Arbeit mit dem Debugger

11. November 2014

Zusammenfassung

Ausgehend von dem Hard- und Software-Entwurf der Schritt-für-Schritt-Anleitung für den Entwurf und Test eines Minimalsystems wird eine weiteres »Software Application Project« mit einem umfangreicheren Testprogramm angelegt und für den Test auch der Debugger genutzt.

1 Anlegen eines weiteren »Application Projects«

SDK ist zu starten und der »Workspace« des Projekts zuvor auszuwählen (Abb. 1).

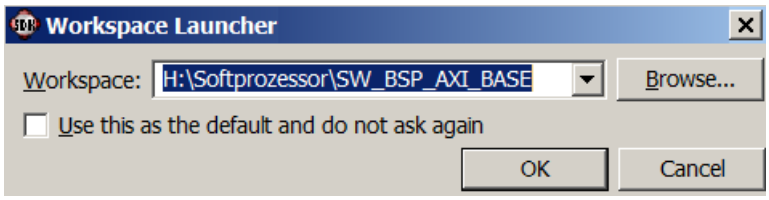

Abbildung 1: Festlegung des Pfads für das Software-Projekt

Das Projekt hat bereits seine »Hardware Platform« und sein »Board Support Package«, so dass sofort mit der Einrichtung eines neuen »Application Project« begonnen werden kann:

```
File ▷ New ▷ Application Project
```

Genau wie beim ersten »Application Project« sind die »Hardware Platform« und das »Bord Support Package« zuzuordnen (Abb. 2). Nach »Next« soll bei diesem und auch allen künftigen eigenen Software-Anwendungen »Empty Application« als Vorlage ausgewählt werden. Bei »Empty Application« werden keine Programmdateien erzeugt. Die benötigeten C- und Header-Dateien sind anschließend jeweils mit

```
File ▷ New ▷ Source File
```

bzw.

```
File ▷ New ▷ Header File
```

zu erzeugen und einzubinden. In dem sich daraufhin öffnenden Fenstern ist jeweils der Datei-Name einzutragen. Wichtig ist die Endung ».c« bzw. ».h«. Im Beispiel soll ein »Source File« mit dem Namen »test_io.c« erzeugt werden. Mit dem Source-File wird automatisch ein Ordner »Debug« mit dem Linker-Skript, in das beim Übersetzen auch das ausführbare Programm geschrieben wird, angelegt (Abb. 3 links). In die Quelldatei soll das Programm in Abb. 3 mitte eingegeben und gespeichert werden.

