BIN-Hourglass (ATtiny2313a) Modding

H39@email.de

Thema

Frage	Kann man das BIN-Hourglass mit anderen Funktionen versehen (Modding)?
Vorhanden	Aufgebautes BIN-Hourglass mit vorprogrammierten ATtiny2313a.
	Das BIN-Hourglass zeigt die Uhrzeit in binärer Form, in Stunden, Minuten und Sekunden an.
	Stunden und Minuten werden über Taster eingestellt.
Ideen	• Timer
	• Countdown
	• BIT-Spielereien
	• ...
Quelle	Datasheet ATtiny2313a: 8246B–AVR–09/11 https://ww1.microchip.com/downloads/en/DeviceDoc/doc8246.pdf
Pollin: Beschreibung, Software	https://www.pollin.de/p/bausatz-bin-hourglass-810399

Aufgaben

1.	BIN-Hourglass bauen.
2.	Arduino-ATtiny ISP-Schnittstelle.
3.	Arduino-IDE einrichten mit Boardverwalter.
4.	Arduino in einen ISP-Programmer verwandeln.
5.	BIN-Hourglass Schaltung verstehen.
6.	ATtiny und BIN-Hourglass Pin-Zuordnung verstehen.
7.	Adapter anfertigen oder µC Standalone.
8.	BIN-Hourglass Programm von Pollin verstehen.
9.	ATtiny Oszillator vs. CPU-Takt.
10.	Mit Arduino-IDE programmieren (BlinkLED.ino).
11.	Sketch mit Programmer Hochladen.
12.	16-bit Timer für Sekundentakt einrichten.
13.	Externen Interrupt abfragen (Taster).
14.	Oszillator, CPU-Takt und Fuses verstehen.

BIN-Hourglass

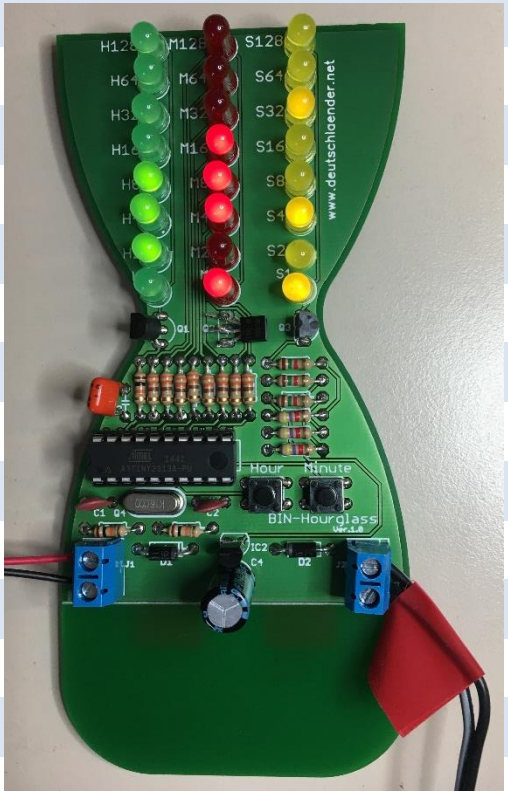
Mikroprozessor: ATtiny2313a

Drei Reihen von LEDs mit je 8 LEDs.

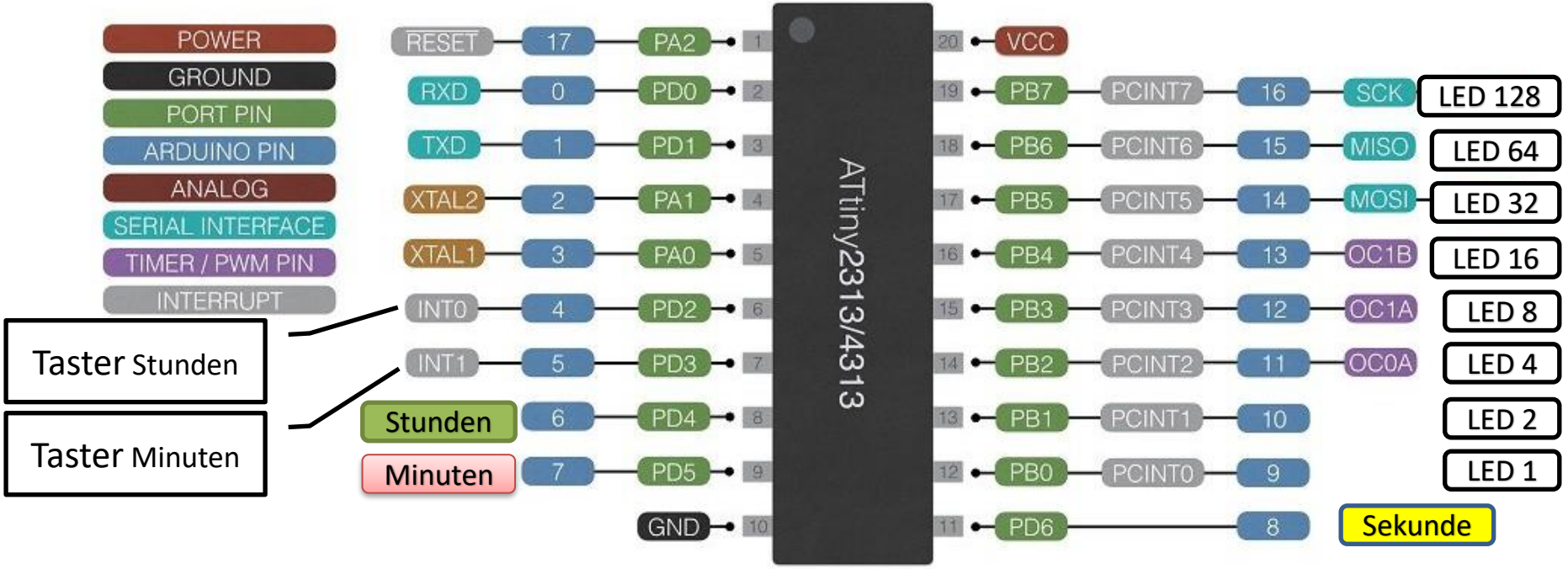
Zwei Taster zum Einstellen von Stunden und Minuten.

Externer Quarz mit 16 MHz.

Stromversorgung 9V-Blockbatterie.

