Bluescreens für ARMe

Besser debuggen mit Faulthandlern

auf dem Cortex-M3/M4/M7

Einführung

- Was wollen wir erreichen?
- Debugausgaben beim Programmabsturz:

```
################################################
UsageFault! An Adresse: 0x0800126a

Cause: DIVBYZERO

Registerdump:
xPSR: __C__T Syshandler: 0 Thread mode (0x21000000)
R0: 0x00000005   R4: 0x20000010   R8:  0x00000005  R12: 0x00000000
R1: 0x00000000   R5: 0x20000011   R9:  0x00000002  SP:  0x2001bf90
R2: 0x40004800   R6: 0x080015ec   R10: 0x00000000  LR:  0x0800031d
R3: 0x00000064   R7: 0x00000001   R11: 0x00000000  PC:  0x0800126a

vergeigter Befehl: sdiv R0, R0, R1

System halted!
```

```
0800126a <divme>:
 800126a:   fb90 f0f1   sdiv  r0, r0, r1
 800126e:   4770        bx    lr
```

Gliederung

- 1. Faults
  - 1.1 Was sind Faults
  - 1.2 Faults auslösen
  - 1.3 Faults behandeln/ausgeben
  - 1.4 Demo
- 2. printf über SWO

Was sind Faults?

- ARM Kern überwacht seinen Zustand
  - weicht dieser vom Soll ab -> gibt einen Fault

- Faults werden „aufgerufen“ wie Interrupts

Welche Faults gibt es?

- Hard Fault
  - Wenn keiner der unteren 3 aktiviert ist
  - Wenn ein Fault im Fault auftritt
- MemManage Fault
  - MPU sagt: „nein“
- Bus Fault
  - Unerlaubte Zugriffe wie: Befehl aus UART Register laden oder in gesperrten Speicherbereich greifen (wo nichts ist)
- Usage Fault
  - Division durch 0 (abfangen muss aktiviert werden)
  - Undefinierte Befehle
  - Absichtliches asm „udf“ bei Programmfehlern

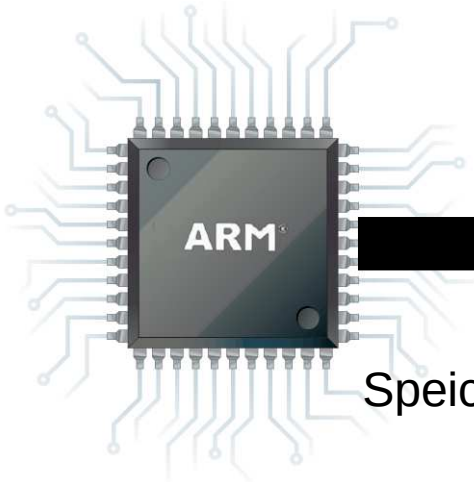
Usage Fault

- Kann zum Programmfehler abfangen genutzt werden

```
static void callb_dma_adc_ext(unsigned int errflags){

	unsigned int error = 0;

	if (errflags & DMA_TEIF){
		deprintf("adcext: Stream transfer error\n");
		error = 1;
	}
	if (errflags & DMA_DMEIF){
		deprintf("adcext: Stream direct mode error\n");
		error = 1;
	}
	if (errflags & DMA_FEIF){
		deprintf("adcext: Stream FIFO error\n");
		error = 1;
	}

	if (error){
		asm("udf");
	}

	if (errflags & DMA_TCIF){
		//selber Callback für TX und RX -> 2 Aufrufe für fertig
		adcext_done++;
		if (adcext_done & 2){
			adcbuf_use = (adcbuf_use+1)&1;
			adcext_done -= 2;
			BaseType_t xHigherPriorityTaskWoken = pdFALSE;
			vTaskNotifyGiveFromISR(aufwachthread,
					&xHigherPriorityTaskWoken);
			portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
		}
	}
}
```