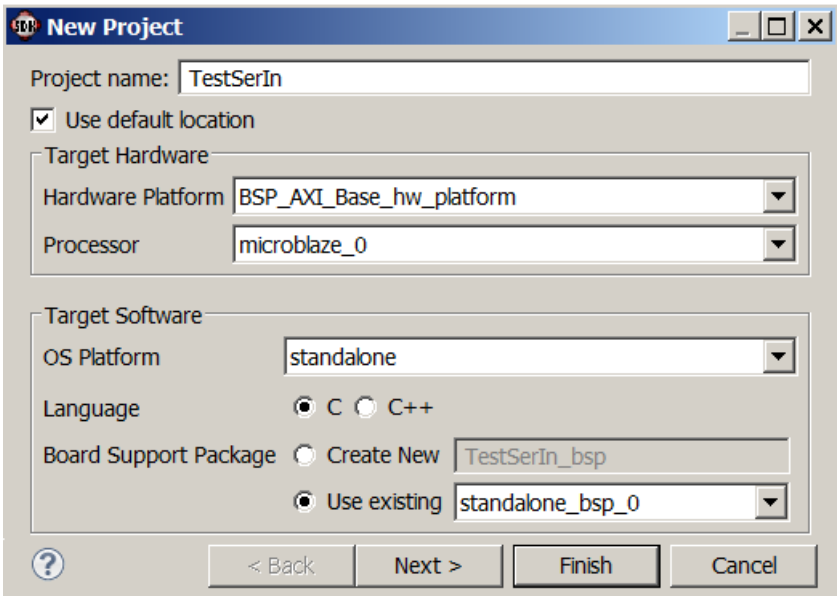

Abbildung 2: Einrichten des neuen Software-Projekts

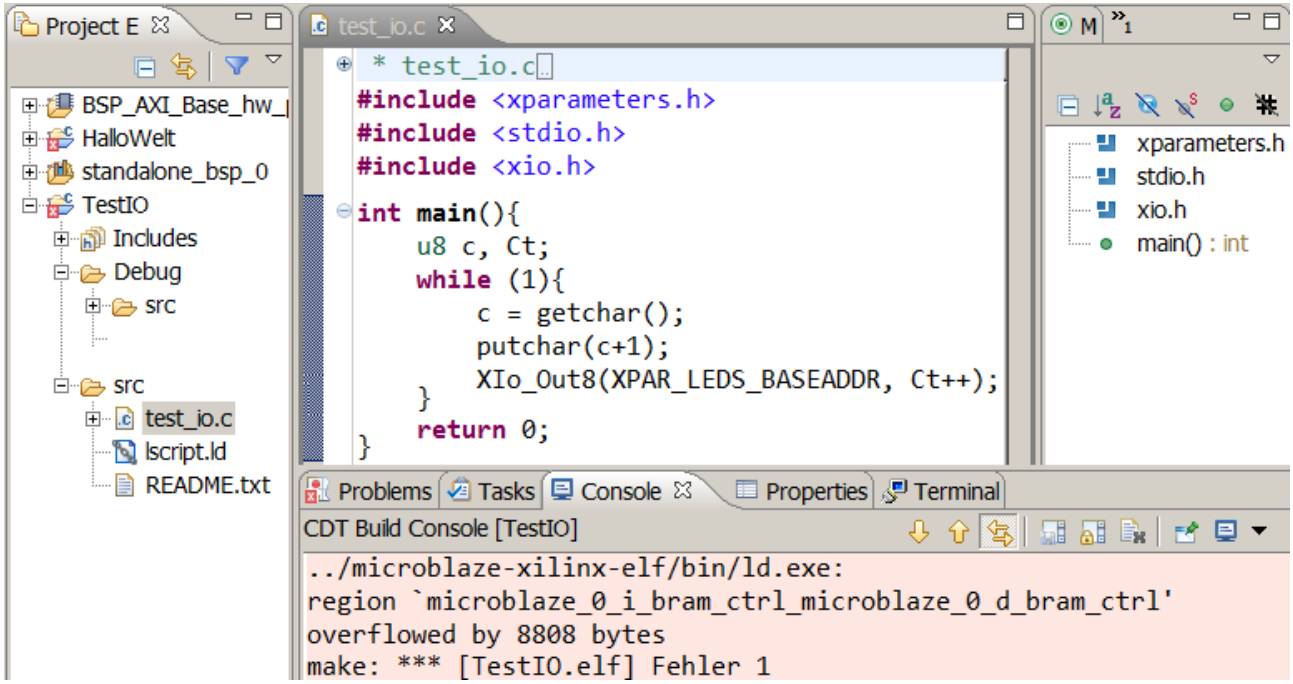

Abbildung 3: Explorer-Ansicht des neuen Software-Projekts und eingegebenes Programm

Das Programm liest in einer Endlosschleife mit der Funktion »getchar()« aus »stdio.h« ein Zeichen von der Standardeingabe, sendet es um eins erhöht zurück, zählt die empfangenen Zeichen in einer Bytevariablen und gibt den Zählwert auf die Leuchtdioden aus. Die Fehlermeldung beim Übersetzen in Abb. 3 »... overflowed by 8808 bytes« besagt, dass das Programm für den verfügbaren Befehlsspeicher zu groß ist. Übersetzbar wird das Programm erst, wenn sowohl die Programmzeile mit der Funktion »getchar()« als auch die mit der Funktion »putchar()« auskommentiert wird. Die über die »stdio.h« eingebundenen Standardeingabefunktionen sind offenbar entweder zu groß für den verfügbaren Programmspeicher oder fehlerhaft.

