Der Weihnachtstannenbaum der Fachschaft 720 – Dekoration zum selber Programmieren!

Author: Timo Maschwitz

1 Grundlegende Funktionsweise

Der Tannenbaum besitzt einen Mikrocontroller (Bauteilbezeichnung: `STM32C011J6M6`) und acht Leuchtdioden (en. ***l**ight **e**mitting **d**iode*, kurz: LED) in insgesamt drei Farben. Mit Hilfe des Mikrocontroller werden die LEDs angesteuert, also ein- oder ausgeschaltet. Das Gehäuse des Mikrocontrollers besitzt acht elektrische Anschlüsse (en. sing. *pin*). Um eine einfache Fehlerfindung (en. *debugging*) zu ermöglichen, werden zwei Pins am Gehäuse des Mikrocontrollers verwendet. Für die Energiezufuhr werden zwei weitere Pins benötigt. So bleiben vier Ein- und Ausgänge (en. sing. ***g**eneral **p**urpose **i**n- and **o**utput*, kurz: GPIO) zur Ansteuerung der acht LEDs. Aus diesem Grund steuert jeder GPIO zwei LEDs an. Damit weder durch die LED noch durch den GPIO zu viel Strom fließt, ist an jeder LED ein Widerstand verbaut. Dieser begrenzt den Strom auf etwa 1 mA.

Da es im Moment des Einschaltens eines Pins dennoch zu kurzfristigen Stromspitzen kommt, kann die Versorgungsspannung am Mikrocontroller zusammenbrechen, was zum Neustart des Programms führen kann. Um die Spannung zu stabilisieren ist direkt am der Spannungsversorgungspin des Mikrocontrollers ein sogenannter Kondensator verbaut, der als Puffer dient. Der gesamte Schaltplan ist in nachfolgenden Abbildung 1 zu sehen.

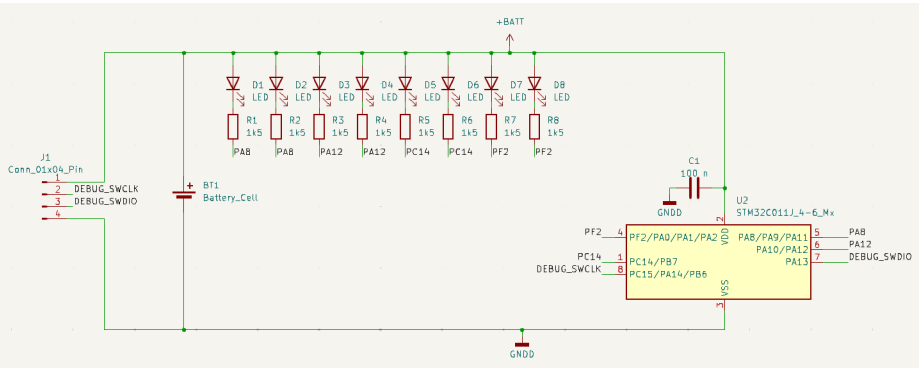

Abbildung 1: Schaltplan des Weihnachtstannenbaums

1.1 Die Welt ist nicht schwarz-weiß – wie gelingen Zwischentöne?

Mit dem beschriebenen Aufbau kann nur ein Blinken der LEDs erzeugt werden: sie können eingeschaltet, oder ausgeschaltet sein. Der Mikrocontroller bietet also nicht die Möglichkeit die LED halb einzuschalten. Dennoch kann der Eindruck erzeugt werden, dass die LED „halb hell“ ist. Dazu wird ein Trick angewendet, der auch in vielen anderen Bereichen angewendet wird: die sogenannte Pulsdauermodulation (en. ***p**ulse **w**idth **m**oulation*, kurz: PWM).

Dazu wird ein Signal erzeugt, das periodisch zwischen den Zuständen *ein* und *aus* wechselt. Die Dauer die es dabei in einem der Zustände (z. B. *ein*) verbleibt entscheidet darüber, wie hell das Leuchten der LED wahrgenommen wird. Damit die LED nicht flackert, muss die Periodendauer kurz genug sein z. B. 1 ms, wie in Abbildung 2 (S. 2) dargestellt.