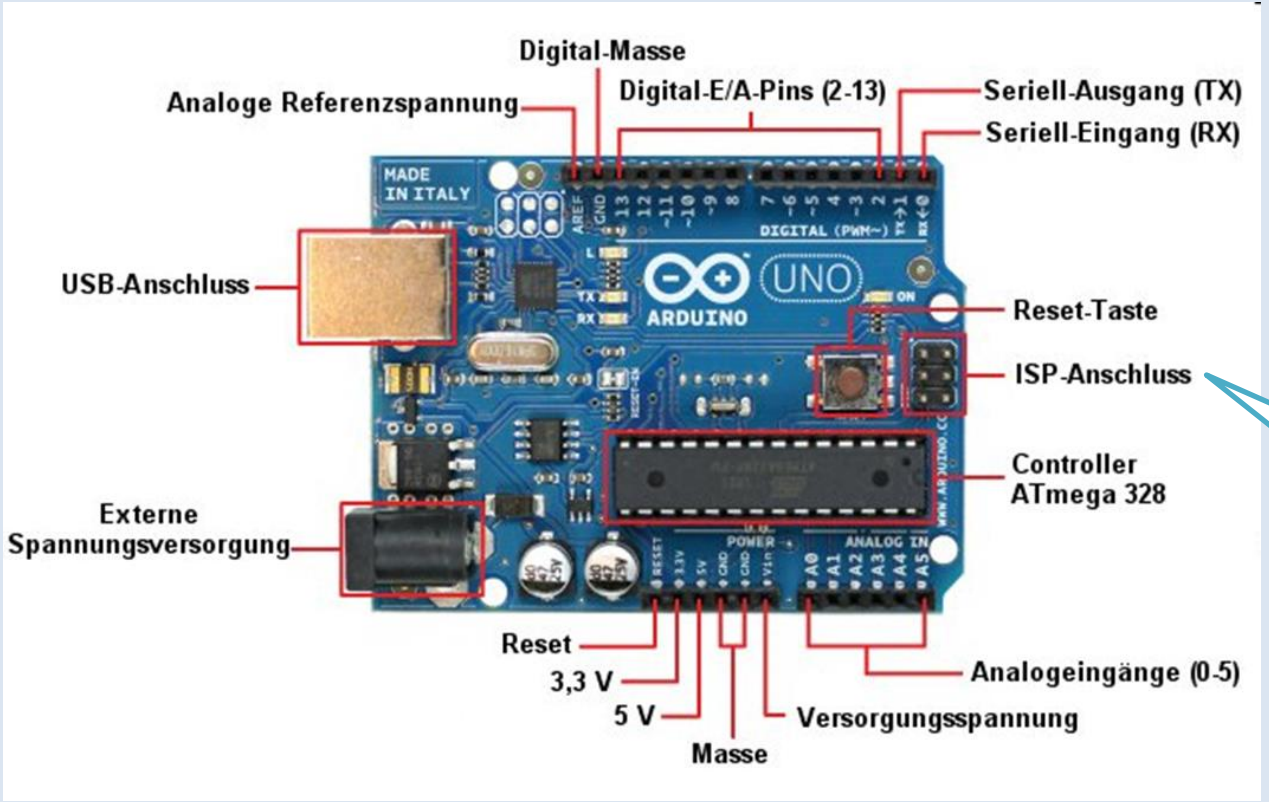


ATtiny2313a Pinout

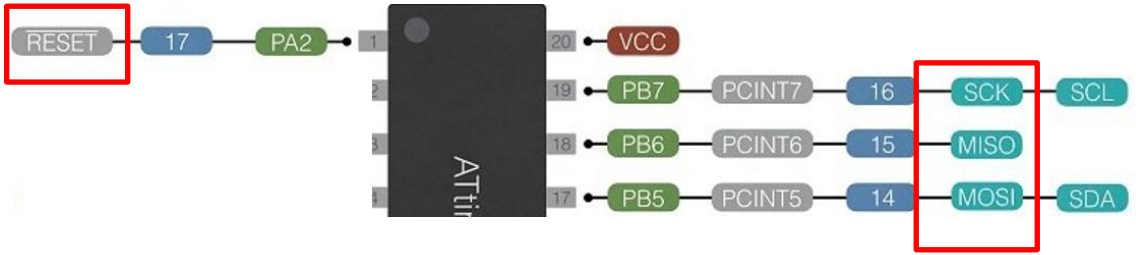


http://github.com/SpenceKonde/ATTinyCore

Arduino UNO


Digital-Masse
Analoge Referenzspannung
Digital-E/A-Pins (2-13)
Seriell-Ausgang (TX)
Seriell-Eingang (RX)
USB-Anschluss
Reset-Taste
ISP-Anschluss
Controller
ATmega 328
Externe
Spannungsversorgung
Reset
3,3 V
5 V
Masse
Versorgungsspannung
Analogeingänge (0-5)
Wird
nicht
genutzt!

ATtiny ISP-Pins

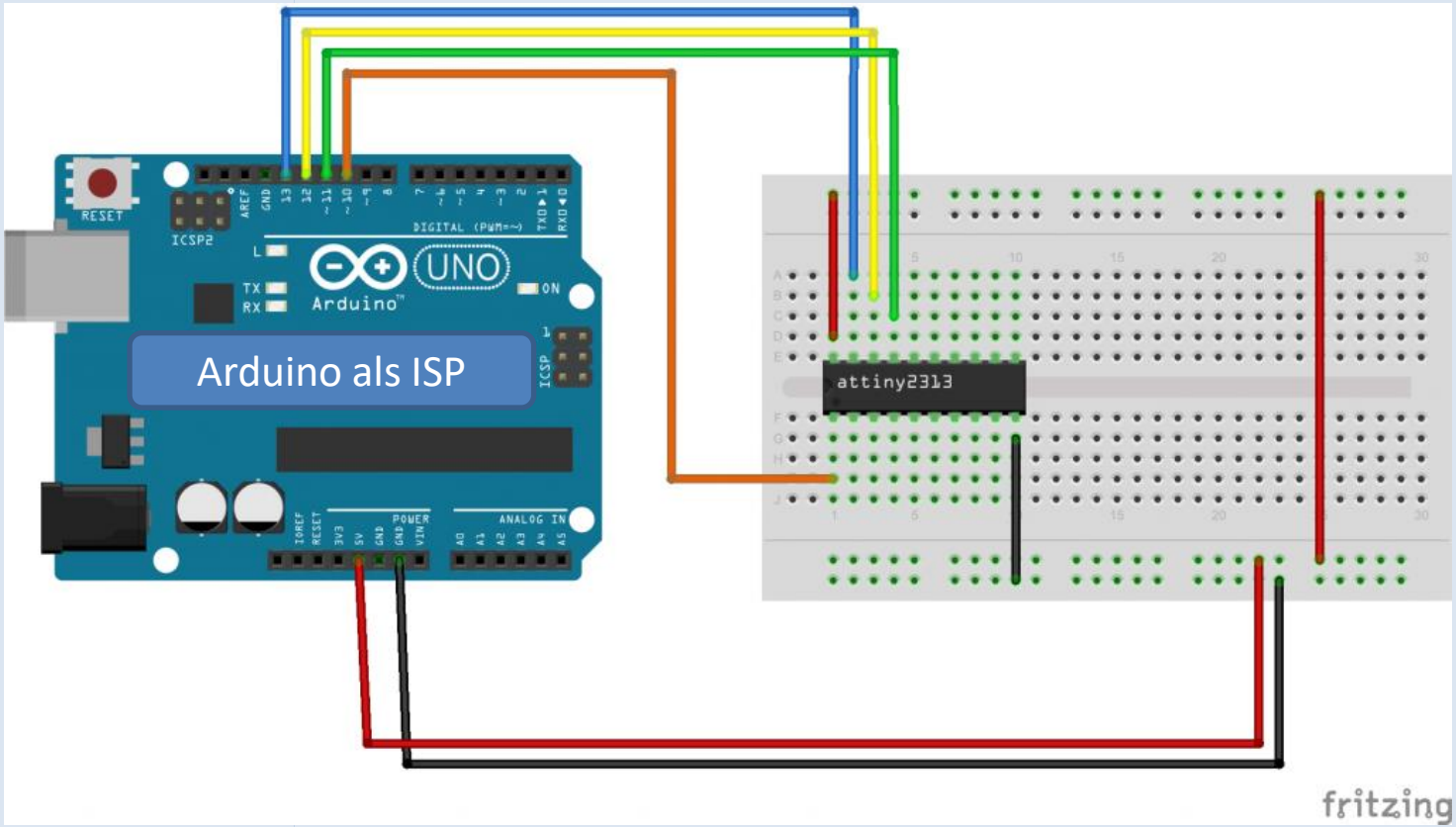
ISP	In-circuit Serial Programmer.
ISP-Pins am zu programmierenden ATtiny2313a:	