2 Geeignete Ein- und Ausgabefunktionen

Das »Board Support Package« stellt eine Vielzahl weiterer Ein- und Ausgabefunktionen für die vorhandenen Hardware-Einheiten zur Verfügung. Über den Header »xuartlite_l.h« lassen sich beispielweise folgende Funktion für das Senden und den Empfang von Bytes einbinden:

```
XUartLite_SendByte(<Adresse_Senderegister>, <Bytewert>);
<Bytewert> = XUartLite_RecvByte(<Adresse_Empfangsregister>);
```

Abb. 4 zeigt ein vergleichbares noch etwas erweitertes Programm mit diesen Ein- und Ausgabefunktionen, das übersetz-, link- und ausführbar ist. Die While-Schleife vor der Endlosschliefe gibt die Zeichenkette »Start Testschleife:« aus, um zu signalisieren, dass das Programm läuft.

```
#include <xparameters.h>
#include <xio.h>
#include <xuartlite_l.h>
int main(){
    u8 c, Ct;
    char *str = "Starte Testschleife:\r\n";
         while (*str != 0) {
            XUartLite_SendByte(XPAR_UARTLITE_0_BASEADDR, *str);
            str++;
          }
    while (1){
        c = XUartLite_RecvByte(XPAR_UARTLITE_0_BASEADDR);
        XUartLite_SendByte(XPAR_UARTLITE_0_BASEADDR, c+1);
        XIo_Out8(XPAR_LEDS_BASEADDR, Ct++);
    }
    return 0;
}
```

Abbildung 4: Geändertes Programm, in dem getchar() und putchar() durch funktionsgleiche Funktionen aus anderen Bibliotheken ersetzt sind

Vor dem Programmstart ist, falls noch nicht erfolgt, wieder über das Menü

```
Xilinx Tools ▷ Program FPGA
```

Die Hardware-Konfiguration in den FPGA zu laden. Danach ist mit mit

```
Run ▷ Run Configurations.. ▷
Rechtsklick auf »Xilix C/C++ application (GDB)« ▷ New
```

eine neu »Run Configuration« zu erstellen und auszuwählen ((Abb. 5)). Auch hier ist wieder unter dem Reiter »STDIO Connection« der Haken bei »Connect STDIO to Console« zu setzen und »JTAG UART« als Port auszuwählen. Der Programmstart erfolgt wieder mit »Run« bzw. über die Schaltfläche .

Abb. 6 zeigt die Ein- und Ausgaben auf der Konsole. Die Ausgabezeichen sind schwarz und die Eingaben grün dargestellt. Zuerst wird der Text »Starte Testschleife« ausgegeben. Dann wurden zweimal mehrere Zeichen eingegeben. Die Übergabe an das zu testende Rechnersystem erfolgt erst bei »Enter«. Ausgegeben werden jeweils die Folgezeichen in der ASCII-Tabelle, d.h. für eine »a« ein »b« und für ein »s« ein »t«. Auch das »Enter« Zeichen wird um eins erhöht. Der Ausgabewert auf den Leds erhöht sich bei jedem »Enter« um die Zeichenzahl. Allerdings erfolgt die Ausgabe bitgespiegelt, d.h. auf Led(7) wird das niederwertigste und auf Led(0) das höchstwertigste Zählbit ausgegeben.

