

Protokoll

Übung 2: „Kochrezept“ eines Echtzeitbetriebssystems

Round Robin

Schritt 1 – Erste Strukturen und Funktionen anlegen

Exemplarisch ds Ergebnis von TaskFillB

Core			33		MOVS	R0,	#0xB0
R0	0x000000B0		34		MOVS	R1,	#0xB1
R1	0x000000B1		35		MOVS	R2,	#0xB2
R2	0x000000B2		36		MOVS	R3,	#0xB3
R3	0x000000B3		37		MOVS	R4,	#0xB4
R4	0x000000B4		38		MOVS	R5,	#0xB5
R5	0x000000B5		39		MOVS	R6,	#0xB6
R6	0x000000B6		40				
R7	0x000000B7		41		MOVS	R7,	#0xB8
R8	0x000000B8		42		MOV	R8,	R7
R9	0x000000B9		43		MOVS	R7,	#0xB9
R10	0x000000BA		44		MOV	R9,	R7
R11	0x000000BB		45		MOVS	R7,	#0xBA
R12	0x000000BC		46		MOV	R10,	R7
R13 (SP)	0x20000490		47		MOVS	R7,	#0xBB
R14 (LR)	0x080019A3		48		MOV	R11,	R7
R15 (PC)	0x08000122		49				
xPSR	0x21000000		50		MOVS	R7,	#0xBC
Banked			51		MOV	R12,	R7
System			52		MOVS	R7,	#0xB7
Internal			53		BX	LR	
Mode	Thread		54		ENDP		
Stack	MSP		55				

Abbildung 1: TaskFillB

Schritt 2 – einfacher Round-Robin Scheduler

Die Struktur des TCB eegibt sich aus den übergebenen Parametern in TaskCreate:

```
typedef struct
{
    uint32_t prio;
    Routine routine;
    uint32_t pStack;
    uint32_t stackSize;
    uint32_t timeSlice;
#ifdef DEBUG
    char * pTaskName;
#endif
}APOS_TCB_STRUCT;
```

Abbildung 2: TCB

Abbildung 3 enthält den Assembler Code des TaskSwitches in PendSV-Handler mit call auf die C-Funktion um den PSP zu setzen. Zur Priorisierung des Interrupts wurde NVIC-Init verwendet, dem die geforderte Priorität mit übergeben wird.

```

PendSV_Handler PROC
EXPORT PendSV_Handler
    mrs r0, psp
    stm r0!, {r4-r7}
    mov r4, r8
    mov r5, r9
    mov r6, r10
    mov r7, r11
    stm r0!, {r4-r7}
    mov r5, r14
    IMPORT APOS_SetPSP
    LDR R0, =APOS_SetPSP
    BLX R0
    mov r14, r5
    adds r0, r0, #16
    ldm r0!, {r4-r7}
    mov r8, r4
    mov r9, r5
    mov r10, r6
    mov r11, r7
    subs r0, r0, #32
    ldm r0!, {r4-r7}
    BX LR
ENDP

```

Abbildung 3: PendSV Handler

Schritt 3 – fertiger Round Robin Scheduler

Anstatt der TaskFillX Funktionen wurde das OS nun mit den vorhandenen Tasks Counter, ... , bis Mandelbrot initialisiert. Die Korrekte Funktion wurde mittels Debugger verifiziert und ist in Abb. 4 festgehalten



Abbildung 4: Scheduling mit 6 Tasks

Schritt 4 – Erweiterung APOS Critical Region

Der TaskSwitch wird nun mit jeder Erhöhung des SysTick-Counters ausgeführt. In der Folge wird das Mandelbrot-Fraktal extrem langsam gezeichnet, da der TimeSlice=SysTick mit 1ms zu kurz für ein zügiges Zeichnen ist.

Die Region-Funktionen bestehen aus dem deaktivieren/reactivieren der Interrupts beim eintreten/verlassen und dem Abfragen der priority mask, um zu testen ob die Region schon betreten wurde.