Verbinde den ATtiny2313 mit den Arduino wie folgt:

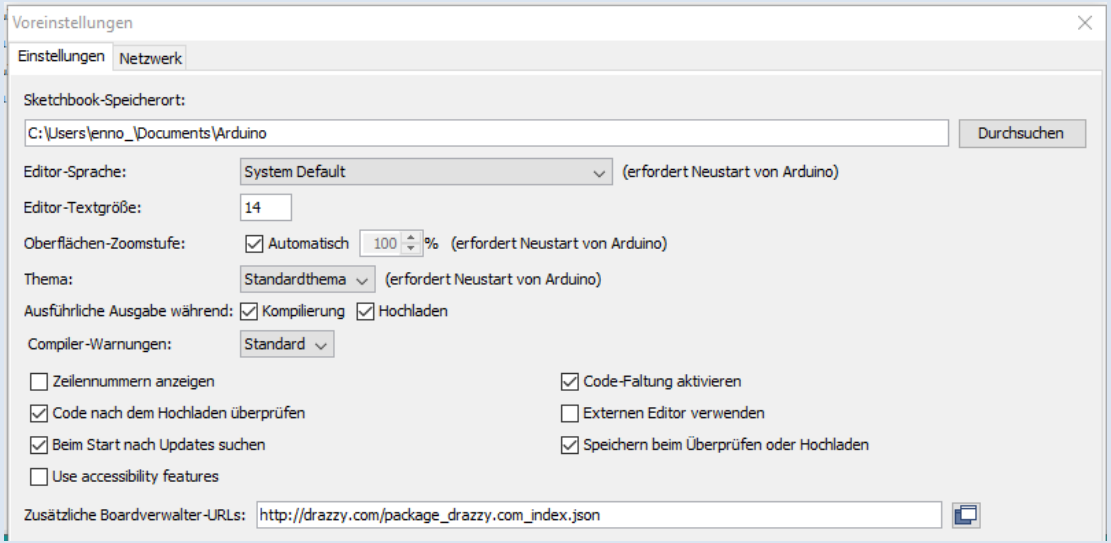
1.	ATtiny2313 µC-Pin	1	to	Arduino-Pin	10 reset
2.	ATtiny2313 µC-Pin	17	to	Arduino-Pin	11 MOSI
3.	ATtiny2313 µC-Pin	18	to	Arduino-Pin	12 MISO
4.	ATtiny2313 µC-Pin	19	to	Arduino-Pin	13 SCK
5.	ATtiny2313 µC-Pin	10	to	Arduino-Pin	GND
6.	ATtiny2313 µC-Pin	20	to	Arduino-Pin	VCC

Quelle:	https://www.arduino.cc/en/pmwiki.php?n=Tutorial/ArduinoISP http://arduinolearning.com/amp/code/program-attiny2313-arduino.php

