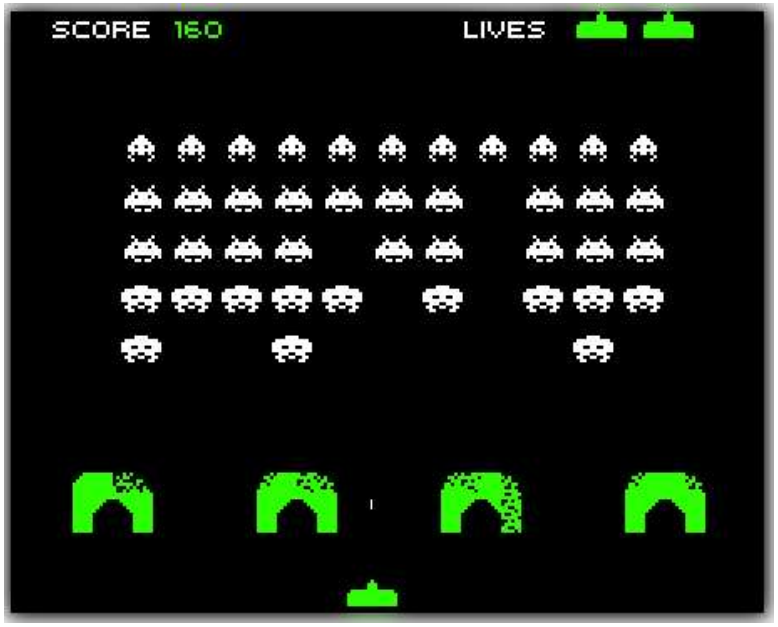
INF1
Bit-Operationen

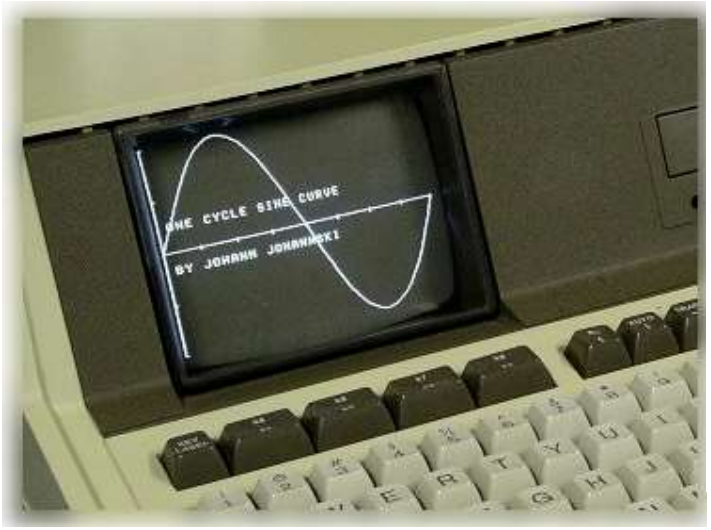
School of Engineering

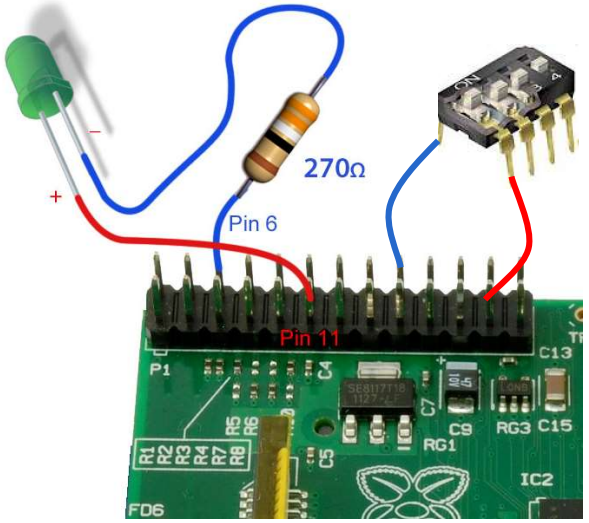


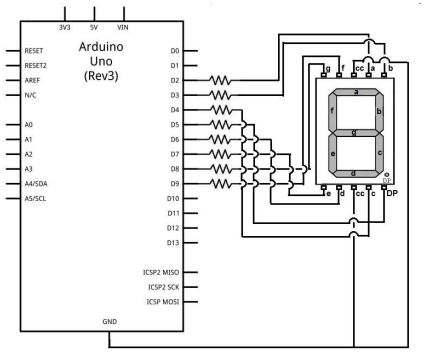
- Bit-Operationen
- Übungen
- Bits setzen, löschen, abfragen
- Beispiel: IO