Schritt 5 – Erweiterung APOS Delay

Dies wurde realisiert, indem sich der aktuelle Task mittels Aufruf von APUS-Delay in der Task-Liste als 'suspended' einträgt, sowie das gewünschte Delay in ticks und den aktuellen systick Wert. der Handler wiederum benutzt diese Info, um den Task erst wieder Ablauf der Delay-Ticks zu rufen.

Schritt 6 – Optimierung Scheduler und Messergebnisse

Eine vollständig gezeichnetes Fraktal benötigt 1 Minute, 14 Sekunden, gemessen mit dem APOS-eigenen Watch-Task. der Zählerstand beträgt dann 3066.

Messungen mittels Logic Analyzer fielen aus, da wir Aufgrund des Lockdowns keinen Zugang zum Analyzer hatten.

Speicherbedarf: die belegten Speicher basieren auf Werten, die per Debugger/Watch Window ausgelesen wurden, bzw. aus dem MAP-File.

Task	max. belegt	reserviert	Overhead
Counter	188	4x100	4x8
Key	232	400	32
LED	304	400	32
Watch	176	400	32
Poti	188	400	32
Mandelbrot	148	400	32
OS	-	678	-

Tabelle 1: Speicherbedarf der einzelnen Tasks in byte

Reserviert wurde jeder Task-Stack mit $4 \times 100 = 400$ byte. Der maximal belegte Speicher wurde ermittelt durch die Position der letzten belegten Speicherstelle im Stack, abzüglich der 32 byte Overhead. Der Overhead jedes Tasks besteht aus dem TCB und ergibt sich somit zu $4 \times 8 = 32$ byte je Task. Als Speicherbedarf für das OS wurde die Speicher-Summe alle Symbole im MAP-File herangezogen, die zum apus.o Object gehören.

1 Source Codes

1.1 APOS.h

APOS.h

```
1  #ifndef __APOS
2  #define __APOS
3
4
5  #include "Services/StdDef.h"
6  // #include "BSP/systick.h"
7  // #include "stm32f0xx_gpio.h"
8  // #include "TaskAll.h"
9  // #include <stdio.h>
10
11 // for debugging
12 #define DEBUG
13
14 typedef void (*Routine)(void);
15
16
17 // Schritt 1:
18 void TaskA(void);
19 void TaskB(void);
20 void TaskC(void);
21 void FillTaskA(void);
22 void FillTaskB(void);
23 void FillTaskC(void);
24
25 // setzt des PendSV Bit um einen TaskSwitch auszuloesen
26 void setPendSV(void);
27
28
29 // Schritt 4:
30 void APOS_EnterRegion(void);
31 void APOS_LeaveRegion(void);
32 int APOS_TestRegion(void);
33
34 // Schritt 5:
35 void APOS_Delay (uint32_t ticks);
36
37 typedef enum { RUNNING, SUSPENDED }APUS_TASK_STATUS;
38
39 typedef struct
40 {
41     uint32_t prio;
42     Routine routine;
43     uint32_t* pStack;
44     uint32_t stackSize;
45     uint32_t timeSlice;
46     uint32_t delay;
47     uint32_t statusTime;
48     APUS_TASK_STATUS status;
49 #ifdef DEBUG
50     char * pTaskName;
51 #endif
52 }APOS_TCB_STRUCT;
53
54
```

```

55 void APOS_Init(void); // Initialisiert das Echtzeitbetriebssystem
56
57 void APOS_TASK_Create( APOS_TCB_STRUCT* pTask, // TaskControlBlock
58     #ifdef DEBUG
59         const char* pTaskName, // Task Name nur fr Debug-Zwecke
60     #endif
61         uint32_t Priority, // Priorit des Tasks (vorerst nicht in
62         void (*pRoutine)(void), // Startadresse Task (ROM)
63         uint32_t* pStack, // Startadresse Stack des Ta
64         uint32_t StackSize, // Gr des Stacks
65         uint32_t TimeSlice // Time-Slice fr Round Robin Scheduling
66     );
67
68 void APOS_Start(void); // Starten des Echtzeitbetriebssystems
69 void APOS_Scheduler(void); // OS Scheduler
70 void APOS_SetPSP(void);
71 void APOS_Delay(uint32_t ticks);
72
73
74 #endif // __APOS

