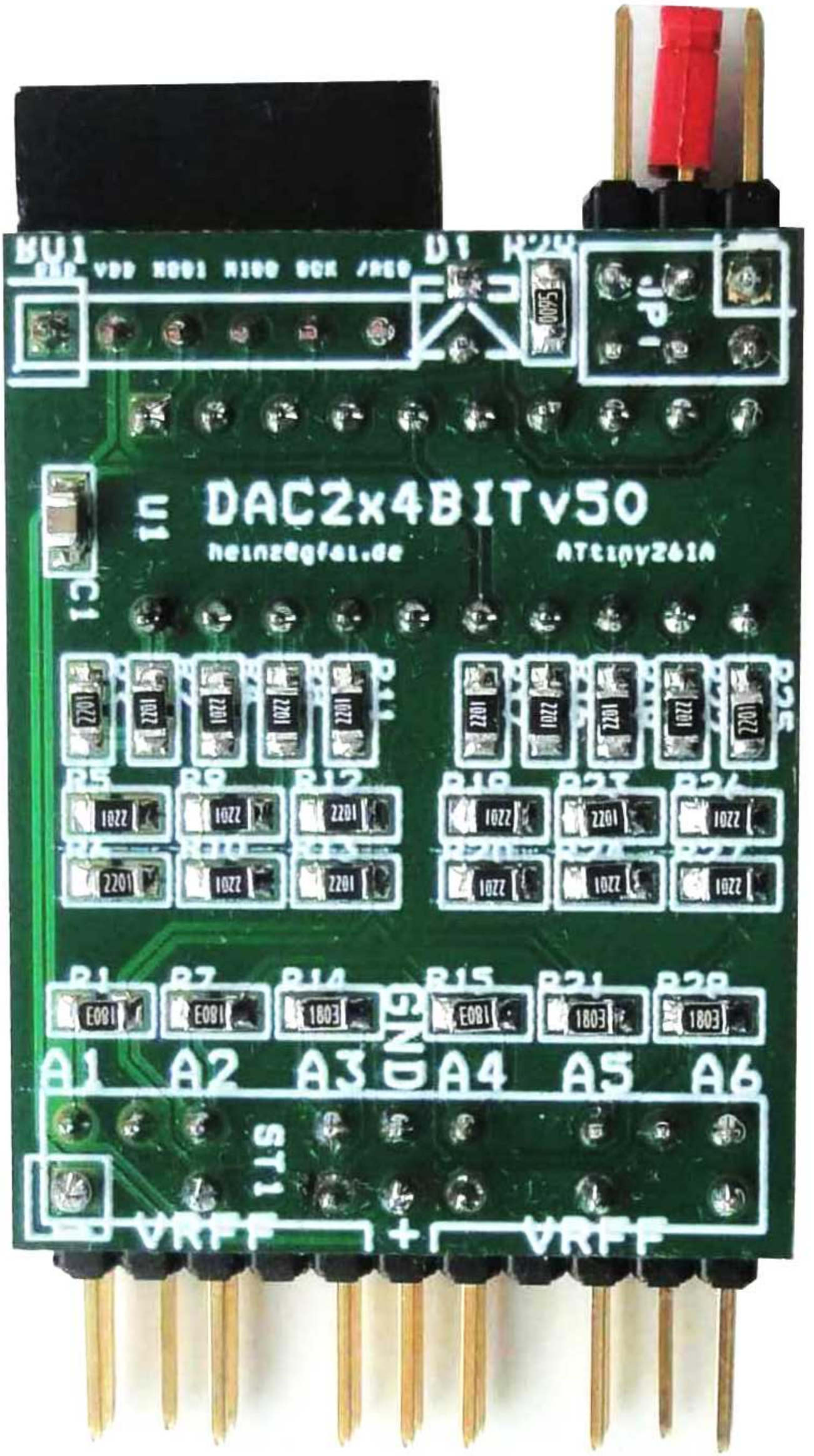
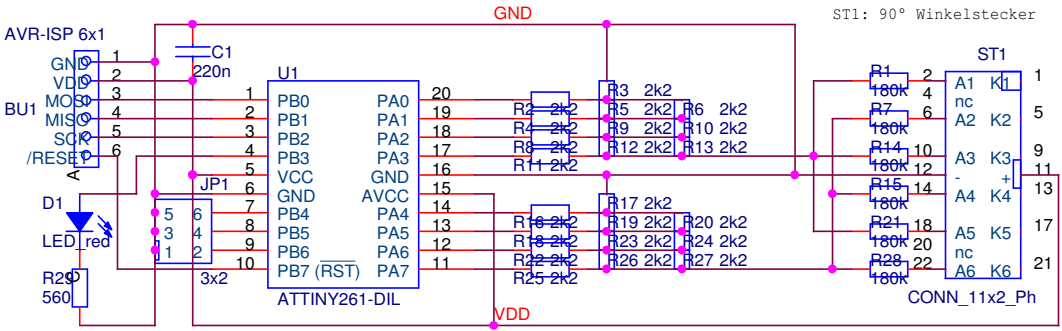
BU1
GND VDD MOSI MISO SCK /RES
D1
R29
JP1
U1
DAC2x4BITv50
heinz@gfai.de
ATtiny261A
C1
R4 R8 R11
R16 R17 R18 R22 R25
R5 R9 R12 R19 R23 R26
R6 R10 R13 R20 R24 R27
R1 R7 R14 R15 R21 R28
GND
A1 A2 A3 A4 A5 A6
SV1
VREF
+
VREF


DAC2x4BITv50
heinz@gfai.de
ATtiny261A
VRFF
VRFF
A1
A2
A3
GND
A4
A5
A6



Achtung: Gefahr für ION!
VDD am AVRISP ist 5 Volt, VCC vom
ION ist 3,3 Volt! Entweder AVRISP,
oder ION stecken!

