Ist dir langweilig?
Bau dir ein Langweilometer

Das Langweilometer war ein Spaßprojekt während der Leitung eines Projektpraktikums an der Hochschule München. Es hilft zwar nicht gegen Langeweile, kann diese aber dokumentieren.

von **Jürgen Plate**

Bekanntlich werden Übel
dadurch erleichtert,
dass man sie
gemeinschaftlich erträgt.

Arthur Schopenhauer

Die Idee des Langweilometers ist schnell erklärt: Per Webinterface melden Teilnehmer einer Veranstaltung ihren aktuellen Eindruck. Dies geschieht durch Eingabe eines prozentualen Langeweilepegels zwischen Null („Super spannend und unterhaltsam!“) und 100 („Schlaftablette!“). Die Webseite meldet den aktuellen Stand nach der Eingabe grafisch zurück. Damit nicht jede Eingabe zur totalen Schwankung der Anzeige führt, zeigt das Langweilometer den Medianwert über die letzten elf Eingaben an (die Dämpfung lässt sich auch auf mehr als elf Eingaben erweitern). Außerdem gibt es eine analoge Ausgabe des Messpegelstandes.

Als Webserver kommt ein Raspberry Pi zum Einsatz, der klein und kostengünstig ist [1]. Er steuert die analoge Anzeige und alles andere über seine digitale Schnittstelle, das *General Purpose Input Output* (GPIO). Neben den typischen Schnittstellen zeichnet sich der kleine Computer durch eine Steckerleiste aus, die Digitalwerte einliest und ausgibt. Gibt er auf einer Leitung ein Signal aus, so sind dort entweder 3,3 Volt messbar (logisch *1*) oder null Volt (logisch *0*). Bei der Eingabe ist das im Prinzip genauso: Liegt dort eine Spannung von mehr als zwei Volt an, ist das eingelesene Zeichen eine logische 1, liegt die Eingangsspannung unter 0,8 Volt, erhält man eine logische 0. Legt man mehr als 3,3 Volt an einen Pin an, muss man einen neuen Raspberry Pi kaufen.

Messpegel-Anzeige in Hardware basteln

Da nur derjenige Veranstaltungsteilnehmer das Webinterface sieht, der eine Bewertung abgibt, soll noch eine externe Anzeige hinzukommen. Die Wahl fiel hier nicht auf das übliche Display oder eine Siebensegment-Ziffernanzeige, sondern es sollte etwas Analoges sein.

Glücklicherweise stand ein altes Voltmeter aus dem Messtechniklabor zur Verfügung. Es sitzt in einem etwa 40 Zentimeter hohen Gehäuse aus Holz und Glas, wie es früher im Physikunterricht üblich war. Ein riesiges Drehspulinstrument bewegt den Zeiger, der auch aus der Entfernung noch erkennbar ist. Das Gehäuse bietet nach dem Ausbau der Vorwiderstände und des Messbereichs-Wahlschalters noch Platz für den Raspberry Pi und vermittelt gleichzeitig einen Hauch von Steampunk (bis auf etwas zuwenig Messing am Gehäuse).

Abbildung 1: Ansicht des umgebauten Messinstruments von vorn.

Die zweite Idee, per Schrittmotor und Laserpointer einen Punkt auf einer an die Wand ge-

malten Skala zu markieren, scheitert an der Machbarkeit: Das Bemalen einer Wand im Foyer findet doch eventuell nicht ganz die Zustimmung der Chefs. Die dritte Idee, als Skala ein Plexiglasrohr zu nehmen, das bis zum entsprechenden Anzeige-Pegel mit gefärbtem Wasser gefüllt wird, scheidet ebenfalls als zu kompliziert aus. Also bleibt nur das Messgerät. Wer noch ein altes Spiegelgalvanometer auftreibt, kann auch das verwenden. Hauptsache groß und imposant.

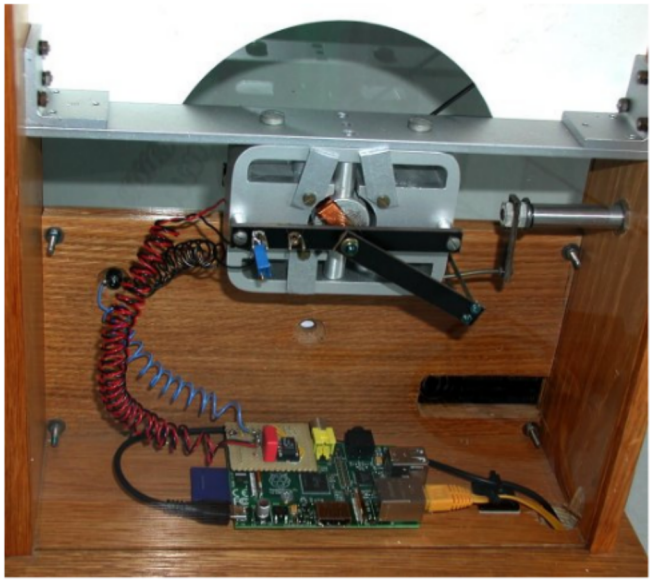

Abbildung 2: Einbau des Raspberry Pi in das Messgerät.

Als erstes ersetzt eine Prozent-Skala die Volt-Skala. Dann gehören die Bohrungen abgedeckt und das Typenschild umgedreht (dahinter war leider ein großes Loch für Sicherungen). Die große Skala des Drehschalters für die Messbereiche wird entfernt und durch ein Smiley ersetzt. Abbildung 1 zeigt das Ergebnis dieser Vorarbeiten.

