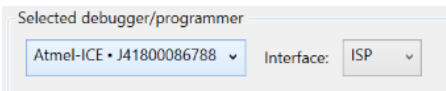


## Lösung 11: SW Download, Debugging

### Aufgabe 1: Vorbereitung

- a)
- b)
- c) Der Code zu Beginn der Main-Routine entspricht der setup(.)-Methode. Der Code innerhalb der „while(1)“-Schleife entspricht der loop(.)-Methode.

### Aufgabe 2: In-System Programming (ISP) über SPI

- a)
  - b)
  - c)
- 
- d)
  - e) Bei jedem Tastendruck sollte sich der Zählerwert verändert. Da keine Entprellung verwendet wurde, führt ein Tastendruck in der Regel dazu, dass der Zähler um mehr als 1 erhöht wird.

f)

Tool	Device	Interface	Device signature	Target Voltage
Atmel-ICE	ATmega2560	ISP	0x1E9801	4.9 V

Buttons: Apply, Read, Read, Settings

- g) Die HEX-Datei kann man unter „Memories“ im „Device Programming“ Dialog auslesen. Einfach in eine beliebige Datei auf dem PC speichern. Die Ausgabe könnte dann z.B. wie folgt aussehen. Das 2. und 3. Zeichen in jeder Zeile gibt an wie viele Datenbytes übertragen werden bevor eine Checksumme eingefügt wird. Im konkreten Beispiele (siehe unten) ist das 2. und 3. Zeichen: „10“, d.h. die Checksumme wird jeweils über 16 Byte berechnet.  
:1000000072C00000C2C000009CC000009AC0000086  
:1000100098C0000096C0000094C0000092C000008C  
:1000200090C000008EC000008CC000008AC000009C  
:1000300088C0000086C0000084C0000082C00000AC
- h)

e) Die HEX-Datei kann man unter „Memories“ im „Device Programming“ Dialog auslesen. Einfach in ein beliebiges Zeichen auf dem PC speichern. Die Ausgabe könnte dann z.B. wie folgt aussehen. Das 2. und 3. Zeichen in jeder Zeile gibt an wie viele Datenbytes übertragen werden bevor eine Checksumme eingefügt wird. Im konkreten Fall ist das 2. und 3. Zeichen: „10“, d.h. nach 16 Byte kommt jeweils eine Checksumme.

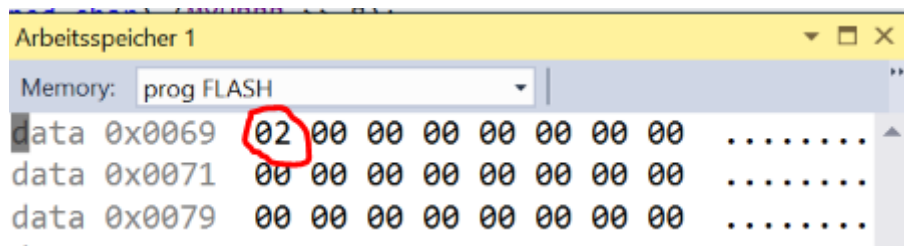
### Aufgabe 3: Debugging über eine Simulation

- c) Der Inhalt des Registers ist 0x01. Das 1. Bit wurden ja gerade eben durch das sei() Kommando auf 1 gesetzt.

```

20 int main(void)
21 {
22     // configure serial port / UART
23     UBRR0L = (unsigned char) MYUBRR;           // set baud rate
24     UBRR0H = (unsigned char) (MYUBRR >> 8);
25     UCSR0B = (1<<TXEN0);                       // enable transmitter
26     UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);         // set frame format: 8 data bits, 1 stop
27
28     // configure external interrupt INT0 / PD1
29     EIMSK |= (1 << INT0);                       // turn on INT0
30     EICRA |= (1 << ISC00) | (1 << ISC01);         // set INT0 to trigger on any rising edge
31     sei();                                       // globally activate interrupts in SREG (
32

```

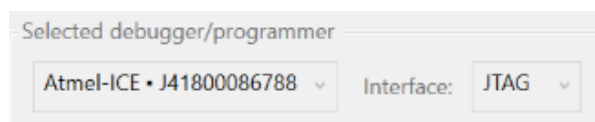


d) Man landet in der Interrupt Service Routine. Der Pin PD0 entspricht dem externen Interrupt. Die Simulation erkennt dies und springt deshalb trotz eines Einzelschritts in die ISR.

e) Simulation kann nur im Zusammenhang mit Debugging verwendet werden. Nochmals: Bei Simulation wird keinerlei Code auf das Target, also den Mikrocontroller geladen.

#### Aufgabe 4: Hardware-Debugging mit JTAG

b)



d) Pin PD0 HIGH, was der Pull-Up Widerstand sicherstellt. Drückt man den Taster während des Einzelschritts, so zeigt das I/O Window für PD0 LOW an. Man merkt also, dass man gerade echtes HW-Debugging durchführt.

Name	Address	Value	Bits
PIND	0x29	0x03	00000001
DDRD	0x2A	0x00	00000000
PORTD	0x2B	0x00	00000000

Man landet in der ISR. Das Drücken der Taste löst einen Interrupt aus, den sich die HW merkt (auch wenn die Taste nicht während des Einzelschritts gedrückt war). Dieser Interrupt wird dann im nächsten Einzelschritt abgearbeitet.