VDD = 3,3V: ia = +/- 8,84µA pro Kanal

Title: DAC 2x4 Bit - PCB-Version		
Size: A4	Document Number: heinz@gfai.de	Rev: 50

BU1
GND VDD MOSI MISO SCK /RES
D1
R29
JP1
U1
C1
DAC2x4BITv50
heinz@gfai.de
ATtiny261A
R4 R8 R11
R17 R18 R22 R25
R5 R9 R12
R19 R23 R26
R6 R10 R13
R20 R24 R27
R1 R7 R14
R15 R21 R28
GND
A1 A2 A3 A4 A5 A6
ST1
VREF
+
VREF

DAC 2x4 Bit - PCB-Version
heinz@gfai.de Revision: 50

Bill Of Materials

PCB 1-layer TOP + SST
1550 x 1200 mil = 39,37 x 30,48 mm

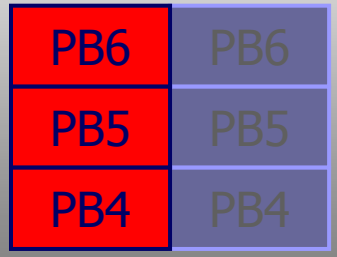
Item	Quantity	Reference	Part
1	1	BU1	AVR-ISP 6x1
2	1	C1	220n
3	1	D1	LED_red
4	1	JP1	3x2
5	6	R1,R7,R14,R15,R21,R28	180k
6	22	R2,R3,R4,R5,R6,R8,R9,R10, R11,R12,R13,R16,R17,R18, R19,R20,R22,R23,R24,R25, R26,R27	2k2
7	1	R29	560
8	1	ST1	CONN_11x2_Ph
9	1	U1	ATTINY261-DIL

Betriebsarten

JP1 PBxyz	LEDclk	Funktion	Symbolbild
-	100	zaehne	XXX...XXXXX
4	250	pyramids	___/X___
5	150	halbtreppe	___|X|___
6	250	treppen	___|X|___
45	150	invpyr	===XX====
46	150	rechteck	__|=|-|=|__
56	100	saege	X|X|X|X|X|X
456	250	invpuls	===|X====

