Hier geht es ran an die Mathematik, so wie wir es früher in der Schule gelernt haben: händisch multiplizieren und dividieren.

Dipl.-Ing. (FH) CID+ ZED F.Jürgen Gensicke, Kirberichshofer Weg 31, 52066 Aachen
www.Elektronikentwickler-Aachen.de, Tel.: 0241 / 47580488

16Bit Multiplikation und andere Berechnungen

Berechnungsformel

Spannung = ADC(HEX) * Auflösung

Auflösung = V_{ref} / 2^(ADC-Bitzahl)

Beispiel zur Berechnung der Auflösung (aus dem Teil 1-Video):

V_{ref} = 5,00V
ADC-Bitzahl = 10 Bits, entsprechend 1024 Schritte
Auflösung = 5,00V / 1024 = 0,0048828125V/digit

Beispiel zur Berechnung der Spannung in Volt:

Spannung = 0123h * 0,0048828125V/digit
mit Taschenrechner
= 291d * 0,0048828125V/digit
= 1,4208984375V mit Taschenrechner

Berechnungsformel

Spannung(HEX) = ADC(HEX) * Auflösung

Auflösung anders Skaliert:
0,0048828125V/digit = 4,8828125mV/digit = 4882,8125µV/digit

Auflösung gerundet und in HEX konvertiert:
4882,8125µV/digit ist ca. 4883µV/digit = 1313h/digit (zu beachten: ist in µV)

Beispiel zur Berechnung der Spannung in HEX:
Spannung(HEX)= 0123h * 1313h/digit
= 0015AE99h (µV) [4 Byte]

mit PIC in 16Bit * 16Bit Multiplikation

16Bit * 16Bit Multiplikations-Funktion (Subroutine) mit Ergebnis in 32 Bit (4 Byte).
Real: 10Bit * 13Bit Multiplikation mit Ergebnis in 24 Bit (3 Byte).
In unserem Beispiel also: 15AE99h (µV) (3 Byte)

8 Bit * 8 Bit Multiplikation im PIC16

Wie funktioniert das?

Wenn das unterste Bit im Byte Argument8.1 „1“ ist, wird erweiterte Argument8.2L/H zum Ergebnis RES81/80 hinzuaddiert. Bei „0“ nicht.
Danach rotiert das Arg8.1 nach rechts und das erweiterte Argument8.2L/H wird in der Wertigkeit durch Linksshift erhöht. Nach 8 Durchläufen ist die 8*8 Multiplikation beendet.

```
;SR UNSIGNED_8_8_MULT();
;*****************************************************************************************
;*                          8Bit * 8Bit Multiply Routine                                 *
;*                          RES81:RES80 = ARG81 * ARG82L                                 *
;*                          Benutzte Variablen:                                          *
;*                          RES81,RES80,ARG81,ARG82H,ARG82L                              *
;*                          Benutzte Subroutinen:                                        *
;*                          keine                                                        *
;*                          Datum: 24.09.21                                              *
;*                          Autor: F.J.Gensicke                                          *
;*****************************************************************************************
UNSIGNED_8_8_MULT       CLRF    ARG82H
                        CLRF    RES80
                        CLRF    RES81
                        MOVLW   09H
                        MOVWF   COUNT
U88M_LOOP               DECFSZ  COUNT,SAME
                        GOTO    U88M_CALCULATE
                        RETURN                              ; zurueck
U88M_CALCULAT           RRCF    ARG81,SAME                  ; Carry-Bit mit LSB laden
                        BTFSS   STATUS,CARRY
                        GOTO    U88M_ROT_LEFT_ARG82         ; Carry-Bit = 0
                        MOVF    ARG82L,WORK
                        ADDWF   RES80,SAME                  ; RES80 = RES80 + ARG82L / Carry beachten!
                        BTFSC   STATUS,CARRY
                        INCF    RES81,SAME                  ; Carry = 1 -> RES81 = RES81 + 1
                        MOVF    ARG82H,WORK
                        ADDWF   RES81,SAME                  ; RES80 = RES80 + ARG82L / Carry unwichtig!;
U88M_ROT_LEFT_ARG82     BCF     STATUS,CARRY                ; CarryBit = 0
                        RLCF    ARG82L,SAME                 ; Carry + ARG82L = Leftshift ARG82L
                        RLCF    ARG82H,SAME                 ; Carry + ARG82H = Leftshift ARG82H + Carry
                        GOTO    U88M_LOOP
```

8 Bit * 8 Bit Multiplikation im PIC18

Wie funktioniert das?

Die PIC18er Serie hat einen internen Hardware-Multiplizierer, der die 8Bit * 8Bit Multiplikation in einem Systemtakt ausführt.

