REGISTER UNTER STM32F3XX

INDINF06

4. DEZEMBER 2014

STEFAN ERCEG
4AHITT

Inhalt

Github-Link: https://github.com/serceg-tgm/4AHITT-SYT/tree/indinf/IndInf06

Github-Tag: erceg_indinf06

1. Aufgabenstellung

Finden und untersuchen Sie das Datenblatt des STM32F303xx auf die notwendigen Register um die User-LEDs auf dem STM32F3Discovery-Board dauerhaft einzuschalten.

Verwenden Sie die zur Verfügung gestellte VMware-Instanz und die darauf installierten Pakete um diese Aufgabe zu lösen. Halten Sie sich dabei an das Git-Repository dergraaf/stm32f3_minimal (dieses enthält eine brauchbare README). Das erste Demo-Beispiel "blink" sollte ihnen dabei behilflich sein. Analysieren Sie den Code und die verwendeten Register. Welche Bit-Operatoren kommen dabei zum Einsatz und wieso? Was sind GPIO Ports und welche Einstellungen können dabei getätigt werden. Erläutern Sie diese!

Dokumentieren Sie alle notwendigen Schritte und Einstellungen und schreiben Sie ein Protokoll. Dies ist keine Gruppenarbeit!

2. Zeitabschätzung

Teilaufgabe	benötigte Zeit
Manual untersuchen	50 Minuten
Bsp. „Blink“ analysieren	30 Minuten
Bsp. „LEDs dauerhaft einschalten“ implementieren	30 Minuten
Protokoll schreiben	90 Minuten
Gesamt	**200 Minuten (3 h 20 min)**

3. Tatsächlicher Zeitaufwand

Teilaufgabe	benötigte Zeit
Manual untersuchen	65 Minuten
Bsp. „Blink“ analysieren	35 Minuten
Bsp. „LEDs dauerhaft einschalten“ implementieren	25 Minuten
Protokoll schreiben	85 Minuten
Gesamt	**210 Minuten (3 h 30 min)**

4. Arbeitsdurchführung

4.1 Installationen & Konfigurationen in der VM für den Mikrocontroller

Da auf der virtuellen Maschine, welche das Betriebssystem Debian 7 besitzt und uns vom Herr Professor Borko zur Verfügung gestellt wurde, gearbeitet wird, müssen einige Installationen und Konfigurationen vor der Anwendung des Mikrocontrollers durchgeführt werden.

Zuerst wurde die VM gestartet und unter `VM->Removable devices` nachgeschaut, ob der `SGS Thomson STM32 STLink` erkannt wurde.

Folgende Schritte wurden dann in der Schule durchgegangen:

- Die GNU Toos für ARM Embedded Prozessoren sind auf folgender Seite enthalten: [1]. Dort wurde die tar.bz2-Datei für Linux heruntergeladen und im Verzeichnis `/home/serceg/opt` mittels folgendem Befehl entpackt: `tar xfvj archiv.tar.bz2`

- Um die Programme auf den Chip, der sich auf dem Mikrocontrollerboard befindet, zu übertragen, wird ein Github-Repository [2] geklont (bei mir im Verzeichnis `/home/serceg/repositories`):

```
git clone https://github.com/texane/stlink
```

- Das Skript `configure` wird gestartet, welches überprüft, ob das Programm mit der aktuellen Systemumgebung kompatibel ist und welches bestimmte Definitionen beim Makefile hinzufügt: `./configure`
- Falls das Skript `configure` nicht vorhanden ist, wird das Skript `autogen.sh` gestartet: `./autogen.sh`
- Wenn das Durchlaufen des `configure`- oder `autogen.sh`-Skripts fehlerfrei stattgefunden hat, wird der Kompiliervorgang mit folgendem Befehl gestartet: `make`

- Im Verzeichnis /bin wird ein symbolischer Link auf das stlink-Verzeichnis angelegt:

```
ln -s /home/serceg/repositories/stlink stlink
```