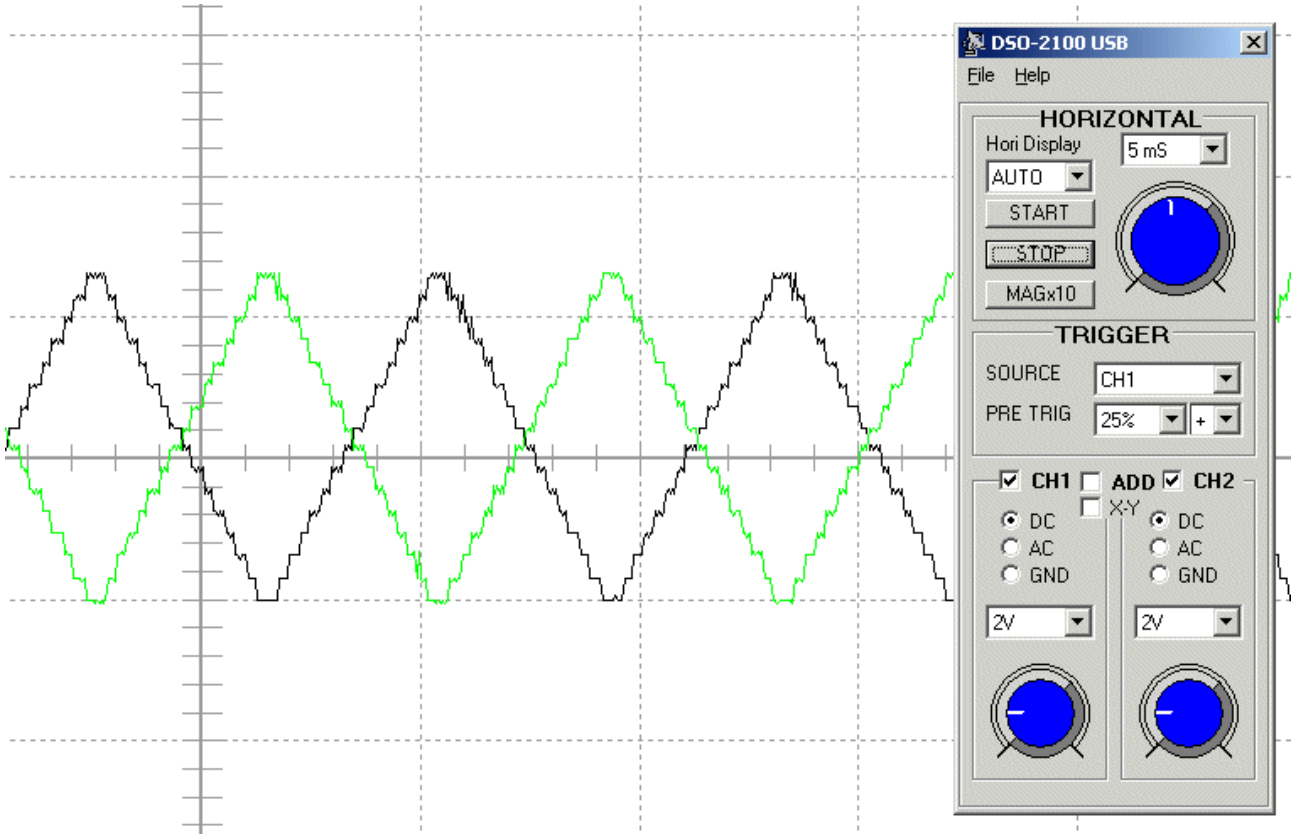
Jumper JP1 gesetzt: rot

Jumper JP1 offen: graublau

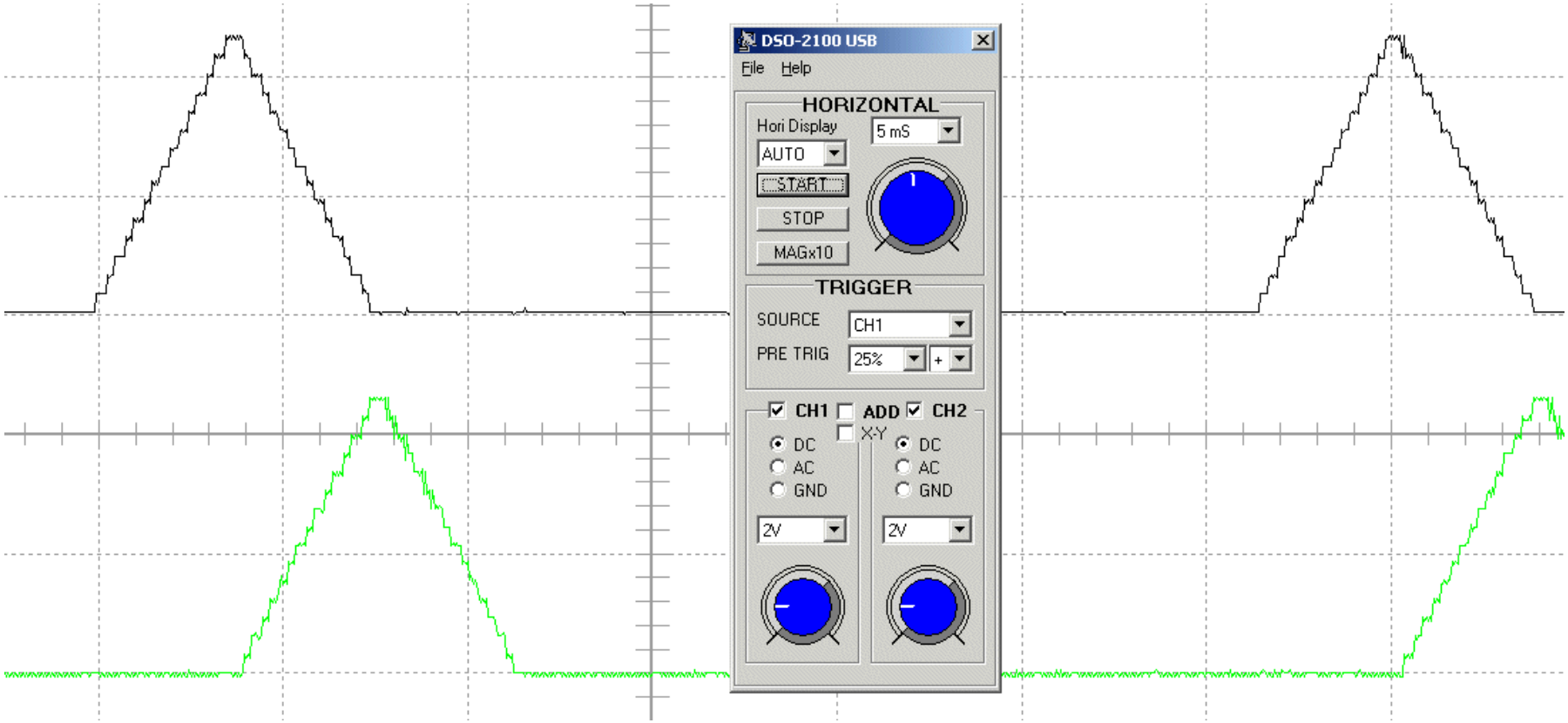


zaehne();

PB6
PB5
PB4

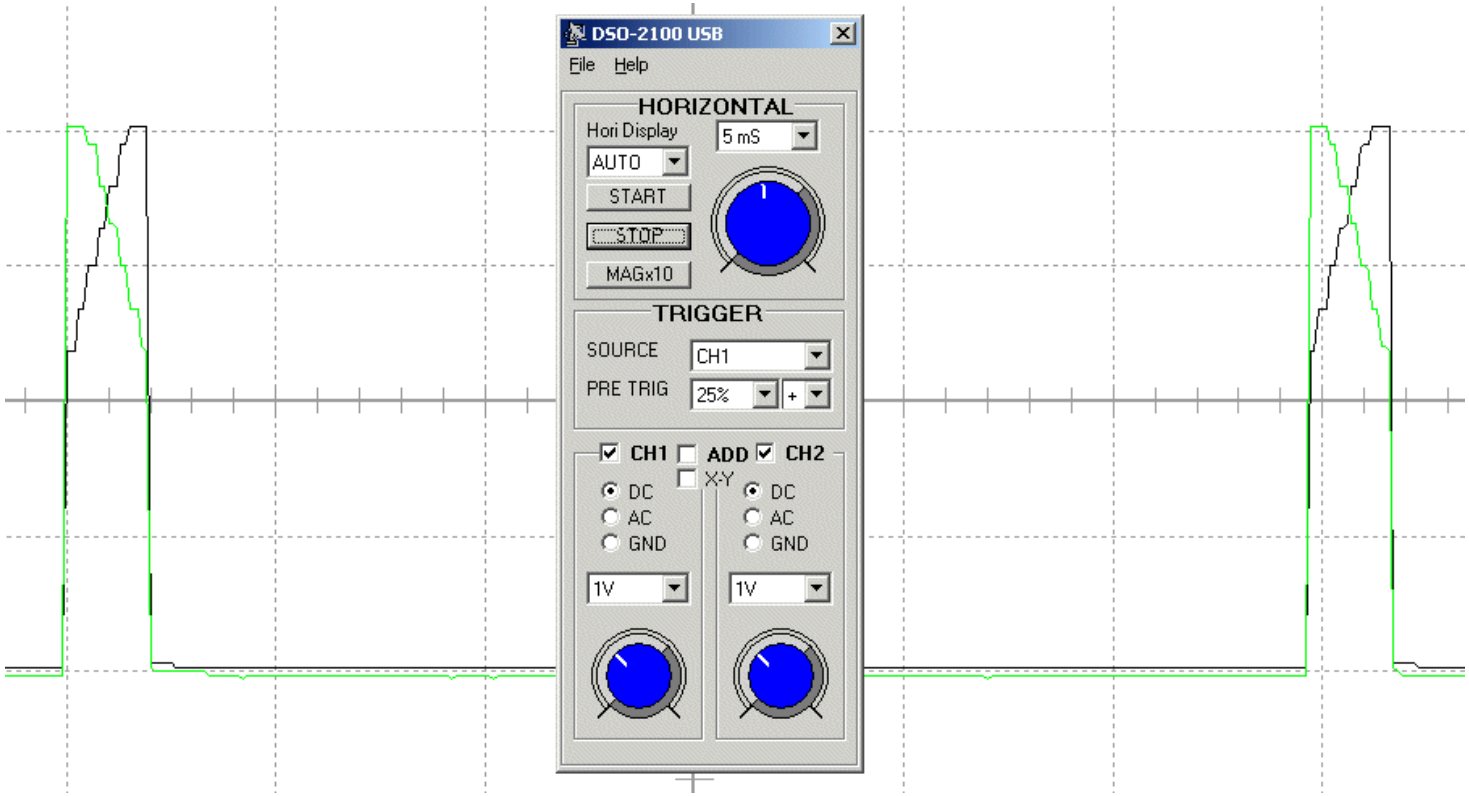


pyramids();


PB6
PB5
PB4
DSO-2100 USB
File Help
HORIZONTAL
Hori Display
5 mS
AUTO
START
STOP
MAGx10
TRIGGER
SOURCE
CH1
PRE TRIG
25%
+
CH1
ADD
CH2
X-Y
DC
AC
GND
DC
AC
GND
2V
2V