MemManage Faults

ARM Kern

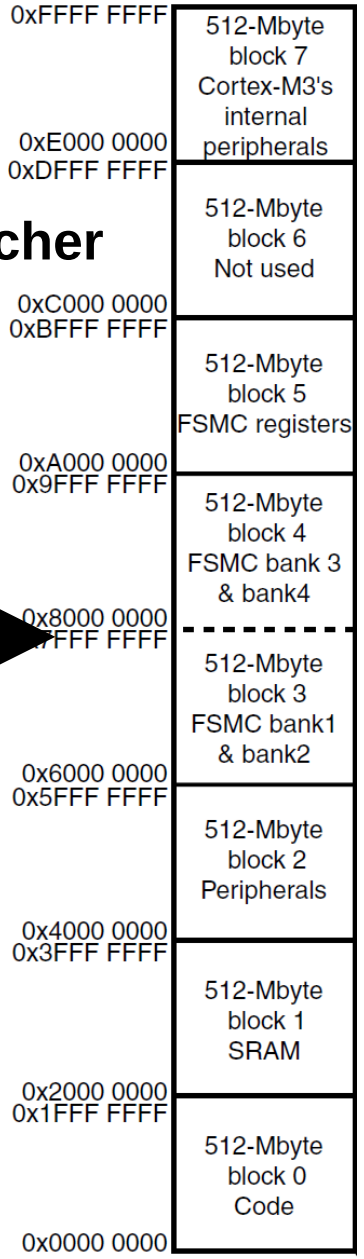


Speicherzugriff

MPU
(Memory Protection Unit)

Fault: „Du kommst hier nicht rein!“

Speicher



MPU

- Besitzt 8* Regionen für erlaubte Zugriffe
  - Der Rest ist gesperrt
  - * 8 Regionen bei M3/M4, geht bis 16 hoch beim M7
- Pro Region einstellbar:
  - Größe in 2^n Schritten (z.B. 128/256/512 (k/M)Byte)
  - Startadresse (muss aligned sein mit der Größe!)
  - Berechtigungen: kein Zugriff, nur lesen, Vollzugriff
  - Ausführbarkeit: liegt hier Code oder Daten
  - Cachebarkeit (erst für den M7 interessant)

Nullpointer abfangen mit MPU

- Erlaubte Regionen sind:
  - Flash, SRAM, Peripherie, ARM Kern Register
- Die Interruptvektortabelle wird zum Flash verschoben:
  - SCB->VTOR = 0x08000000;
  - (da liegt die eh rum und wurde durch die BOOT Pins nur nach 0x00000000 gemapped)

Faulthandler aktivieren

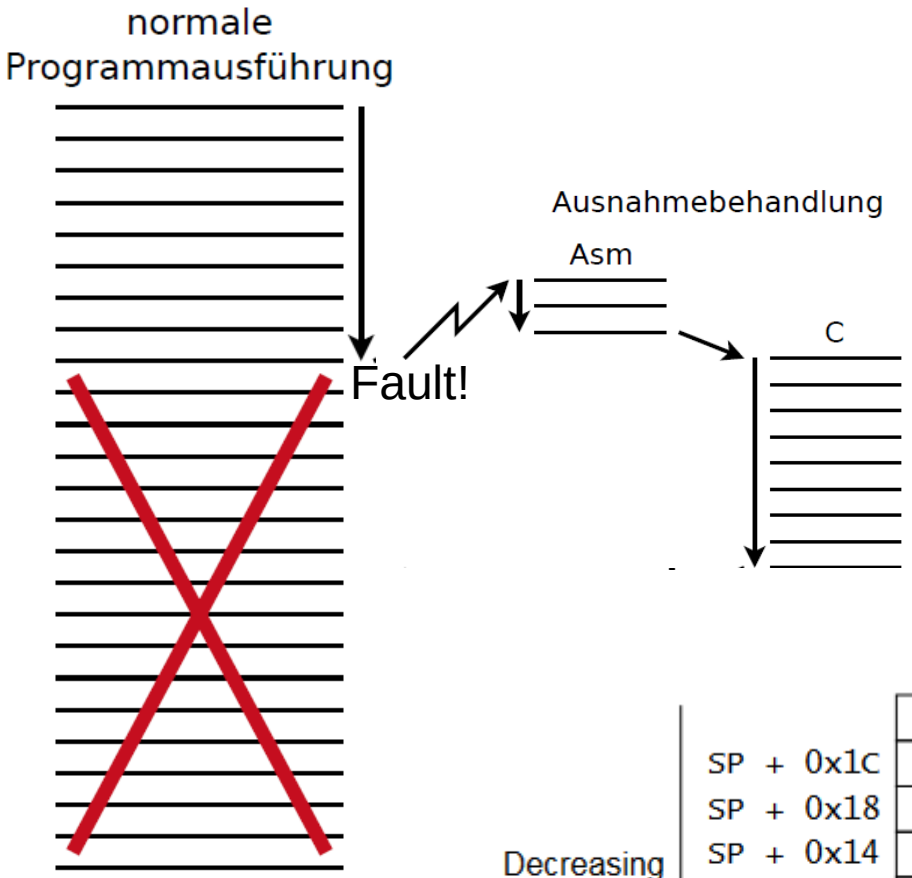
- Nur der Hard Fault ist nach dem Reset aktiv
- Aktivieren der anderen Faults:

```
SCB->SHCSR    |= SCB_SHCSR_USGFAULTENA_Msk
               |  SCB_SHCSR_BUSFAULTENA_Msk
               |  SCB_SHCSR_MEMFAULTENA_Msk;
```