```

1.2 APOS.c

```
APOS.c
1 #include <stdlib.h>
2 #include <stdint.h>
3 #include <string.h>
4 #include "Services/StdDef.h"
5 #include "APOS.h"
6 #include "BSP/systick.h"
7 #include "stm32f0xx_gpio.h"
8
9 #define DEBUG
10
11 static const uint32_t maxTasks = 10;
12 static uint32_t numTasks = 0;
13 static uint32_t currentTask = 0;
14 static APOS_TCB_STRUCT* pTasks[maxTasks];
15
16
17 static void copyTasks(APOS_TCB_STRUCT** source, APOS_TCB_STRUCT** dest, uint32_t size)
18 {
19     for(int i = 0; i < size; i++)
20     {
21         (*dest)[i] = (*source)[i];
22     }
23 }
24
25 static BOOL APOS_Running(void);
26
27 void APOS_Init(void) // Initialisiert das Echtzeitbetriebssystem
28 {
29     // NVIC_InitTypeDef NVIC_InitStructure;
30     // NVIC_InitStructure.NVIC_IRQChannel = PendSV_IRQn;
31     // NVIC_InitStructure.NVIC_IRQChannelPriority = 0xF0; // hchster Wert: kleinste Prio // 0xF im obere
32     //
33     // NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
34     // NVIC_Init(&NVIC_InitStructure);
35     //
36     // NVIC_SetPriority(PendSV_IRQn, 0xF0 );
37
38     for(int i = 0; i < numTasks; i++)
39     {
40         pTasks[i] = NULL;
41     }
42 }
43
44 static void APOS_STACK_INIT(APOS_TCB_STRUCT* pTask)
45 {
46     pTask->pStack[pTask->stackSize-1] = 0x01000000; // xPSR ( thumb mode )
47     pTask->pStack[pTask->stackSize-2] = (uint32_t)pTask->routine; // PC
48     pTask->pStack[pTask->stackSize-3] = (uint32_t)APOS_Scheduler; // LR
49     for(int i = 0; i < 5; i++)
50     {
51         pTask->pStack[pTask->stackSize-i-4] = i+1; // r12, r3, r2, r1, r0
52     }
53     pTask->pStack = pTask->pStack + pTask->stackSize-7-1;
54 }
55
56 void APOS_TASK_Create( APOS_TCB_STRUCT* pTask, // TaskControlBlock
57                       #ifdef DEBUG
58                       const char* pTaskName, // Task Name nur fr Debug-Zwecke
59                       #endif
60 )
```

```

58         uint32_t Priority,           // Priorit des Tasks (vorerst nicht in
59         void (*pRoutine)(void),     // Startadresse Task (ROM)
60         uint32_t* pStack,           // Startadresse Stack des Ta
61         uint32_t StackSize,        // Gr des Stacks
62         uint32_t TimeSlice         // Time-Slice fr Round Robin Scheduling
63     )
64 {
65     if(pRoutine == NULL)           // Abbruch wenn keine gltige Funktions
66         return;
67
68     if(!pTask)                     // notwendigen Speicher fr T
69         pTask = calloc(sizeof(APOS_TCB_STRUCT), 1);
70
71
72     #ifdef DEBUG
73     if(!pTaskName)
74         return;
75     pTask->pTaskName = calloc(sizeof(char), strlen(pTaskName));
76     strcpy(pTask->pTaskName, pTaskName);
77     #endif
78     pTask->prio = Priority;          // Task-Daten in den control block bern
79     pTask->routine = pRoutine;
80     pTask->pStack = pStack;
81     pTask->timeSlice = TimeSlice;
82     pTask->stackSize = StackSize / 4;
83     pTask->status = RUNNING;
84     pTask->statusTime = SysTick_GetTick();
85     pTask->delay = 0;
86     APOS_STACK_INIT(pTask);
87     if(numTasks < maxTasks) {
88         pTasks[numTasks] = pTask;   // in der Task-Liste eintragen
89         numTasks++;
90     }
91 }
92
93 void APOS_Start(void)             // Starten des Echtzeitbetriebssystems
94 {
95     APOS_SetPSP();
96     __set_CONTROL(2);             // [0]=0    privileged mode um IRQs enable/disable zu koennen
97                                     // [1]=1 thread mode - Alternate stack pointer PSP is used.
98     APOS_Scheduler();             // Scheduler in Endlos-Schleife ausfhren
99 }
100
101 void APOS_Scheduler(void)
102 {
103     static int lastTick = 0;
104     while(1)
105     {
106         int tick = SysTick_GetTick();
107         if(lastTick == 0)
108             lastTick = tick;
109
110
111         if((tick - lastTick) >= pTasks[currentTask]->timeSlice || !APOS_Running()) // wenn timeSlice des aktuellen
112             //if((tick - lastTick) > 0) // wenn SysTick erhht wurde
113             { currentTask++; // auf nsten Task schalten
114               currentTask %= numTasks;
115               lastTick = SysTick_GetTick();
116               setPendSV(); // PendSV Handler auslsen