- Nun wird zum Home-Verzeichnis des Benutzers (bei mir /home/serceg) gewechselt und die .bashrc-Datei geöffnet: `nano .bashrc`

  Damit der Befehl `stlink` dauerhaft definiert wird, werden folgende Zeilen in der Datei hinzugefügt:

```
if [ -d "/bin/stlink" ] ; then
    export PATH="$HOME/repositories/stlink/:$PATH"
fi
```

- In der Datei 80-embedded-devices.rules, welche sich im Verzeichnis /etc/udev/rules.d befindet, werden folgende Zeilen hinzugefügt:

```
# STM32F3DISCOVERY
SUBSYSTEM=="usb", ATTRS{idVendor}=="0483", ATTRS{idProduct}=="3748",
MODE="0666", SYMLINK+="stm32f3", GROUP="dialout"
```

4.2 Blink-Beispiel in der VM klonen, ausführen und analysieren

Das `blink`-Beispiel, welches auf dem öffentlichen Github-Repository [3] verfügbar ist, wurde in der VM geklont: `git clone https://github.com/dergraaf/stm32f3_minimal`

Da im Repository ebenfalls noch ein anderes Beispiel verfügbar ist, wurde zum Unterverzeichnis blink gewechselt und dort folgende Befehle durchgeführt:

- make build
- make program

  ➔ Da nach dem Ausführen dieses Befehls eine Fehlermeldung auftauchte, dass OpenOCD noch nicht installiert wurde, wurde dieses Package ebenfalls installiert:

  ```
  apt-get install openocd
  ```

Danach konnte das Programm erfolgreich auf den Chip des Mikrocontrollers übertragen werden und eine LED auf dem Mikrocontroller fing zu blinken an.

Nachdem dies funktionierte, schaute ich mir den implementierten Code in der main.c-Datei an. Interessant war dabei die main-Methode:

```
int main (void) {

	init_clock();

	// Struct of PIO Port E
	GPIO_TypeDef *PE = GPIOE;

	// Red LED is PE9
	// set mode to 01 -> output
	PE->MODER = (1 << (9*2));

	// switch LED on (is connected between IO and GND)
	PE->ODR = (1 << 9);

	while(1) {
		PE->ODR = (1 << 9);
		delay_us(500000);
		PE->ODR = (0 << 9);
		delay_us(500000);
	}

}
```

Folgende Schritte wurden bei der main-Methode durchgeführt:

- Clock wurde zu Beginn initialisiert
- ein Pointer der Struct des GPIO Port E wurde erstellt und initialisiert
- MODER, welcher als Port Mode Register dient und somit den Modus (Input/Output – in diesem Fall Output) einstellt, wurde verwendet, damit die LED, welche sich auf Pin 9 befindet, leuchten kann
- ODR, welcher als Output Data Register dient, wurde verwendet, damit Pin 9 auf Pin 1 gesetzt werden kann und die LED somit eingeschaltet wird
- in der while-Schleife, welche eine Endlosschleife darstellt, wurde die LED auf Pin 9 permanent ein- bzw. ausgeschaltet. Mit „delay_us“ werden Wartezeiten zwischen dem Ein- und Ausschalten festgelegt

4.3 Beispiel „LEDs dauerhaft einschalten“

Um ein Programm zu schreiben, welches die LEDs dauerhaft einschaltet, wird das bestehende .c-File vom Github-Repository [3] verwendet und verändert.