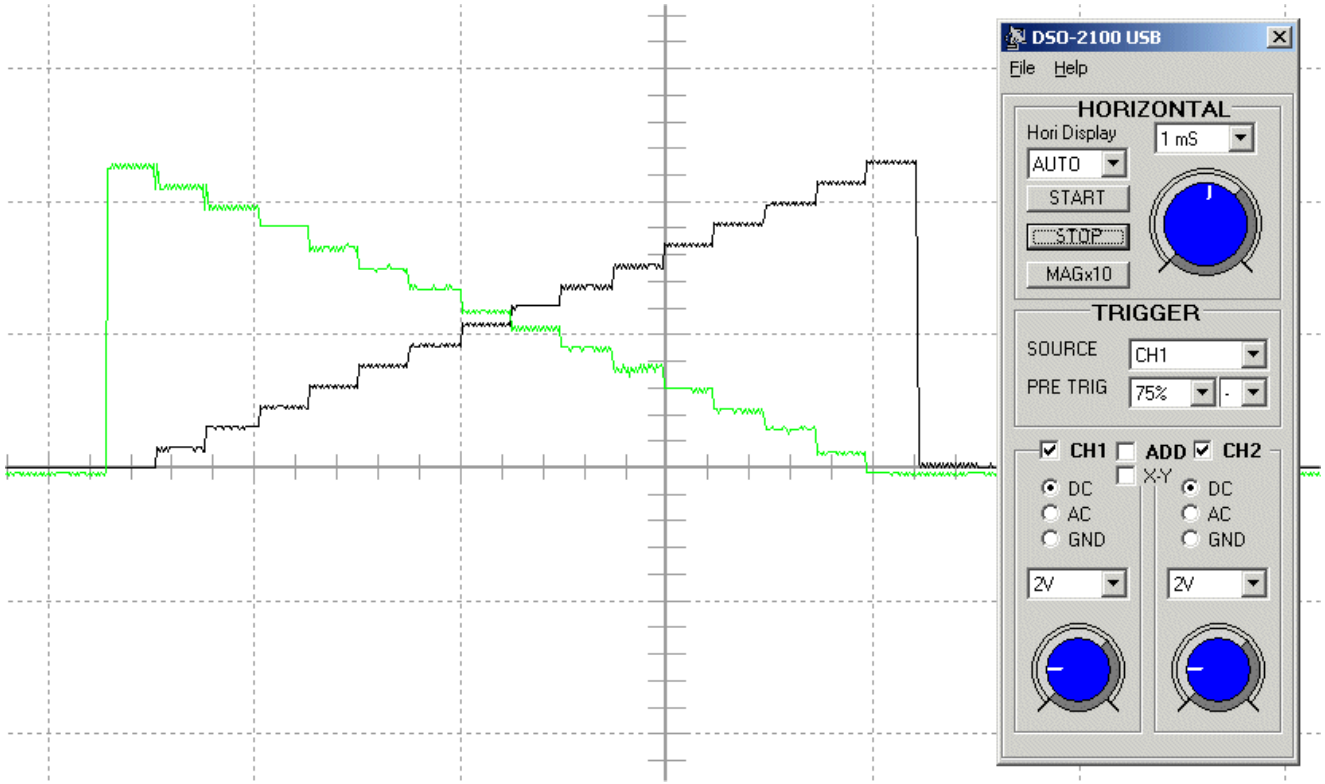
halbtreppe();

PB6
PB5
PB4



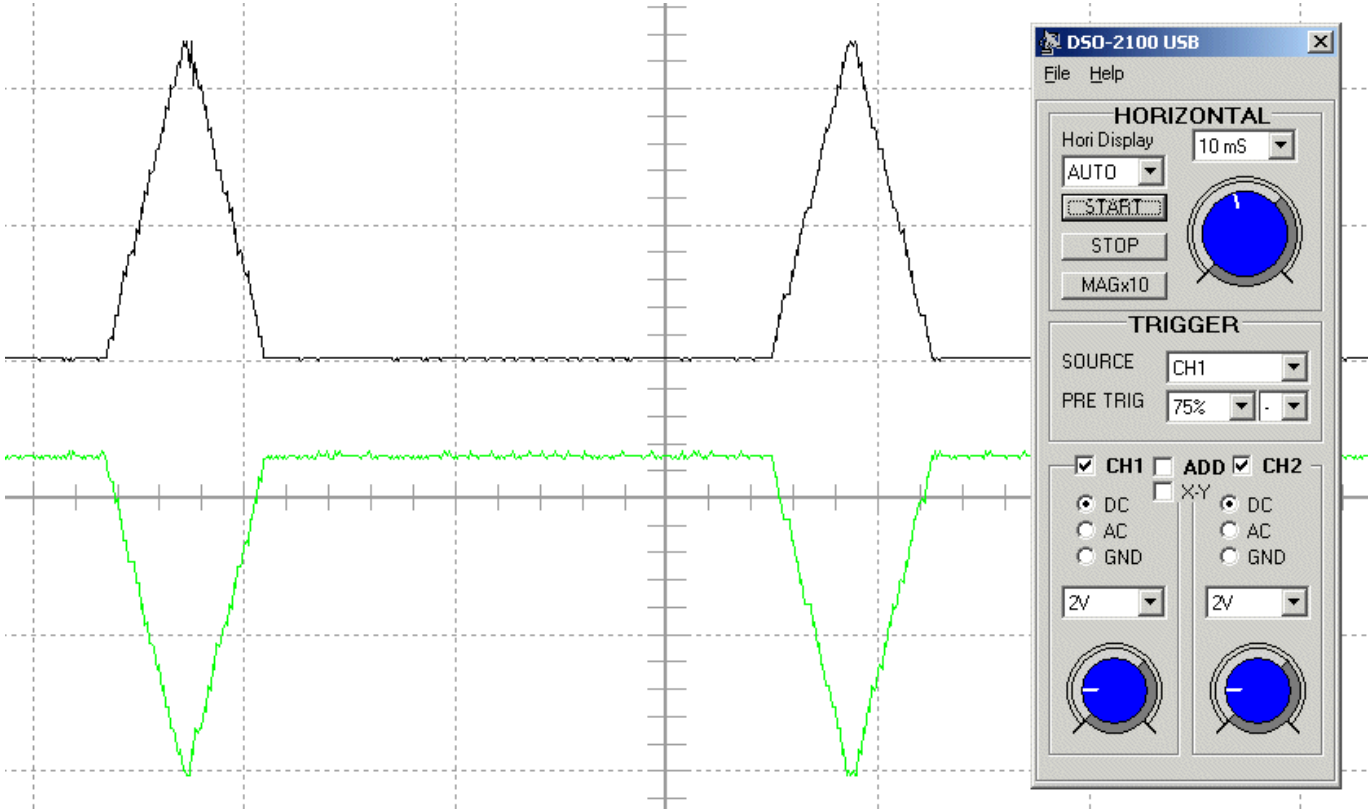
treppen();