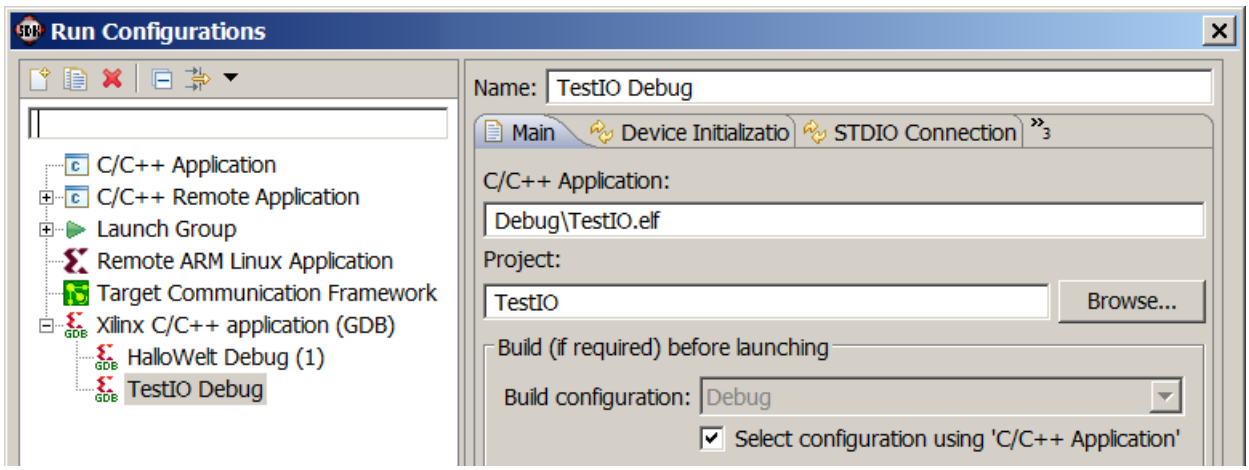

Abbildung 5: Haupteinträge der neuen »Run Configuration«

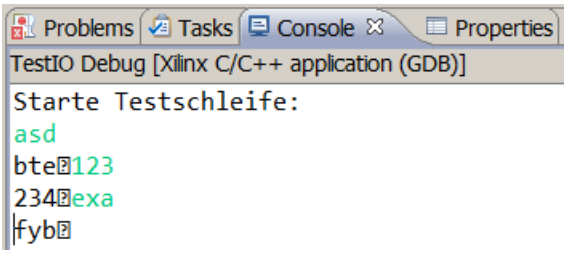

Abbildung 6: Testausgabe

3 Test mit dem Debuggers

Wenn, wie im Beispiel, die »Run Configuration« richtig eingestellt und ausgewählt ist, wird der Debugger mit

```
Run ▷ Debug
```

oder einen Klick auf das Käfersymbol gestartet. Falls auf dem Prozessor ein Programm läuft, ist die Frage »Would you like to terminate previous launch?« mit »Yes« zu beantworten. Die Frage »switch perspective?« ist auch mit »Yes« zu beantworten. Richten Sie die Debug-Ansicht so ein, dass, wie in Abb. 7 links der Quellcode mit Zeilennummern[1], rechts die Variablen und ihre Werte und darunter die Console mit den Ein- und Ausgaben angezeigt werden.

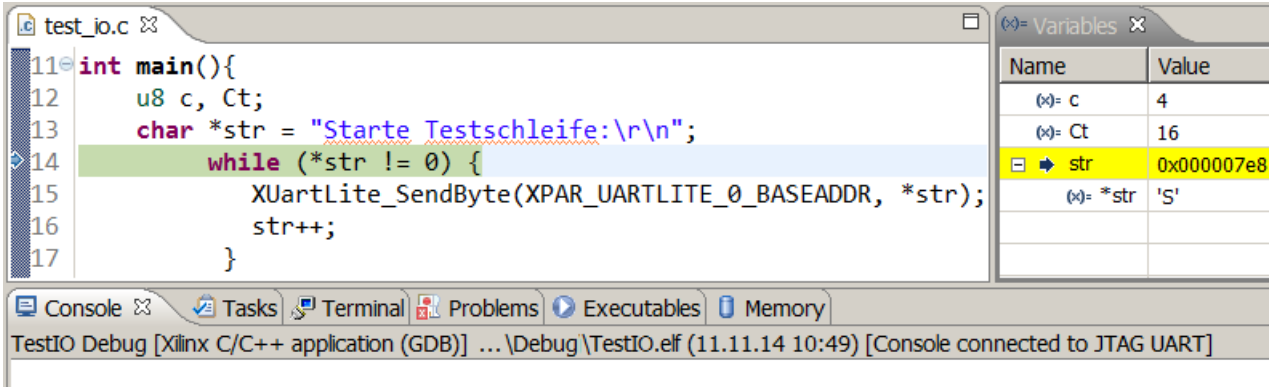

Abbildung 7: Debug-Ansicht nach dem ersten »Step Over«

[1]Rechtsklick auf den Rand vor einer Programmzeile und aktivieren von »Show Line Numbers«.

Gehen Sie mit (Step Over) wie im Bild mit dem blauen Abarbeitungszeiger bis vor die while-Schleife in Zeile 14. Der Zeiger »str« zeigt auf den Anfang der Zeichenkette. Öffnen Sie wie in Abb. 8 ein Memory-Fenster, das den Speicherbereich ab Adresse »0x7E8« anzeigt. Mit einer ASCII-Tabelle lässt sich überprüfen, dass die ab dieser Adresse gespeicherten Werte 0x53, 0x74, 0x61, 0x72, 0x74, ... genau den Ausgabetext darstellen.