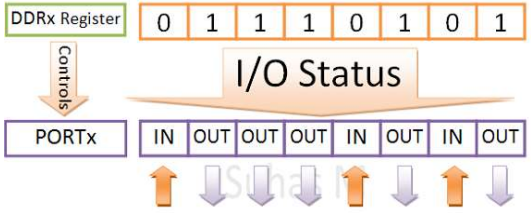
Displays and I/O

- Binary Display & Input/Output









Operationen für Bitmanipulation

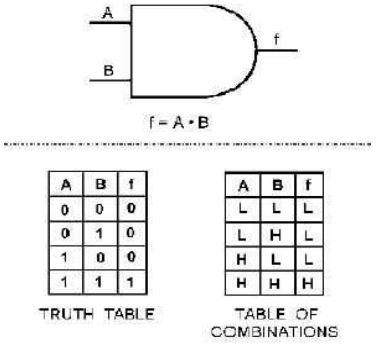
School of Engineering

Bit - Operatoren

- Die Bitoperatoren verknüpfen die einzelnen Bits der Operanden.
- **Beispiel**: AND-Funktion Operator: &

```
unsigned char resultat;
unsigned char a      = 0xbe;      //   1 0 1 1   1 1 1 0
unsigned char b      = 0x18;      //   0 0 0 1   1 0 0 0
resultat = a & b;                 //   0 0 0 1   1 0 0 0
```

AND bitweise



- Die Operationen sind nur auf ganzzahlen Typen definiert.
- Für ein Byte nimmt man meist unsigned char

Verwendung

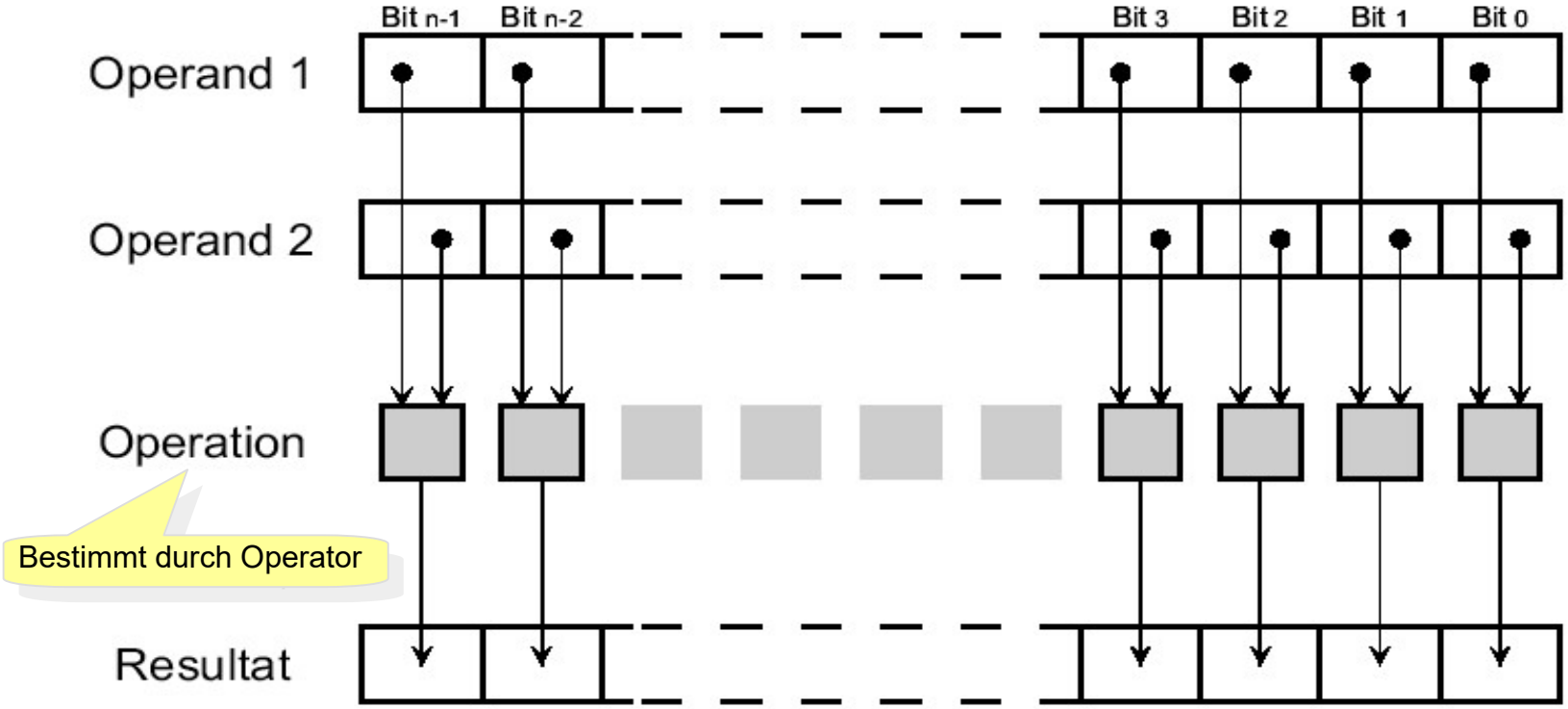
- Die Kommunikation mit der Hardware erfolgt über Register auf den Peripherie-Bausteinen
- Beispiel
  - Produktionsanlage, in der verschiedene Ventile angesteuert werden
  - Parallel-I/O Baustein, in dessen Register jedes Bit einer I/O Leitung entspricht, die mit einem Taster oder (Digital-) Sensorverbunden ist (z.B. Lichtschranke)
  - Um einen Schalter ein- oder auszuschalten müssen einzelne Bits gesetzt oder zurückgesetzt werden