Folgende Schritte wurden in der main-Methode durchgeführt, um ein dauerhaftes Einschalten der LEDs zu ermöglichen:

- while-Schleife wurde entfernt, da diese ja das Blinken der LED ermöglicht hat
- vom User Manual für den STM32F3 Microcontroller [4] wurde entnommen, welche Bits gesetzt werden müssen -> 8. bis 15. Bit für die LEDs

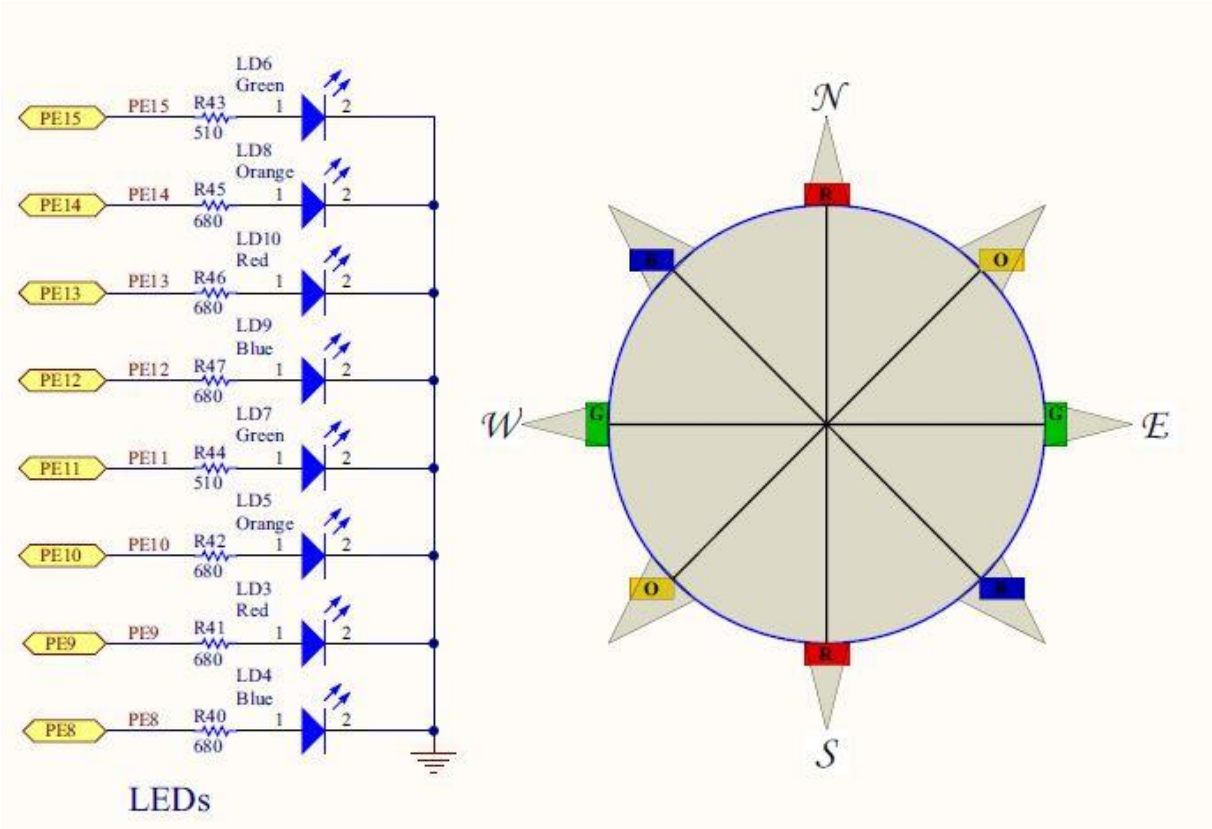


Abbildung 1: Peripherals [entnommen von: STMicroelectronics (2013), UM1570 User Manual (S. 34)]

- damit mehrere LEDs leuchten, wurde beim Setzen der Bits statt einem normalen Zuweisungsoperator ein verodernder Zuweisungsoperator (|=) verwendet
- alle Output-Werte der LEDs wurden auf 1 gesetzt
- das Port Mode Register und Output Data Register wurden aufgrund der Kompaktheit in eine for-Schleife gepackt, welcher vom 8. bis zum 15. Bit durchläuft

Der finale Code sah danach folgendermaßen aus:

```
int main (void) {

	init_clock();

		// Struct of PIO Port E
		GPIO_TypeDef *PE = GPIOE;

		int i;

		for (i = 8; i <= 15; i++) {
			PE->MODER |= (1 << (i*2));
			PE->ODR |= (1 << i);
		}

}
```

Nach dem Builden und Ausführen des Programms, sind die LEDs auf dem Microcontrollerboard nun permanent eingeschaltet.

