Vorlesungsskript zu „Vertiefung Programmieren“ Programmentwurf Hinweis

Dozent: Dipl.-Inf. (FH) Andreas Schmidt

Vorgehen & Hinweise

Vorgehensweise

- Bereiten Sie die Linker-Files mit den notwendigen Sections für den Authenticator und die Application vor.
  Hinweis: Zu Beginn ist es von Vorteil, die Memory-Regions noch auf den Standard-Werten zu belassen. Somit müssen noch keine Startup-Files angepasst werden und auch die Vector-Tabelle noch nicht verschoben werden.
- Implementieren Sie die Grundfunktionalitäten für den Authenticator (ohne Starten der Applikation) sowie die Application.
- Sind Authenticator und Application grundlegend umgesetzt, können die Memory-Regions sowie das Starten der Applikation umgesetzt werden.
  - Anpassen der Memory-Regions (inkl. Laden von zusätzlichem ELF Binary im GDB)
  - Erstellen eines neuen Startup-Files für die Applikation (inkl. Remap der Vector-Tabelle) und Einbinden in das Makefile
  - Implementieren der Funktion zum Starten der Applikation im Authenticator (Hinweis: Start-Adresse der Applikation muss ermittelt werden)
    Für Test- und Entwicklungszecke ist es empfehlenswert, das Einlesen des Decryptions Key sowie das Entschlüsseln über bedingte Kompilierung zu deaktivieren.

Quelle: intern

Linker Files

Hinweis

Um Datenbytes durch den Linker in eine Section zu schreiben, kann die Funktion `BYTE()` verwendet werden.

```
.some_section :
{
    . = ALIGN(4);
    BYTE(0x55)

} > some_memory_region
```

In oben gezeigtem Beispiel, wird das Datenbank 0x55 an den Beginn der Section .some_section geschrieben. Abhängig von der Memory Region in die diese Section gelegt wird, werden die Daten auch im ELF Binary gespeichert und auf das Target-System übertragen.

Quelle: intern

Startup-Files

Hinweis

- Da es sich bei dem Authenticator und der Application um zwei unabhängige Software-Binaries handelt, benötigen beide auch ein entsprechendes Startup File
  ➔ .bss und .data Initialisierung
  ➔ Stack Initialisierung und Stack-Vorbereitung

- Die Application wird allerdings nicht über den Reset-Handler aufgerufen, sondern durch den Authenticator. Daher ist es von Vorteil, die Funktion zum Initialisieren von .bss und .data nicht ResetHandler zu nennen.
- Dennoch ist es empfehlenswert, diese Funktion in der Vector-Tabelle der Application an der Stelle des Reset-Handlers einzuhängen. Somit sind die Vector-Tabellen konsistent

Quelle: intern

Remap Vector Table

Hinweise

- Da der Authenticator und die Application unabhängige Software-Binaries sind, können beide keine gemeinsame Vector-Tabelle verwenden
- Beide Binaries liegen an unterschiedlichen Stellen im Flash. Der STM32 erwartet nach einem Reset die Vector-Tabelle an der Adresse 0x00000000 bzw. 0x08000000. Um nun die Vector-Tabelle der Application dem STM32 bekannt zu machen, muss das VTOR Register mit der entsprechenden Offset-Adresse versehen werden.

4.4.4 Vector table offset register (VTOR)

Address offset: 0x08

Reset value: 0x0000 0000

Required privilege: Privileged

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		TBLOFF[29:16]													
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TBLOFF[15:9]							Reserved								
rw	rw	rw	rw	rw	rw	rw									

Bits 31:30 Reserved, must be kept cleared

Bits 29:9 **TBLOFF**: Vector table base offset field.
It contains bits [29:9] of the offset of the table base from memory address 0x00000000. When setting TBLOFF, you must align the offset to the number of exception entries in the vector table. The minimum alignment is 128 words. Table alignment requirements mean that bits[8:0] of the table offset are always zero.
Bit 29 determines whether the vector table is in the code or SRAM memory region.
0: Code
1: SRAM
Note: Bit 29 is sometimes called the TBLBASE bit.