PB6
PB5
PB4

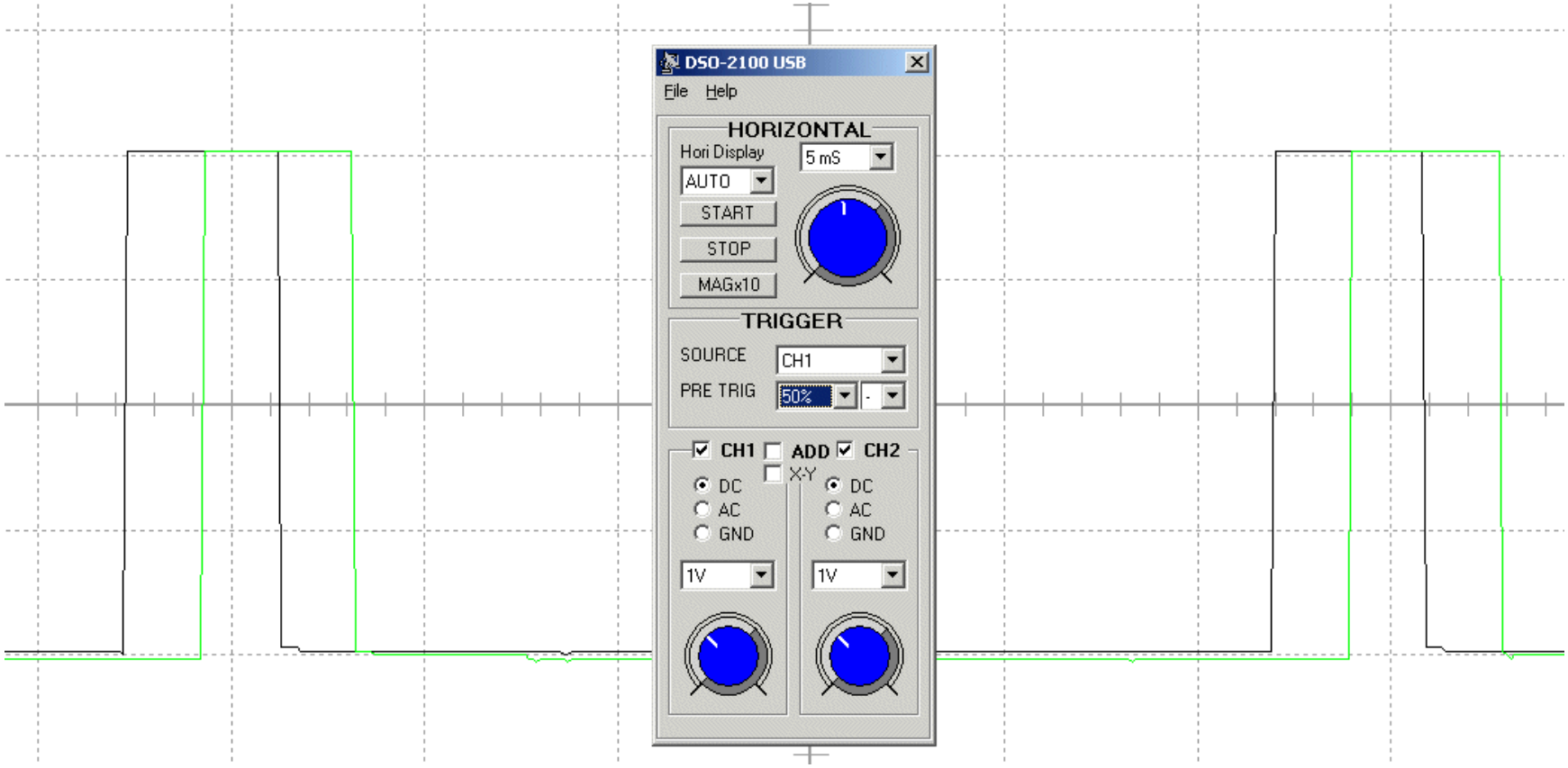


invpyr();