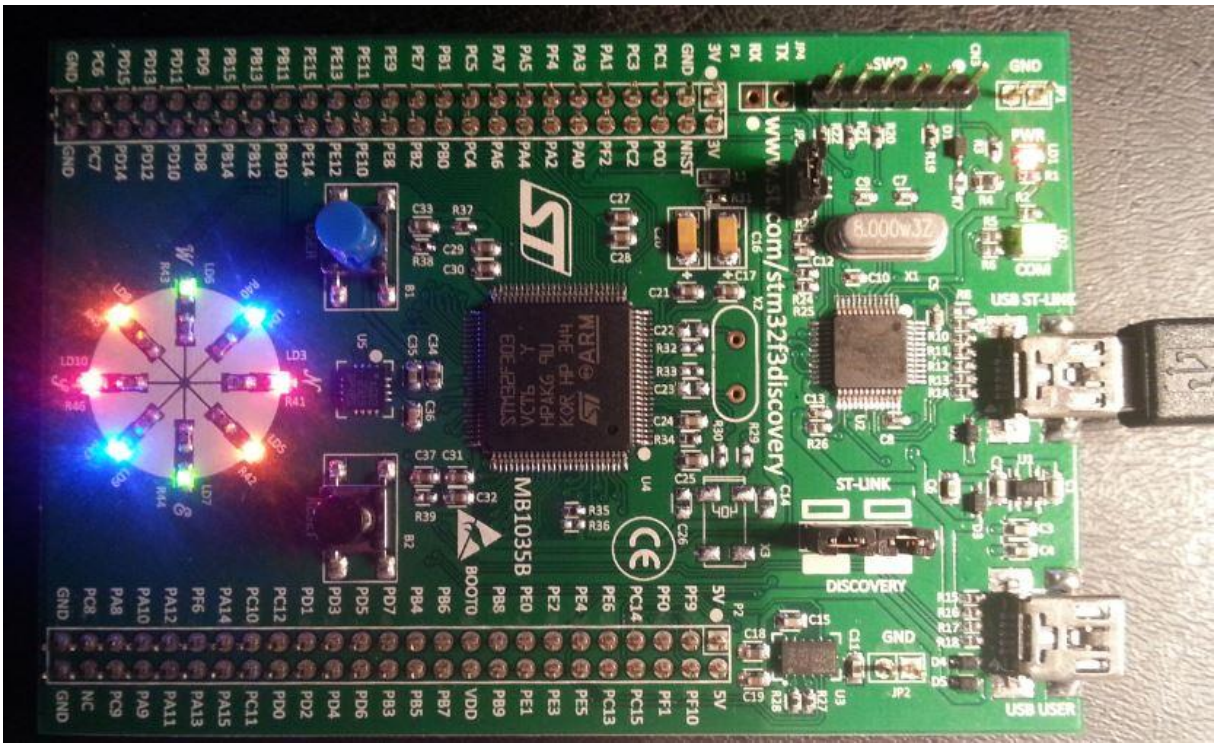


Abbildung 2: Anzeige des Mikrocontrollerboards, auf dem alle LEDs permanent durchleuchten [Stefan Erceg, fotografiert am 04.12.2014]

