Cortex-M3 programming

Texas Instruments, www.ti.com
CortexM3InstructionSet.pdf

STMicroelectronics, www.st.com
CortexM3Programmer.pdf

PM0056

January 17, 2013 Jonathan Valvano
EE345M/EE380L.6

Modes and Levels

Thread mode  Used to execute application software. The processor enters Thread
mode when it comes out of reset.

Handler mode  Used to handle exceptions. The processor returns to Thread mode
when it has finished exception processing.

The privilege levels for software execution are:

Unprivileged The software:
  • Has limited access to the MSR and MFR instructions, and cannot
    use the CPS instruction
  • Cannot access the system timer, NVIC, or system control block
  • Might have restricted access to memory or peripherals.

Unprivileged software executes at the unprivileged level.

Privileged The software can use all the instructions and has access to all
resources.
Privileged software executes at the privileged level.

Registers

R0-R3 parameters
R4-R11 must be saved

General-purpose registers
R0-R12 are 32-bit general-purpose registers for data operations.

Stack pointer
The Stack Pointer (SP) is register R13. In Thread mode, the CONTROL register
indicates the stack pointer to use:
  • 0 = Main Stack Pointer (MSP). This is the reset value.
  • 1 = Process Stack Pointer (PSP).
On reset, the processor loads the MSP with the value from address 0x00000000.

Link register
The Link Register (LR) is register R14. It stores the return information for subroutines,
function calls, and exceptions. On reset, the processor loads the LR value withFFFFFFFFFF.

Program counter
L14 is important
The Program Counter (PC) is register R15. It contains the current program address. B(15) is
always 0 because instruction fetches must be halfword aligned. On reset, the processor
loads the PC with the value of the reset vector, which is at address 0x00000004.
Function calls

```c
void delay (int cnt){
    while (cnt--);
}
delay(10);
```

Follow the link register LR

```assembly
SUB      r0,r0,#0x01
BNE      delay  BX        lr
MOV      r0,#0x0A
BL          delay
```

```c
void function1 (void)
{
    output(0x01);
}
int main (void) {
    ...
    function1();
    ...
}
```

Draw a stack picture

The accesses happen in order of increasing register numbers, with the lowest numbered register using the lowest memory address and the highest number register using the highest memory address

```
PUSH     {r4-r6,lr}
MOV      r0,#0x01
BL          output
POP       {r4-r6,pc}
```

Program Status Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-0</td>
<td>Reserved</td>
</tr>
<tr>
<td>15-0</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>14</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>13</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>12</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>11</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>10</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>9</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>8</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>7</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>6</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>5</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>4</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>3</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>2</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>1</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>0</td>
<td>IR, NUMBER</td>
</tr>
</tbody>
</table>

Interrupt Program Status Register

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-8</td>
<td>Reserved</td>
</tr>
<tr>
<td>8-0</td>
<td>Interrupts</td>
</tr>
<tr>
<td>7</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>6</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>5</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>4</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>3</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>2</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>1</td>
<td>IR, NUMBER</td>
</tr>
<tr>
<td>0</td>
<td>IR, NUMBER</td>
</tr>
</tbody>
</table>

Run debugger:
- stop in ISR and
- look at IPSR
Execution Program Status Register

The Execution PSR (EPSR) contains two overlapping fields:
- the Interruptible-Continuable Instruction (ICI) field for interrupted load multiple and store multiple instructions
- the execution state field for the H-Then (IT) instruction, and the T-bit (Thumb state bit).

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved</td>
</tr>
<tr>
<td>27</td>
<td>Reserved</td>
</tr>
<tr>
<td>26</td>
<td>Reserved</td>
</tr>
<tr>
<td>25</td>
<td>Reserved</td>
</tr>
<tr>
<td>24</td>
<td>Reserved</td>
</tr>
<tr>
<td>23</td>
<td>Reserved</td>
</tr>
<tr>
<td>16</td>
<td>ICI/IT</td>
</tr>
<tr>
<td>15</td>
<td>Reserved</td>
</tr>
<tr>
<td>14</td>
<td>Reserved</td>
</tr>
<tr>
<td>13</td>
<td>Reserved</td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
</tr>
<tr>
<td>11</td>
<td>Reserved</td>
</tr>
<tr>
<td>10</td>
<td>Reserved</td>
</tr>
<tr>
<td>9</td>
<td>Reserved</td>
</tr>
<tr>
<td>8</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>Reserved</td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
</tr>
<tr>
<td>5</td>
<td>Reserved</td>
</tr>
<tr>
<td>4</td>
<td>Reserved</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Race, Critical Sections

- Shared information
  - Permanently allocated
  - Memory or I/O registers
- Sequences involving at least one write
  - E.g., RMW, WW, WR
- Nonatomic sequence
  - Begins with access to permanent memory
  - Ends with access to permanent memory

```
PUSH {r4-r6,lr}
LDREX
STREX
```