Bit-Operationen

Bit-Operationen

- Mit Bit-Operatoren kann man Bits setzen, löschen, abfragen und andere Operationen darauf durchführen
- Bei den Operatoren &, | und ^ werden jeweils die in der binären Schreibweise einander entsprechenden Bits der Operanden miteinander verknüpft
- Diese Operationen können nur auf ganze Zahlen angewendet werden
- Bit-Operationen braucht es vor allem für Hardware-Manipulationen und damit in der Digitaltechnik

Bit-Operatoren: Übersicht



… Bit-Operationen: Operatoren

Operator	Beschreibung	Beispiel	(a = 6 = 0110 b = 3 = 0011)
&	Bitweise AND Verknüpfung	a & b	(6 & 3 ergibt 2=0010)
|	Bitweise OR Verknüpfung	a | b	(6 | 3 ergibt 7 =0111)
^	Bitweise EXOR Verknüpfung	a ^ b	(6 ^ 3 ergibt 5=0101)
~	Bitweise Negation (Einerkomplement)	~a	(~6 ergibt 9 =1001)

<< Bitweises nach links schieben, a << 3 (6 << 3 ergibt 48)

(a << b entspricht a * 2^b)

es werden Nullen nachgeschoben,

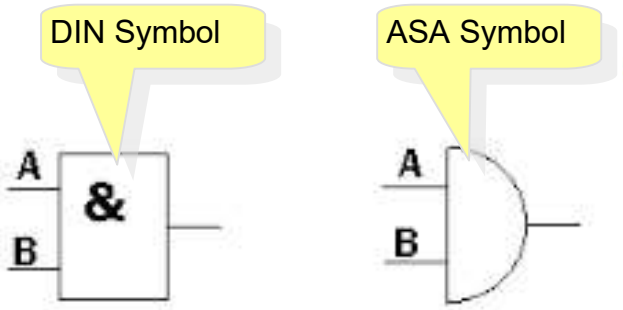
a selber wird nicht verändert!

>> Bitweises nach rechts schieben a = 9; (= 1001_2)

(a >> b entspricht a / 2^b)

a >> 3 (9 >> 3 ergibt 1).

Der &-Operator (and)



Bit1	Bit2	Bit1 & Bit2
0	0	0
0	1	0
1	0	0
1	1	1

```
int c1, c2;
c1 = 0x45;
c2 = 0x71;
printf("Result of %x & %x = %x\n", c1, c2, c1 & c2);
// Result of 45 & 71 = 41
```

	c1 = 0x45	binary 01000101
&	c2 = 0x71	binary 01110001
=	0x41	binary 01000001

Beispiel für &

- Definition einer Hilfsfunktion statt die direkte Bitmaske "inline" ist verständlicher

Binärkonstante

```
bool odd (int value) {
    return ((value & 0b00000001) == 1);
}
```

- Kann auch einfach so geschrieben werden (Wert != 0 ist true)

```
bool odd (int value) {
    return ((value & 0b00000001));
}
```

Unterschied zwischen && und &

```
int main (void) {
    int i1, i2; // two random integers
    i1 = 4;
    i2 = 2; // set values

    // Nice way of writing the conditional
    if ((i1 != 0) && (i2 != 0)) { printf("Both are not zero #1\n"); }

    // Shorthand way of doing the same thing
    // Correct C code, but bad style
    if (i1 && i2) { printf("Both are not zero #2\n"); }

    // Incorrect use of bitwise and resulting in an error
    if (i1 & i2) { printf("Both are not zero #3\n"); }

    return 0;
}
```

Der | Operator (or)

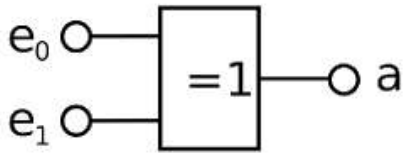
Bit1	Bit2	Bit1 | Bit2
0	0	0
0	1	1
1	0	1
1	1	1


```
int c1, c2;
c1 = 0x47;
c2 = 0x53;
printf("Result of %x | %x = %x\n", c1, c2, c1 | c2);
// Result of 47 | 53 = 57
```

i1=0x47	01000111
i2=0x53	01010011
| 57	01010111

Der ^ Operator (xor)

Bit1	Bit2	Bit1 ^ Bit2
0	0	0
0	1	1
1	0	1
1	1	0

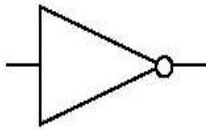



```
int c1, c2;
c1 = 0x47;
c2 = 0x53;
printf("Result of %x ^ %x = %x\n", c1, c2, c1 ^ c2);
// Result of 47 ^ 53 = 14
```

	i1=0x47	01000111
	i2=0x53	01010011
^	14	00010100

Der ~ Operator Einerkomplement

Bit	~Bit
0	1
1	0



c=	0x45	01000101
~c=	0xBA	10111010

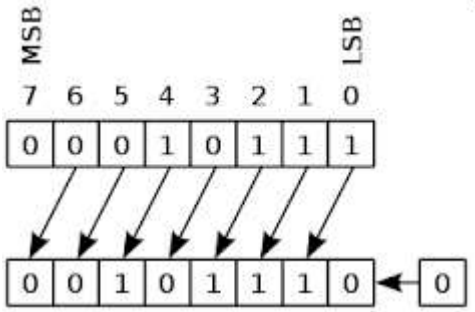
Der << Operator

<< Bitweises nach links schieben, **a << 1** (23 << 1 ergibt 46)