PB6
PB5
PB4

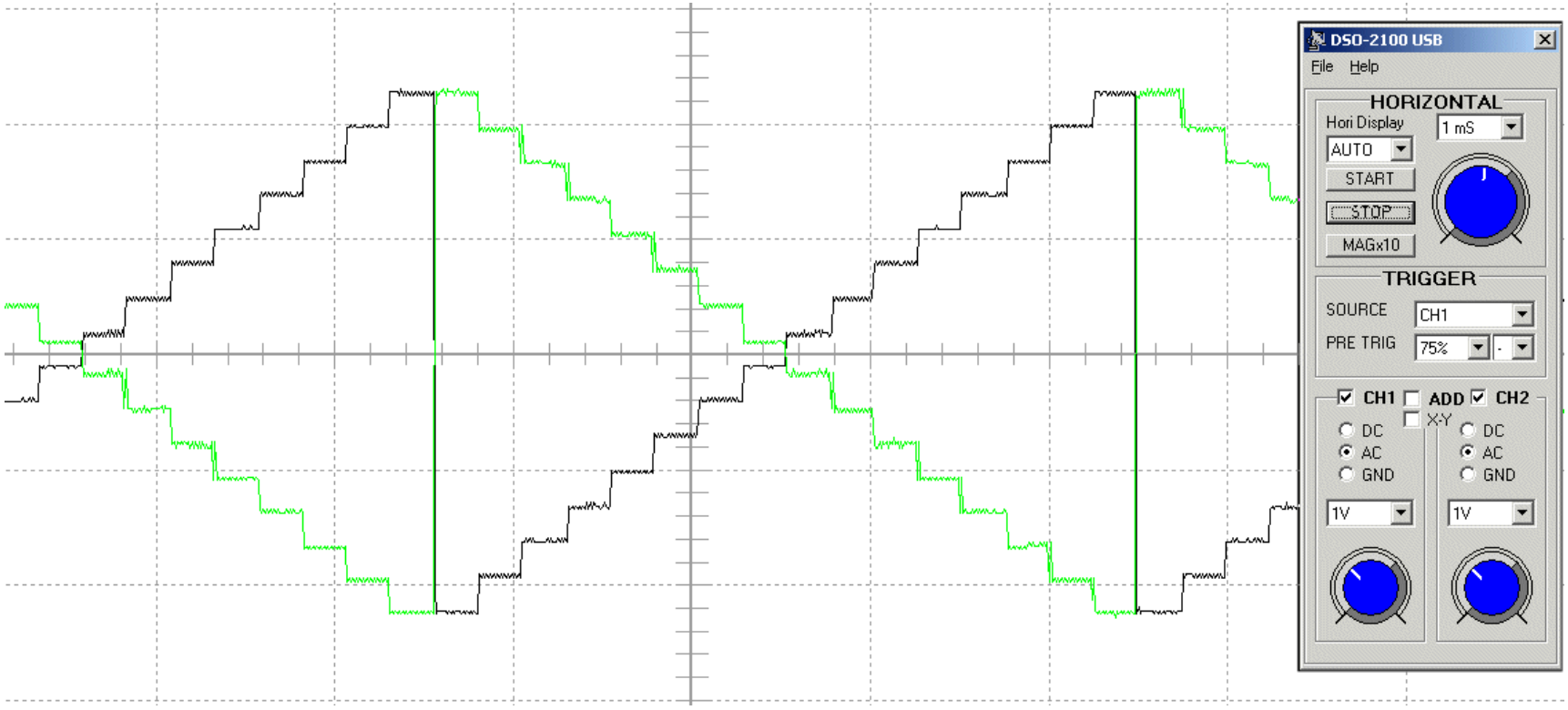


rechteck();


PB6
PB5
PB4
DSO-2100 USB
File Help
HORIZONTAL
Hori Display
5 mS
AUTO
START
STOP
MAGx10
TRIGGER
SOURCE
CH1
PRE TRIG
50%
CH1 ADD CH2
X-Y
DC
AC
GND
1V

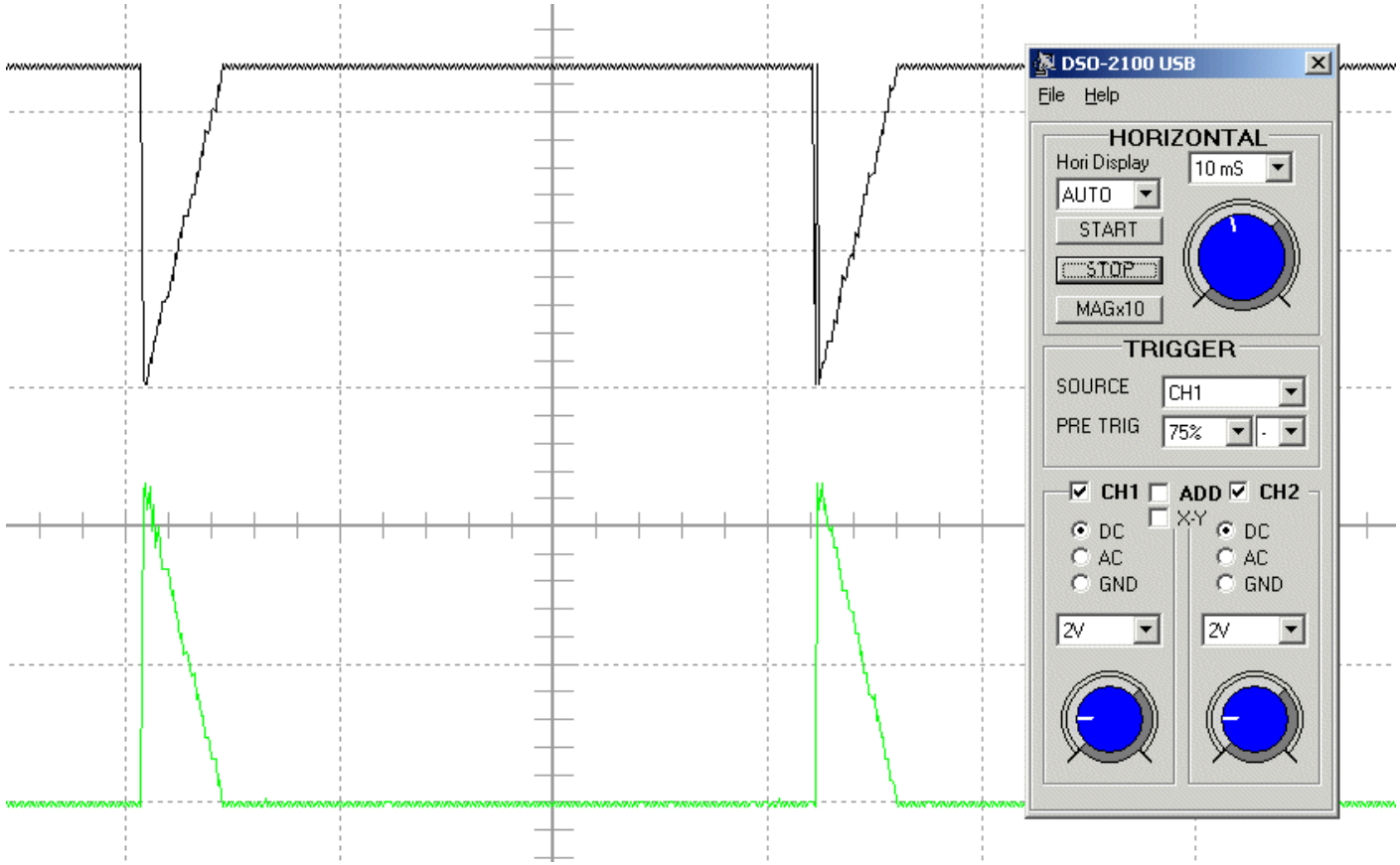
saege();

PB6
PB5
PB4



invpuls();