Arduino an ATtiny2313

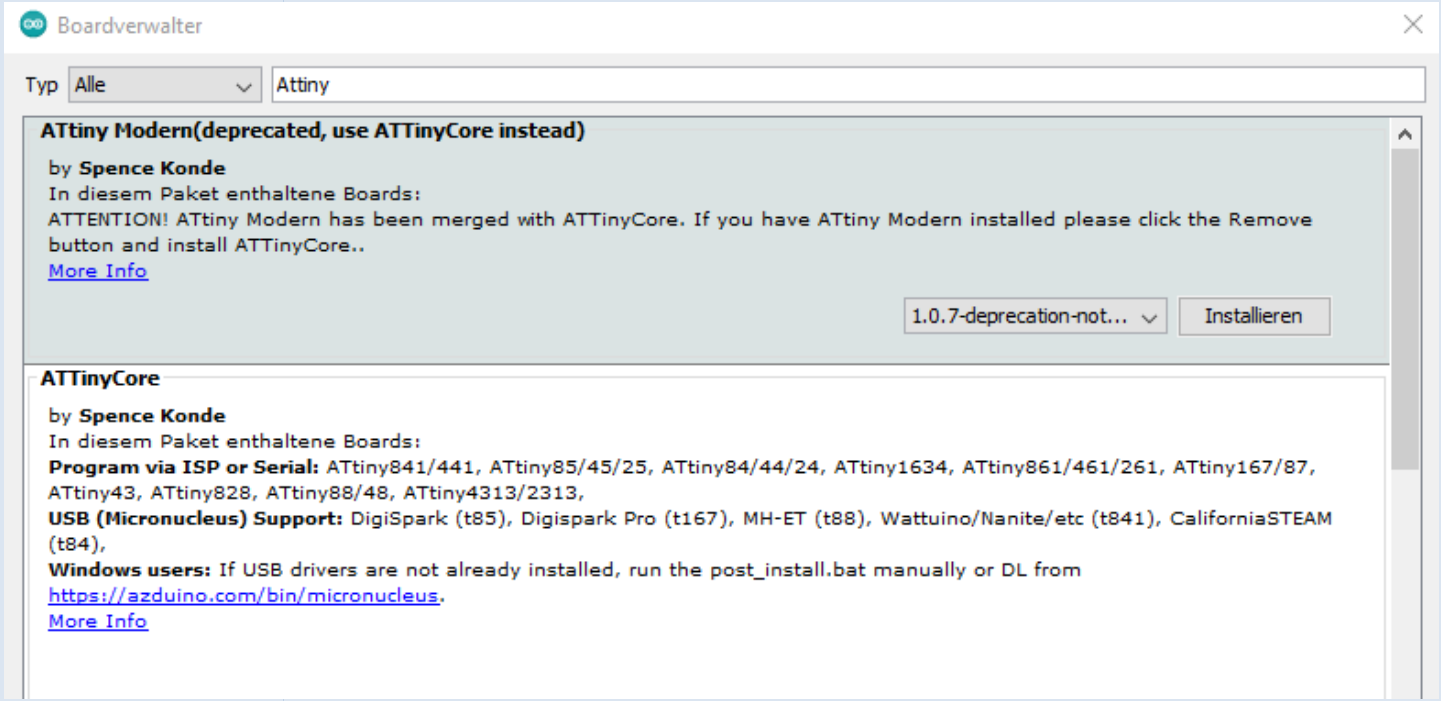


Arduino-IDE ATtiny2313a

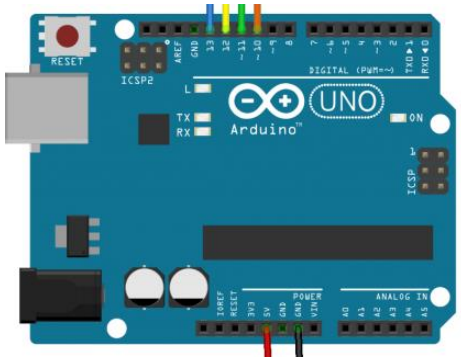
Arduino-IDE	Arduino-1.8.15
IDE portabel machen	Ordner „portable“ erzeugen: „<Dein Pfad>\arduino-1.8.15\portable“
Arduino-IDE	> Datei > Voreinstellungen > Zusätzliche Boardverwalter-URLs:
Boardverwalter hinzufügen:	http://drazzy.com/package_drazzy.com_index.json
	
Quelle	https://github.com/SpenceKonde/ATTinyCore/blob/master/Installation.md

Boardverwalter

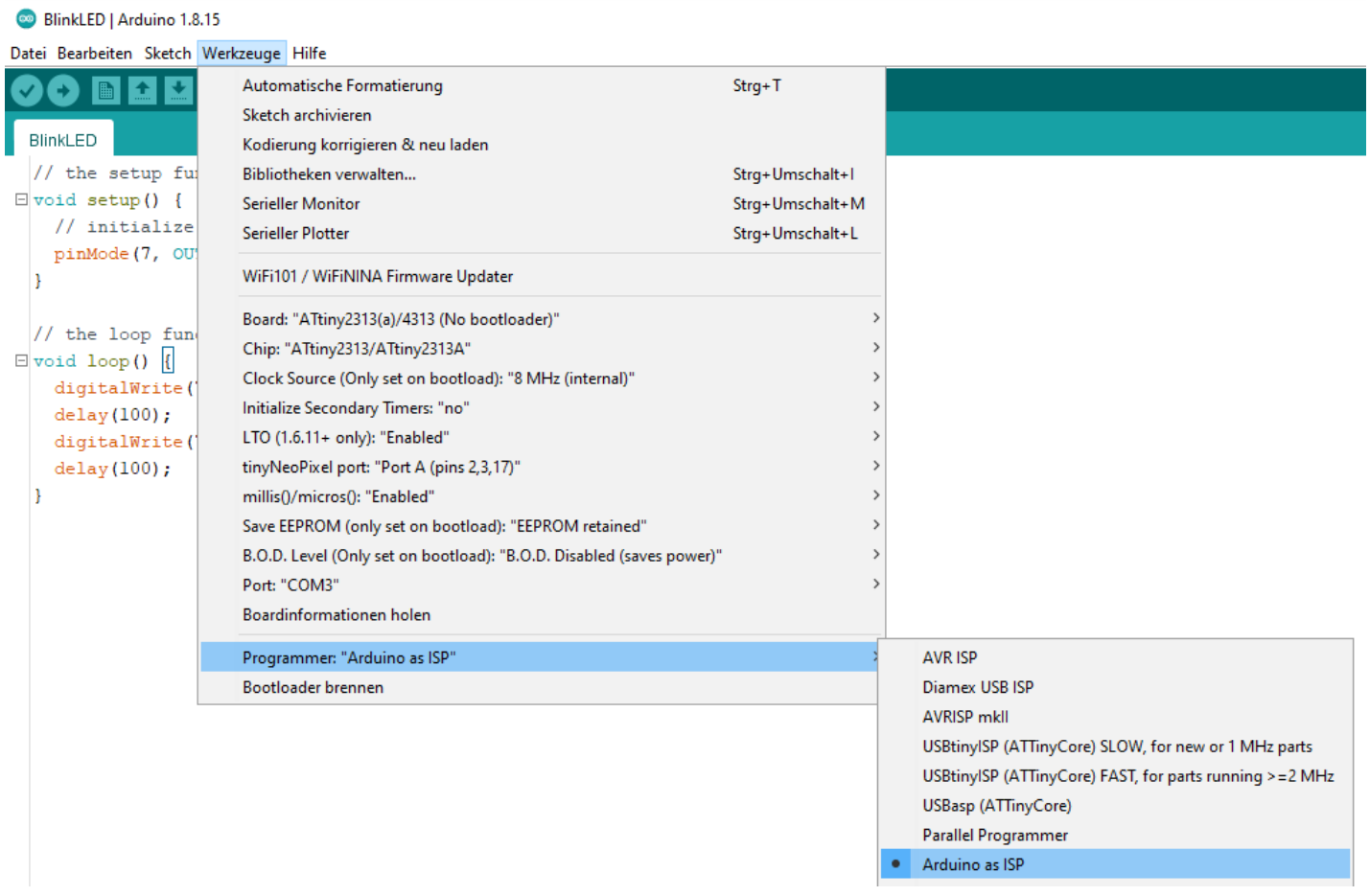
Boardverwalter starten:	Werkzeuge > Boards > Boardverwalter
	Nach „Attiny“ suchen. „ATTinyCore by Spence Konde“ installieren.



