ARMv8 exception vectors and handling

I am working on a ARM Cortex A53 processor and can not figure out how to set up interrupts to work.

I've read the documentation regarding this subject but still find it confusing and cannot get interrupts to work in a bare metal environment.

Here is the vector table I currently have:

```
_vectors:
    /* Current EL with SP0 */
    b sync_addr /* Synchronous */
    .balign 128
    b irq_addr  /* IRQ/vIRQ */
    .balign 128
    b fiq_addr  /* FIQ/vFIQ */
    .balign 128
    b serr_addr /* SError/vSError */
    /* Current EL with SPn */
    b sync_addr /* Synchronous */
    .balign 128
    b irq_addr  /* IRQ/vIRQ */
    .balign 128
    b fiq_addr  /* FIQ/vFIQ */
    .balign 128
    b serr_addr /* SError/vSError */
    /* Lower EL with Aarch64 */
    b sync_addr /* Synchronous */
    .balign 128
    b irq_addr  /* IRQ/vIRQ */
    .balign 128
    b fiq_addr  /* FIQ/vFIQ */
    .balign 128
    b serr_addr /* SError/vSError */
    /* Lower EL with Aarch32 */
    b sync_addr /* Synchronous */
    .balign 128
    b irq_addr  /* IRQ/vIRQ */
    .balign 128
    b fiq_addr  /* FIQ/vFIQ */
    .balign 128
    b serr_addr /* SError/vSError */

sync_addr: .word reset_handler
irq_addr:  .word irq_handler
fiq_addr:  .word reset_handler
serr_addr: .word reset_handler
```

I got this via the ARMv8 Programmer's Guide section 10.4.

As far as I know, I need to set the VBAR_ELn register to point to the table, which I do as such:

```
ldr x0, =_vectors
msr vbar_el1, x0
```

Is there anything else I am missing? Any help or point to a reference would be greatly appreciated.

assembly arm arm64

edited Jul 9 '17 at 20:11

asked Jul 8 '17 at 22:25

Rade Latinovich

11 1

what alignment did you use on your table? look at the disassembly and confirm the handlers are at the right offset. you have enabled interrupts to the core after setting all of this up? this is after using polling to completely understand the peripheral and when it interrupts and how to clear it. – old_timer Jul 9 '17 at 23:10

the PSTATE field. I am also aware that this processor has a GIC (Generic Interrupt Controller), and have looked into it, but still have no clue as to how I am supposed to move forward from here. – Rade Latinovich Jul 10 '17 at 14:31

I recommend you back up. Using polling (leds or preferrably the uart, you DONT need printf) to learn/understand how the peripheral in question works, how to make its interrupt status flag assert and then what registers/etc to write to clear that interrupt status. – old_timer Jul 10 '17 at 14:39

then if you really do have a GIC (for example the gic is disabled in the raspberry pi 3) then poll the gic status registers to see the interrupt from the peripheral, clear the peripheral interrupt, does it clear the gic? is there anything you have to do at that level to clear the interrupt. – old_timer Jul 10 '17 at 14:40

THEN start to worry about the interrupt vector table, you need to use arm-whatever-objdump -D myprogram(.elf). I like to use the .elf extension most folks dont either way you need to examine where the linker has placed your vectors and is it at the exact address required for the handler? and then change the enable in PSTATE – old_timer Jul 10 '17 at 14:42

1 Answer

So I already have an example for this that you can study, take it or leave it...

config.txt (goes on the sd card):

```
arm_control=0x200
kernel_old=1
disable_commandline_tags=1
```

memmap (linker script):

```
MEMORY
{
    ram : ORIGIN = 0x0000, LENGTH = 0x1000000
}

SECTIONS
{
    .text : { *(.text*) } > ram
    .bss : { *(.bss*) } > ram
}
```