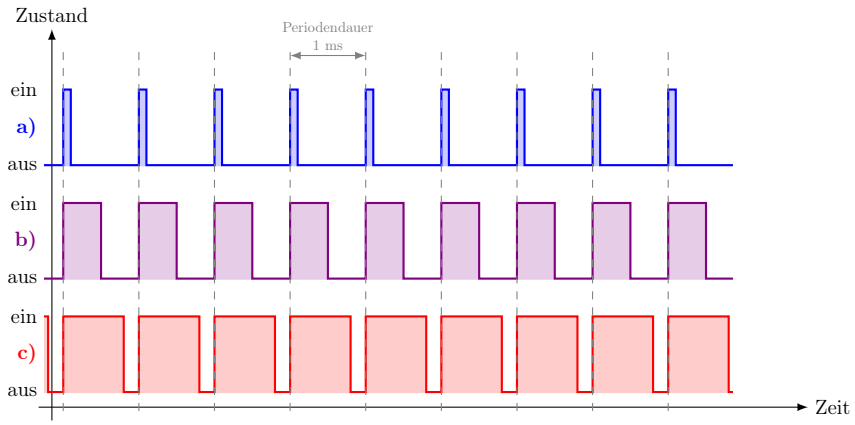

Abbildung 2: PWM-Signale für eine wahrgenommene Helligkeit a) von 10 %, b) von 50 % und c) von 80 %.

Ein solches Signal kann auf unterschiedliche Weise erzeugt werden. In der digitalen Domäne wird dazu ein Taktsignal (z. B. mit einer Periodendauer von 0,001 ms) auf ein Zählwerk gegeben, dass sich automatisch zurück setzt, wenn ein Zählerstand von 999 erreicht wird. Eine zweite Komponente vergleicht den Zählerstand mit einer vorgegebenen Zahl und entscheidet danach, ob der Ausgabewert *ein* oder *aus* ist. Wird definiert, dass der Ausgabewert *ein* ist, wenn der Zählerstand kleiner ist als die Vergleichszahl, wird die Dauer in der das Ausgangssignal *ein* ist länger sein, je größer die Vergleichszahl ist.

Eine PWM kann in der Programmiersprache C wie folgt abbildet werden, vorausgesetzt die Funktion `GPIO_write()` ist bereits definiert:

```
uint16_t zaehlerstand = 0;
uint16_t vergleichswert = 300;

while(1) // main-Schleife
{
    if (zaehlerstand > vergleichswert) // Wenn der Zaehlerstand groesser als der Vergleichswert ist,
    {
      GPIO_write(gpio_name, 0);        // schalte einen GPIO-Ausgang aus.
    }
    zaehlerstand++; // Erhoehe den Zaehlerstand um 1
    if (zaehlerstand >= 1000)   // Wenn der Zaehlerstand den Maximalwert erreicht hat,
    {
      zaehlerstand = 0;          // setze den Zaehlerstand zurueck und
      GPIO_write(gpio_name, 1); // schalte einen GPIO-Ausgang ein.
    }
}
```

Diese Art der Implementierung wird Software-PWM genannt und kann im Prinzip immer sehr einfach angewendet werden. Jedoch ist die Periodendauer nicht stabil (aus Laufzeitgründen) und es bleibt keine Rechenzeit mehr für andere Aufgaben zur Verfügung. Die Taktrate kann stark sinken wenn weitere Rechenschritte durchgeführt werden, sodass die LED beginnen würde zu flackern.

Deshalb besitzt der Mikrocontroller hardwareseitig ein Zählwerk das sich selber zurück setzen kann und einen Vergleichsbaustein, der den Ausgangswert verändert, abhängig von seiner Konfiguration und dem Vergleichswert. Dazu werden entsprechende Register im Mikrocontroller beschrieben. Register sind spezielle Speicherstellen im Mikrocontroller. Jedes Register wird im Referenzhandbuch erläutert. Aufgrund der umfangreichen Funktionalitäten des Mikrocontrollers wäre eine Einarbeitung sehr aufwendig.