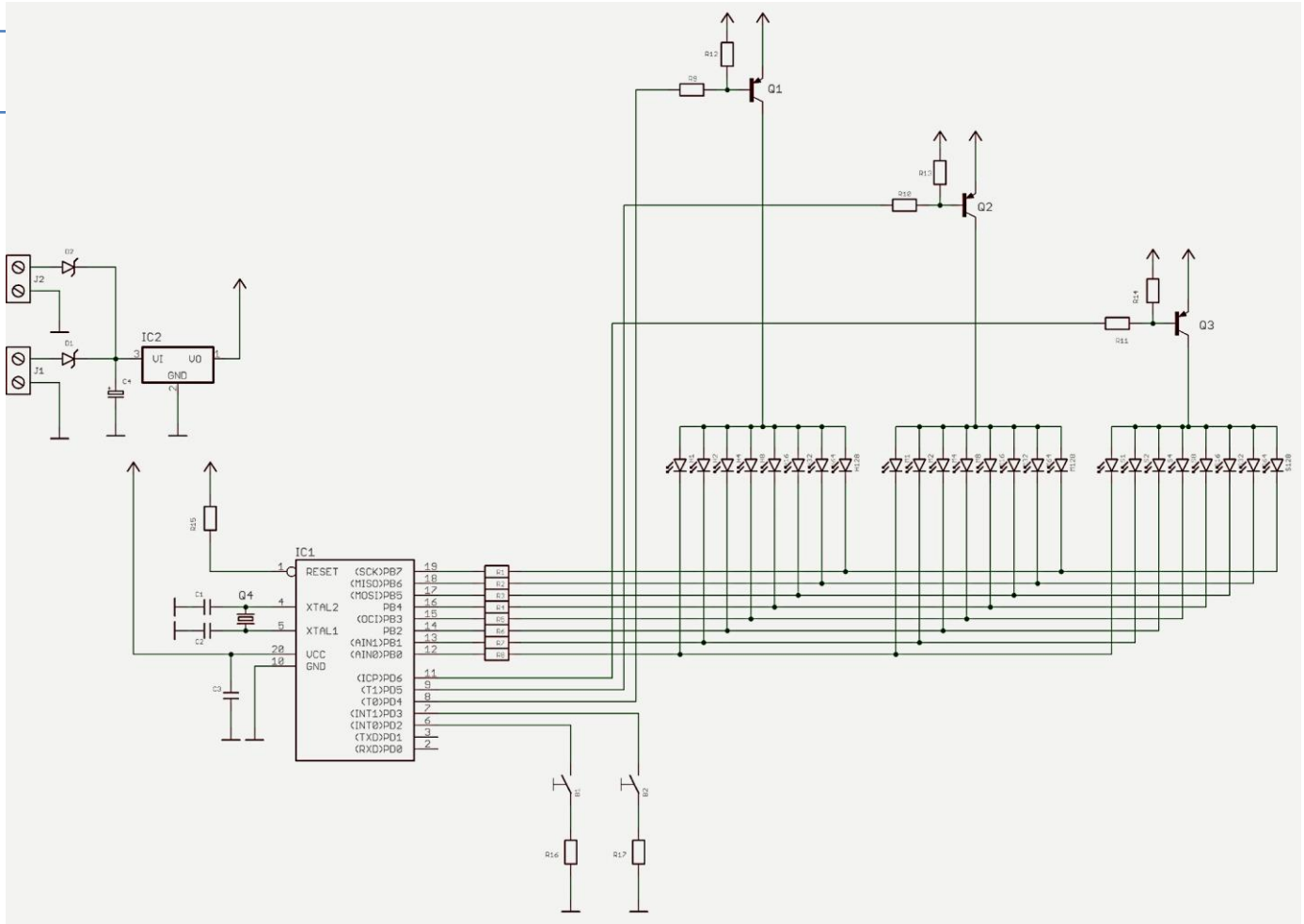
Arduino als ISP-Programmer

ISP	**I**n-circuit **S**erial **P**rogrammer.
Bootloader	**Der ATtiny2313a kommt ohne Bootloader.**
ISP-Programmer	Der Sketch „ArduinoISP.ino“ macht aus dem Arduino einen ISP-Programmer.
Arduino UNO Pins als ISP-Programmierer	
Arduino ISP-Sketch laden	> Datei > Beispiele > 11.ArduinoISP > ArduinoISP.ino
Werkzeuge	Als Board „Arduino UNO“ (noch nicht der ATtiny2313a) wählen. Als Programmer noch „AVRISP mkII“.
	Sketch „ArduinoISP.ino“ auf Arduino (später ISP-Programmer) hochladen.
Quellen:	https://www.arduino.cc/en/pmwiki.php?n=Tutorial/ArduinoISP http://arduinolearning.com/amp/code/program-attiny2313-arduino.php

ISP aktivieren

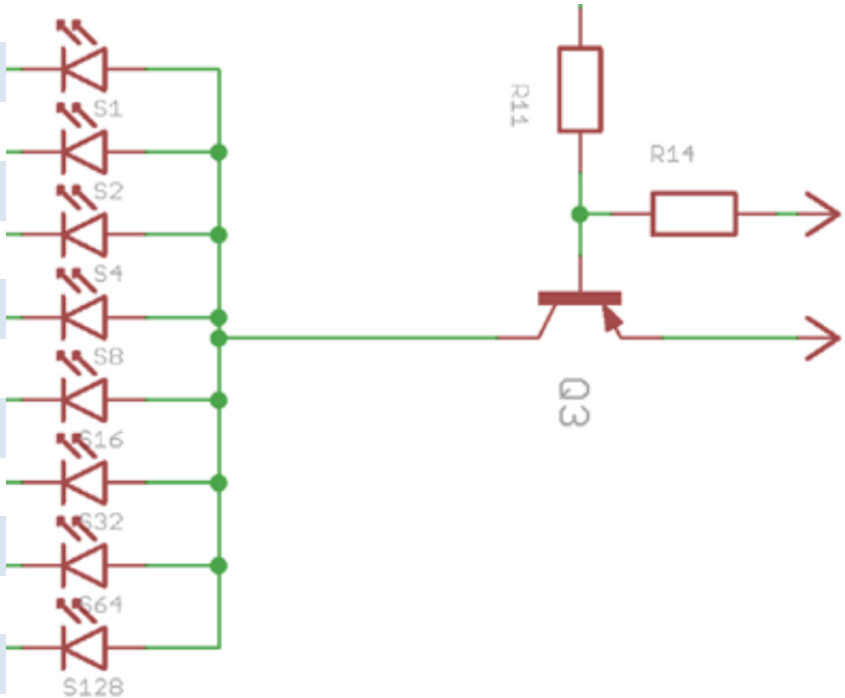


Schaltplan


Q1
Q2
Q3
IC1
IC2
RESET
XTAL2
XTAL1
VCC
GND
(SCK)PB7
(MISO)PB6
(MOSI)PB5
PB4
(OC1)PB3
PB2
(AIN1)PB1
(AIN0)PB0
(ICP)PD6
(T1)PD5
(T0)PD4
(INT1)PD3
(INT0)PD2
(TXD)PD1
(RXD)PD0
VI
VO
GND

LED-Ansteuerung

	µC 11 (PD6) auf LOW (PNP) macht Spalte Sekunden aktiv.
Masse auf LOW => LED an.	Basis auf LOW
µC 12 (PB0) hier LED S1	
in Kombination mit µC 11 (PD6)	
	+5 V
	+5 V (Stromkreis geschl.)
	R14 = 1,5 kΩ
	R11 = 4,7 kΩ
	Transistor BC557B



Taster-Schaltung

Logik:

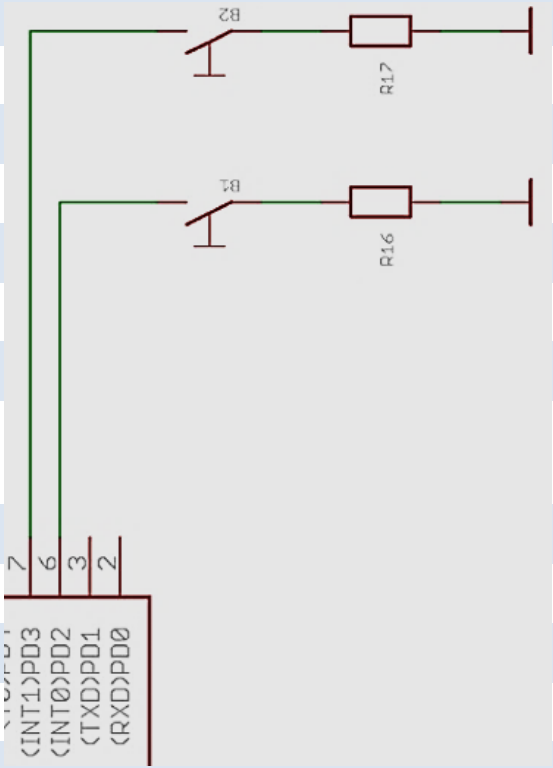
Wenn der Taster gedrückt wird,

wird der µC-Eingang LOW.