PB6
PB5
PB4



```
/*

Schneller, zweikanaliger Zeitfunktionsgenerator
mit zwei Digital Analog Convertern a 4 Bit
Basis Atmel ATtiny261A
http://www.gfai.de/~heinz/techdocs
heinz@gfai.de

Funktion
Auf PA0...3 und PA4...7 werden bei jedem Systick
zwei Nibble a 4 Bit ausgegeben. An diesen Portausgängen
liegen zwei 4-Bit R-2R-Wandler, die als DAC je eine
Zeitfunktion erzeugen.
Der Systemtakt wird mit Timer0 eingestellt. An drei Jumpern
(PB4,5,6) können acht Zeitfunktionen eingestellt werden.
Jeder Systick sorgt für Abarbeitung des an den Jumpern
eingestellten Zeitfunktionsprogramms.
An Portpin PB3 ist eine LED angeschlossen, deren Blinkrate
über Timer1 gesteuert wird.

Digital-Analog-Wandler (DAC)
Die DAC besitzen eine Ausgangsimpedanz von etwa 1 kOhm. In der konkreten
Applikation sind je drei 180 kOhm Widerstände in Serie an den Ausgang
geschaltet, um einen sehr geringen Ausgangsstrom zu erzeugen zur
Imitation von Photodioden.
Die zwei R-2R-DAC könnnen im Prinzip auch ohne Timer betrieben werden,
die Taktrate des Prozessors begrenzt deren maximale Arbeitsfrequenz.

Betriebsarten
Am PORTB 4,5,6 sind drei Jumper gegen Masse angeschlossen.
Darüber lassen sich 2^3 = 8 Betriebsarten wählen.
Jumper gesetzt erzeugt low am Portpin, die Jumper sind low-aktiv.
Vorsicht: Das Programm cmdDecode() erhält high-aktive Werte als Input.

    /PB     LEDclk  Programm    Zeitfunktion

    -       100     zaehne      XXX...XXX
    4       250     pyramids    ___/X\___
    5       150     halbtreppe  ___|X|___
    6       250     treppen     ___|X|___
    45      150     invpyr      ===XX====
    46      150     rechteck    _|=|-|=|_
    56      100     saege       X|X|X|X|X
    456     250     invpuls     ===|X====

Variablen
    cnt:    Zähler wird durch Timer0-Overflow-Interrupt inkrementiert
            und bei stepmax rückgesetzt
    lo:     Low-Nibble  0x0_ <=> DAC0
    hi:     High-Nibble 0x_0 <=> DAC1

Inbetriebnahme
Zum Brennen sind AVR Programmieradapter im AVR-ISP-Mode geeignet.
Achtung: Der ATtiny261A hat standardmäßig die Fuse CLKDIV (/8) gesetzt.
Fuse bitte entsetzen.

Programmierumgebung
AVR-Studio 4.19, Built 730, Atmel 2011 incl. WINAVR-GCC
Programmer z.B. JTAGICEmkII, AVRISPmkII, USBprog u.a.

*/


/********* Compiler ************************************************/


#include <avr/io.h>
#include <avr/interrupt.h>

// falls nicht im AVR-Studio-Projekt definiert:
//#define __AVR_ATtiny261A__    // Prozessor (ruft <avr/iotn261a.h> über io.h auf)
#define F_CPU 8000000UL         // 8 MHz interner RC-Oszillator
```

```
// Systick:
#define systick32us     0x01    // 32 µs
#define systick256us    0x02    // 256 µs
#define systick2ms      0x03    // 2 ms
#define systick8ms      0x04    // 8 ms
#define systick32ms     0x05    // 32 ms

// Zyklusdauer:
#define stepmax 120         // Zyklusdauer, 120 ~ 30ms, 127 ~ 31,75ms, minimal 50

// PORTA
#define DAC0    0x0f
#define DAC1    0xf0

// PORTB
#define MOSI    (1<<PIN0)       // AVR-ISP Programmer
#define MISO    (1<<PIN1)       // AVR-ISP Programmer
#define SCK     (1<<PIN2)       // AVR-ISP Programmer
#define LED     (1<<PIN3)
#define JP4     (1<<PIN4)
#define JP5     (1<<PIN5)
#define JP6     (1<<PIN6)
#define nRESET  (1<<PIN7)       // AVR-ISP Programmer

// remember
#define bitset(PORT,BITNUM) ((PORT) |=  (1<<(BITNUM)))  // Set I/O bit;     OR      |=
#define bitclr(PORT,BITNUM) ((PORT) &=~ (1<<(BITNUM)))  // Clear I/O bit;   AND     &=
#define bittog(PORT,BITNUM) ((PORT) ^=  (1<<(BITNUM)))  // Toggle I/O bit;  XOR     ^=

// LED
void ledon(void)  { ((PORTB) |=  (LED));    }   // Set I/O bit;     OR      |=
void ledoff(void) { ((PORTB) &=~ (LED));    }   // Clear I/O bit;   AND     &=
void ledtog(void) { ((PORTB) ^=  (LED));    }   // Toggle I/O bit;  XOR     ^=

// globale Variablen
static uint8_t mode;
static uint8_t lo, hi, cnt;
static uint8_t steps;

/************ Inits ************************************************/

void initPorts(void) {      // Init data directions and ports
    DDRA  = DAC0|DAC1;      // Datenrichtung 1: output; 0: input; PORTA ist DAC
    //PORTA = 0x00;         // Portbits setzen (bei in = 1: pullup)
    DDRB  = LED;            // LED ist Output = 1, Rest Input = 0 (PB7 = /RESET)
    PORTB = JP4|JP5|JP6;    // PB4,5,6 pullup
};

void systicks(uint8_t prescal) {    // Init Timer0 Overflow-Interrupt
    // set prescaler TCCR0B 2:0 --> prescal = {1...5}
        //  value   bits    _bm     step        ovf after
        //  /1:     001     0x01    125ns        32µs
        //  /8:     010     0x02    1µs         256µs
        //  /64:    011     0x03    8µs          2,04ms
        //  /256:   100     0x04    32µs         8,16ms
        //  /1024:  101     0x05    0,128 ms    32,64ms
    TCCR0B |= prescal;  // set CS1 (/1024) 0,128*255 = 32,64ms
    // set TOIE1 overflow-interrupt enable
    TIMSK |= (1<<TOIE0);    // 0x02 = 0b0000 0010, Bit#1
};

void ledclk(uint8_t beats) {    // Init Timer1 Overflow-Interrupt
    // set prescaler TCCR1B 3:0
    TCCR1B |= 0b00001111;   // CS1 = 0x0f ~ /16384, p.117, beatfreq 488 Hz = 2,048ms
    //TCCR1B |= 0b00001110; // CS1 = 0x0e ~ /8192,  p.117, beatfreq 977 Hz = 1,024ms
    OCR1C = beats;          //
    // set TOIE1 overflow-interrupt enable
    TIMSK |= (1<<TOIE1);    // 0x04 = 0b0000 0100, Bit#2
};

void disableTimers(void) {
    TIMSK &=~ (1<<TOIE0);   // 0x02 = 0b0000 0010, Bit#1
    TIMSK &=~ (1<<TOIE1);   // 0x04 = 0b0000 0100, Bit#2
```

```
}