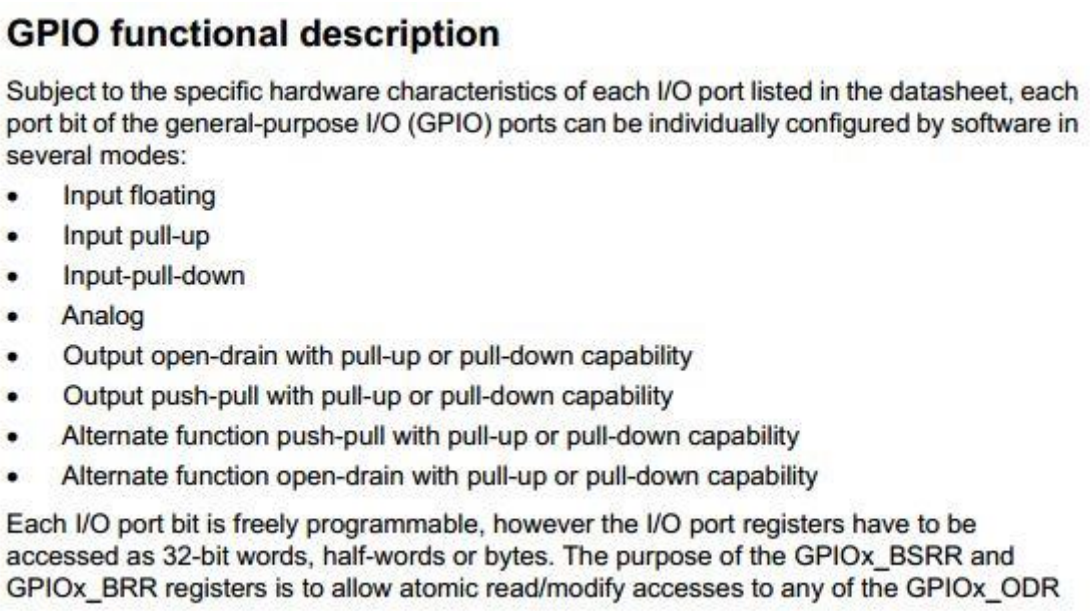
4.4 GPIO Ports

Unter den GPIO (General Purpose Input/Output; dt.: Allzweckeingabe/-ausgabe) Ports versteht man generische Pins an einem integrierten Schaltkreis (IC). Diese können vom Benutzer zur Laufzeit kontrolliert werden, unabhängig davon, ob der Pin als Input- oder Output-Pin definiert wurde. Ein GPIO-Port kennt nur binäre Werte. [5]

Folgendes kann bei einem GPIO Port eingestellt werden:

- GPIO Pins können als Input- oder Output-Pin definiert werden
- GPIO Pins können aktiviert bzw. deaktiviert werden
- Eingabewerte sind lesbar (normalerweise: high = 1, low = 0)
- Ausgabewerte sind sowohl lesbar, als auch schreibbar
- Eingabewerte können oft als Interrupts verwendet werden (normalerweise zum Aufwecken des Geräts)

Folgende Einstellungen können bei GPIO Ports getätigt werden:



GPIO functional description

Subject to the specific hardware characteristics of each I/O port listed in the datasheet, each port bit of the general-purpose I/O (GPIO) ports can be individually configured by software in several modes:

- Input floating
- Input pull-up
- Input-pull-down
- Analog
- Output open-drain with pull-up or pull-down capability
- Output push-pull with pull-up or pull-down capability
- Alternate function push-pull with pull-up or pull-down capability
- Alternate function open-drain with pull-up or pull-down capability

Each I/O port bit is freely programmable, however the I/O port registers have to be accessed as 32-bit words, half-words or bytes. The purpose of the GPIOx_BSRR and GPIOx_BRR registers is to allow atomic read/modify accesses to any of the GPIOx_ODR

Abbildung 3: GPIO functional description [entnommen von: STMicroelectronics (2014), RM0316 Reference Manual (S. 146)]

4.5 Verwendete Bit-Operatoren

Bei den Punkten 4.2 und 4.3 wurden folgende Bit-Operatoren verwendet:

<< ***Linksverschiebungsoperator***

Dieser wird bei dem Beispiel verwendet, um die einzelnen Pins eines Registers zu setzen. Bei diesem Operator werden alle Bits um n Stellen nach links zu verschieben.