Es bietet sich daher an, die Programmiersoftware des Herstellers zu nutzen, in der mit Hilfe einer graphischen Benutzeroberfläche (en. ***g**raphical **u**ser **i**nterface*, kurz: GUI) die übersichtliche Konfiguration der Register möglich ist. Die Software heißt *STM32CubeIDE* und kann kostlos von der Website der Firma ST Microelectronics heruntergeladen werden.

2 Programmieren und die Programmierschnittstelle

Zum Programmieren wird ein Programmiergerät benötigt. Dabei gibt es unterschiedliche Geräte, unter anderem das Programmiergerät von ST Microelectronics (ST-Link), aber auch Alternativprodukte mit dem gleichen Namen zu einem geringeren Preis und mit der gleichen Funktionalität. Mit Verbindungskabeln müssen die Anschlüsse des Programmiergeräts mit denen des Weihnachtstannenbaums verbunden werden. Dabei ist zu beachten, dass die Signalnamen die gleichen sind, also `VDD`, `SWCLK`, `SWDIO` und `GND`. Um die Verbindungskabel an den Weihnachtstannenbaum anschließen zu können, wird eine vierpolige Sockelleiste mit einem Rastermaß von 1,27 mm benötigt Die im vorangegangenen Unteranschnitt erwähnte Software wird genutzt, um den aus dem Konfigurationsmenü erstellten C-Code in eine Binärdatei zu erzeugen und diese dann auf dem Mikrocontroller zu schreiben. Zusammen sieht das wie folgt aus:

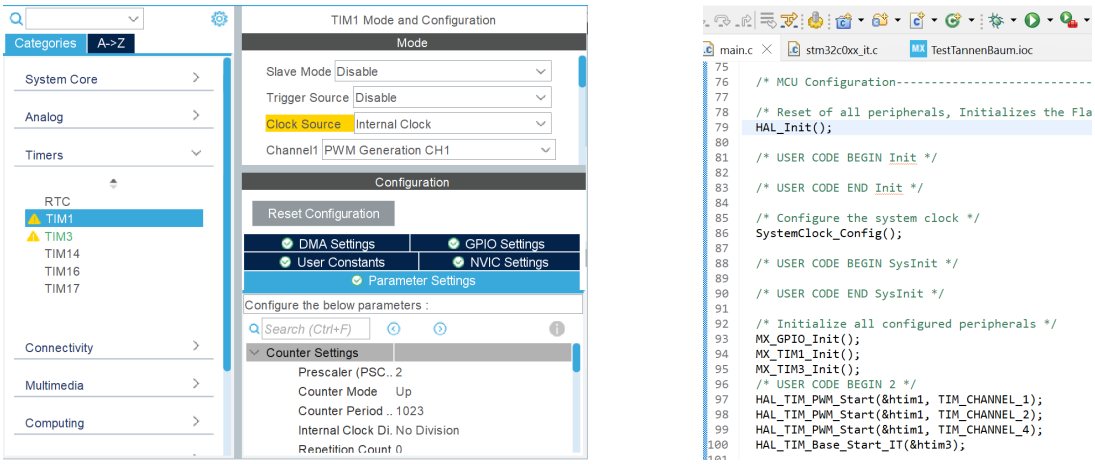

Abbildung 3: Bildschirmausschnitte von der Programmierumgebung *STM32CubeIDE*. Links: Konfiguration der PWM, rechts: Ausschnitt aus dem C-Programm zur Konfiguration der Hardware. Die Schaltfläche am oberen Bildrand mit dem Käfer erzeugt die Binärdatei und schreibt diese auf den Mikrocontroller.