- In der Interruptvektortabelle „anmelden“:

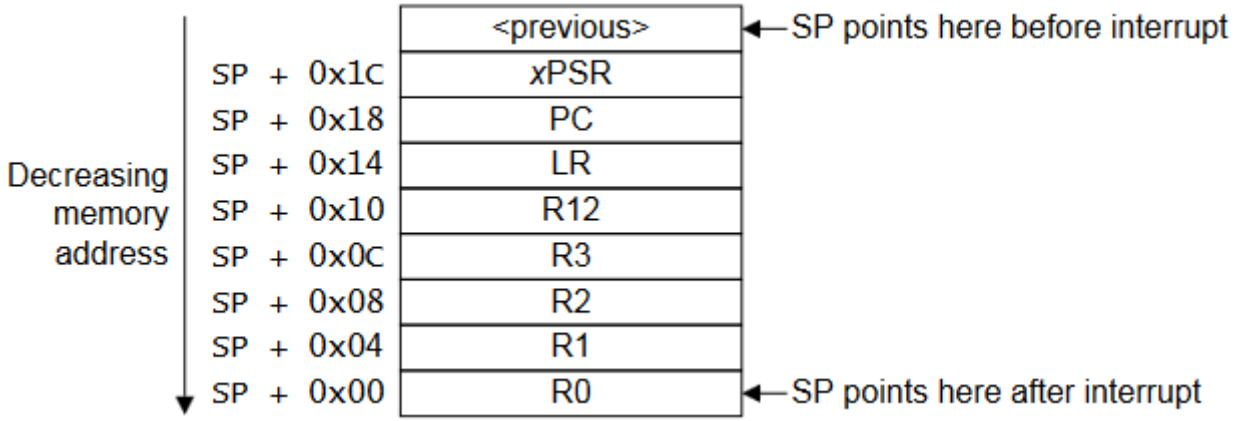
```
.thumb
.syntax unified

.section .text.vectors
_vectors:
      .long 0x2001C000              //Stack ans RAM1 Ende
      .long _startup+1              //Reset Handler
      .long nmihandler+1            //NMI Handler
      .long hardfault_tramp+1       //Hard Fault Handler
      .long mpufault_tramp+1        //MPU Fault Handler
      .long busfault_tramp+1        //Bus Fault Handler
      .long usagefault_tramp+1      //Usage Fault Handler
```

Fault Aufruf



Fault tritt auf:
->NVIC sichert Register auf Stack
->springt in der IVT zum Faulthandler