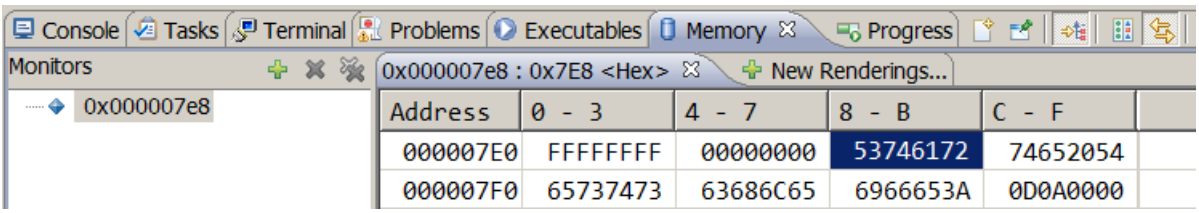

Abbildung 8: Speicherfenster mit dem Ausgabetext

Setzen vor Zeile 16 mit der Anweisung »str++« einen Unterbrechungspunkt, bei dem Sie über das Menü in Abb. 9 einstellen, dass er erst beim fünften Treffer das Programm anhält und starten Sie das Programm mit (Resume).

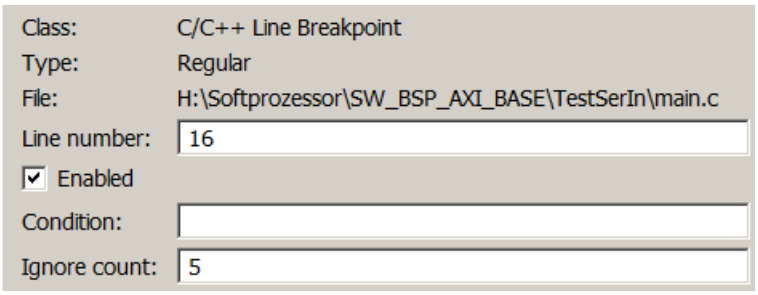

Abbildung 9: Editierfenster für Unterbrechungspunkteinstellungen

Das Programm durchläuft fünfmal die Schleife, erhöht den Zeiger »str« um fünf und gibt die ersten fünf Zeichen des Ausgabetextes, d.h. die Teilzeichenkette »Starte« aus (Abb. 10).

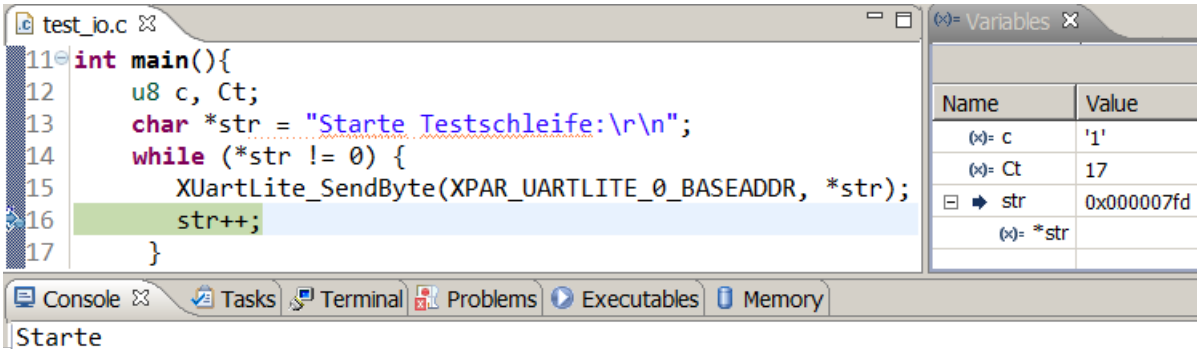

Abbildung 10: Debug-Ausgabe am Unterbrechungspunkt

Als nächstes soll der jetzige Unterbrechungspunkt gelöscht und wie in Abb. 11 ein einfacher Unterbrechungspunkt hinter die Anweisung für den Byte-Empfang gesetzt werden. Anschließend ist das Programm mit (Resume) fortzusetzen. Die Zeichenkette wird fertig ausgegeben (Abb. 12). Dann sollen Zeichen '1', '2', ... eingegeben werden, die grün angezeigt werden.