Der Raspberry Pi und eine kleine Zusatzplatine für den Digital-Analog-Wandler finden wunderbar im Gehäuse Platz. Für das Stromversorgungskabel und das Netzwerkkabel kommt ein Loch in den Boden des Messgeräts. Für die Netzwerkverbindung eignet sich ein dünnes und flaches Kabel, das besonders flexibel ist. Solche Kabel lassen sich sogar unter dem Teppich verlegen.

Der Blick durch die hintere Scheibe zeigt nun unten den Raspi und das Messwerk (Abbildung 2). Der blaue Knubbel links am Messgerät ist ein Vorwiderstand (Mehrgangpoti), mit dem sich der Vollausschlag des Messinstruments auf die maximalen 3,3 Volt des Digital-Analog-Wandlers einstellen lässt. Das blaue Kabel führt zu einer ebenfalls blauen LED, die in die Öffnung des ehemaligen Umschalters gewandert ist. Sie zeigt nicht nur an, ob der Raspberry noch lebt, sondern auch, wann ein neuer Datenwert eintrifft. Die Spiralform der Kabel hat nur einen optischen, keinen elektrischen Zweck.

Analog-Interface für den Raspberry Pi

Der Raspberry Pi verfügt leider über keine analogen Ein- und Ausgänge, und der Ausgang für ein pulsweit moduliertes Signal (PWM) verträgt sich irgendwie nicht mit dem relativ niederohmigen Messwerk. Also musste eine kleine Schaltung für einen Digital-Analog-Wandler her. Keine Angst, die Schaltung in Abbildung 3 ist so einfach, dass sie auch von jemandem zusammengebraten werden kann, der mit dem Lötkolben ungeübt ist. Man sollte nur nicht zu lange an einen Pin herumlöten.

Die Anode der LED (positiver Pol, längerer Anschlussdraht) an der Gerätefront ist direkt über einen 120-Ohm-Vorwiderstand an GPIO 27 angeschlossen. Die Kathode (kürzerer Anschlussdraht) ist mit Masse verbunden. Der Wandlerbaustein ist auf einer Lochrasterplatte verdrahtet. Auf der Unterseite der Lochrasterplatte sitzt eine doppelreihige Buchsenleiste, die auf den GPIO-Anschluss des Raspberry Pi passt. Dann kann man die wenigen Verbindungen löten und die Platine auf den Raspi stecken.

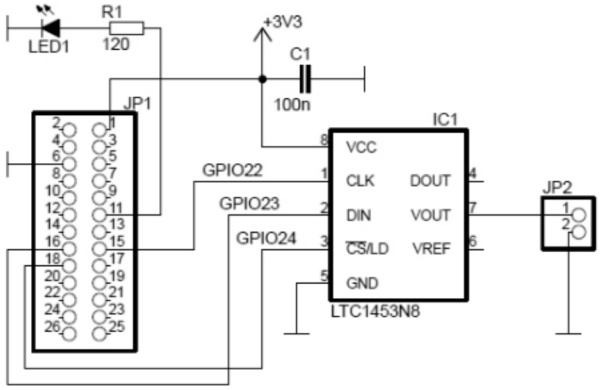

Abbildung 3: Die Schaltung des Digital-Analog-Wandlers.

Als Digital-Analog-Wandler dient der Baustein *LTC1453* von *Linear Technology* im achtpoligen, lötfreundlichen *DIL*-Gehäuse (Dual In-Line) [2]. Der Chip ist für den 3,3-Volt-Betrieb ausgelegt und hat eine 12 Bit Auflösung, was für die geplante Anwendung mehr als ausreichend ist. Auf den ersten Blick schien der LTC1453 ein ganz normaler *SPI*-Chip zu sein (Serial Peripheral Interface) – also null Probleme bei der Ansteuerung, denn der Raspberry hat schon ein passendes Treibermodul. Leider stellt sich heraus, dass es sich um einen nur

fast SPI-kompatiblen Chip handelt. Dazu später mehr.

Das SPI ist ein von *Motorola* entwickeltes System für einen synchronen, seriellen Datenbus. Er verbindet digitale Schaltungen nach dem Master-Slave-Prinzip. Im Grunde wandern die Daten von einem Takt gesteuert Bit für Bit vom Controller in den Peripheriebaustein und umgekehrt. Insofern ist SPI ein entfernter Verwandter der guten, alten seriellen Schnittstelle – nur ohne Start- und Stop-Bit, und mit getrennter Taktleitung. Das SPI besteht aus drei gemeinsamen Leitungen:

- SCLK (Serial Clock) – Takt, den der Master erzeugt.
- MOSI (Master Output, Slave Input) – dient der bitweisen Datenübertragung zum Peripheriebaustein.
- MISO (Master Input, Slave Output) – empfängt die Bits vom Peripheriebaustein

Damit sich mehrere Slaves ansprechen lassen, verfügt jeder Baustein über einen digitalen *Select*-Eingang, der über Null oder Eins den Chip aktiviert oder deaktiviert. Da beim Digital-Analog-Wandler nur Daten zum Peripheriebaustein wandern, sind sogar nur zwei Leitungen (und natürlich Chip-Selects) nötig.