Wir müssen:
->Register dumpen in ASM für C Code
->Faultstatus ausgeben



Register dumpen

```
.global usagefault_tramp
usagefault_tramp:
      add r0, sp, #(8*4)            //Stackpointer nach r0,
                                    //denn Writebackregister darf nicht in Liste sein
                                    //Mit Abzug von 8*4 wegen Exception HW Stackpush
      stmfd sp!, {r0}               //Stackpointer auf Stack für Regdump
      stmfd sp!, {r4 - r11}
      mov r0, sp                    //Registerarray als Pointer nach C übergeben
      bl usagefault_handler
```

SP + 0x	44	<previous>	
SP + 0x	40	PSR	
SP + 0x	3C	PC	
SP + 0x	38	LR	
SP + 0x	34	R12	
SP + 0x	30	R3	
SP + 0x	2C	R2	
SP + 0x	28	R1	
SP + 0x	24	R0	
SP + 0x	20	SP	
SP + 0x	1C	R11	
SP + 0x	18	R10	
SP + 0x	14	R9	
SP + 0x	10	R8	
SP + 0x	0C	R7	
SP + 0x	08	R6	
SP + 0x	04	R5	
SP + 0x	00	R4	<- Stackpointer

Faulthandler

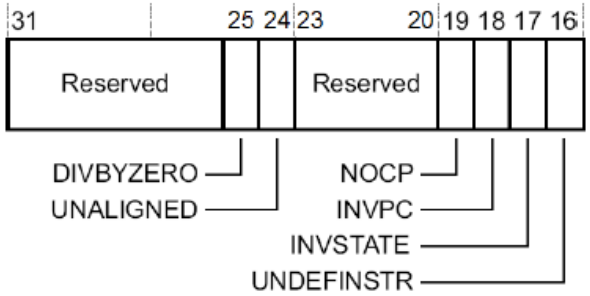
```
void usagefault_handler(struct faultregs *regs){

      sys_notaus();           // System in definierten Zustand bringen z.B. Strom aus, Motor aus
      set_fault_usart();      // UART auf Poll Betrieb umschalten, IRQs stehen nicht mehr zur Verfügung

      deprintf("\n\n#####################################################\n");
      deprintf("UsageFault! An Adresse: %X\n", regs->PC);

      deprintf("\nCause: ");
      scb_print_ufsr_faults();      // Usagefaultregister durch Unterfunktion ausgeben
      deprintf("\n");

      print_regdump(regs);
      deprintf("\nvergeigter Befehl: ");
      disasm_print(regs->PC);

      deprintf("\nSystem halted!");

      while (1);                    // Anhalten, alles ist kaputt
};
```

```
#####################################################
UsageFault! An Adresse: 0x0800126a

Cause: DIVBYZERO

Registerdump:
xPSR: __C__T Syshandler: 0 Thread mode (0x21000000)
R0: 0x00000005   R4: 0x20000010   R8:  0x00000005  R12: 0x00000000
R1: 0x00000000   R5: 0x20000011   R9:  0x00000002  SP:  0x2001bf90
R2: 0x40004800   R6: 0x080015ec   R10: 0x00000000  LR:  0x0800031d
R3: 0x00000064   R7: 0x00000001   R11: 0x00000000  PC:  0x0800126a

vergeigter Befehl: sdiv R0, R0, R1

System halted!
```

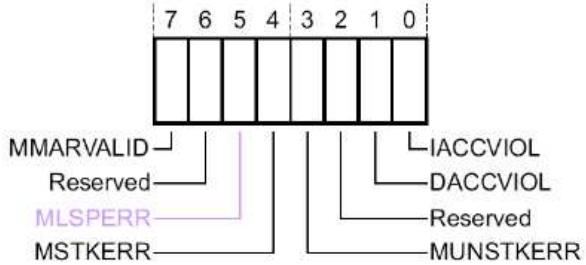
Fault Präzisieren

- Ja schön, ein Registerdump, aber was wars denn genau?
- Dazu gibt es Register im System Control Block

UsageFault Status Register (UFSR)



- MemManage Register:



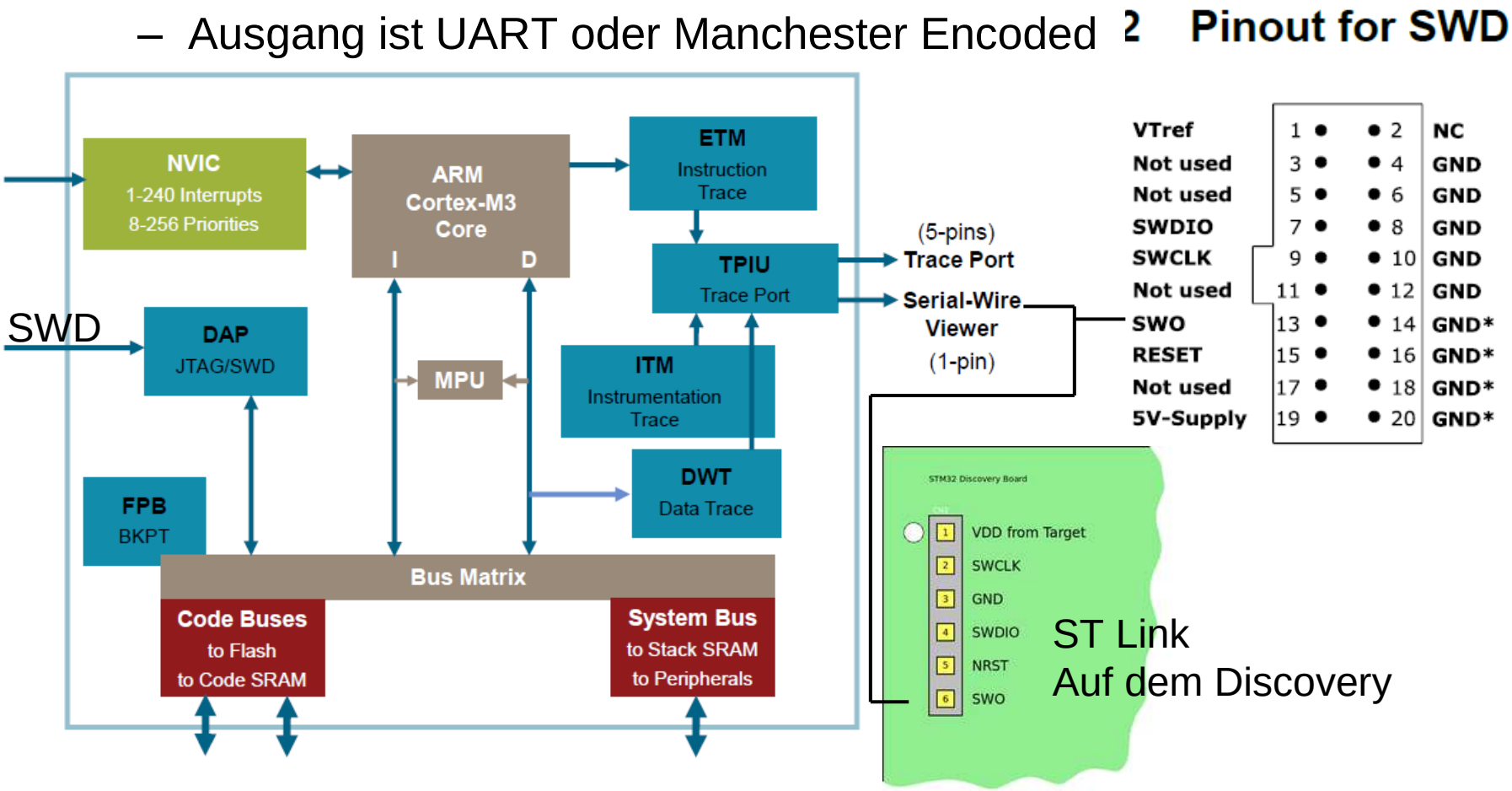
- Wenn MMARVALID = 1 -> genaue Adresse ist bekannt