Priority Mask Register

```
CPSID   I
CPSIE   I
MRS PRIMASK,R0
```

Code from uCOS-II

```
#define OS_ENTERCRITICAL() {sr = SRSave();}
#define OS_EXITCRITICAL() {SRRestore(sr);}
void  Task (void *p_arg)  {
  long sr=0;
  OS_CRITICALENTER();
  // critical section
  OS_CRITICALEXIT();
  }
```

Where is the I bit saved?
Control Register

Table 10. CONTROL register bit definitions

<table>
<thead>
<tr>
<th>Bm</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b 1</td>
<td>APSRsel: Active stack pointer selection</td>
</tr>
<tr>
<td>0b 2</td>
<td>MSP is the current stack pointer.</td>
</tr>
<tr>
<td>0b 14</td>
<td>PSP is the current stack pointer.</td>
</tr>
<tr>
<td>0b 15</td>
<td>LR is the current stack pointer.</td>
</tr>
</tbody>
</table>

Reset debugger:
- look at CONTROL
- stop in ISR and
- look at CONTROL

Exceptions

Table 16

<table>
<thead>
<tr>
<th>Exception number(1)</th>
<th>IRQ number(2)</th>
<th>Exception type</th>
<th>Priority</th>
<th>Vector address or offset(3)</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td>Reset</td>
<td>-</td>
<td>0b00000004</td>
<td>Asynchronous</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>HIU</td>
<td>0</td>
<td>0b00000005</td>
<td>Asynchronous</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>Hard fault</td>
<td>1</td>
<td>0b0000000C</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>Memory management fault</td>
<td>Configurable(8)</td>
<td>0b00000010</td>
<td>Synchronous</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>Bus fault</td>
<td>Configurable(8)</td>
<td>0b00000014</td>
<td>Synchronous</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>Usage fault</td>
<td>Configurable(8)</td>
<td>0b00000018</td>
<td>Asynchronous</td>
</tr>
<tr>
<td>7-15</td>
<td>-</td>
<td>SVC call</td>
<td>Configurable(8)</td>
<td>0b00000020</td>
<td>Synchronous</td>
</tr>
<tr>
<td>16-20</td>
<td>-</td>
<td>System mode</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Interrupt vector table

Show vector table in project, debugger
Define Group priority 0-15
Subpriority Nested exceptions
Return 8 Registers
Tail chaining ● R0-R3, R12
Late arrival ● LR
Stacking ● Return address ● PSR
LR=EXC_RETURN 0b11110001 Ret to Handler MSP 0b11111001 Ret to Thread MSP 0b11111101 Ret to Thread PSP aligned to double-word address

Supervisor Call

3.9.10 SVC

Supervisor Call;

Syntax

```
svc <imm>
```

where:

- `imm` is an optional condition code, see Conditional execution on page 56.
- `imm` is an expression evaluating to an integer in the range 0-255 (8-bit value).

Operation

The SVC instruction causes the SVC exception. 

Condition flags

This instruction does not change the flags.

Examples

```
svc 0x12345678  // Supervisor Call (SC) handler can return the immediate value
                 // by writing 0xA to the stacked 0x1234
```

Remember

Systick is 15
Code from uCOS-II

```
#define OS_TASK_SW() OSCtxSw()
```

```
OS_CPU_PendSVHandler
    CPSID I ; Prevent interruption during context switch
    MRS R0, PSP ; PSP is process stack pointer
....
    MSR PSP, R0 ; Load PSP with new process SP
    ORR LR, LR, #0x04 ; exception return uses process stack
    CPSIE I
    BX LR
```

System tick (initialization)

```
void SysTick_Init(unsigned long period){
    volatile unsigned long delay;
    SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOD; // activate port D
    Counts = 0;
    GPIO_PORTD_DIR_R |= 0x01; // make PD0 output
    GPIO_PORTD_DEN_R |= 0x01; // enable digital I/O on PD0
    NVIC_ST_CTRL_R = 0; // disable SysTick during setup
    NVIC_ST_RELOAD_R = period - 1; // reload value
    NVIC_ST_CURRENT_R = 0; // any write to current clears it
    NVIC_PENDSVSET = 0x10000000;
    NVIC_INT_CTRL = 0xE000ED04;
    NVIC_ST_CTRL_R = NVIC_ST_CTRL_ENABLE+NVIC_ST_CTRL_CLK_SRC
                       +NVIC_ST_CTRL_INTEN;
    EnableInterrupts();
}
```

System tick

```
#define GPIO_PD0 (*((volatile unsigned long *) 0x40007004))

void SysTick_Handler(void) {
    GPIO_PD0 = GPIO_PD0^0x01;
    Counts = Counts + 1;
}

void main(void){
    ...
    SysTick_Init(50000); // 1msec, assuming 50 MHz bus clock
    ...
}
```

Summary

- Learn the assembly language
- Study how interrupts are processed
- Determine what goes on the stack
- Develop and deploy your debugging skills
  - TFBF (time to first bus fault)
  - Control and observability