Das Programm erreicht nicht den Unterbrechungspunkt, hält auch nicht an und zeigt auch keine Variablen an, bevor auf der Console ein »Enter« eingegeben wird. Nach der Eingabe von

```
19      while (1){
20          c = XUartLite_RecvByte(XPAR_UARTLITE_0_BASEADDR);
21          XUartLite_SendByte(XPAR_UARTLITE_0_BASEADDR, c+1);
22          XIo_Out8(XPAR_LEDS_BASEADDR, Ct++);
23      }
```

Abbildung 11: Setzen des zweiten Unterbrechungspunktes

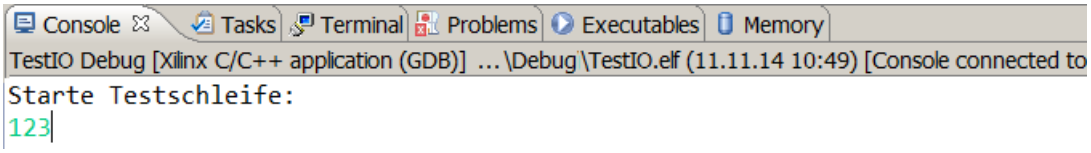

Abbildung 12: Testaus- und Eingabe vor der Eingabe von »Enter«

»Enter« schickt die Console alle vier Zeichen an das Programm. Das liest das erste Zeichen, in der Abbildung eine '1', ein und hält am Unterbrechungspunkt an. Mit (Step Over) wird das nächste Zeichen in der ASCII-Tabelle '2' ausgegeben (Abb. 13). Noch einen Schritt weiter wird der initiale, um eins erhöhte Zählwert der Variablen »Ct«, hier 0x18, ausgegeben[2].

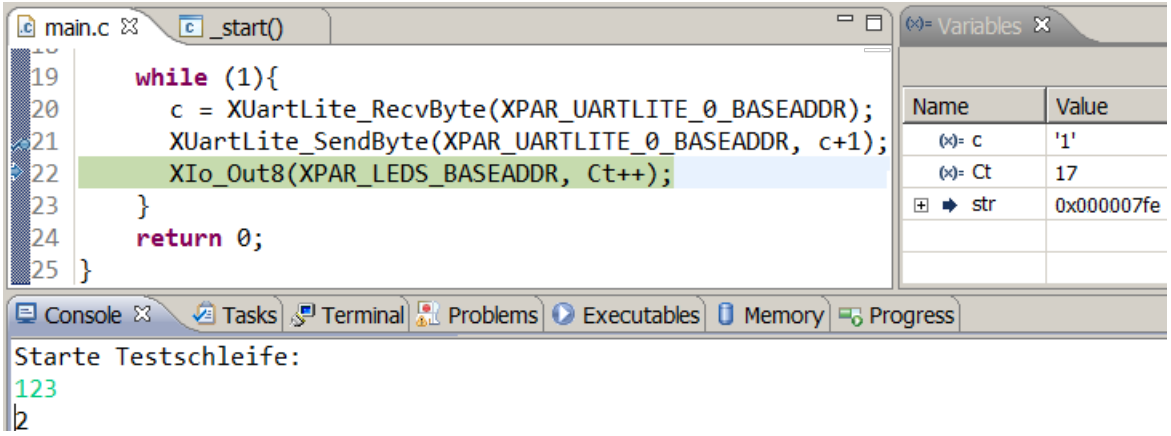

Abbildung 13: Ein- und Ausgabe nach »Step Over« vom zweiten Unterbrechungspunkts

Experimentieren Sie mit dem Programm weiter:

- Fortsetzen mit (Step Over),
- Fortsetzen mit (Step Into),
- Setzen und Löschen von Unterbrechungspunkten,
- Fortsetzung mit (Resume),
- Anhalten, wenn das Programm keinen Unterbrechungspunkt erreicht, mit (Suspend),
- Beenden der Programmabarbeitung mit (Terminate) und
- Neustart ab Programmbeginn mit (Debug).

Die Arbeit mit einem Debugger erfordert Übung. Noch mehr Übung erfordert es, ein Programm so zu schreiben, dass es sich mit verfügbaren Debug-Funktionen gut testen lässt.

[2]Wegen der fehlenden Anfangsinitialisierung der Variablen »Ct« ist das ein zufälliger Ausgabewert, der bei jedem Programmstart anders sein kann. Damit ist der erste offensichtliche Fehler im Programm gefunden. Denn das Programm soll sicher keine Zufallswerte auf die Leds ausgeben.