Demo

```
BOOTIN!
SWO Init OK
SWO Testausgabe OK

Wie tief moechten Sie heute ins Klo greifen?
ud -> div0 (usagefault)
uu -> undefined Instruction (usagefault)
mn -> Nullpointerzugriff (Daten) (MPU Fault)
mi -> Nullpointerzugriff (Instruction) (MPU Fault)
input: uu

##################################################
UsageFault! An Adresse: 0x08000352

Cause: UNDEFINSTR

Registerdump:
xPSR: _ZC__T Syshandler: 0 Thread mode (0x610f0000)
R0: 0x20000ac0   R4: 0x200009bc   R8:  0x00000007  R12: 0x7374616f
R1: 0x00000000   R5: 0x200009bd   R9:  0x00000002  SP:  0x2001bf90
R2: 0x40004800   R6: 0x080080ac   R10: 0x00000000  LR:  0x0800033b
R3: 0x00000075   R7: 0x00000001   R11: 0x00000000  PC:  0x08000352

vergeigter Befehl: udf #1 (0x1)

System halted!
```

SWO

- Serial Wire Output
  - Der kleine Bruder vom TracePort
  - Kann manuell über Code/Registerzugriffe gesteuert werden
  - Ausgang ist UART oder Manchester Encoded



SWO

- SWO aktivieren hat etwas von Black Magic
- Am Ende hat es folgende Funktionen (wie ein UART):

```
/** \brief
  Ausgeben von chars über den SWO Pin per ITM (Tracecell).
*/

#ifndef SWO_H
#define SWO_H

/** \brief SWO print aktivieren
    \param swo_freq gewünschte Frequenz der Ausgabe am SWO Pin in Hz
    \param cpu_freq CPU Takt in Hz
*/
void swo_init(unsigned int swo_freq, unsigned int cpu_freq);

/** \brief Stimulus Port ein oder aus schalten
    \param stimulus_port Stimulus Port der aktiviert werden soll (0 ... 31)
    \param onoff Stimulus Port aktivieren bei !=0, sonst deaktivieren
*/
void swo_manage_stimulus_port(int stimulus_port, int onoff);

/** \brief Zeichen über Stimulus Port ausgeben
    \param stimulus_port Stimulus Port über den ausgegeben soll (0 ... 31)
    \param c auszugebendes Zeichen
*/
void swo_printchar(int stimulus_port, char c);

#endif//SWO_H
```