```



```

117         } else
118         {
119             pTasks[currentTask]->routine();           // aktuelle Task-Funktion callen
120         }
121     }
122 }
123
124 void APOS_SetPSP(void)
125 {
126     __set_PSP((uint32_t)pTasks[currentTask]->pStack);
127 }
128
129 void setPendSV(void)
130 {
131     // NVIC_SetPendingIRQ(PendSV_IRQn); ... wirkungslos, beobachtet beim debuggen
132     SCB->ICSR |= SCB_ICSR_PENDSVSET_Msk;
133 }
134
135 // betreten einer critical region durch deaktivieren aller Interrupts
136 void APOS_EnterRegion() // um einen task switch zu verhindern
137 {
138     // uint32_t primask = __get_PRIMASK();
139     __disable_irq();
140     // primask = __get_PRIMASK();           ...zum debuggen, ob IRQ wirklich deaktiviert wurde
141                                           // nach disable muss primask == 1 sein
142 }
143
144 // verlassen einer critical region durch aktivieren aller Interrupts
145 void APOS_LeaveRegion() // um task switch wieder zu ermoglichen
146 {
147     // uint32_t primask = __get_PRIMASK();
148     __enable_irq(); // wirkungslos?
149     // primask = __get_PRIMASK();           ...zum debuggen, ob IRQ wirklich wieder aktiviert wurde
150                                           // nach disable muss primask == 0 sein
151 }
152
153 // prufen, ob critical region schon vergeben ist, durch prfen
154 // return 1 ... region vergeben
155 // return 0 ... region ist gerade frei
156 int APOS_TestRegion() // ob priority mask der IRQs gesetzt wurde
157 {
158     uint32_t primask = __get_PRIMASK();
159     return primask;
160 }
161
162 void APOS_Delay (uint32_t ticks)
163 {
164     pTasks[currentTask]->status = SUSPENDED;
165     pTasks[currentTask]->delay = ticks;
166     pTasks[currentTask]->statusTime = SysTick_GetTick();
167 }
168
169 static BOOL APOS_Running(void) {
170     switch(pTasks[currentTask]->status) {
171         case RUNNING:
172             return TRUE;
173             break;
174         case SUSPENDED:
175             if((SysTick_GetTick() - pTasks[currentTask]->statusTime) > pTasks[currentTask]->delay) {
176                 pTasks[currentTask]->status = RUNNING;
177                 pTasks[currentTask]->statusTime = SysTick_GetTick();
178                 pTasks[currentTask]->delay = 0;
179                 return TRUE;
180             }
181     }
182 }