Wie angedeutet, outet sich der LTC1453 als nur fast-SPI, denn er enthält einen Hardware-Bug. Bug und Workaround sind im Datenblatt versteckt [3]. Dort steht der kleine Satz: „Note: CLK must be low before CS/LD is pulled low to avoid an extra internal clock pulse. “ Übersetzt heißt das:

- Den Chip-Select-Pin CSLD auf 1 setzen, dann elf (!) Bit ganz normal eintakten (positiver Impuls am Takteingang CLK)
- Beim letzten Bit den Takteingang CLK auf 1 setzen, dann den CSLD-Pin auf 0 und danach CLK auf 0.

`lwm_setdac.c`: Pin aktivieren

Die Funktion `GPIO_Export()` aktiviert einen Pin des GPIO, indem man die Nummer des gewünschten GPIO in die Pseudovariable */sys/class/gpio/export* schreibt (Zeile 08):

```
01 int GPIO_Export(int pin)
02  {
03   char buffer[MAXBUF]; /* Output Buffer */
04   ssize_t bytes; /* Datensatzlaenge */
05   int fd;  /* Filedescriptor */
06   int res;  /* Ergebnis von write */
07
08   fd = open("/sys/class/gpio/export",
09             O_WRONLY);
10  if (fd < 0)
11     {
12     perror("Fehler bei open (export)!\n");
13     return(-1);
14     }
15   bytes = snprintf(buffer, MAXBUF, "%d", pin);
16   res = write(fd, buffer, bytes);
17   if (res < 0)
18     {
19     perror("Kann Pin nicht aktivieren!\n");
20     return(-1);
21     }
22   close(fd);
23   return(0);
24  }
```

Genauso lässt sich das GPIO wieder deaktivieren. Nur heißt das Ziel in Zeile 08 dann `/sys/class/gpio/unexport`. Dafür gibt es die Funktion `GPIO_Unexport()`, die aus dem Listing ensteht, in dem man den Funktionsnamen in Zeile 01 entsprechend ändert und `export` in Zeile 12 durch `unexport` ersetzt.

Also steht kein geeigneter Gerätetreiber zur Verfügung. Die Bits sind von Hand in den LTC1453 zu schieben. Nachdem im hier berichteten Fall der Baustein schon verlötet war, entstand die Software eben ohne SPI-Treiber, was auch nicht weiter schlimm war.

`lwm_setdac.c`: Analogwert auf AD-Wandlerbaustein bringen

```
01 void SetData(unsigned int value)
02   {
03   int i;
04  /* CS muss anfangs LOW sein */
05   GPIO_Write(CS, LOW);
06  /* shift MSB ... LSB (MSB zuerst) */
07   for(i = 11; i >= 0; i--)
08     {
09    /* set data bit, check MSB  */
10     if (value & 0x800)
11       GPIO_Write(DIN, HIGH);
12     else
13       GPIO_Write(DIN, LOW);
14     value = value << 1;
15    /* CLK-Impuls senden und bei Bit 0 CS/LD-Impuls auf HIGH */
16     GPIO_Write(CLK, HIGH);
17     if (i == 0) GPIO_Write(CS, HIGH);
18     GPIO_Write(CLK, LOW);
19     }
20   GPIO_Write(CS, LOW);
21  }
```

Die Funktion `void SetData()` schiebt den Analogwert als 12-Bit-Zahl bitweise in den LTC1453-Baustein, wobei das MSB (*most significant bit*, höchstwertiges Bit) als erstes kommt (Zeilen 06/07).

`lwm_setdac.c`: Analogwert setzen ...

Das C-Programm *lwm_setdac.c* ([4], [5]) setzt einen Analogwert über den Baustein *LTC1453*. Nach getaner Arbeit macht das Programm mittels einer `delay`-Funktion eine kleine Pause von 0,2 Sekunden, damit auch ein amok laufendes Skript (oder ein DAU) nicht zu wild mit dem Gerätezeiger wedelt. Die Funktion rechnet Millisekunden in Sekunden und Nanosekunden um und ruft dann die Library-Fnktion `nanosleep()` auf.

Der einzige Kommandozeilenparameter des Programms ist ein Prozentwert (`int`) zwischen null und 100. Damit taugt das Programm nicht nur für das Langweilometer, sondern auch für andere Anwendungen oder analoge Anzeigezwecke. Auch ein Aufruf per Shellskript ist möglich: Der Aufruf erfolgt dann zum Beispiel als `./lwm_setdac 45` für eine Anzeige von 45 Prozent.

Das Kompilieren erfolgt mit dem Kommando:

```
gcc -Wall -o lwm_setdac lwm_setdac.c
```

Damit unprivilegierte User (wie *pi* oder *www-data*) das Programm aufrufen können, muss es auf die SUID *root* lauten – oder man gibt den GPIO generell für den entsprechenden User frei:

```
01 sudo chown root.root lwm_setdac
02 sudo chmod 4711 lwm_setdac
```

Das Programm definiert folgende Konstanten, die für die Kommunikation mit dem GPIO notwendig sind:

```
01 /* Datenrichtung */
02 #define IN  0
03 #define OUT 1
04
05 /* Binärdatenwerte */
06 #define LOW  0
07 #define HIGH 1
08
09 /* großzügig bemessene */
10 /* maximale Datenpuffergröße */
11 /* fuer die GPIO-Funktionen */
12 #define MAXBUF 100
13
14 /* Namen der LTC1453-Pins */
15 #define CLK 22 /* Clock Input */
16 #define DIN 23 /* Data Input  */
17 #define CS 24 /* Enable */
```

Kern des Ganzen ist die Funktion `void SetData()`. Sie schiebt den Analogwert als 12-Bit-Zahl bitweise in den LTC1453-Baustein, wobei das höchstwertige Bit als erstes kommt (siehe Kasten „`void SetData()`: Analogwert auf AD-Wandlerbaustein bringen“).