Meine Empfehlung ist es daher, wenn es auf Geschwindigkeit ankommt und es muss Multipliziert werden, dann immer einen PIC18er einsetzen.

```
;SR UNSIGNED_8_8_MULT();
;*****************************************************************************************
;*                          8Bit * 8Bit Multiply Routine                 *
;*                          RESH:RESL = ARG1 * ARG2                      *
;*                          Benutzte Variablen:                          *
;*                          RESH,RESL,ARG1,ARG2                          *
;*                          Benutzte Subroutinen:                        *
;*                          keine                                        *
;*                          Datum: 24.09.21                              *
;*                          Autor: F.J.Gensicke                          *
;*****************************************************************************************
UNSIGNED_8_8_MULT   MOVF    ARG1,WORK,0
                    MULWF   ARG2,0          ; ARG1 * ARG2 -> PRODH:PRODL
                    MOVF    PRODL,WORK,0
                    MOVWF   RESL,0
                    MOVF    PRODH,WORK,0
                    MOVWF   RESH,0
                    RETURN                  ; zurueck
```

16 Bit * 16 Bit Multiplikation im PIC16/18

Wie funktioniert das?

Nebenstehend ist nur der Header meiner Subroutine aufgezeigt, aber mit Referenzen wo man mehr Informationen herholen kann.
Im Prinzip ist es eine Kreuzmultiplikation von 2 Byte Argument1 mit 2 Byte Argument2. Je nach Byte-Kombination wird das Ergebnis entweder um 2^8 oder um 2^16 nach links im Ergebniswort RES3:0 aufaddiert.

Im PIC16 muss man dann für die 8*8 Multiplikation die vorherige Subroutine verwenden. Im PIC18 kann man direkt dann den Hardware-Multiplizierer (MUL) verwenden.

```
;SR UNSIGNED_16_16_MULT();
;*****************************************************************************************************
;*                                16Bit * 16Bit Multiply Routine unsing Kernel MUL-Function    *
;*                                RES3:RES0  = ARG1H:ARG1L * ARG2H:ARG2L                       *
;*                                = (ARG1L * ARG2L)              +                             *
;*                                  (ARG1H * ARG2H * 2^16)       +                             *
;*                                  (ARG1L * ARG2H * 2^8)        +                             *
;*                                  (ARG1H * ARG2L * 2^8)                                      *
;*                                Referenzen: http://www.8052.com/mul16 und                    *
;*                                PIC18_L_F2X_4XK22.pdf Kap 8 Seite 111                        *
;*                                Benutzte Variablen:                                          *
;*                                RES3,RES2,RES1,RES0,ARG1H,ARG1L,ARG2H,ARG2L                  *
;*                                Benutzte Subroutinen:                                        *
;*                                UNSIGNED_8_8_MULT                                            *
;*                                Datum: 24.09.21                                              *
;*                                Autor: F.J.Gensicke                                          *
;*****************************************************************************************************
UNSIGNED_16_16_MULT       …
```

Sonderfall: x Bit * 2,4,8,16,etc Integer-Multiplikation/Division im PIC

Wie funktioniert das?

Im Binärsystem allgemein gilt eine Verdopplung des Wertes von Bit LSB nach Bit MSB.
Beispiel:
Bit 0 hat den Wert 1
Bit 1 hat den Wert 2
Bit 2 hat den Wert 4
etc.

Das heißt, wenn ich in einem Register folgende Zahl stehen habe:
b‘00101101‘, was in Hex 2Dh bedeutet und in Dezimal 45d,
und müsste diese mit 4 Multiplizieren, dann ist der schnellste Weg zum Ziel über den zweifachen Linksshift:
2 * RLNCF b‘00101101‘ = b‘10110100‘, also in HEX B4h und in Dezimal 180d > 4 * 45 = 180!

Das gleiche gilt auch für die geradzahlige Integer-Division.

Im Teil 1 haben wir ja auch die fehlerbehaftete Rundung betrachtet. Je nach zugelassenem Fehler (Toleranz) kann diese sehr schnelle Multiplikation bzw. Division eine Option sein.

Sonderfall aus Beispiel Teil 1: x Bit * 5 Integer-Multiplikation im PIC

In Teil 1 hatten wir das Beispiel, wo wir von 4,8828mV auf 5mV mit einem akzeptablen Fehler von 2,4% aufgerundet haben.
Die „5“ lässt sich leicht entweder durch einen internen Hardware Multiplizierer oder durch 2 mal LEFTSHIFT + 1 mal ADD sehr schnell berechnen.

Wie funktioniert das?

Mit dem Hardware Multiplizierer kann man die Multiplikation mit 5 (oder einer anderen Integerzahl bis 255) in 10 CPU-Takte erledigen.
Die Shift-Add-Variante benötigt für die Multiplikation ebenfalls 14 CPU-Takte.
Je nach Multiplikator kann das auch schneller gehen, aber auch langsamer.