```

```
176         return FALSE;
177         break;
178     }
179 }
```

1.3 APOS.s

```
1  AREA  |.text|, CODE, READONLY
2
3
4  FillTaskA  PROC
5      EXPORT FillTaskA
6      MOVS    R0,    #0xA0
7      MOVS    R1,    #0xA1
8      MOVS    R2,    #0xA2
9      MOVS    R3,    #0xA3
10     MOVS    R4,    #0xA4
11     MOVS    R5,    #0xA5
12     MOVS    R6,    #0xA6
13
14     MOVS    R7,    #0xA8
15     MOV      R8,    R7
16     MOVS    R7,    #0xA9
17     MOV      R9,    R7
18     MOVS    R7,    #0xAA
19     MOV      R10,   R7
20     MOVS    R7,    #0xAB
21     MOV      R11,   R7
22
23     MOVS    R7,    #0xAC
24     MOV      R12,   R7
25     MOVS    R7,    #0xA7
26     BX      LR
27     ENDP
28
29
30  FillTaskB  PROC
31      EXPORT FillTaskB
32      MOVS    R0,    #0xB0
33      MOVS    R1,    #0xB1
34      MOVS    R2,    #0xB2
35      MOVS    R3,    #0xB3
36      MOVS    R4,    #0xB4
37      MOVS    R5,    #0xB5
38      MOVS    R6,    #0xB6
39
40      MOVS    R7,    #0xB8
41      MOV      R8,    R7
42      MOVS    R7,    #0xB9
43      MOV      R9,    R7
44      MOVS    R7,    #0xBA
45      MOV      R10,   R7
46      MOVS    R7,    #0xBB
47      MOV      R11,   R7
48
49      MOVS    R7,    #0xBC
50      MOV      R12,   R7
51      MOVS    R7,    #0xB7
52      BX      LR
53      ENDP
54
55  FillTaskC  PROC
56      EXPORT FillTaskC
57      MOVS    R0,    #0xC0
```

APOS.s

```

58      MOVS    R1,    #0xC1
59      MOVS    R2,    #0xC2
60      MOVS    R3,    #0xC3
61      MOVS    R4,    #0xC4
62      MOVS    R5,    #0xC5
63      MOVS    R6,    #0xC6
64
65      MOVS    R7,    #0xC8
66      MOV      R8,    R7
67      MOVS    R7,    #0xC9
68      MOV      R9,    R7
69      MOVS    R7,    #0xCA
70      MOV      R10,   R7
71      MOVS    R7,    #0xCB
72      MOV      R11,   R7
73
74      MOVS    R7,    #0xCC
75      MOV      R12,   R7
76      MOVS    R7,    #0xC7
77      BX      LR
78      ENDP
79
80      END

