sigma-delta
  - Extension of SAR for higher resolution
- Pipeline
  - Half parallel for high sampling rates
- Flash
  - Fully parallel, fast & expensive

http://www.techdesignforums.com/eda/technique/ which-adc-architecture-is-right-for-your-application/

Interfacing Sensors & Actuators – ADC



- Example of a successive-approximation ADC architecture

Sensors & Actuators

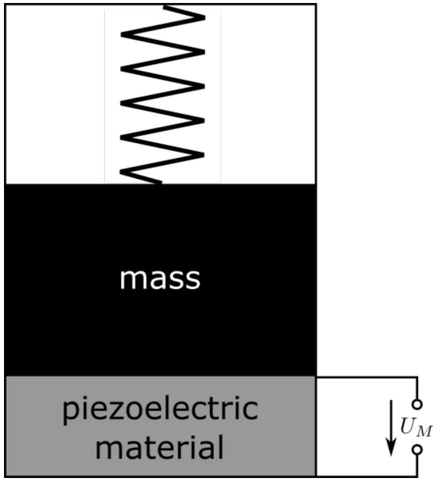
- Accelerometer
- Gyroscope
- Strain gauge
- Encoder
- Motor driver

Sensors & Actuators - Accelerometer

- Measures the acceleration along an axis

$$F = m \cdot a$$

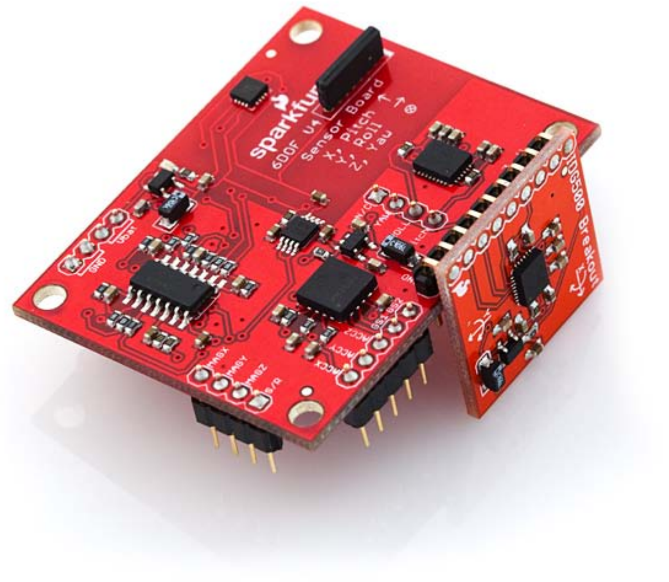
- Piezo or capacity based (often realized as MEMS)
- Usage:
  - Vibration sensing
  - Tablet computers / smart phones
  - Inertial navigation (missile, aircraft)



mcustore.com

Sensors – Gyroscope

- Measures the angular velocity around an axis
- Often packaged together with an accelerometer
  (Inertial measurement unit – IMU)
- Available as MEMS
- Often fused with other (absolute) sensors due to integration errors
- Usage:
  - Inertial Navigation



Sensors & Actuators – Strain gauge

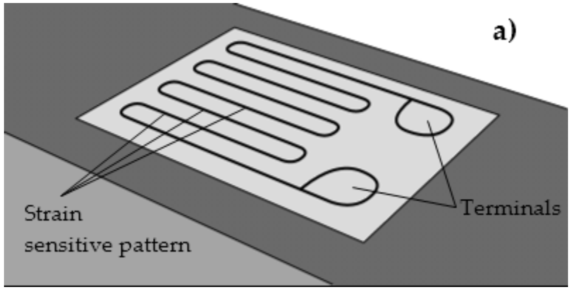
- Measures the strain (stretch) of a material (proportional to stress in elastic materials)

$$R = \rho \frac{l}{A}$$

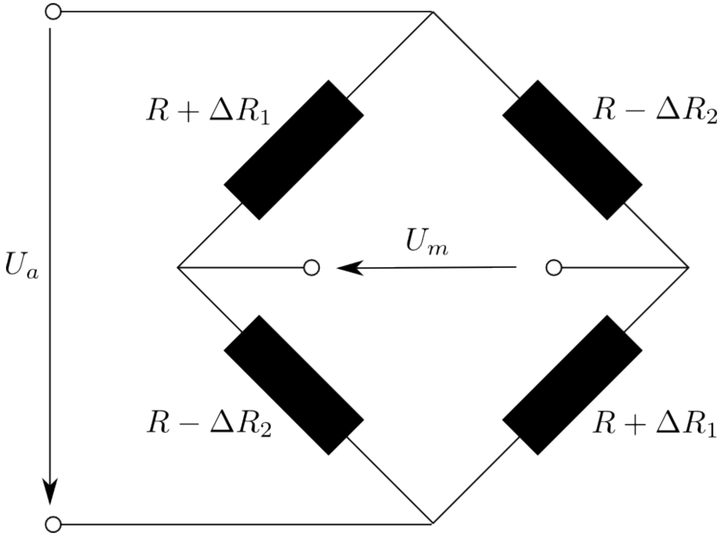
  l: length, A: cross-section,
  ρ: specific resistance