BTW: 16MHz Takt = 250nsec CPU-Takt

```
;SR UNSIGNED_10_5FIX_MULT()
;*************************************************************************************
;*       10Bit * 5 Fix Integer Multiply Routine with Hardware Multipl.               *
;*       RES1:RES0       =       ARG1H:ARG1L * 5                                     *
;*       Benutzte Variablen: RES1,RES0,ARG1H,ARG1L                                   *
;*       Benutzte Subroutinen: keine                                                 *
;*       Datum: 02.11.21                                                             *
;*       Autor: F.J.Gensicke                                                         *
;*************************************************************************************
UNSIGNED_10_5FIX_MULT   MOVF    ARG1L,WORK,0
                        MULLW   ARG2L                   ; ARG1L * ARG2L -> PRODH:PRODL
                        MOVFF   PRODH,RES1              ;
                        MOVFF   PRODL,RES0              ;
                        MOVF    ARG1H,WORK,0            ;
                        MULLW   ARG2L                   ; ARG1H * ARG2L -> PRODH:PRODL
                        MOVF    PRODL,WORK,0            ;
                        ADDWF   RES1,SAME,0             ; Add PRODL to RES1 (high)
;                       * PRODH not further used, because is allway zero while we multiply by 5!
                        RETURN                          ;zurueck
;
UNSIGNED_10_5FIX_MULT_SIHFTADD
                        BCF     STATUS,CARRY,0          ; Carry Bit loeschen
                        RLCF    ARG1L,WORK,0            ; Mult Low by 2
                        MOVWF   RES0,0
                        RLCF    ARG1H,WORK,0            ; Mult High by 2
                        MOVWF   RES1,0
                        BCF     STATUS,CARRY,0          ; Carry Bit loeschen
                        RLCF    RES0,SAME,0             ; Mult Low by 2
                        RLCF    RES1,SAME,0             ; Mult High by 2
                        MOVF    ARG1L,WORK,0            ;
                        ADDWF   RES0,SAME,0             ; RES0 + Carry = Res0 + ARG1L
                        MOVF    ARG1H,WORK,0            ;
                        ADDWFC  RES1,SAME,0             ; RES1 = Res1 + ARG1H + Carry
                        RETURN                          ;zurueck
```

HEX zu Dezimal Konverter

Umrechnung HEX in Dezimal für das LC-Display

Aus unserem Beispiel: 15AE99h (µV) [3 Byte]

15AE99h in µV, damit kann keiner was anfangen!
Wir brauchen einen dezimalen Wert.

Mit dem Taschenrechner umgerechnet:
15AE99h > 1420953d (µV) > Darstellung auf LC-Display 1,420953V

Aber wie geht sowas auf einem Mikrocontroller?

Denn auf dem Display soll nachher „1,420953V“ stehen.
Für die Datenübertragung zum Display und zur Darstellung, sind das insgesamt 9 Zeichen, die übertragen werden müssen. Und zwar Einzeln!
Also muss jede Stelle des Wertes „1420953“ in einzelnen Bytes oder (als Array) abgelegt werden und dann nacheinander ans Display gesendet werden.

Umrechnung HEX in Dezimal für das LC-Display

Um zu verstehen wie der Algorithmus funktioniert, machen wir zuerst mal ein Beispiel mit Dezimalzahlen.

Das Grundprinzip folgt der Logik, vom Ausgangswert die zuerst höchste Stellenwertigkeit so lange zu subtrahieren, bis sie negativ wird. Dann folgt vom Rest die Subtraktion der nächsten kleineren Stellenwertigkeit.

Beispiel: die Zahl 2345d soll als einzelne Stellenbytes abgelegt werden.

2345d	1000er	100er	10er	1er
-1000	I			
1345				
-1000	I			
345				
-100		I		
245				
-100		I		
145				
-100		I		
45				
-10			I	
35				
-10			I	
25				
-10			I	
15				
-10			I	
5				5
4 Stellenbytes:	**2**	**3**	**4**	**5**

Umrechnung HEX in Dezimal für das LC-Display

Jetzt mal in HEX mit dem gleichen Wert 2345d, in HEX 0929h.

Das Grundprinzip folgt hier exakt nach dem gleich Algorithmus wie eine Seite vorher.