Bits 8:0 Reserved, must be kept cleared

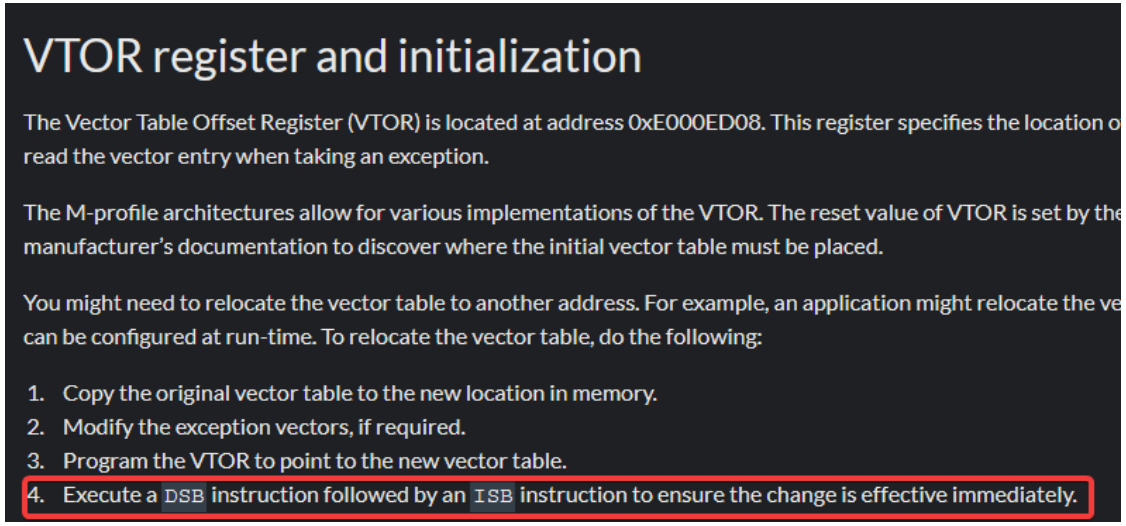
Quelle: intern

Remap Vector Table

Hinweis

- Die Offset-Adresse der Vector-Table kann mit Hilfe des VTOR Registers gesetzt werden.
- Entsprechend der ARM Dokumentation wird empfohlen, nach dem Setzen des Offsets durch die Befehle DSB und ISB

```
SCB->VTOR = 0x12345678;
__DSB();
__ISB();
```



VTOR register and initialization

The Vector Table Offset Register (VTOR) is located at address 0xE000ED08. This register specifies the location o
read the vector entry when taking an exception.

The M-profile architectures allow for various implementations of the VTOR. The reset value of VTOR is set by the
manufacturer's documentation to discover where the initial vector table must be placed.

You might need to relocate the vector table to another address. For example, an application might relocate the ve
can be configured at run-time. To relocate the vector table, do the following:

1. Copy the original vector table to the new location in memory.
2. Modify the exception vectors, if required.
3. Program the VTOR to point to the new vector table.
4. Execute a `DSB` instruction followed by an `ISB` instruction to ensure the change is effective immediately.

Quelle: intern

Enable/Disable Interrupts

Hinweise

- Insbesondere beim Remap der Vector-Table ist darauf zu achten, dass vor dem Remap die Interrupts deaktiviert werden und nachdem die Vector-Tabelle neu gesetzt wurde, die Interrupts wieder aktiviert werden.
- Die kann mit Hilfe der folgenden Funktionen erreicht werden

Aktivieren der Interrupts

```
__enable_irq();
```

Deaktivieren der Interrupts

```
__disable_irq();
```

Quelle: intern

Peripherie-Reset

Hinweise

- Dadurch, dass Authenticator und Application unabhängig voneinander sind, werden auch beide Software-Binaries die Hardware entsprechend initialisieren (z.B. GPIO, Timer, ADC, etc.)
- Um sicher zu gehen, dass die Hardware in beiden Fällen (also der Initialisierung und Konfiguration der Hardware) den gleichen Ausgangszustand haben, sollte vor der Hardware-Initialisierung der Application die gesamte Peripherie des Controllers zurückgesetzt werden.
- Hierzu können folgende beiden Macros verwendet werden. Diese sollten, so wie dargestellt, direkt hintereinander stehen.