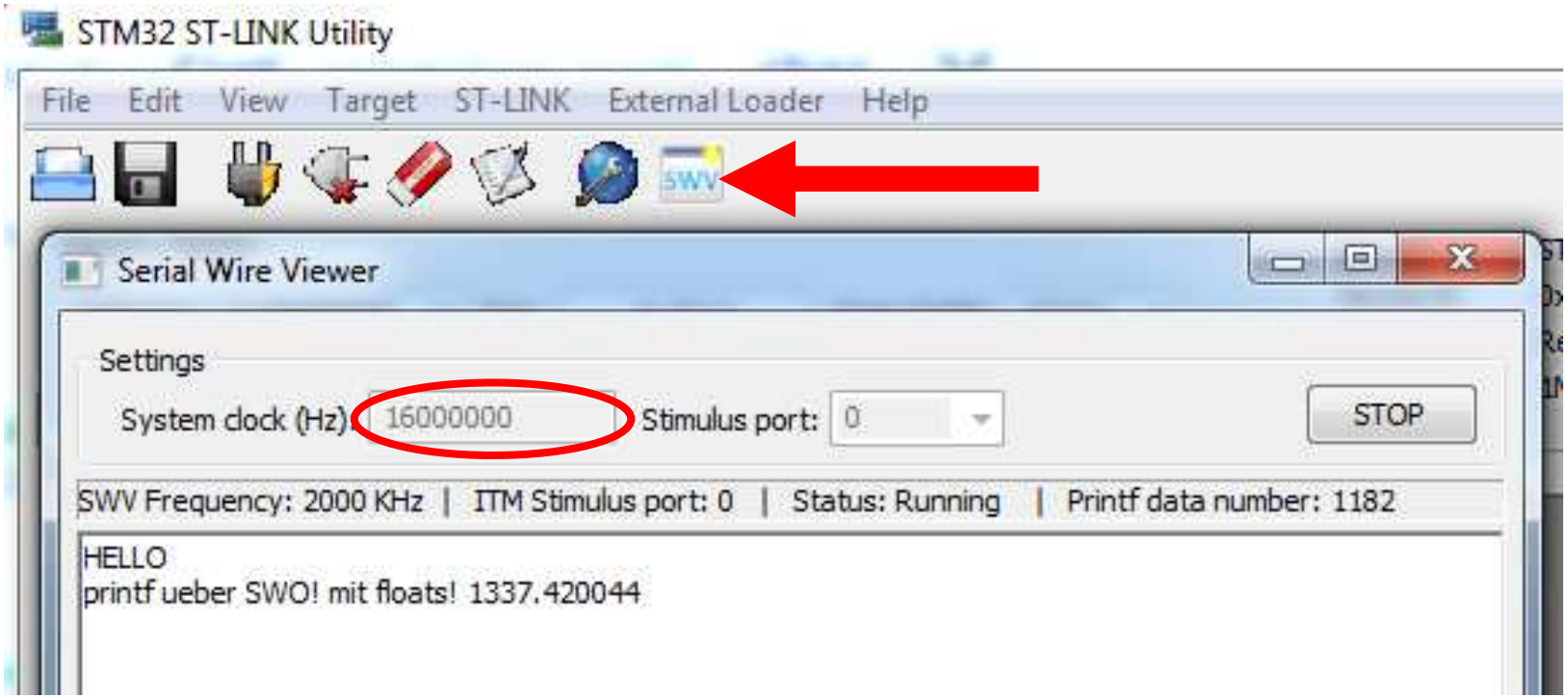
SWO (Black Magic)

```
void swo_init(unsigned int swo_freq, unsigned int cpu_freq) {

  unsigned int swo_prescaler = (cpu_freq/swo_freq) - 1;

  // Trace überhaupt ersteinmal aktivieren
  CoreDebug->DEMCR = CoreDebug_DEMCR_TRCENA_Msk;

  // SWO Pin Protokoll (2: SWO NRZ, 1: SWO Manchester encoding)
  TPI->SPPR = 0x2;

  // SWO Pin Baudrate
  TPI->ACPR = swo_prescaler;

  // ITM Kontrollzugriffe freischalten (sonst ist nur Stimulus Port Zugriff möglich)
  ITM->LAR = 0xC5ACCE55;

  // Tracecontrol: SWO aktivieren, ITM aktivieren
  ITM->TCR = ITM_SWOENA | ITM_ITMENA;

  // Trace Privileg: jeder darf alles
  ITM->TPR = TPR_PRIVMASK;
}
```

printf über SWO

```
void swo_printchar(int stimulus_port, char c){

  // Es gibt nur 32 Stimulus Ports
  if (stimulus_port > 31){
    return;
  }

  // Wenn Trace deaktiviert, dann passiert hier nichts außer eine Endlosschleife
  if (0 == (ITM->TCR & ITM_ITMENA)){
    return;
  }

  // Nur wenn passendes Trace Enable Register Bit gesetzt ist funktioniert die Ausgabe.
  // Sonst Endlosschleife
  if (0 == (ITM->TER & (1 << stimulus_port))){
    return;
  }

  // Wenn die FIFO des Stimulus Port nicht voll ist dann schreiben, sonst warten
  while ( 0 == ITM->STIM[stimulus_port].u32);
  ITM->STIM[stimulus_port].u8 = c;
}

// write Syscall überschreiben damit printf Zeichen über SWO ausgeben kann
int _write(int file, char *ptr, int len){

      (void)file;
      int i;
      for (i = 0; i < len; i++){
            swo_printchar(0, ptr[i]);
      }

      return len;
}
```

printf über SWO



Weiterführende Literatur

IRQ Aufruf Cortex-M:
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0552a/Babefdjc.html

Fault Register:
http://www.keil.com/appnotes/files/apnt209.pdf
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0552a/CIHFDJCA.html

MPU Register:
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0552a/BIHIIJDC.html

SWO:
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0314h/Chdhgebe.html