Stellenwertigkeit:

1000d > 03E8h
100d > 0064h
10d > 000Ah
1d > 0001h

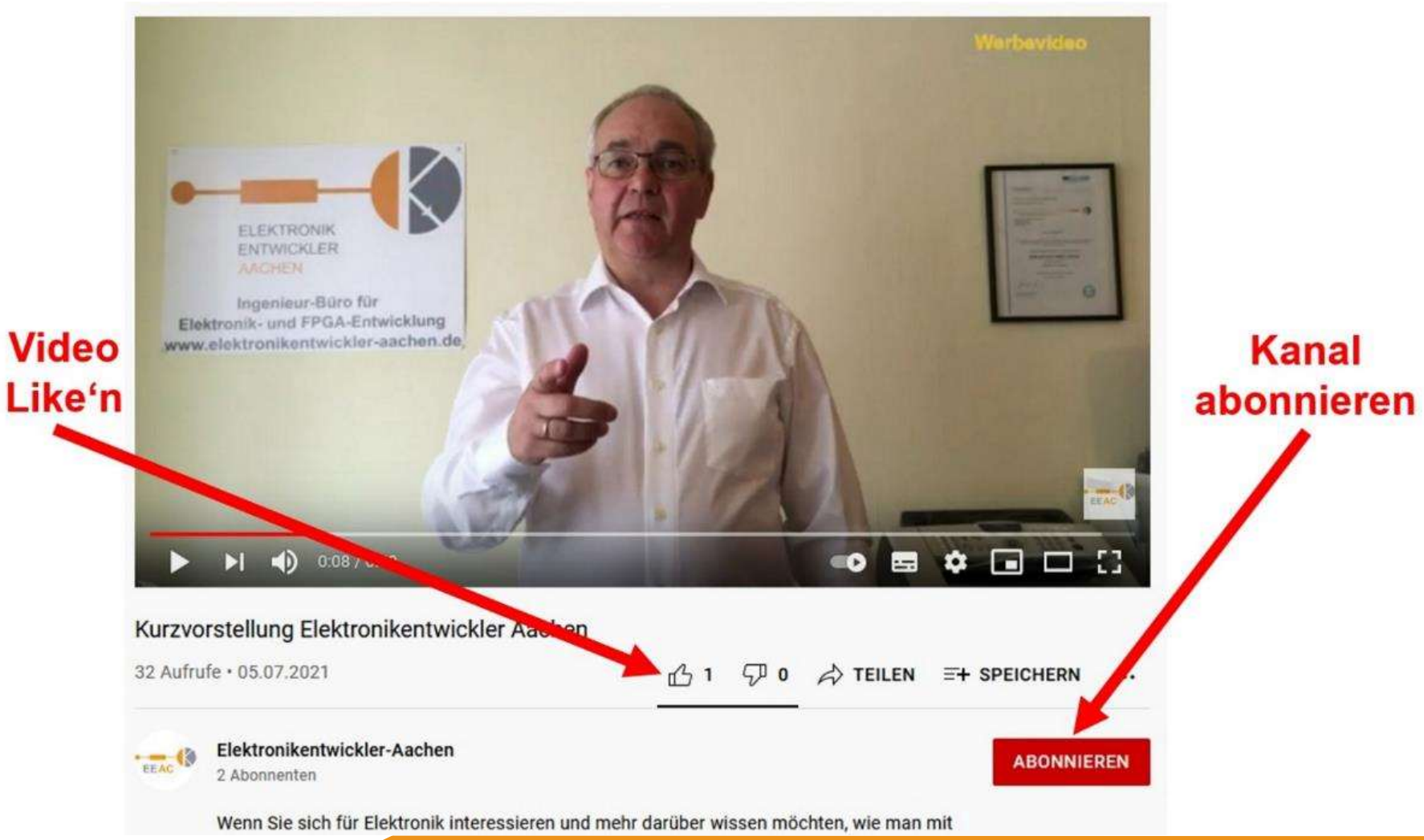
0929h	03E8h(1000er)	64h(100er)	0Ah(10er)	01h(1er)
-03E8	I			
541				
-03E8	I			
159				
-64		I		
F5				
-64		I		
91				
-64		I		
2D				
-0A			I	
23				
-0A			I	
19				
-0A			I	
0F				
-0A			I	
5				5
4 Stellenbytes:	**2**	**3**	**4**	**5**

Umrechnung HEX in Dezimal für das LC-Display

Das Bild rechts zeigt eine komplette Berechnung zu den oben gezeigten Beispielen.

0123h ist der Wert vom ADC.
1313h ist der Skalierungsfaktor.
15AE99h das Ergebnis der Multiplikation aus obigem.
1,420953V ist das dezimale Ergebnis der Multiplikation.

Das Ergebnis ist mathematisch korrekt ohne Fehler!


Video
Like'n
Kanal
abonnieren
Werbevideo
ELEKTRONIK
ENTWICKLER
AACHEN
Ingenieur-Büro für
Elektronik- und FPGA-Entwicklung
www.elektronikentwickler-aachen.de
Kurzvorstellung Elektronikentwickler Aachen
32 Aufrufe • 05.07.2021
1
0
TEILEN
SPEICHERN
Elektronikentwickler-Aachen
2 Abonnenten
ABONNIEREN
Wenn Sie sich für Elektronik interessieren und mehr darüber wissen möchten, wie man mit