Beim letzten Bit ist in dieser Funktion ab Zeile 10 der Workaround für den Hardware-Bug er-

sichtlich. Da wir es mit einer 12-Bit-Zahl zu tun haben, ermittelt eine *AND*-Verknüpfung mit `0x800` (= 1000 0000 0000) das höchstwertige Bit und setzt die Datenleitung zum LTC1453 dementsprechend auf `0` oder `1`. Danach wandert die Zahl um ein Bit nach links. Für den Takt wird bei den ersten elf Bits ein `1`-Impuls erzeugt, und beim letzten Bit die CS-Leitung vor dem Wechsel des Taktsignals auf `1` gezogen.

`lwm_setdac.c`: Datenrichtung festlegen

Die Funktion `Funktion GPIO_Direction()` legt die Datenrichtung eines GPIO-Pins fest. Der Wert `0` bedeutet, dass der GPIO ein Eingabepin ist, und `1` bedeutet Ausgabepin:

```
01 int GPIO_Direction(int pin, int dir)
02  {
03   char path[MAXBUF]; /* Buffer fuer Pfad */
04   int fd;  /* Filedescriptor */
05   int res;  /* Ergebnis von write */
06
07   snprintf(path,
08            MAXBUF,
09            "/sys/class/gpio/gpio%d/direction",
10            pin);
11   fd = open(path, O_WRONLY);
12   if (fd < 0)
13     {
14     perror("Fehler direction (open)!\n");
15     return(-1);
16     }
17   switch (dir)
18     {
19     case IN : res = write(fd,"in",2); break;
20     case OUT: res = write(fd,"out",3); break;
21     }
22   if (res < 0)
23     {
24     perror("Fehler direction (write)!\n");
25     return(-1);
26     }
27   close(fd);
28   return(0);
29  }
```

... und digital den Raspi ansteuern

Nach der Umwandlung spricht das C-Programm über das */sys*-Dateisystem die digitale Schnittstelle *General Purpose Input Output* (GPIO) des Raspberry Pi an. Die GPIO-Pins lassen sich damit direkt über ein System virtueller Dateien steuern. Auch hier gilt, dass dies nur mit Root-Rechten gelingt. Schreiben oder Lesen auf die virtuellen Dateien aktiviert über den Linux-Treiber zunächst den entsprechenden Port-Pin und stellt die Datenrichtung ein. Danach spricht Lesen oder Schreiben dann den Port an.

Die folgenden Funktionen dienen dem Ein- und Ausschalten der Ports und dem Schreiben auf eine Portleitung. Bei allen Funktionen bedeutet der Rückgabewert `0`, dass die Operation erfolgreich war, und `-1`, dass ein Fehler aufgetreten ist.

Die erste Funktion `GPIO_Export()` aktiviert einen Pin des GPIO. Dies geschieht, indem man die Nummer des gewünschten GPIO – nicht etwa die Pinnummer der Steckerleiste! – in die Pseudovariable */sys/class/gpio/export* schreibt (siehe Kasten „`lwm_setdac.c`: Pin aktivieren“).

Durch den Export entsteht ein neues Verzeichnis mit dem Namen `gpioXX`. In diesem Verzeichnis befinden sich unter anderem zwei Dateien namens *direction* und *value*. Diese Dateien dienen zum Ansteuern des GPIO-Pins. *direction* legt die Datenrichtung fest, also die Ein- oder Ausgabe. *value* dient zum Lesen und Schreiben von `0` oder `1`.

Die zweite Funktion `GPIO_Direction()` legt

die Datenrichtung eines GPIO-Pins fest (siehe Kasten „`lwm_setdac.c`: Datenrichtung festlegen“). Ist ein GPIO-Pin als Ausgabe geschaltet, erzeugt das Schreiben von Null oder Eins am entsprechenden Pin einen Pegel von 0 Volt bzw. 3,3 Volt. Die dritte wichtige Funktion `GPIO_Write()` setzt schließlich einen GPIO-Pin (siehe Kasten „`lwm_setdac.c`: Pin setzen“).

`lwm_setdac.c`: Pin setzen

Die Funktion `GPIO_Write()` setzt einen GPIO-Pin, nachdem `GPIO_Direction()` die Datenrichtung festgelegt hat. Als Werte (Parameter `value`, Zeile 09) kommen nur `LOW` und `HIGH` zum Einsatz:

```
01 int GPIO_Write(int pin, int value)
02  {
03   char path[MAXBUF]; /* Buffer fuer Pfad */
04   int fd; /* Filedescriptor */
05   int res; /* Ergebnis von write */
06
07   snprintf(path,
08            MAXBUF,
09            "/sys/class/gpio/gpio%d/value",
10            pin);
11   fd = open(path, O_WRONLY);
12   if (fd < 0)
13     {
14     perror("Fehler GPIO_Write (open)!\n");
15     return(-1);
16     }
17   switch (value)
18     {
19     case LOW : res = write(fd,"0",1); break;
20     case HIGH: res = write(fd,"1",1); break;
21     }
22   if (res < 0)
23     {
24     perror("Fehler GPIO_Write (write)!\n");
25     return(-1);
26     }
27   close(fd);
28   return(0);
29  }
```

Das komplette C-Programm vereint alle Funktionen. Der Vorspann mit seinen Header-Dateien und den Konstanten ist eigentlich für alle Programme verwendbar, die auf die Ein- und Ausgabe-Ports des Raspberry Pi zugreifen. Lediglich die Namen der Pins ändern sich. Die Hauptfunktion *main()* prüft zu Beginn, ob jemand mit Root-Rechten das Programm aufgerufen hat. Wenn nicht, bricht es die Ausführung ab. Danach initialisiert es die benötigten GPIO-Ports und prüft den Kommandozeilenparameter.