2.1 Das Hardware Abstraction Layer

Die Programmierung der Register ist theoretisch vollständig manuell möglich. Wie in Abschnitt 1.1 (S. 1) beschrieben, kann dazu die Programmiersoftware genutzt werden; sie generiert automatisch C-Code aus den Konfigurationen. Damit weitere Veränderungen am Code möglich und für Menschen nachvollziehbar sind, wird das sogenannte *Hardware Abstraction Layer* (kurz: HAL). Dazu werden die Adressen der Register und deren Einstellungsmöglichkeiten als Begriffe und Namen abstrahiert. Auf diese Weise kann z. B. Timer 1 konfiguriert werden, ohne dass dem Nutzer die genaue Adresse der benötigten Register (Adressbereich `0x4001 2C00 - 0x4001 2FFF`) bekannt ist.
Das HAL gibt auch die Möglichkeit durch vorgefertigte Funktionen die Hardware einfacherer nutzen zu können. So ist zum Beispiel das ein- bzw. ausschalten einer LED am GPIO „A2“ mit der folgenden Funktion möglich:

```
HAL_GPIO_Write(GPIOA, GPIO_PIN_12, GPIO_PIN_RESET);
HAL_GPIO_Write(GPIOA, GPIO_PIN_12, GPIO_PIN_SET);
```

Dabei bedeutet in diesem Beispiel der Begriff `GPIO_PIN_RESET` nichts anderes als *null* und `GPIO_PIN_SET` bedeutet *eins*.

2.2 Ein tieferer Einstieg – wie kann die Helligkeit verändert werden?

Mit den bisherigen Schritten können die LEDs statisch mit einer Helligkeit leuchten. Ähnlich zur Software-PWM könnte auch die Leuchtdauer periodisch geändert werden, mit den gleichen Nachteilen bezogen auf Laufzeit, wie zuvor. Hierzu können Interrupts genutzt werden. Da die PWM-Periodendauer ca. 1 ms beträgt, ergibt es keinen Sinn die Helligkeit schneller zu verändern als jede Millisekunde. Außerdem wäre ein schnellerer Helligkeitswechsel als alle ca. 40 ms nicht sinnvoll, da das menschliche Auge nicht so schnelle Veränderungen wahrnehmen kann. Deshalb wird eine Aktualisierungsrate von 50 ms genutzt.
Im Hauptprogramm könnte eine Abfrage eines weiteren Zählers genutzt werden, um zu merken, wann 50 ms vorbei sind. Jedoch würde auch damit Rechenzeit verschwendet werden. Aus diesem Grund wird ein Interrupt (de. „Unterbrecher“) genutzt. In diesem Fall wird Timer 3 dazu wieder mit Hilfe der Programmierumgebung STM32CubeIDE so konfiguriert, dass ein Interrupt alle 50 ms ausgelöst wird. Das Hauptprogramm wird dann unterbrochen. Es wird dann eine Programmsequenz bearbeitet, die frei programmiert werden kann. Es ergibt Sinn diese Unterbrechung so kurz wie möglich zu halten.

In der Interrupt-induzierten Programmsequenz wird ein fest abgespeicherter Wert an das Vergleichswert-Register gesendet. Das kann z. B. so aussehen:

```
void TIM3_IRQHandler(void)
{
    TIM1->CCR1 = 0x1FF;
}
```

Damit ist der Wert jedoch immer der gleiche und es würde zu keiner Veränderung der Helligkeit kommen. Ein weiterer Schritt ist nötig. Dazu wird ein sogenanntes Array (de. „Datenfeld“), in dem unterschiedliche Werte nacheinander abgespeichert sind, angelegt. Mit Hilfe einer Index-Variable kann dann ein bestimmter Wert aus dem Array abgerufen werden. Danach muss dieser Index inkrementiert, also erhöht werden:

```
uint16_t daten_feld[] = {0x000, 0x00F, 0x07F, 0x0FF};
uint16_t index = 0;

void TIM3_IRQHandler(void)
{
    TIM1->CCR1 = daten_feld[index];
    index++;
}
```

Wichtig ist, dass `index` wieder auf `0` zurück gesetzt wird, wenn das Ende des Datenfeldes erreicht ist.

3 Das Wichtigste zum Schluss

Der Weihnachtstannenbaum ist als Deko, aber auch als Programmieranregung gedacht. Er ist so konzipiert, dass es nicht möglich sein sollte, etwas zu beschädigen, wenn mal was in der Programmierung nicht ganz richtig war. – Hier kann das Programmieren ohne Gefahren ausprobiert werden!