(a << b entspricht a * 2^b)

es werden Nullen nachgeschoben,

a selber wird nicht verändert!



c = a << 1

soll a selber verändert werden verwendet man den <<= Operator

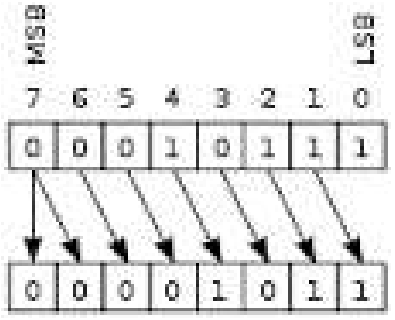
a <<= 1

Der >> Operator

- Bitweises nach rechts schieben a = 9; (= 1001_2)
  (a >> b entspricht a / 2^b)
  a >> 1 (23 >> 1 ergibt 11.

Achtung, bei signed Operanden wird üblicherweise das Vorzeichenbit dupliziert (Prozessor und Compilerabhängig), bei unsigned wird immer eine 0 nachgeschoben.

- soll a selber verändert werden verwende man den >>= Operator
  a >>= 1



Zusammengesetzte Operatoren

- Entsprechend den Rechenoperationen...

```
+=    -=    *=    /=    %=
```

- ... sind für ganzzahlige Werte ausserdem zusammengesetzte Bit-Operatoren verfügbar:

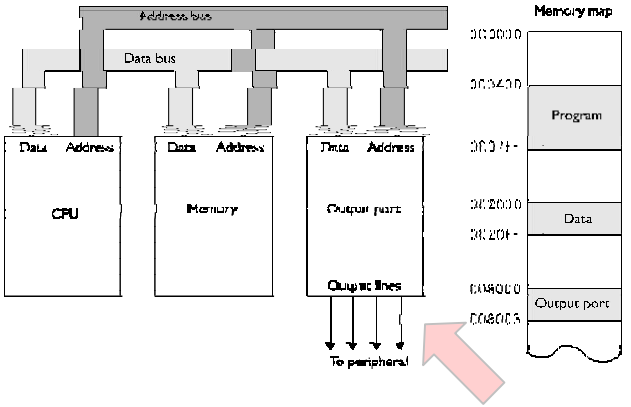
```
&=    |=    ^=   <<=    >>=
```

Operationen für Bitmanipulation: Anwendungen

School of Engineering

Wozu sind sie in der Programmierung mit C gut?

- Mit **Bit-Operatoren** kann man **Einzelbits in int-Variablen setzen, löschen, abfragen und andere Operationen darauf durchführen**.
  - Mit Hilfe von Pointer kann man solche Operationen auf beliebiger Speicher-Adresse ausführen
    → auch solche, wo durch die Adresse **ein IO-Register** angesprochen wird (**memory-mapped IO**)
  - Durch Setzen oder Löschen der Bits eines IO-Registers (IO-Port) steuert man dann direkt die IO-Peripherie (z.B. Motor einschalten, Sensor ON/FF abfragen)



- **Beispiel**: Bit Nr. 4 im Register an der Adresse C080C0 löschen (=0 setzen):

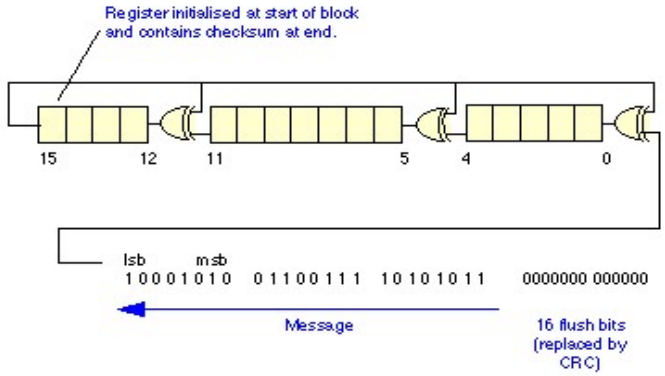
```
char *ptr = 0xC080C0;    // hex-Adresse des IO-Registers
char mask  = 0xef;       //  maske = 1110 1111
*ptr = * ptr & mask;     // &-Op. setzt alle Bits im IO-Register, die in der
                         // Variable mask mit 0 markiert sind, auf null
```

Pointer auf unsigned byte

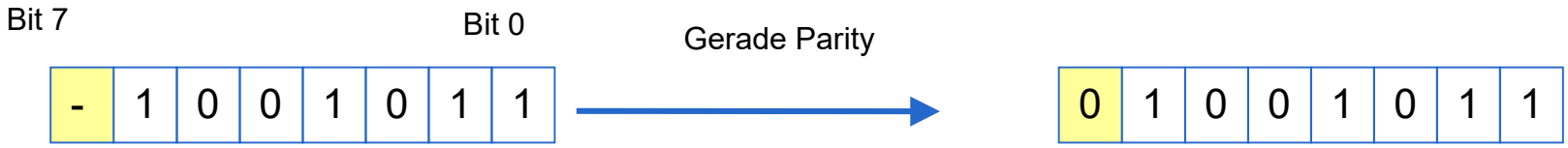
…Operationen für Bitmanipulation: Anwendungen

Andere Anwendung: Simulation der Hardware-Operationen in Software.