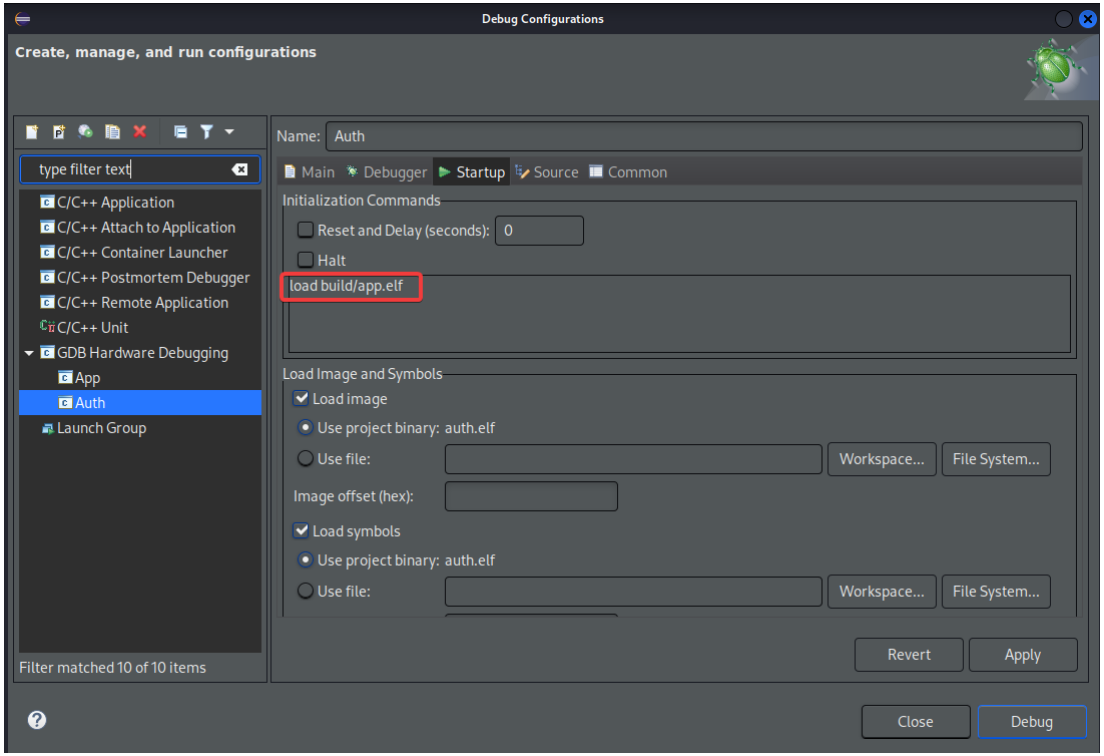
```
__HAL_RCC_AHB1_FORCE_RESET();
__HAL_RCC_AHB1_RELEASE_RESET();
```

Quelle: intern

Debug Config

Hinweis

- Damit beide Binaries (Authenticator und Application) in den Flash-Speicher des Zielsystem geladen werden, muss bei der Debug-Konfiguraiton das jeweils andere ELF Binaries bei den „Initialization Commands angegeben werden.



Quelle: intern

Section Encryption

Vorgehensweise

- Extrahieren der .auth Section als Raw Binary aus dem auth.elf File
- Verschlüsseln der Raw Binary Datei mit encrypt_file.py Script
- Ersetzen der .auth Section im auth.elf File durch das verschlüsselte Raw Binary

Extrahieren der .auth Section

```
arm-none-eabi-objcopy --dump-section .auth=auth_section.bin auth.elf
```

Verschlüsseln des Raw-Binary Files

```
python ../../Scripts/encrypt_file.py -o auth_section.bin.enc auth_section.bin
```

Ersetzen der .auth Section

```
arm-none-eabi-objcopy --update-section .auth=auth_section.bin.enc auth.elf
```

Quelle: intern