vectors.s

```
.globl _start
_start:
    b skip
    b hang
    b hang
    b hang
    b hang
    b hang
    b hang
    b hang
.balign 128
    b hang
.balign 128
    b hang
.balign 128
    b hang
.balign 128
    b hang
.balign 128
    b irq_handler

skip:
    // isolate core 0
    mrs x0,mpidr_el1
    mov x1,#0xC1000000
    bic x1,x0,x1
    cbz x1,zero
not_zero:
    wfi
    //msr daifset,#2
    b not_zero
zero:

    mov sp,#0x08000000
    bl notmain
hang: b hang
```

```
  str w1,[x0]
  ret

.globl GET32
GET32:
    ldr w0,[x0]
    ret

.globl enable_irq
enable_irq:
    //should already be set here
    ldr x1,=0x00000000
    msr vbar_el3,x1
    //route to EL3
    mrs x0,scr_el3
    orr x0,x0,#8
    orr x0,x0,#4
    orr x0,x0,#2
    msr scr_el3,x0
    //clear/enable irq bit in PSTATE
    msr daifclr,#2
    ret

irq_handler:
    //19 up are callee saved
    //so we have to preserve all below?
    stp x0,x1,[sp,#-16]!
    stp x2,x3,[sp,#-16]!
    stp x4,x5,[sp,#-16]!
    stp x6,x7,[sp,#-16]!
    stp x8,x9,[sp,#-16]!
    stp x10,x11,[sp,#-16]!
    stp x12,x13,[sp,#-16]!
    stp x14,x15,[sp,#-16]!
    stp x16,x17,[sp,#-16]!
    stp x18,x19,[sp,#-16]!

    //mrs x0,esr_el3
    bl c_irq_handler

    ldp x18,x19,[sp],#16
    ldp x16,x17,[sp],#16
    ldp x14,x15,[sp],#16
    ldp x12,x13,[sp],#16
    ldp x10,x11,[sp],#16
    ldp x8,x9,[sp],#16
    ldp x6,x7,[sp],#16
    ldp x4,x5,[sp],#16
    ldp x2,x3,[sp],#16
    ldp x0,x1,[sp],#16
    eret

.globl DOWFI
DOWFI:
    wfi
    //msr daifset,#2
    ret
```

periph.c

```
#define PBASE 0x3F000000

extern void PUT32 ( unsigned int, unsigned int );
extern unsigned int GET32 ( unsigned int );
extern void dummy ( unsigned int );

#define ARM_TIMER_CTL   (PBASE+0x0000B408)
#define ARM_TIMER_CNT   (PBASE+0x0000B420)

#define GPFSEL1         (PBASE+0x00200004)
#define GPSET0          (PBASE+0x0020001C)
#define GPCLR0          (PBASE+0x00200028)
#define GPPUD           (PBASE+0x00200094)
#define GPPUDCLK0       (PBASE+0x00200098)

#define AUX_ENABLES     (PBASE+0x00215004)
#define AUX_MU_IO_REG   (PBASE+0x00215040)
#define AUX_MU_IER_REG  (PBASE+0x00215044)
#define AUX_MU_IIR_REG  (PBASE+0x00215048)
#define AUX_MU_LCR_REG  (PBASE+0x0021504C)
#define AUX_MU_MCR_REG  (PBASE+0x00215050)
#define AUX_MU_LSR_REG  (PBASE+0x00215054)
#define AUX_MU_MSR_REG  (PBASE+0x00215058)
#define AUX_MU_SCRATCH  (PBASE+0x0021505C)
#define AUX_MU_CNTL_REG (PBASE+0x00215060)
#define AUX_MU_STAT_REG (PBASE+0x00215064)
```

```
//GPIO14  TXD0 and TXD1
//GPIO15  RXD0 and RXD1

unsigned int uart_lcr ( void )
{
    return(GET32(AUX_MU_LSR_REG));
}

unsigned int uart_recv ( void )
{
    while(1)
    {
        if(GET32(AUX_MU_LSR_REG)&0x01) break;
    }
    return(GET32(AUX_MU_IO_REG)&0xFF);
}

unsigned int uart_check ( void )
{
    if(GET32(AUX_MU_LSR_REG)&0x01) return(1);
    return(0);
}

void uart_send ( unsigned int c )
{
    while(1)
    {
        if(GET32(AUX_MU_LSR_REG)&0x20) break;
    }
    PUT32(AUX_MU_IO_REG,c);
}

void uart_flush ( void )
{
    while(1)
    {
        if(GET32(AUX_MU_LSR_REG)&0x40) break;
    }
}

void hexstrings ( unsigned int d )
{
    //unsigned int ra;
    unsigned int rb;
    unsigned int rc;

    rb=32;
    while(1)
    {
        rb-=4;
        rc=(d>>rb)&0xF;
        if(rc>9) rc+=0x37; else rc+=0x30;
        uart_send(rc);
        if(rb==0) break;
    }
    uart_send(0x20);
}

void hexstring ( unsigned int d )
{
    hexstrings(d);
    uart_send(0x0D);
    uart_send(0x0A);
}

void uart_init ( void )
{
    unsigned int ra;

    PUT32(AUX_ENABLES,1);
    PUT32(AUX_MU_IER_REG,0);
    PUT32(AUX_MU_CNTL_REG,0);
    PUT32(AUX_MU_LCR_REG,3);
    PUT32(AUX_MU_MCR_REG,0);
    PUT32(AUX_MU_IER_REG,0);
    PUT32(AUX_MU_IIR_REG,0xC6);
    PUT32(AUX_MU_BAUD_REG,270);
    ra=GET32(GPFSEL1);
    ra&=~(7<<12); //gpio14
    ra|=2<<12;    //alt5
    ra&=~(7<<15); //gpio15
    ra|=2<<15;    //alt5
    PUT32(GPFSEL1,ra);
    //PUT32(GPPUD,0);
    //for(ra=0;ra<150;ra++) dummy(ra);
    //PUT32(GPPUDCLK0,(1<<14)|(1<<15));
    //for(ra=0;ra<150;ra++) dummy(ra);
```

```
}
```