void enableTimers(void) {
    TIMSK |= (1<<TOIE0);    // 0x02 = 0b0000 0010, Bit#1
    TIMSK |= (1<<TOIE1);    // 0x04 = 0b0000 0100, Bit#2
}


/********************** Zeitfunktionen ****************************/

void invpuls(void) {        // inverse Pulse
    steps = 120;            // Anzahl der Schritte im Zyklus
    if (cnt < 16) {
        lo = cnt;
        hi = ((~lo) << 4);  // negieren und vier Bit links schieben
    }
    // DAC-Ausgabe erfolgt zentral
}

void treppen(void) {        // gegenläufige Pulse
    steps = 120;    // Anzahl der Schritte im Zyklus
    if (cnt < 16) {
        lo = cnt;
        hi = ((~lo) << 4);  // negieren und vier Bit links schieben
    }
    if (cnt > 15) {
        lo = 0;
    }
    // DAC-Ausgabe erfolgt zentral
}

void halbtreppe(void) {     // gegenläufige Pulse
    steps = 120;            // Anzahl der Schritte im Zyklus
    if (cnt < 8) {
        lo = cnt + 8;
        hi = ((~cnt) << 4); // negieren und vier Bit links schieben
    }
    if (cnt == 8) {
        lo = 0;
        hi = 0;
    }
    // DAC-Ausgabe erfolgt zentral
}

void saege(void) {          // gegenläufige Treppen
    steps = 32;             // Anzahl der Schritte im Zyklus
    lo = (cnt & DAC0);
    hi = ((~cnt << 4) & DAC1);
    // DAC-Ausgabe erfolgt zentral
}

void zaehne(void) {         // gegenläufig gezahnt
    steps = 32;             // Anzahl der Schritte im Zyklus
    if ((cnt & DAC0) < 16) {
        lo = (cnt & DAC0);
        hi = ((~cnt) << 4); // negieren und vier Bit links schieben
    }
    if (((cnt & 0x10) == 0x10)) {
        lo = ((~cnt) & DAC0);
        hi = ((cnt) << 4);  // negieren und vier Bit links schieben
    }
    // DAC-Ausgabe erfolgt zentral
}

void invpyr(void) {         // Pyramide und Inverse
    steps = 120;            // Anzahl der Schritte im Zyklus
    if (cnt < 16) {
        lo = (cnt & DAC0);
        hi = ((~lo) << 4);  // negieren und vier Bit links schieben
    }
    else {
        if (cnt < 32) {
            lo = (~cnt & DAC0);
```

```
            hi = ((~lo) << 4);  // negieren und vier Bit links schieben
        }
    }
    // DAC-Ausgabe erfolgt zentral
}

void rechteck(void) {       // verschobene Rechteckfunktionen
    steps = 120;            // Anzahl der Schritte im Zyklus
    if (cnt == 0) {
        lo = DAC0;
        hi = 0;
    }
    if (cnt == 8) {
        lo = DAC0;
        hi = DAC1;
    }
    if (cnt == 16) {
        lo = 0;
        hi = DAC1;
    }
    if (cnt == 24) {
        lo = 0;
        hi = 0;
    }
    // DAC-Ausgabe erfolgt zentral
}

void pyramids(void) {       // zwei Pyramiden
    steps = 120;

    if (cnt > 47 ) {        // Rest nullen
        lo = 0;
        hi = 0;
    }
    if (cnt < 48) {
        lo = 0;
        hi = ((~cnt) << 4); //
    }
    if (cnt < 32) {
        lo = ((~cnt) & DAC0);   // negieren
        hi = (cnt << 4);    // vier Bit links schieben
    }
    if (cnt < 16) {
        lo = (cnt & DAC0);  // maskieren
        hi = 0;
    }
}

uint8_t checkJumper(void) { // Jumper abfragen PB654
    // shift right 4 bit, PORTB bits 6,5,4 durchlassen
    return (~(PINB >> 4) & 0x07);   // results: bits {2,1,0}; return-values: 0...7
}

void cmdDecode(uint8_t mask) {  // mask-values {0...7}
    switch(mask) {
        case 0: {           // kein Jumper gesetzt "-", dabei sind PB654 high
            ledclk(50);     // LED-Frequenz in ms
            zaehne();       // XXX...XXXX
            break;
        }
        case 1: {           // Jumper PB4 gesetzt
            ledclk(250);
            pyramids();     // ___/X\___
            break;
        }
        case 2: {           // Jumper PB5 gesetzt
            ledclk(150);
            halbtreppe();   // ___|X|___
            break;
        }
        case 3: {           // Jumper PB4 und PB5 gesetzt
            ledclk(150);
            invpyr();       // ===XX====
```

```
            break;
        }
        case 4: {           // Jumper PB6 gesetzt
            ledclk(250);
            treppen();      // ___|X|___
            break;
        }
        case 5: {           // Jumper PB4 und PB6 gesetzt
            ledclk(150);
            rechteck();     // ____|=|-|=|____
            break;
        }
        case 6: {           // Jumper PB5 und PB6 gesetzt
            ledclk(100);
            saege();        // X|X|X|X|X|
            break;
        }
        case 7: {           // Jumper PB4, PB5, PB6 gesetzt, PB456 low
            ledclk(50);
            invpuls();      // ===|X====
        }
        default: {          // case 0
            mode = 0;
        }
    }
}


/********************** Hauptprogramm ******************************/

int main(void) {
    steps = stepmax;
    initPorts();
    ledon();
    mode = checkJumper();
    ledclk(255);                // beats mal zwei Millisekunden
    systicks(systick256us);     // Abtastfrequenz
    sei();

    while (1) {
        asm volatile ("nop");
        // hier passiert nichts mehr
        // den Rest machen die Interrupts
    }

    return 0;
}


/********************** Interrupts *********************************/


// Interrupt Timer0 Overflow
ISR(TIMER0_OVF_vect) {  // systick
    PORTA = (lo | hi);  // ohne Verzögerung den letzten Wert ausgeben
    cmdDecode(mode);    // nächsten Wert berechnen
    cnt++;              // Zähler permanent umlaufend
    if (cnt == steps) { cnt = 0;}       // Zykluszeit
}


ISR(TIMER1_OVF_vect) {  // LED toggle
    ledtog();
    mode = checkJumper();
}
```