- Four strain gauges can be used in a Wheatstone Bridge, which is balanced (U_m=0) when there is no excitation (Compensates for temperature dependency)

- Usage:
  - Measure extensions (strain)
  - Measure force (stress)

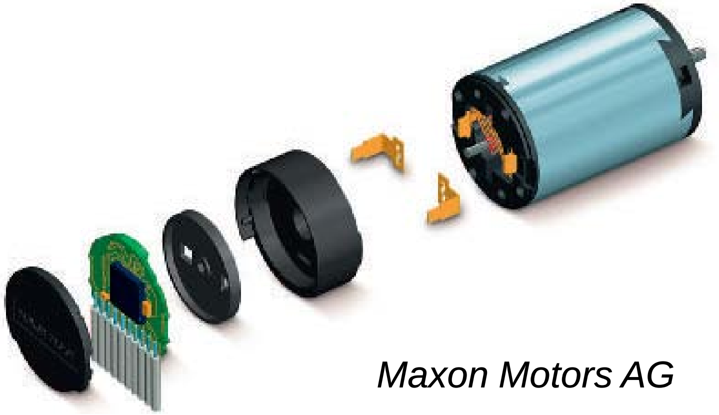


Wikimedia Commons



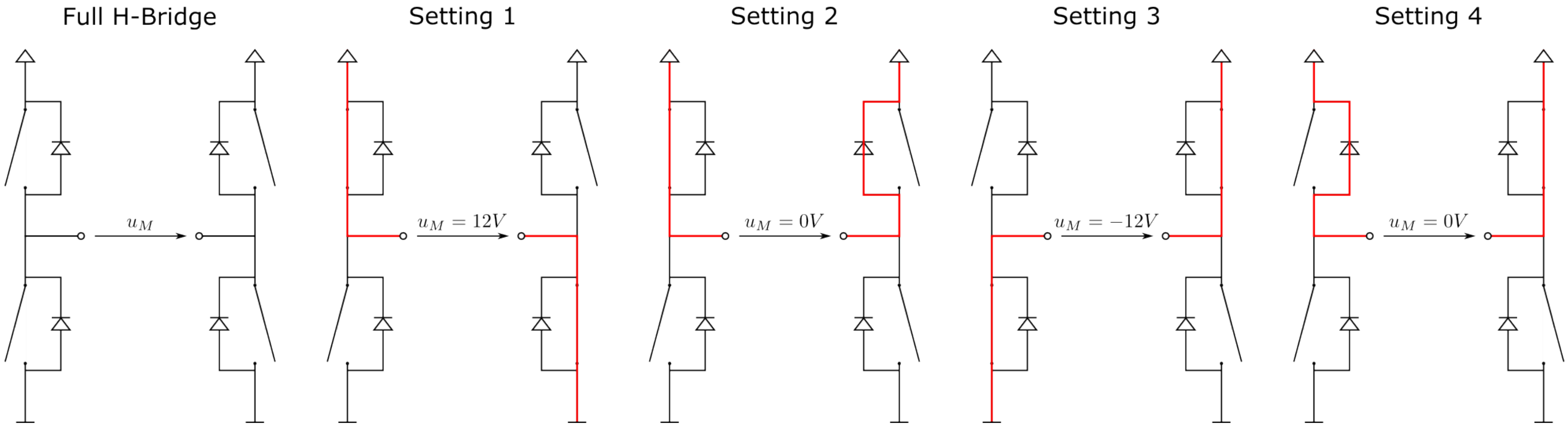
Sensors & Actuators – Encoder

- Motor position can be measured by an encoder
- Relative position is obtained by „counting ticks“
- Digital magneto-resistant (MR) or optical
- Some microcontrollers feature timer-based encoder interfaces
- Usage:
  - Motor control
  - Robotic joint control



Maxon Motors AG

Sensors & Actuators – H-Bridge



- H-Bridges are used to drive large loads, e.g.
  - Magnetic coils
  - Motors
- Free-wheeling diodes are needed for inductive loads
- Fast switches are driven by a PWM signal

Literature

- Chapter 7-9 in Lee and Seshia, Introduction to Embedded Systems – A Cyber-Physical Systems Approach, 1st Edition, 2011, LeeSheshia.org
- Hennessy and Patterson, Computer Architecture (A quantitative Approach), 5th Edition, 2012, Morgan Kaufmann