Die eigentliche Arbeit zeigt der Ausschnitt im Kasten „`lwm_setdac.c`: Richtige Ganzzahlen errechnen“: Da der Digital-Analog-Wandler 12 Bit verarbeitet, muss der an den Wandler gesendete Wert zwischen Null und 4095 liegen. Der hereinkommende Wert liegt zwischen Null und Hundert – es geht ja um den prozentualen Langeweilepegel.

`lwm_frontled.c`: LED erzählt, was geschieht

Die LED an der Gerätefront ist eigentlich eine Notlösung. Nach dem Ausbau der Messgeräte-Innereien hinterlässt der Umschalter für Gleich- und Wechselspannung nämlich ein unschönes Loch, in das aber genau eine fünf-Millimeter-LED mit Fassung passt. Fehlt nur noch eine Aufgabe für die LED.

Bei Mikrocontrollern implementiert der Autor nun immer gern eine blinkende LED als so genannten Heartbeat, um zu sehen, ob der Controller noch am Leben ist. Beim Raspberry kommt

die Tatsache gelegen, dass Linux ein Multitasking-Betriebssystem ist. Die LED arbeitet also völlig unabhängig vom Webinterface oder anderen Programmen – getreu dem Motto: Ein Programm, eine Aufgabe, die aber richtig. Das Beispiel besitzt aber sogar zwei Aufgaben. Im Ruhezustand arbeitet die LED als Heartbeat und zeigt an, dass das System aktiv ist. Und wenn eine neue Eingabe über die Webseite eintrifft, flackert sie kurz und heftig auf.

`lwm_setdac.c`: Richtige Ganzzahlen errechnen

```
01  if (argc == 2)
02    {
03    /* String lesen, nach int umwandeln */
04     value = atoi(argv[1]);
05    /* Wertebereich 0 .. 100 */
06     if ((value >= 0) || (value <= 100))
07       {
08      /* Prozent nach 12-Bit-int wandeln */
09       value = (unsigned int)(40.95 * value);
10       SetData(value);
11       }
12     else
13       {
14       fprintf(stderr,
15       "Argument must be between 0 and 100!\n");
16       return 3;
17       }
19    }
```

Da der Digital-Analog-Wandler 12 Bit verarbeitet, muss der an den Wandler gesendete Wert zwischen Null und 4095 liegen. Der hereinkommende Wert liegt zwischen Null und Hundert – es geht ja um den prozentualen Langeweilepegel (vgl. Zeile 05/06). Also multipliziert Zeile 09 den von der Kommandozeile hereinkommenden Wert mit 40,95 – so erhält der Wandler den richtigen Ganzzahlenwert.

Für die LED gibt es auch wieder ein C-Programm: `lwm_frontled.c` [6]. Das Kompilieren erfolgt wieder ganz schlicht mit dem Kommando

```
gcc -Wall -o lwm_frontled lwm_frontled.c
```

Das Programm muss natürlich ständig laufen, sonst blinkt nix. Damit der Bootvorgang es aktiviert, steht es sinnvollerweise in der Datei */etc/rc.local*. Damit der Zeiger des Messinstruments nach dem Booten nicht bei Null stehen bleibt, kommt auch ein Aufruf von `lwm_setdac.c` in diese Datei. Somit müssen zwei Zeilen das `exit` in */etc/rc.local* ergänzen:

```
01 /home/pi/lwm_setdac 50 &
02 /home/pi/lwm_frontled &
```

Das Ampersand-Zeichen hinter den Kommandos ist wichtig, um die Programme in den Hintergrund zu legen. Sonst unterbrechen sie den Ablauf der Startskripte, weil die Ausführung bei *lwm_frontled* hängen bleibt.

Und Action!

Das Programm *lwm_frontled.c* [5] implementiert die zwei Aufgaben der LED folgendermaßen:

- Der Heartbeat lässt die LED für 50 Millisekunden aufleuchten. Darauf folgen 100 Millisekunden Pause, wieder 50 Millisekunden Aufleuchten, dann eine Pause von einer Sekunde.
- Erfolgt eine neue Eingabe über das Webinterface, blinkt die LED 50 mal mit je 50 Millisekunden Periodendauer. Sie flackert also für 2,5 Sekunden.