Programmierung:

Man muss auf eine fallende Flanke reagieren (Falling edge).

Die Port-Pins PD2 und PD3 **müssen** durch Setzen von Pullup auf 5 V gezogen werden.



µC-Anschlüsse

Die µC-Pins sind die Kontakte am Prozessor!

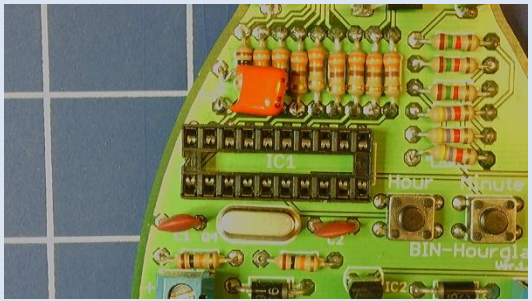
µC-Pin	Port-Pin	Arduino-IDE	Funktion
6	PD2 INT0	4	Taster Stunden
7	PD3 INT1	5	Taster Minuten
8	PD4	6	H Stunden
9	PD5	7	M Minuten
11	PD6	8	S Sekunden
12	PB0	9	LED 1 (H1 / M1 / S1)
…		…	
19	PB7	15	LED 128 (H128 / M128 / S128)

Die Arduino-IDE Pin-Bezeichnungen werden für „digitalWrite()“ etc benötigt.

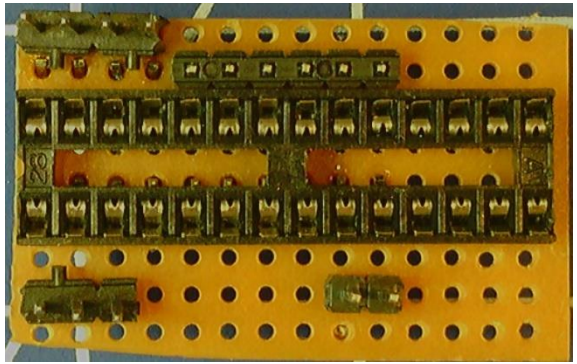
Adapter bauen I

Ziel	Der Programmieradapter soll das Programmieren direkt am BIN-Hourglass ermöglichen.

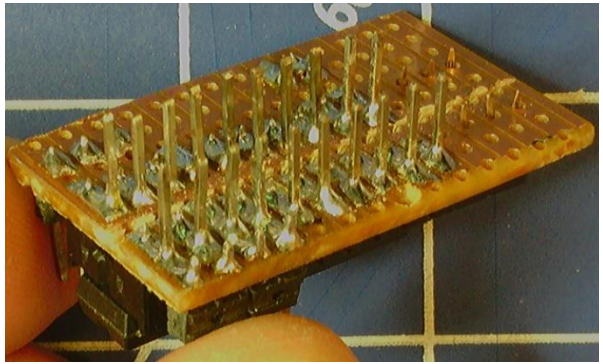
BIN-Hourglass ohne µC



Adapter Oberseite

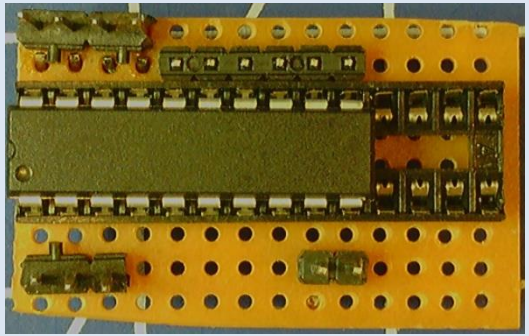


Adapter Unterseite

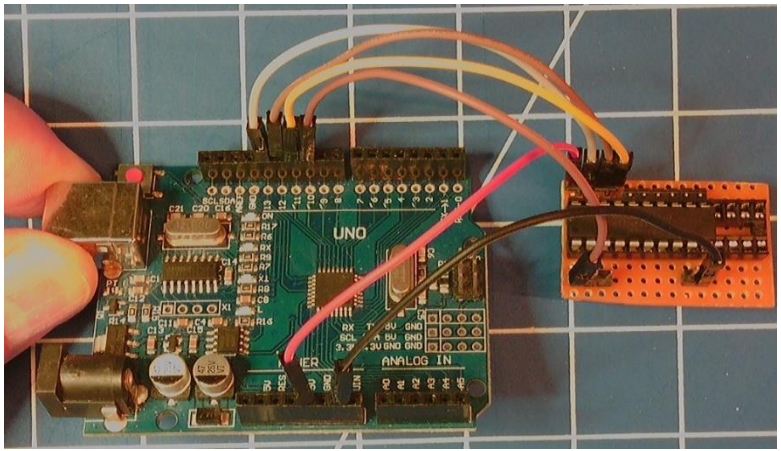


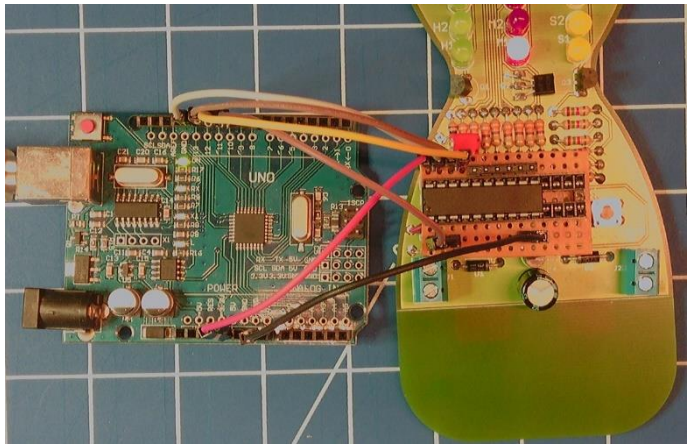
Adapter bauen II

Adapter mit µC



Adapter und Verkabelung





HourGlass Quellcode von Pollin

Besonderheit:	Nicht mit Arduino-IDE Funktionen geschrieben, sondern in AVR C++.
	Die „OurGlass.ino“ ist nur ein Dummy, damit man mit der Arduino-IDE programmieren kann.
Tab: Main.c c	Ein AVR C++ Programm, das die „main()-Funktion“ enthält.
Tab: binTime.h	Ein AVR C++ Programm mit der Erweiterung „*.h“, und damit als Header gekennzeichnet. Der Header enthält –ausgelagerte- Definitionen und Funktionen, die beim Kompilieren vor der main()-Funktion eingefügt werden.
Tab: setTime.h	siehe oben
Nachteile:	Die Funktionen der Arduino-IDE stehen (zunächst) nicht zur Verfügung. Also pinMode(), digitalWrite(), digitalRead() sind nicht nutzbar. (man könnte die „Arduino.h“ einbinden).
Vorteile:	Die Arduino-IDE setzt z.B. ungewollt Timer. Hier erlangt man vollkommene Kontrolle über das was der µC ausführen soll.
Download Pollin:	https://www.pollin.de/p/bausatz-bin-hourglass-810399
Autor:	Leonhard Hesse