- Beispiel: **CRC-Prüfsumme-Berechnung** durch Simulation der Hardware:
- Benötigte Bit-Operationen:
  - **XOR** ^
  - **Shift** <<



- Einfacheres Beispiel: **Parity-Prüfbit** bei Datenübertragung:
  - Bit 7 in einem Byte so ergänzen, dass die Anzahl Einser gerade (oder ungerade) ist



Übungen

Übung Bit-Operatoren

- Welchen Wert hat r jeweils nach Ausführen dieser Ausdrücke?

```
unsigned char a = 5, b = 6, r;

r = a | b;
r = a ^ b;
r = a & b;
r = ~r;
r >>= 4;
```

Übung 2

```
void funktion(char c) {
    int i;
    for (i=7; i >= 0; i-=1) {
        if (( 1 << i & c) == 0) {
            printf("0");
        } else {
            printf("1");
        }
    }
    printf("\n");
}
```

Was macht diese Funktion?

Bits setzen, löschen, abfragen

Ein bestimmtes Bit in einer Variablen auf 1 setzen

Beispiele von **oft verwendeten Problemstellungen mit Bit-Operationen**:

- Das **Bit Nr. 2** mit einer Konstanten **markieren** (1 an der Bit-Stelle)

```
#define    BIT2   0x04    // Binär : 00000100
// alternativ: const unsigned char BIT2 = 0x04;
// ODER        const unsigned char BIT2 = 1 << 2;
```

- Mit OR-Operation das Bit auf 1 setzen

```
unsigned char c = 0;
c=c | BIT2; // auf 1 setzen mit OR-Operation
```

Ein bestimmtes Bit in einer Variablen auf 0 setzen

- Das **Bit Nr. 2 mit einer Konstanten markieren** (1 an der Bit-Stelle)

```
#define   BIT2   0x04     // Binär : 00000100
```

- Das positive Bit-Muster 00000100 negieren, um es in eine Löschmaske umzuwandeln `~BIT2 = ~(00000100) = 11111011` und anschliessend das Bit mit `AND-Operation` löschen:

```
unsigned char c = 0;
c = c & ~BIT2;// auf 0 setzen mit AND-Operation
```

Löschmaske für Bit2:
1111 1011

Ein bestimmtes Bit in einer Variablen invertieren

- Das Bit Nr. 2 mit einer Konstanten markieren (1 an der Bit-Stelle)

```
#define   BIT2   0x04     // Binär : 00000100
```

- Mir XOR-Operation das Bit invertieren:

```
unsigned char c = 0;
c = c ^ BIT2;
```

Ein bestimmtes Bit in einer Variablen abfragen

- Um ein Bit abzufragen, verwenden Sie den Operator **&** mit einer **Maske, welche das zu testende Bit selektiert**:
- Das **Bit Nr. 2 mit einer Konstanten markieren** (1 an der Bit-Stelle)

```
#define   maskeBit2    0x04     // Binär : 00000100
```

- **Das Bit testen**:

```
if ((flags & maskeBit2) != 0) { ..... }
```

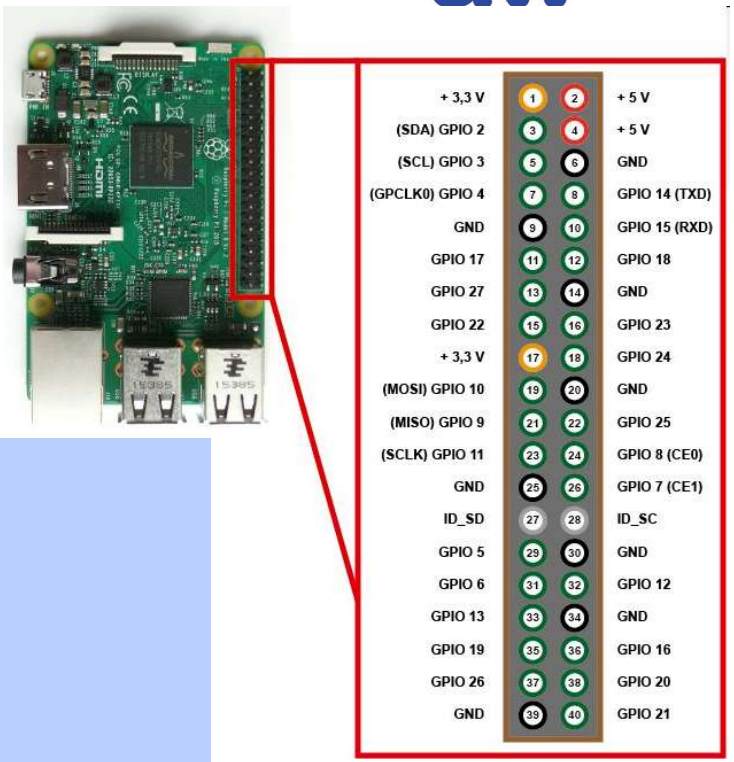
kann man sich sparen

Lässt nur Bits durch, die in der Maske mit 1 markiert sind

Beispiel I/O

GPIO Raspberry

- GPIO-Anschlussleiste
- Wahlweise
  - Als Eingang oder Ausgang programmieren
  - Reihe weiterer Funktionen
- Erweiterungen und elektronische Schaltungen .