Der Programmcode ist relativ einfach. Die Delay-Funktion stammt aus *lwm_setdac.c*. Auch `lwm_frontled.c` ist für andere Aufgaben verwendbar – sofern es nur darum geht, auf die Änderung einer Datei zu reagieren. Der Vorspann ist der gleiche wie beim vorhergehenden Programm, lediglich die Definitionen des LED-Pins und der Datendatei sind unterschiedlich. Die LED selbst ist an GPIO 27 angeschlossen, das ist Pin 13 der Steckerleiste:

```
...
01 /* maximale Datenpuffergröße */
02 /* fuer die GPIO-Funktionen */
03 /* (großzügig bemessen) */
04 #define MAXBUFFER 100
05
06 /* LED-Pin, Stiftleiste Pin 13 */
07 #define LED 27
08
09 /* Datei für Werteübergabe */
10 #define FILENAME "/home/pi/boring"
...
```

`lwm_frontled.c`: Datei-Modifikationsdatum für LED-Flackern

```
01   for(;;)
02     {
03    /* Lesen des Modifikationsdatums */
04     stat(FILENAME, &attr);
05     epoc = attr.st_mtime;
06     if (epoc > oldepoc)
07 /* Datei geändert, Lightshow :-) */
08       {
09       for (i = 0; i < 50; i++)
10         {
11         GPIO_Write(LED, 1);
12         delay(50);
13         GPIO_Write(LED, 0);
14         delay(50);
15         }
16       oldepoc = epoc;
17       }
18     else
19 /* normaler Heartbeat: */
20 /* zweimal blinken, dann Pause */
21       {
22       GPIO_Write(LED, 1);
23       delay(50);
24       GPIO_Write(LED, 0);
25       delay(100);
26       GPIO_Write(LED, 1);
27       delay(50);
28       GPIO_Write(LED, 0);
29       delay(1000);
30       }
31     }
32   return 0; /* never reached */
33   }
```

Das Heartbeat-Programm merkt sich in den Zeilen 03 bis 05 das Dateidatum und fragt es mittels `stat()` nach jedem Herzschlag-Zyklus ab (Zeile 06). Ist das neue Dateidatum größer, ist die Datei geändert – die Lightshow beginnt (Zeilen 07 bis 17), und das Programm speichert das neue Datum (Zeile 15). Günstig ist, nicht Datum und Uhrzeit zu speichern, sondern die sogenannte Unix-Epoche, also die Anzahl der Sekunden, die seit dem 1.1.1970, null Uhr vergangen sind (Zeilen 04 und 21). Das ist nämlich ein monoton steigender 32-Bit-Wert, der sich ideal für den geplanten Zweck eignet.

Neu hinzu kommt eine Signal-Handler-Funktion für die Tastenkombination *[Strg]+[C]*. Sie räumt auf, wenn das Programm bei manuellem Aufruf von Hand wieder zu beenden ist und sitzt vor der Hauptfunktion:

```
01 void beenden(int dummy)
02   {
03   GPIO_Write(LED, 0);
04   GPIO_Unexport(LED);
05   delay(100);
06   printf("\nHasta la vista, Baby!\n");
07   exit(0);
08   }
```

Nach der Initialisierung begibt sich das Programm in eine Endlosschleife. Wichtig ist bei solchen Programmen, dass in jedem durchlaufenen Programmzweig irgendwo ein Delay auftaucht – sonst nimmt das Programm den Prozessor unnö-

tig stark in Beschlag.

Das Blinken ist einfach: Delay, LED an, Delay, LED aus. Wie erfährt aber das LED-Programm von einer Neueingabe über das Webinterface? Da die Realisierung über eine der zahlreichen Möglichkeiten der Prozesskommunikation zu aufwendig erscheint, tut es ein einfacher Trick: Jeder neue Wert, der vom Webinterface kommt, wandert ja in die Datei */home/pi/boring*. Dadurch ändert sich das Modifikationsdatum der Datei (siehe Kasten "`lwm_frontled.c`: Datei-Modifikationsdatum für LED-Flackern").

`lwm_index.php`: Einfügefunktion

```
01 function insert($newvalue, $datafile)
02   {
03   $lines = array();
04   if (is_readable($datafile))
05     { $lines = file($datafile); }
06   while (count($lines) < MAXDATA)
07     { $lines[] = 50; }
08   for ($i = MAXDATA - 1; $i > 0; $i--)
09     { $lines[$i] = $lines[$i-1]; }
10   $lines[0] = $newvalue;
11   if (is_writable($datafile))
12     {
13     $handle = @fopen($datafile, "w");
14     for ($i = 0; $i < count($lines); $i++)
15       { fwrite($handle,
16                trim($lines[$i]) . "\n"); }
17     fclose($handle);
18     }
19   }
```

Die Funktion `insert()` liest die Datei in ein Array (Zeile 03) und schiebt alle Datenwerte um eine Indexposition weiter: Der letzte und damit älteste Datenpunkt fällt weg (Zeile 08 und 09), der neue Wert wird zum ersten Wert (Zeile 10). Sie führt in den Zeilen 06 und 07 auch eine kleine Korrektur durch: Da am Anfang noch keine Werte vorhanden sind, füllt die Routine das Array gegebenenfalls mit 50-Prozent-Werten auf. Schließlich schreibt sie die Daten zurück in die Datei, wobei sie in den Zeilen 11 bis 18 gleich noch Leerzeichen, Tabs und Newlines entfernt:

`lwm_index.php`: Das Webinterface

Für das Webinterface *lwm_index.php* [7] bilden – neben den notwendigen Webseiten-Geschichten wie Header, Body, Eingabeformular und Anzeige – zwei Funktionen den Kern der Langweilometer-Anwendung. Es handelt sich um eine Einfügefunktion für neue Daten sowie eine Funktion, welche die letzten n Eingaben mittelt. Sinn von Letzterem ist, dass nicht wegen einer einzigen Bewertung der Zeiger extrem ausschlägt.