```

1.4 startup-stm32f072.s, beinhaltet PendSV-Handler

```
startup-stm32f072.s
1 ;***** (C) COPYRIGHT 2014 STMicroelectronics *****
2 ;* File Name      : startup_stm32f072.s
3 ;* Author         : MCD Application Team
4 ;* Version        : V1.3.1
5 ;* Date           : 17-January-2014
6 ;* Description    : STM32F072 Devices vector table for
7 ;*                : for MDK-ARM toolchain.
8 ;*                : This module performs:
9 ;*                : - Set the initial SP
10 ;*                : - Set the initial PC == Reset_Handler
11 ;*                : - Set the vector table entries with the exceptions ISR address
12 ;*                : - Configure the system clock
13 ;*                : - Branches to __main in the C library (which eventually
14 ;*                :   calls main()).
15 ;*                : After Reset the CortexM0 processor is in Thread mode,
16 ;*                : priority is Privileged, and the Stack is set to Main.
17 ;* <<< Use Configuration Wizard in Context Menu >>>
18 ;*****
19 ; @attention
20 ;
21 ; Licensed under MCD-ST Liberty SW License Agreement V2, (the "License");
22 ; You may not use this file except in compliance with the License.
23 ; You may obtain a copy of the License at:
24 ;
25 ;     http://www.st.com/software_license_agreement_liberty_v2
26 ;
27 ; Unless required by applicable law or agreed to in writing, software
28 ; distributed under the License is distributed on an "AS IS" BASIS,
29 ; WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
30 ; See the License for the specific language governing permissions and
31 ; limitations under the License.
32 ;
33 ;*****
34 ;
35 ; Amount of memory (in bytes) allocated for Stack
36 ; Tailor this value to your application needs
37 ; <h> Stack Configuration
38 ;   <o> Stack Size (in Bytes) <0x0-0xFFFFFFFF:8>
39 ; </h>
40
41 Stack_Size      EQU      0x0000800
42
43                 AREA      STACK, NOINIT, READWRITE, ALIGN=3
44 Stack_Mem        SPACE    Stack_Size
45 __initial_sp
46
47
48 ; <h> Heap Configuration
49 ;   <o> Heap Size (in Bytes) <0x0-0xFFFFFFFF:8>
50 ; </h>
51
52 Heap_Size        EQU      0x00000800
53
54                 AREA      HEAP, NOINIT, READWRITE, ALIGN=3
55 __heap_base
56 Heap_Mem          SPACE    Heap_Size
57 __heap_limit
```

```

58
59 PRESERVE8
60 THUMB
61
62
63 ; Vector Table Mapped to Address 0 at Reset
64 AREA RESET, DATA, READONLY
65 EXPORT __Vectors
66 EXPORT __Vectors_End
67 EXPORT __Vectors_Size
68
69 __Vectors DCD __initial_sp ; Top of Stack
70 DCD Reset_Handler ; Reset Handler
71 DCD NMI_Handler ; NMI Handler
72 DCD HardFault_Handler ; Hard Fault Handler
73 DCD 0 ; Reserved
74 DCD 0 ; Reserved
75 DCD 0 ; Reserved
76 DCD 0 ; Reserved
77 DCD 0 ; Reserved
78 DCD 0 ; Reserved
79 DCD 0 ; Reserved
80 DCD SVC_Handler ; SVC Call Handler
81 DCD 0 ; Reserved
82 DCD 0 ; Reserved
83 DCD PendSV_Handler ; PendSV Handler
84 DCD SysTick_Handler ; SysTick Handler
85
86 ; External Interrupts
87 DCD WWDG_IRQHandler ; Window Watchdog
88 DCD PVD_VDDIO2_IRQHandler ; PVD and VDDIO2 through EXTI Line detect
89 DCD RTC_IRQHandler ; RTC through EXTI Line
90 DCD FLASH_IRQHandler ; FLASH
91 DCD RCC_CRs_IRQHandler ; RCC and CRS
92 DCD EXTI0_1_IRQHandler ; EXTI Line 0 and 1
93 DCD EXTI2_3_IRQHandler ; EXTI Line 2 and 3
94 DCD EXTI4_15_IRQHandler ; EXTI Line 4 to 15
95 DCD TSC_IRQHandler ; TS
96 DCD DMA1_Channel1_IRQHandler ; DMA1 Channel 1
97 DCD DMA1_Channel2_3_IRQHandler ; DMA1 Channel 2 and Channel 3
98 DCD DMA1_Channel4_5_6_7_IRQHandler ; DMA1 Channel 4, Channel 5, Channel 6 and Channel 7
99 DCD ADC1_COMP_IRQHandler ; ADC1, COMP1 and COMP2
100 DCD TIM1_BRK_UP_TRG_COM_IRQHandler ; TIM1 Break, Update, Trigger and Commutation
101 DCD TIM1_CC_IRQHandler ; TIM1 Capture Compare
102 DCD TIM2_IRQHandler ; TIM2
103 DCD TIM3_IRQHandler ; TIM3
104 DCD TIM6_DAC_IRQHandler ; TIM6 and DAC
105 DCD TIM7_IRQHandler ; TIM7
106 DCD TIM14_IRQHandler ; TIM14
107 DCD TIM15_IRQHandler ; TIM15
108 DCD TIM16_IRQHandler ; TIM16
109 DCD TIM17_IRQHandler ; TIM17
110 DCD I2C1_IRQHandler ; I2C1
111 DCD I2C2_IRQHandler ; I2C2
112 DCD SPI1_IRQHandler ; SPI1
113 DCD SPI2_IRQHandler ; SPI2
114 DCD USART1_IRQHandler ; USART1
115 DCD USART2_IRQHandler ; USART2
116 DCD USART3_4_IRQHandler ; USART3 and USART4