ATtiny Oszillator vs. CPU-Takt

Lieferzustand ATtiny2313a	The device is shipped with CKSEL = “0100”, SUT = “10”, and CKDIV8 programmed. The default clock source setting is the Internal RC Oscillator with **longest start-up time** and an initial system clock **prescaling of 8**, resulting in **1.0 MHz system clock**. This default setting ensures that all users can make their desired clock source setting using an In-System or Parallel programmer.
	Interner Oszillator-Takt: 8 MHz Takt-Vorteiler: 8 System Clock: 8/8 => 1 MHz
BIN-Hourglass	
Externer Oszillator	Externer Oszillator-Takt: 16 MHz Takt-Vorteiler: 8 System Clock: 16/8 => 2 MHz
System Clock Frequency für Timer-Berechnungen	System Clock: 2 MHz
Ändern der System Clock	Der µC kann mit Hilfe sogenannter Fuses umkonfiguriert werden.
Fuses berechnen mit:	http://elektronik-kompendium.de/public/arnerossius/programme/web/avrfuse/

ATtiny2313a delay() Problem

Problem	Ein delay(1000) führt zu einem zu schnellem Blinkintervall.
Ursache	Der ATtiny2313a taktet nicht mit dem Oszillator-Takt, sondern wegen der „Divide clock by 8“ Konfiguration mit einer „System clock“.
ATtiny2313a	Entsprechend der Fuses wird der Oszillator-Takt mit 8 geteilt, so dass die System clock = 16 MHz/8 => 2 MHz beträgt (externer Oszillator 16 MHz)
Arduino-Funktion	
8-bit Timer	Die delay()–Funktion basiert auf dem automatisch gestarteten 8-bit Timer0.
	Mit einem voreingestellten Timer0-Prescaler von 64.
Folge	Nun erzeugt ein „delay(1000)“ ein Blinkinterval von 1000/8 = 125 ms.
Abhilfe	Den voreingestellten Timer0-Prescaler von 64 auf 8 ändern.
In setup() ergänzen:	// Patch for 8-bit Timer, sets prescaler $clk_{I/O}$ devided by 8, leads to clk_{Tn}
	`TCCR0B = (0 << CS02) | (1 << CS01) | (0 << CS00);`
Quelle:	ATtiny2313A/4313 Data Sheet (microchip.com)

BlinkLED.ino

```
// BIN-Hourglass Modding
// Blinking LED M1 (BlinkLED.ino)
// In order to select column "M", set the Pin to LOW or "0"
// LOW drives the PNP-transistor, so we get C
// Column Minute  Arduino-IDE-Pin => 7
//                Port-Pin        => PD5 set to LOW or binary "0"
//                µP-Pin          => 9
// In order to select LED "1", set the Pin to LOW or "0", so we get (GND)
// LED              |  1  |
// Arduino-IDE-Pin  |  9   |
// Port-Pin         |  PB0 |
// µP-Pin           |  12  |

#define Column 7               // column minutes
#define PinLED 9               // LED 1 (H1 / M1 / S1)

void setup() {
  // initialize digital Pins as an output
  PinMode(PinLED, OUTPUT);        // sets ATtiny-Pins as OUTPUT
  PinMode(Column, OUTPUT);
  digitalWrite(Column, LOW);      // because of PNP-transistor, the base needs to be LOW
                                  // in order to get led on, thus we get positive voltage
  // Patch for 8-bit Timer, sets prescaler clkI/O devided by 8
  TCCR0B = (0 << CS02) | (1 << CS01) | (0 << CS00);
}
void loop() {
  digitalWrite(PinLED, LOW);     // connects LED to GND, means LED on
  delay(100);                    // wait for 1/100 second
  digitalWrite(PinLED, HIGH);    // disconnects LED from GND, that means LED off
  delay(100);                    // wait for 1/100 second
}
```

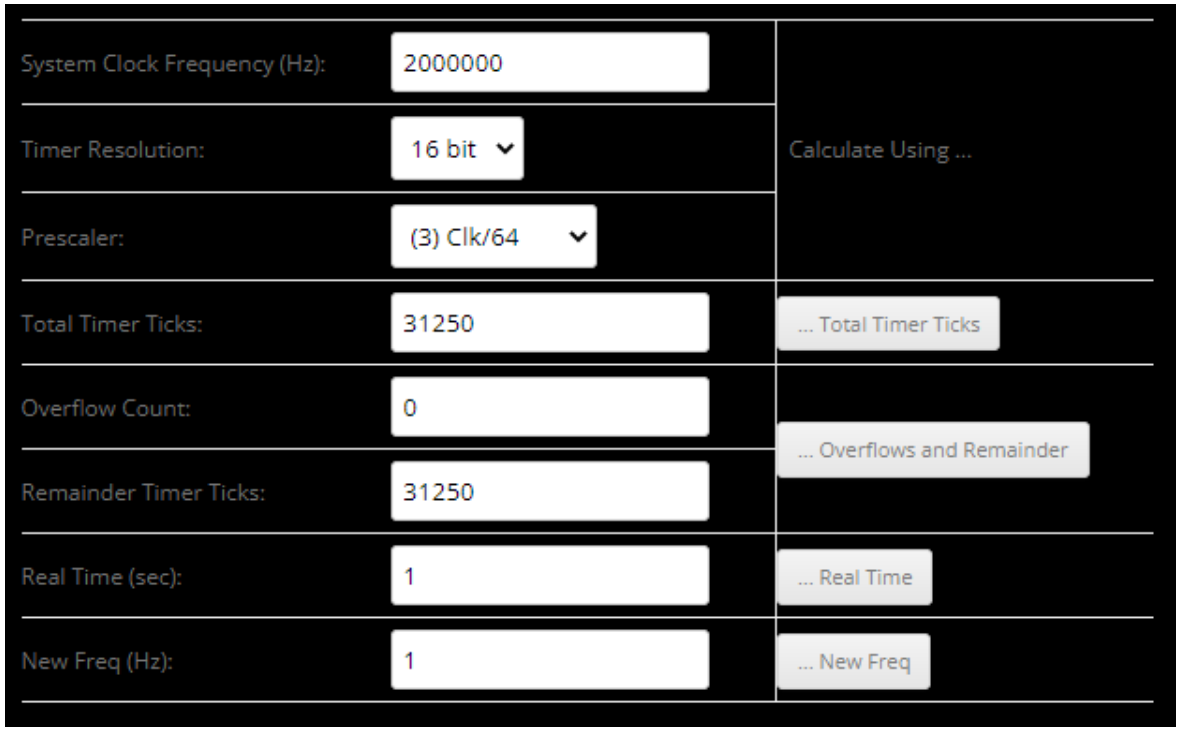
Hochladen

Einstellungen	> Werkzeuge
Board:	> Werkzeuge > Board > ATtinyCore > ATtiny2313(a)/4313 (No bootloader)
Chip:	> Werkzeuge > Chip > ATtiny2313/ATtiny2313A
Clocksource:	> Werkzeuge > Clock Source… > 8 MHz
Port:	> Werkzeuge > Port > COMx (Dein USB-Anschluss)
Programmer	> Werkzeuge > Programmer > **Arduino as ISP**
Sketch	Sketch schreiben.
Hochladen	Symbol „Hochladen“. Nicht „Hochladen mit Programmer“!

