U23 2017 - Dokumentation zum STM32 Buildenv

22. November 2017

1 Grundsätzlicher Aufbau STM32 Buildenv

`apps/`:
Ordner mit Programmen. Der Ordner enthält einige Beispiele.
`apps/u23/`:
Ordner mit den U23 Lorawan bezogenen Beispielen.
`apps/u23/blinky/`:
Beispielprojekt, auf dessen Basis eigene Applikationen für das Lorabone entwickelt werden können.
`apps/u23/ttn_hello_world/`:
Beispielprojekt, welches das Versenden von Daten über LoRaWAN demonstriert.
`libs/ibm-lmic/`:
IBMs Implementierung des LoRaWAN Macs.
`middlewares/`:
Enthält Middlewares wie den STM32 HAL und den lora-bone BSP.
`middlewares/bsp/lora-bone/`:
lora-bone BSP. Enthält Intialisierungsfunktionen für I2C, SPI, UART passend zum lora-bone.

2 Kompilieren eines Projektes

Alle Beispielkommandos gehen davon aus, dass ihr euch in dem Hauptverzeichnis des Buildenv befindet. Um das Blinky Beispiel zu bauen, gibt man folgendes in die Konsole ein:

```
$ make blinky
```

Um das Blinky Beispiel auf den Microcontroller zu laden:

```
$ make upload-blinky
```

3 Eigenes Projekt anlegen

Um ein eigenes Projekt anzulegen, kann z.B. das Blinky Projekt als Vorlage verwendet werden. Dazu wird der Ordner `apps/u23/blinky` kopiert und Beispielsweise mit dem Namen `apps/u23/neuesprojekt` abgelegt. Als nächstes muss die Datei `apps/u23/neuesprojekt/target.mak` bearbeitet werden:

```
1 # Target file name.
2 TARGET = neuesprojekt
3
4 # List C source files here.
5 CCSOURCES = main.c interrupts.c
```

Nach dem der Name des Projekts eingestellt und die Quelldateien zur Variable CCSOURCES hinzugefügt wurden, muss `apps/u23/target.mak` bearbeitet werden und der Ordner des neues Projektes hinzugefügt werden:

```
1 U23_APPS = blinky ttn_hello_world neuesprojekt
2 SUBDIRS = $(U23_APPS)
3 $(call include-subdirs)
```

Nun kann das neue Projekt mit dem Buildenv verwendet werden.

4 Debugger mit dem Buildenv verwenden

Um den Debugger zu verwenden, müssen zwei Programme gestartet werden. Openocd, welches die Verbindung zum ST-Link USB Stick herstellt und gdb, der eigentliche Debugger. Wir haben die dazu nötigen Kommandos für euch im Buildenv vorbereitet. Um Openocd zu starten gebt ihr folgendes in die Konsole ein:

```
$ make run-openocd-stm32f1
```

In einer zweiten Konsole startet ihr nun den Debugger. Openocd muss vor dem Debugger gestartet worden sein und noch laufen, wenn der Debugger gestartet. Um zum Beispiel unser eben neu erstellstes Projekt zu debuggen, verwenden wir folgendes Kommando:

```
$ make debug-neuesprojekt
```

Nun befindet ihr euch in der Kommandozeile des Debuggers. Nützliche Kommandos:

load läd das Programm in den Flash

run startet das Programm

break [Datei]:[Zeile] setzt einen Breakpoint

delete [Breakpoint Nummer] löscht einen Breakpoint wieder.

continue setzt z.B. nach dem erreichen eines Breakpoints das Programm fort

print [Ausdruck] Zeigt den aktuellen Wert eines Ausdrucks(z.B. einer Variable) an

monitor reset halt löst einen Reset des Microcontrollers aus

backtrace zeigt einen Backtrace(Aufrufhistorie) an

quit beendet den Debugger

Ein laufendes Programm kann im Debugger durch die Tastenkombination Strg+C unterbrochen werden. Einige Anweisungen des Debuggers können abgekürzt werden, z.B. kann ”continute” mit ”c” und ”backtrace” mit ”bt” abgekürzt werden.

4.1 Semihosting

Semihosting erlaubt es unter anderem, Konsolenausgaben im Programm auf dem Microcontroller zu machen und diese dann in der Konsole des Debuggers zu sehen. Dazu müsse einige Zeilen in der `target.mak` des jeweiligen Projektes geändert werden:

```
25 # Uncomment this to enable semihosting
26 LDFLAGS += -specs=rdimon.specs -lrdimon
27 # Comment this when enable semihosting
28 #LDFLAGS += -specs=nosys.specs
29 LDFLAGS += -mthumb -mcpu=cortex-m3
30 LDFLAGS += -Wl,-T$(ROOT)/misc/linker/f1/STM32F103XB_FLASH.ld,-Map,$(SELF_DIR)/$(TARGET).map
```

Jetzt kann die Funktion `printf()` im Programm benutzt werden. Um im Debugger Semihosting zu aktivieren, muss im Debugger das Kommando `monitor arm semihosting enable` eingegeben werden. Das Programm wird nun aber nur noch im Debugger funktionieren. Um das Programm wieder ”stand alone” betreiben zu können, müssen die obrigen Änderungen an der target.mak des Projektes rückgänig gemacht werden.