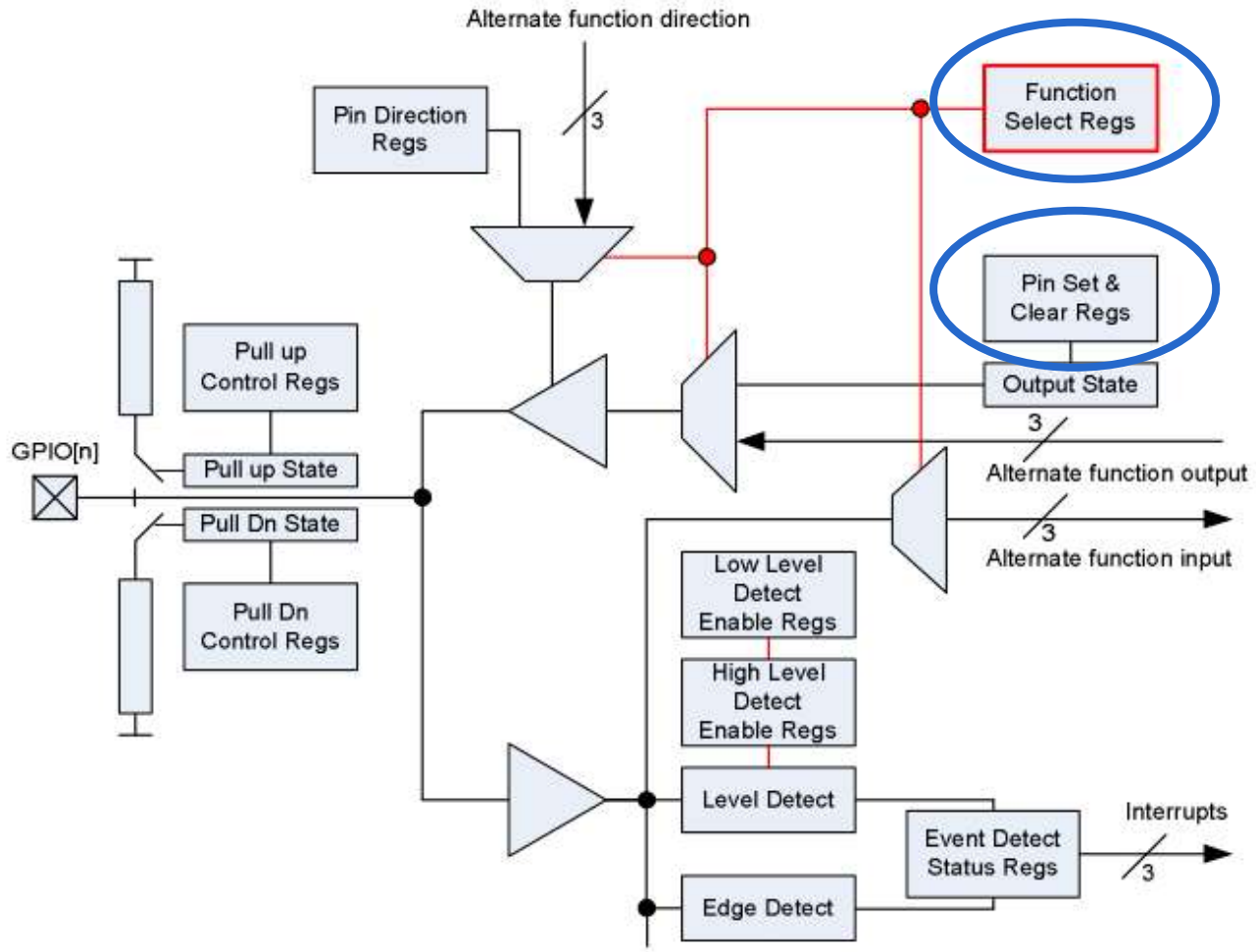



```
// sets   bits which are 1 ignores bits which are 0
#define GPIO_SET  *(gpio.addr + 7)
// clears bits which are 1 ignores bits which are 0
#define GPIO_CLR  *(gpio.addr + 10)
 for (rep=0; rep<10; rep++) {
     for (g=7; g<=11; g++) {
       GPIO_SET = 1<<g;
       sleep(1);
     }
     for (g=7; g<=11; g++) {
       GPIO_CLR = 1<<g;
       sleep(1);
     }
  }
```

https://elinux.org/RPi_GPIO_Code_Samples

Raspberry GPIO BCM2835

- General Purpose IO Baustein



Function Select Register

GPIO Function Select Registers (GPFSELn)

SYNOPSIS The function select registers are used to define the operation of the general-purpose I/O pins. Each of the 54 GPIO pins has at least two alternative functions as defined in section 16.2. The FSEL{n} field determines the functionality of the nth GPIO pin. All unused alternative function lines are tied to ground and will output a "0" if selected. All pins reset to normal GPIO input operation.