Wenn wir z.B. folgende Bits in Binärdarstellung besitzen:

1100 0111 (Ergebnis: 199)

… und wir alle Bits um 3 Stellen nach links verschieben, sieht das Ergebnis in Binärdarstellung folgendermaßen aus:

0001 1100 (Ergebnis: 28)

[7]

| ***ODER-Operator***

Dieser wird bei dem Beispiel verwendet, um mehrere Pins anzusprechen. Bei diesem Operator werden zwei oder mehrere Zahlen verglichen und daraus ein neues Ergebnis generiert.

Die Zahl 1 wird in Binärdarstellung folgendermaßen dargestellt: 0000 0001

Die Zahl 8 wird in Binärdarstellung folgendermaßen dargestellt: 0000 1000

Das Ergebnis in Binärdarstellung lautet daher: 0000 1001 (Ergebnis: 9)

& ***UND-Operator***

Dieser wird nur bei dem Blink-Beispiel in der „init_clock()“-Methode verwendet. Bei diesem Operator werden ebenfalls wie beim ODER-Operator zwei oder mehrere Zahlen verglichen und daraus ein neues Ergebnis generiert.

Die Zahl 2 wird in Binärdarstellung folgendermaßen dargestellt: 0000 0010

Die Zahl 6 wird in Binärdarstellung folgendermaßen dargestellt: 0000 0110

Das Ergebnis in Binärdarstellung lautet daher: 0000 0010 (Ergebnis: 2)

Folgende Zuweisungsoperatoren kamen ebenfalls zur Anwendung:

= „standardmäßiger“ Zuweisungsoperator
!= „verodernder“ Zuweisungsoperator

5. Lessons learned

- zur Kenntnis genommen, dass es sehr wichtig ist, Datenblätter durchzulesen
- gelernt, wann es sinnvoll ist, bestimmte Bitoperatoren, wie z.B. den Linksverschiebungsoperator, zu verwenden
- Skills bei der Microcontrollerprogrammierung erweitert

6. Quellenangaben

[1] Canoncial Ltd. (2004, 2014). GNU Tools for ARM Embedded Processors [Online]. Available at: https://launchpad.net/gcc-arm-embedded/+download [zuletzt abgerufen am 04.12.2014]

[2] Github-User „texane" (2011, 2014). Github-Repository „stlink" [Online]. Available at: https://github.com/texane/stlink [zuletzt abgerufen am 04.12.2014]

[3] Fabian Greif (2013, 2014). Github-Repository „stm32f3_minimal" [Online]. Available at: https://github.com/dergraaf/stm32f3_minimal [zuletzt abgerufen am 04.12.2014]

[4] STMicroelectronics (February 2013). UM1570 User manual [Online]. Available at: http://www.st.com/st-web ui/static/active/jp/resource/technical/document/user_manual/DM00063382.pdf [zuletzt abgerufen am 04.12.2014]

[5] Oracle (2013). General Purpose Input/Output (GPIO) [Online]. Available at: https://docs.oracle.com/javame/config/cldc/rel/3.4/core/da/html/device_access/gpio.htm [zuletzt abgerufen am 04.12.2014]

[6] STMicroelectronics (May 2014). RM0316 Reference manual [Online]. Available at: http://www.st.com/web/en/resource/technical/document/reference_manual/DM00043574.pdf [zuletzt abgerufen am 04.12.2014]

[7] Computer Science – University of Maryland (2003). Bitshift Operators [Online]. http://www.cs.umd.edu/class/sum2003/cmsc311/Notes/BitOp/bitshift.html [zuletzt abgerufen am 04.12.2014]