Wie bei C, ermöglicht auch bei PHP die Definition von Konstanten leichte Anpassung. Das Programm legt den Binary-Pfad, eine Datendatei und die Anzahl Daten in der Datendatei fest:

```
01 define("SETDAC", "/home/pi/lwm_setdac");
02 define("DATAFILE", "/home/pi/boring");
03 define("MAXDATA", "11");
```

Die Einfügefunktion `insert()` soll die Datendatei bei der Datenspeicherung einerseits nicht zu groß werden lassen, andererseits Aktualität widerspiegeln. Daher speichert sie Daten rollierend: Wenn einen neues Datum hinzukommt, löscht sie das älteste. Dieser Algorithmus ist ein einfacher Ringpuffer, realisiert auf einem Array (siehe Kasten „`lwm_index.php`: Einfügefunktion“).

Wie lange das Ganze gut geht, hängt davon ab, wie viele Schreibzyklen die SD-Karte verkraftet. Dieser Wert hängt von der Qualität der SD-Karte ab – ein Chip-Speicher ist halt doch kein vollwertiger Ersatz für eine Festplatte. Deshalb ist zu empfehlen, sobald alles läuft, die SD-Karte in eine Image-Datei auf dem PC oder Laptop zu kopieren. Dann beschränkt sich die Wartungsarbeit beim Versagen der SD-Karte auf das Kopieren des Images auf eine neue SD-Karte.

Bei der zweiten Funktion hat der Autor zum Glätten der Werte mit zwei Verfahren experimen-

tiert. Zum einen bot sich wegen der sowieso schon rollierenden Datei der gleitende Mittelwert an: Da beim Eintreffen eines neuen Werts jeweils der älteste wegfällt, ergibt sich immer ein Mittel über die letzten n Werte. Dieses Verfahren läuft zwar zufriedenstellen, lässt sich aber von Ausreißern beeinflussen.

`lwm_index.php`: Median-Berechnungsfunktion

```
01 function calculate_median($datafile)
02   {
03   $lines = array();
04   if (is_readable($datafile))
05     { $lines = file($datafile); }
06   while (count($lines) < MAXDATA)
07     { $lines[] = 50; }
08   $aktual = $lines[0];
09   for ($i = 0; $i < MAXDATA; $i++)
10     { $lines[$i] = trim($lines[$i]); }
11   sort($lines);
12   $min = $lines[0];
13   $max = $lines[MAXDATA-1];
14   if(MAXDATA % 2 == 0)
15     {
16     $mean = ($lines[(MAXDATA/2) - 1]
17              + $lines[MAXDATA/2])/2;
18     }
19   else
20     {
21     $mean = $lines[MAXDATA/2];
22     }
23   return array ($aktual, $mean, $min, $max);
24   }
```

Die Funktion `calculate_median()` hat als Parameter den Namen der Datendatei. Sie liest wieder zunächst die Datei in ein Array (Zeile 03), füllt die Daten gegebenenfalls mit 50-Prozent-Werten auf (Zeilen 06 bis 08) und schmeißt Leerzeichen weg (Zeilen 09 und 10). Anschließend sortiert sie die Werte aufsteigend und holt sich die Maxima (Zeilen 11 und 12). Eine Wenn-Funktion unterscheidet: Ist die Zahl der Werte ungerade, berechnet sie den Median als das arithmetische Mittel der beiden mittleren Zahlen (Zeilen 16 und 17). Andernfalls, bei einer ungeraden Zahl der Werte, gilt der wert in der Mitte als Meridian (Zeile 21). Schließlich gibt die Funktion eine Liste von vier Werten zurück – den aktuellen Wert, das Minimum, das Maximum und den Median (Zeile 23).

Resistenter gegen Ausreißer ist der Median. Im Gegensatz zum Mittelwert teilt der Median die Wertemenge in zwei gleich große Teile auf. Daher eignet er sich bei dem Verdacht auf Ausreißer. Für den Betrachter ergibt sich lediglich der Nachteil, dass seine Eingabe nicht immer sofort eine Änderung der Anzeige zur Folge hat, wie es bei gleitenden Durchschnitt normalerweise geschieht. Der Median lässt sich so bestimmen (siehe Kasten „`lwm_index.php`: Median-Berechnungsfunktion“):

- Alle Werte aufsteigend sortieren.
- Ist die Anzahl der Werte gerade, gilt als Median das arithmetische Mittel der beiden mittleren Werte.
- Ist die Anzahl der Werte ungerade, ist die mittlere Zahl der Median.

Der Rest des PHP-Programms ist unspektakulär. Einige Funktionen versuchen, unsinnige Daten bei der Eingabe auszufiltern, andere melden sich, welcher Langeweilepegel gerade erreicht ist (nicht jeder kann ja das Instrument sehen). Die Anzeige ist als waagerechter Balken realisiert, dessen Breite das CSS an den aktuellen Prozentwert anpasst, und der seine Farbe von grün über gelb, orange und rot nach violett ändert (Abbildung 4).