Bit(s)	Field Name	Description	Type	Reset
31-30	---	Reserved	R	0
29-27	FSEL9	FSEL9 - Function Select 9 000 = GPIO Pin 9 is an input 001 = GPIO Pin 9 is an output 100 = GPIO Pin 9 takes alternate function 0 101 = GPIO Pin 9 takes alternate function 1 110 = GPIO Pin 9 takes alternate function 2 111 = GPIO Pin 9 takes alternate function 3 011 = GPIO Pin 9 takes alternate function 4 010 = GPIO Pin 9 takes alternate function 5	R/W	0
26-24	FSEL8	FSEL8 - Function Select 8	R/W	0
23-21	FSEL7	FSEL7 - Function Select 7	R/W	0
20-18	FSEL6	FSEL6 - Function Select 6	R/W	0
17-15	FSEL5	FSEL5 - Function Select 5	R/W	0

Set Output Pins to Low or High

- Pins can be set or cleared

GPIO Pin Output Set Registers (GPSETn)

SYNOPSIS The output set registers are used to set a GPIO pin. The SET{n} field defines the respective GPIO pin to set, writing a "0" to the field has no effect. If the GPIO pin is being used as in input (by default) then the value in the SET{n} field is ignored. However, if the pin is subsequently defined as an output then the bit will be set according to the last set/clear operation. Separating the set and clear functions removes the need for read-modify-write operations

Bit(s)	Field Name	Description	Type	Reset
31-0	SETn (n=0..31)	0 = No effect 1 = Set GPIO pin *n*	R/W	0

Table 6-8 – GPIO Output Set Register 0

GPIO Pin Output Clear Registers (GPCLRn)

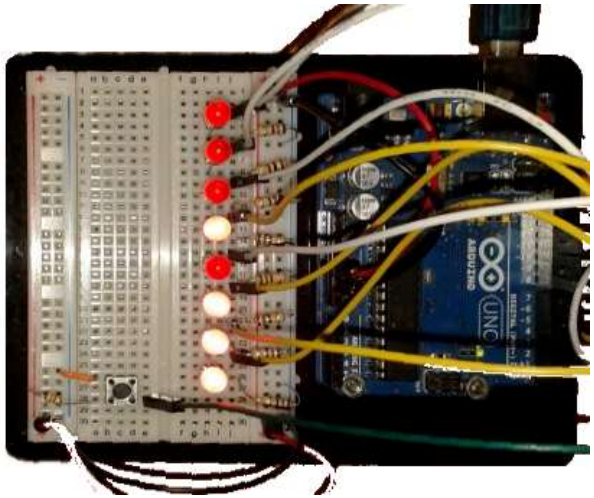
SYNOPSIS The output clear registers) are used to clear a GPIO pin. The CLR{n} field defines the respective GPIO pin to clear, writing a "0" to the field has no effect. If the GPIO pin is being used as in input (by default) then the value in the CLR{n} field is ignored. However, if the pin is subsequently defined as an output then the bit will be set according to the last set/clear operation. Separating the set and clear functions removes the need for read-modify-write operations.

Bit(s)	Field Name	Description	Type	Reset
31-0	CLRn (n=0..31)	0 = No effect 1 = Clear GPIO pin *n*	R/W	0

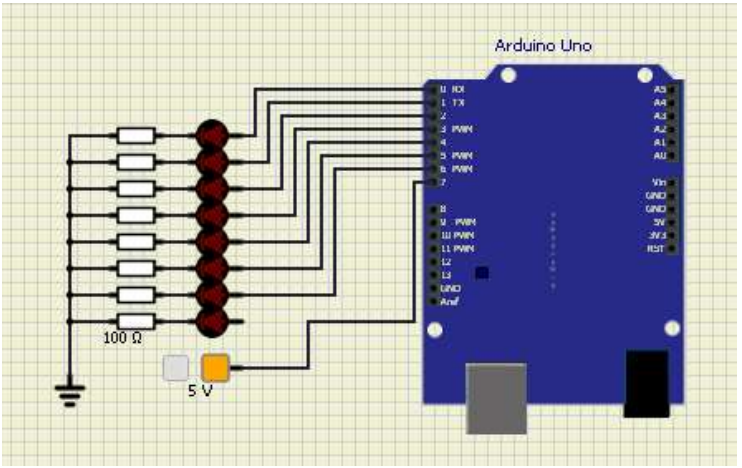
Table 6-10 – GPIO Output Clear Register 0

Arduino Plattform

- Mikrocontroller und analogen und digitalen Ein- und Ausgängen
- Programmierung in C



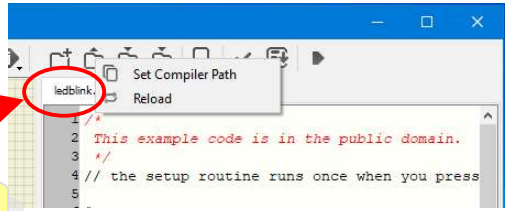
```
void setup() {
  DDRD = 0b01111111;
}

void loop() {
   if (PIND & 0b10000000) {
         PORTD = 0b01111111;
   }
   delay(1000);
   PORTD = 0b00000000;
   delay(1000);
}
```