notmain.c

```
extern void PUT32 ( unsigned int, unsigned int );
extern unsigned int GET32 ( unsigned int );
extern void dummy ( unsigned int );
extern void enable_irq ( void );
extern void DOWFI ( void );

extern void uart_init ( void );
extern void uart_send ( unsigned int );
extern void hexstring ( unsigned int );

#define GPFSEL2 0x3F200008
#define GPSET0  0x3F20001C
#define GPCLR0  0x3F200028


#define ARM_TIMER_LOD 0x3F00B400
#define ARM_TIMER_VAL 0x3F00B404
#define ARM_TIMER_CTL 0x3F00B408
#define ARM_TIMER_CLI 0x3F00B40C
#define ARM_TIMER_RIS 0x3F00B410
#define ARM_TIMER_MIS 0x3F00B414
#define ARM_TIMER_RLD 0x3F00B418
#define ARM_TIMER_DIV 0x3F00B41C
#define ARM_TIMER_CNT 0x3F00B420

#define SYSTIMERCLO 0x3F003004
#define GPFSEL1 0x3F200004
#define GPSET0  0x3F20001C
#define GPCLR0  0x3F200028
#define GPFSEL3 0x3F20000C
#define GPFSEL4 0x3F200010
#define GPSET1  0x3F200020
#define GPCLR1  0x3F20002C

#define IRQ_BASIC 0x3F00B200
#define IRQ_PEND1 0x3F00B204
#define IRQ_PEND2 0x3F00B208
#define IRQ_FIQ_CONTROL 0x3F00B210
#define IRQ_ENABLE_BASIC 0x3F00B218
#define IRQ_DISABLE_BASIC 0x3F00B224

volatile unsigned int icount;


void c_irq_handler ( void )
{
    icount++;
    if(icount&1)
    {
        PUT32(GPSET0,1<<21);
        uart_send(0x55);
    }
    else
    {
        PUT32(GPCLR0,1<<21);
        uart_send(0x56);
    }
    PUT32(ARM_TIMER_CLI,0);
}

int notmain ( void )
{
    unsigned int ra;

    PUT32(IRQ_DISABLE_BASIC,1);

    ra=GET32(GPFSEL2);
    ra&=~(7<<3);
    ra|=1<<3;
    PUT32(GPFSEL2,ra);

    uart_init();
if(1)
{
    PUT32(ARM_TIMER_CTL,0x003E0000);
    PUT32(ARM_TIMER_LOD,1000000-1);
    PUT32(ARM_TIMER_RLD,1000000-1);
    PUT32(ARM_TIMER_DIV,0x000000F9);
    PUT32(ARM_TIMER_CLI,0);
    PUT32(ARM_TIMER_CTL,0x003E00A2);

    for(ra=0;ra<2;ra++)
```

```
        uart_send(0x55);
        while(1) if(GET32(ARM_TIMER_MIS)) break;
        PUT32(ARM_TIMER_CLI,0);

        PUT32(GPCLR0,1<<21);
        uart_send(0x56);
        while(1) if(GET32(ARM_TIMER_MIS)) break;
        PUT32(ARM_TIMER_CLI,0);
    }
    uart_send(0x0D);
    uart_send(0x0A);
}
if(1)
{
    PUT32(ARM_TIMER_CTL,0x003E0000);
    PUT32(ARM_TIMER_LOD,2000000-1);
    PUT32(ARM_TIMER_RLD,2000000-1);
    PUT32(ARM_TIMER_CLI,0);
    PUT32(IRQ_ENABLE_BASIC,1);
    PUT32(ARM_TIMER_CTL,0x003E00A2);
    for(ra=0;ra<3;ra++)
    {
        PUT32(GPSET0,1<<21);
        uart_send(0x55);
        while(1) if(GET32(IRQ_BASIC)&1) break;
        PUT32(ARM_TIMER_CLI,0);

        PUT32(GPCLR0,1<<21);
        uart_send(0x56);
        while(1) if(GET32(IRQ_BASIC)&1) break;
        PUT32(ARM_TIMER_CLI,0);
    }
    PUT32(IRQ_ENABLE_BASIC,0);
    uart_send(0x0D);
    uart_send(0x0A);
}
    PUT32(ARM_TIMER_CTL,0x003E0000);
    PUT32(ARM_TIMER_LOD,500000-1);
    PUT32(ARM_TIMER_RLD,500000-1);
    PUT32(ARM_TIMER_CLI,0);
    PUT32(IRQ_ENABLE_BASIC,1);
    icount=0;
    enable_irq();
    PUT32(ARM_TIMER_CTL,0x003E00A2);
    PUT32(ARM_TIMER_CLI,0);

    while(1)
    {
        DOWFI();
        uart_send(0x33);
    }
    return(0);
}
```