Timer-Calculator

Probieren!

Prescaler so wählen, dass kein Overflow Count eintritt.
Overflow Count muss 0 sein.



Timer berechnen mit: https://eleccelerator.com/avr-timer-calculator/

16-bit Timer

Ziel	In einem Takt von 1 Sekunde soll eine Funktion ausgeführt werden, die z.B. eine LED umschaltet.
CTC Clear Timer on Compare Match:	Timer im CTC-Modus. Der Timer soll den Interrupt auslösen und wieder bei 0 anfangen zu zählen. Gewünschten Endwert in das OCR0A Register speichern.
16-bit Timer/Counter1	
TCNT1	Zählerwert (am Anfang 0, wird hochgezählt)
OCR1A	Endwert (Vergleichswert), bis zu dem gezählt werden soll.
TCCR1B	Vorteiler (Prescaler), geht jetzt von der „System Clock“ aus (Hier 2 MHz).
TIMSK	Interrupt aktivieren.
Timer berechnen mit:	https://eleccelerator.com/avr-timer-calculator/
Quellen:	ATtiny2313A/4313 Data Sheet (microchip.com) Timer Interrupts - Arduino - www.simsso.de

Timer_0.ino

```
// vollständiger Sketch siehe Timer_0.ino
#define OCRIA_    31250      // external oszillator with 16 MHz and a predifined devider of 8
                             // the System Clock runs with 16/8 = 2 MHz.
// …                          .

void setup() {
  // …
  initTimer();               // init Timer 1
}

void loop() {
  if (int1_occured){
    PORTB ^= (1 << PB0);                    // toggle PB0
    int1_occured = false;
  }
}

// timer interrupt service routine
ISR(TIMER1_COMPA_vect) { int1_occured = true; }

void initTimer(){                          // for Timer1
cli();                                      // deactivate all interrupts
  TCNT1 = 0;                                // set counter to 0
  OCR1A = OCRIA_;                           // set Output Compare Register 1A
  TCCR1B  = (1 << CS10) | (1 << CS11);      // prescale 64
  TCCR1B |= (1 << WGM12);                   // CTC mode; resets TCNT1 at overflow
  TIMSK   = (1 << OCIE1A);                  // Compare A Match Interrupt Enable
  sei();                                    // activate all interrupts
}
```

Taster

Ziel	Ein Taster soll eine LED an- bzw. ausschalten.
Arduino-IDE Funktion	attachInterrupt()
Syntax	attachInterrupt(digitalPinToInterrupt(Pin), ISR, mode);
digitalPinToInterrupt(Pin)	Der externe Interrupt-Pin. Hier: Arduino-Pin 5, was dem externem Interrupt INT1 (Minute) entspricht.
ISR	Name der Funktion, die beim Auftreten des Interrupt ausgeführt wird. Hier: *int1_routine()*
mode	Fallende Flanke des Signals. Hier: FALLING
Befehl	*attachInterrupt(digitalPinToInterrupt(5), int1_routine, FALLING);*
Externer Interrupt	http://shelvin.de/eine-taste-per-interrupt-einlesen-und-entprellen/ Arduino - AttachInterrupt
Quellen:	ATtiny2313A/4313 Data Sheet (microchip.com)

ButtonIntLed_0.ino

```
// vollständiger Sketch siehe ButtonIntLED.ino
// …
volatile boolean int1_occured = false;     // we got an interrupt, do something in loop()
volatile unsigned long lastTime = 0;
volatile unsigned long debounceTime = 20;

void setup() {
// …
                                          // defining ISR for PinButton and falling edge
  attachInterrupt(digitalPinToInterrupt(PinButton), int1_routine, FALLING);

// …
}

void loop() {
  if (int1_occured){
    digitalWrite(PinLED, !digitalRead(PinLED));     // get status of LED and toggle it
    int1_occured = false;
  }
}

// Interrupt service routine (ISR) for interrupt "INT1"
void int1_routine() {
  if((millis() - lastTime) > debounceTime) {        // do nothing during debouncing time
    lastTime = millis();                            // store last buttontick
    int1_occured = true;                            // set message for loop()
  }
}
```

1 2 3 4

Fuses berechnen

Parameter auswählen:			
Externer Oszillator-Takt: 16 MHz Takt-Vorteiler: 8	Clock Source: 8-20 MHz Start-up Time: slowly rising power ☐ Output clock on CKOUT (PB2) ☑ *Divide clock by 8* *	**Low** 0x7F
SPI aktivieren (sonst geht nichts mehr)	☐ Disable external Reset Brown-out Detection: disabled * ☐ Watchdog Timer always on ☑ *Enable SPI programming* * ☑ Preserve EEPROM through Chip Erase ☐ Enable debugWIRE	**High** 0x9F
	☐ Enable Self Programming	**Extended** 0xFF
Avrdude-Parameter:	-U lfuse:w:0x7F:m -U hfuse:w:0x9F:m -U efuse:w:0xff:m		
Stapelprozedur:	Fuse.cmd		
Fuses berechnen mit:	http://elektronik-kompendium.de/public/arnerossius/programme/web/avrfuse/		

Projekt für wen?

Anfänger:	Bedingt geeignet! Das Schalten der LEDs auf „an“ geschieht durch Setzen des Pin auf LOW, anstelle von erwarteter Weise, HIGH. Es kann mit der Arduino-IDE programmiert werden. Man kann in Sketchen (*.ino) programmieren.
	Übungen auf der Ebene der Arduino-IDE mit Arduino-IDE Funktionen sind machbar.
Anwender mit Erfahrungen:	Geeignet! Es kann mit der Arduino-IDE programmiert werden. Man kann in Sketchen (*.ino) programmieren. Port-Manipulationen sind auch in Sketchen möglich.
	Ideen zur Erzeugung von Bitmustern…
Fortgeschrittene:	Gut geeignet! Es kann mit der Arduino-IDE programmiert werden. Man kann in Sketchen (*.ino) programmieren. Port-Manipulationen um Interrupts (Taster) und Timer zu programmieren.
	Ideen, die Tasten und Timer erfordern.

Liste der Sketche

BlinkLED.ino	Blinken einer LED mit Arduino-Funtionen:	*PinMode(PinLED, OUTPUT);*
Muster_0.ino	Bit-Muster mit Port-Manipulation:	*DDRD = 0b01110000*
Muster_1.ino	Bit-Muster mit Port-Manipulation:	*DDRD = 0b01110000*
ButtonIntLED_0.ino	Taste abfragen mit Arduino Funktionen	*attachInterrupt(…)*
ButtonIntLED_1.ino	Taste abfragen mit AVR-Registern	*MCUCR |= (1 << ISC11) | (0 << ISC10);*
Timer_0.ino	Erzeugt einen sekündlichen Funktionsaufruf.	