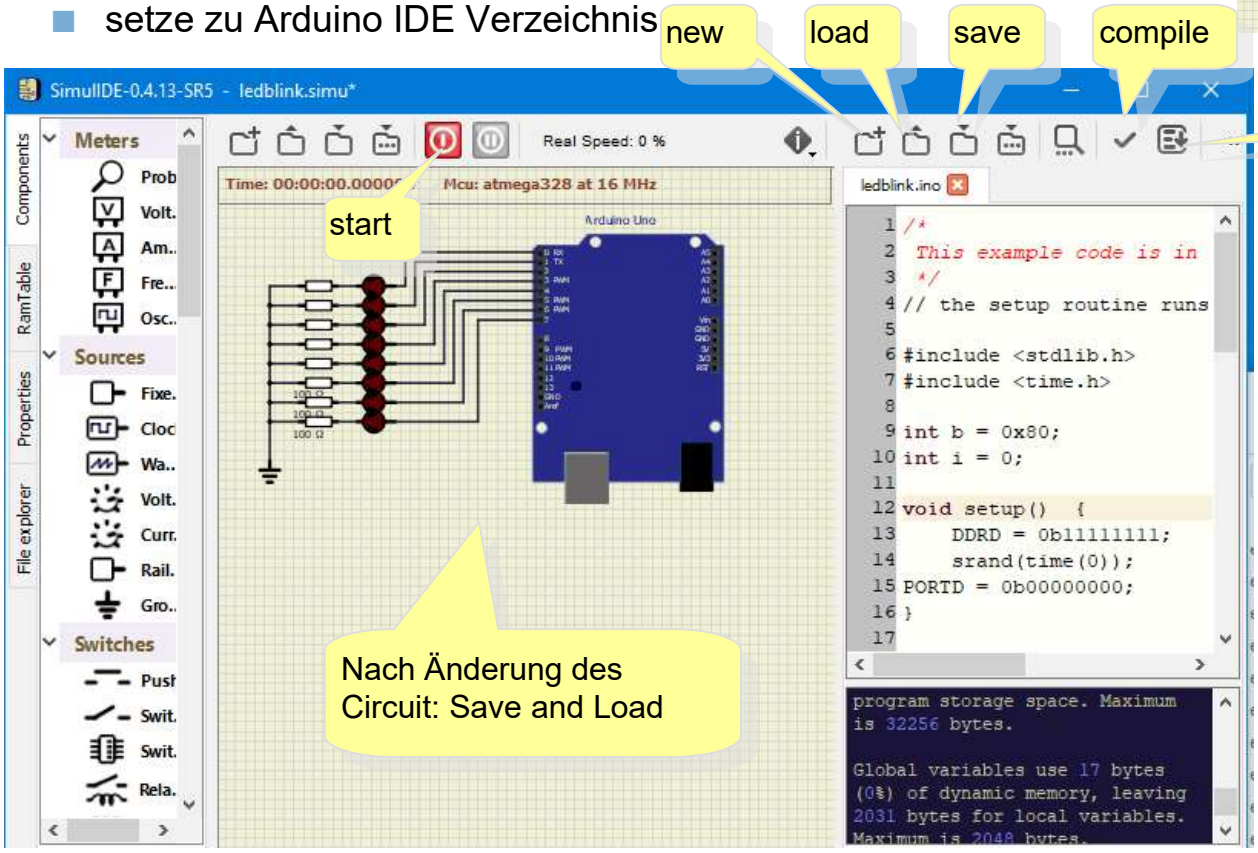


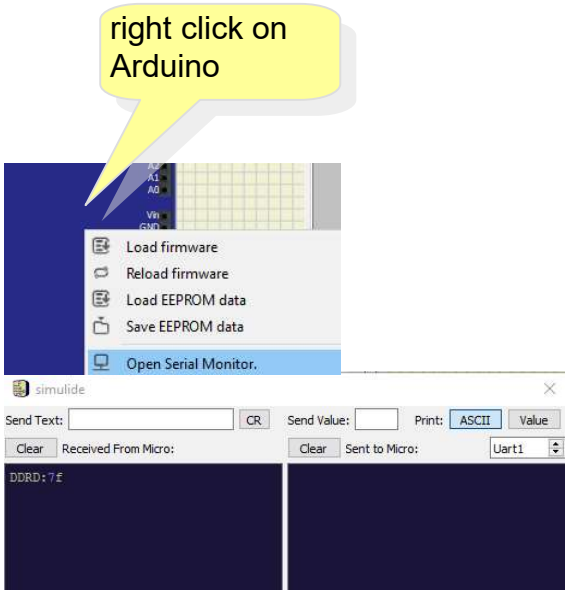
http://www.netzmafia.de/skripten/hardware/Arduino/Programmierung/portmanipulation.html

SimulIDE

- Installiere SimulIDE https://www.simulide.com/p/home.html
- und Arduino IDE https://www.arduino.cc/en/software/
- Start und verbinde Compiler mit Rechtsklick auf Tab
  - setze zu Arduino IDE Verzeichnis







Noch Fragen?

Lösung: Übung Bit-Operatoren

School of Engineering

- Welchen Wert hat r jeweils nach Ausführen dieser Ausdrücke?

```
unsigned char a = 5, b = 6, r;

r = a | b;      // 7
r = a ^ b;      // 3
r = a & b;      // 4
r = ~r;         // 251
r >>= 4;        // 15
```