```

```

117          DCD      CEC_CAN_IRQHandler          ; CEC and CAN
118          DCD      USB_IRQHandler              ; USB
119
120  __Vectors_End
121
122  __Vectors_Size EQU  __Vectors_End - __Vectors
123
124          AREA      |.text|, CODE, READONLY
125
126  ; Reset handler routine
127  Reset_Handler  PROC
128                  EXPORT  Reset_Handler          [WEAK]
129          IMPORT  __main
130          IMPORT  SystemInit
131                  LDR     R0, =SystemInit
132                  BLX     R0
133                  LDR     R0, =__main
134                  BX      R0
135          ENDP
136
137  ; Dummy Exception Handlers (infinite loops which can be modified)
138
139  NMI_Handler    PROC
140                  EXPORT  NMI_Handler            [WEAK]
141                  B       .
142          ENDP
143  HardFault_Handler\
144                  PROC
145                  EXPORT  HardFault_Handler      [WEAK]
146                  B       .
147          ENDP
148  SVC_Handler    PROC
149                  EXPORT  SVC_Handler            [WEAK]
150                  B       .
151          ENDP
152  PendSV_Handler PROC
153                  EXPORT  PendSV_Handler         [WEAK]
154                  mrs r0, psp
155                  subs r0, r0, #32
156                  stm r0!, {r4-r7}
157                  mov r4, r8
158                  mov r5, r9
159                  mov r6, r10
160                  mov r7, r11
161                  stm r0!, {r4-r7}
162                  subs r0, r0, #16
163                  mov r5, r14                      ; save LR in R5
164                  IMPORT  APOS_SetPSP
165                  LDR     R0, =APOS_SetPSP        ; call C-Function
166                  BLX     R0                      ; to set psp for next Task
167                  mov r14, r5                      ; get LR back from R5
168                  subs r0, r0, #16
169                  ldm r0!, {r4-r7}
170                  mov r8, r4
171                  mov r9, r5
172                  mov r10, r6
173                  mov r11, r7
174                  subs r0, r0, #32
175                  ldm r0!, {r4-r7}