Damit nun nicht Hinz und Kunz in aller Welt, sondern nur aktuell Betroffene einen Wert eingeben, untersucht das Programm die IP-Adresse des Webclient. Es dürfen nur diejenigen eine Wertung

abgeben, deren Client aus dem internen Netz des Veranstaltungsortes kommen. Im Anwendungsfall des Autoren war das das Netz der Fakultät, dessen IP-Adressen mit `10.27.` beginnen. Alle anderen, die zum Beispiel per Wlan im Netz herumgeistern, müssen das geheime Passwort kennen, das als SHA1-Hashwert im Programm gespeichert ist:

```
01 $Client = $_SERVER['REMOTE_ADDR'];
02 if (preg_match('/^10\.27\./', $Client))
03   { $Passwort = $PW; }
```

Schließlich setzt PHP die Instrumentenanzeige per System-Call:

```
01 $cmd = SETDAC . ' ' . $Mittel;
02 system($cmd, $retval);
```

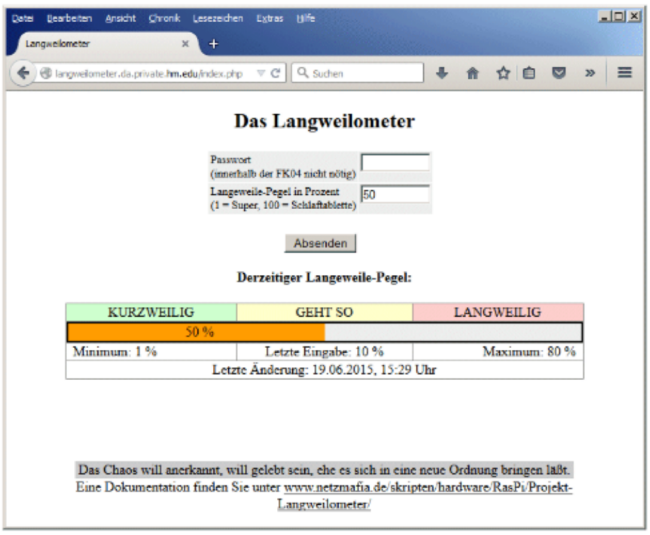

Abbildung 4: Das Webinterface mit Eingabemaske und Langweilometer-Balken.

Epilog

Dieser Artikel ist ein Versuchsballon. Die Mehrheit der UpTimes-Leser sind vermutlich eher dem Software- oder Administrationsbereich zuzurechnen und mit Hardware das letzte Mal möglicherweise vor mehr als 25 Jahren beim Zusammenschrauben eines PCs in Berührung gekommen. Daher kann es gut sein, dass dieser Artikel vollkommen neben der Zielgruppe liegt – noch dazu, weil das Thema auch nicht allzu ernsthaft ist.

Aber auch Claude Shannon, Mathematiker und Begründer der Informationstheorie, hat kleine mechanische Clowns gebastelt. Außerdem keimt derzeit ja das zarte Pflänzchen *Bastler* rund um preiswerte Boards wie Arduino oder Raspberry Pi. (Der Autor mag übrigens den neudeutschen Begriff *Maker*, der heute oft dafür verwendet wird, überhaupt nicht.)

Die Qualität dessen, was da so im Netz, aber auch in Zeitschriften zu finden ist, schwankt extrem. In den 1980er Jahren rekrutierten sich viele Computerbastler noch aus den Radiobastlern, Funkamateuren, HiFi-Enthusiasten und Modellbauern. Ohm, Watt und Volt waren bekannte Begriffe, der Übergang zum Computer nicht so schwer. Die Älteren unter uns erinnern sich vielleicht, dass die ersten Mikrocomputer wie *Altair* oder *Apple 1* anfangs als Bausatz erhältlich waren.

Der Autor stellt sich einfach die Frage, ob von dem geneigten Leser weitere Artikel zum Raspberry Pi gewünscht sind, und ob damit verbundene Hardware-Basteleien eher interessieren oder eher abschrecken. Eine kurze Mail an <***@netzmafia.de> mit dem Betreff *UpTimes* wäre schön. Auch Anregungen sind willkommen. *(Anm. d. Red.: Ja, die E-Mail-Adresse stimmt so und funktioniert, grinst der Autor auf unsere verwunderte Nachfrage, verweisend auf [8]. :-))*

Links

[1] Webserver auf dem Raspberry Pi einrichten:
http://www.netzmafia.de/skripten/hardware/RasPi/RasPi_http.html
[2] Digital-Analg-Umwandler *LTC1453*: http://www.linear.com/product/LTC1453
[3] Datenblatt des LTC1453 mit Hardware-Bug: http://www.linear.com/docs/3265
[4] Quelltext *lwm_setdac.c* für den Digital-Analog-Umwandler:
http://www.netzmafia.de/skripten/hardware/RasPi/Projekt-Langweilometer/lwm_setdac.c
[5] Dokumentation "GPIO beim Raspberry Pi per Programm ansteuern":
http://www.netzmafia.de/skripten/hardware/RasPi/RasPi_GPIO_C.html
[6] Quelltext *lwm_frontled.c* für die LED:
http://www.netzmafia.de/skripten/hardware/RasPi/Projekt-Langweilometer/lwm_frontled.c
[7] Quelltext *lwm_index.php* für das Webinterface:
http://www.netzmafia.de/skripten/hardware/RasPi/Projekt-Langweilometer/lwm_index.php.txt
[8] Jürgen Plate: Shellskripte mit Aha-Effekt; Erlaubte Zeichen in E-Mail-Adressen. In: UpTimes 2013-02, S. 22: https://www.guug.de/uptimes/2013-2/index.html