```

```

176         BX      LR
177     ENDP
178 SysTick_Handler PROC
179     EXPORT SysTick_Handler          [WEAK]
180     B        .
181     ENDP
182 Default_Handler PROC
183
184     EXPORT WWDG_IRQHandler          [WEAK]
185     EXPORT PVD_VDDIO2_IRQHandler   [WEAK]
186     EXPORT RTC_IRQHandler           [WEAK]
187     EXPORT FLASH_IRQHandler         [WEAK]
188     EXPORT RCC_CRs_IRQHandler       [WEAK]
189     EXPORT EXTI0_1_IRQHandler        [WEAK]
190     EXPORT EXTI2_3_IRQHandler        [WEAK]
191     EXPORT EXTI4_15_IRQHandler       [WEAK]
192     EXPORT TSC_IRQHandler            [WEAK]
193     EXPORT DMA1_Channel1_IRQHandler [WEAK]
194     EXPORT DMA1_Channel2_3_IRQHandler [WEAK]
195     EXPORT DMA1_Channel4_5_6_7_IRQHandler [WEAK]
196     EXPORT ADC1_COMP_IRQHandler      [WEAK]
197     EXPORT TIM1_BRK_UP_TRG_COM_IRQHandler [WEAK]
198     EXPORT TIM1_CC_IRQHandler        [WEAK]
199     EXPORT TIM2_IRQHandler           [WEAK]
200     EXPORT TIM3_IRQHandler           [WEAK]
201     EXPORT TIM6_DAC_IRQHandler       [WEAK]
202     EXPORT TIM7_IRQHandler           [WEAK]
203     EXPORT TIM14_IRQHandler          [WEAK]
204     EXPORT TIM15_IRQHandler          [WEAK]
205     EXPORT TIM16_IRQHandler          [WEAK]
206     EXPORT TIM17_IRQHandler          [WEAK]
207     EXPORT I2C1_IRQHandler           [WEAK]
208     EXPORT I2C2_IRQHandler           [WEAK]
209     EXPORT SPI1_IRQHandler           [WEAK]
210     EXPORT SPI2_IRQHandler           [WEAK]
211     EXPORT USART1_IRQHandler         [WEAK]
212     EXPORT USART2_IRQHandler         [WEAK]
213     EXPORT USART3_4_IRQHandler       [WEAK]
214     EXPORT CEC_CAN_IRQHandler        [WEAK]
215     EXPORT USB_IRQHandler            [WEAK]
216
217
218 WWDG_IRQHandler
219 PVD_VDDIO2_IRQHandler
220 RTC_IRQHandler
221 FLASH_IRQHandler
222 RCC_CRs_IRQHandler
223 EXTI0_1_IRQHandler
224 EXTI2_3_IRQHandler
225 EXTI4_15_IRQHandler
226 TSC_IRQHandler
227 DMA1_Channel1_IRQHandler
228 DMA1_Channel2_3_IRQHandler
229 DMA1_Channel4_5_6_7_IRQHandler
230 ADC1_COMP_IRQHandler
231 TIM1_BRK_UP_TRG_COM_IRQHandler
232 TIM1_CC_IRQHandler
233 TIM2_IRQHandler
234 TIM3_IRQHandler

```



```

235 TIM6_DAC_IRQHandler
236 TIM7_IRQHandler
237 TIM14_IRQHandler
238 TIM15_IRQHandler
239 TIM16_IRQHandler
240 TIM17_IRQHandler
241 I2C1_IRQHandler
242 I2C2_IRQHandler
243 SPI1_IRQHandler
244 SPI2_IRQHandler
245 USART1_IRQHandler
246 USART2_IRQHandler
247 USART3_4_IRQHandler
248 CEC_CAN_IRQHandler
249 USB_IRQHandler
250
251         B      .
252
253         ENDP
254
255         ALIGN
256
257 ;*****
258 ; User Stack and Heap initialization
259 ;*****
260         IF      :DEF:__MICROLIB
261
262         EXPORT  __initial_sp
263         EXPORT  __heap_base
264         EXPORT  __heap_limit
265
266         ELSE
267
268         IMPORT  __use_two_region_memory
269         EXPORT  __user_initial_stackheap
270
271 __user_initial_stackheap
272
273         LDR     R0, = Heap_Mem
274         LDR     R1, =(Stack_Mem + Stack_Size)
275         LDR     R2, =(Heap_Mem + Heap_Size)
276         LDR     R3, = Stack_Mem
277         BX     LR
278
279         ALIGN
280
281         ENDIF
282
283         END
284
285 ;***** (C) COPYRIGHT STMicroelectronics *****END OF FILE*****

```