build

```
aarch64-none-elf-as --warn --fatal-warnings vectors.s -o vectors.o
aarch64-none-elf-gcc -Wall -O2 -nostdlib -nostartfiles -ffreestanding  -c periph.c -o
periph.o
aarch64-none-elf-gcc -Wall -O2 -nostdlib -nostartfiles -ffreestanding  -c notmain.c -o
notmain.o
aarch64-none-elf-ld vectors.o periph.o notmain.o -T memmap -o notmain.elf
aarch64-none-elf-objdump -D notmain.elf > notmain.list
aarch64-none-elf-objcopy notmain.elf -O binary kernel8.img
```

Then examine the disassembly to confirm that everything is in the right place. I write zero to the vbar even though I shouldnt need to as it is supposed to come up as zero. So the address/offset needs to be 0x280 for EL3 or what is it not el0/el1? something like that.

```
0000000000000000 <_start>:
   0:   140000a1    b   284 <skip>
   4:   140000a8    b   2a4 <hang>
   8:   140000a7    b   2a4 <hang>
   c:   140000a6    b   2a4 <hang>
  10:   140000a5    b   2a4 <hang>
  14:   140000a4    b   2a4 <hang>
  18:   140000a3    b   2a4 <hang>
  1c:   140000a2    b   2a4 <hang>
  20:   d503201f    nop
  24:   d503201f    nop
  28:   d503201f    nop
  2c:   d503201f    nop
...
 27c:   d503201f    nop
```

My guess is the big gap between vectors is so you can put the handler there and not branch elsewhere, but I branched elsewhere anyway.

And copy kernel8.img and config.txt (as well as bootcode.bin and start.elf and not have any other kernel*.img with that name (rename as needed or delete).

The first if(1) polls the interrupt in the peripheral. The next is chip specific as the raspberry pi 3 has its own interrupt logic, they strapped the GICCDISABLE to disable the GIC, so dont get to play with that. And the last the interrupt is allowed thorugh.

The pi3 boots in EL3 as one would expect they dont mess with it (if you even can from the edge of the core, can certainly switch aarch32 vs aarch64).

So from where you left off you need to insure your vector is in the right place/offset from where you are setting vbar. I would assume but would have to check that there is an alignment requirement for vbar, maybe there isnt (some number of lower address bits having to be zero). then you have all the peripheral work to do. The pi is nice (not that others arent) in that you can poll the interrupt status lines to see what line changes with your peripheral once you figure out how to make your peripheral send an interrupt. Verify that interrupt line/number with the broadcom docs and see if makes any sense, some of them are not documented or maybe you have to see the community driven errata, etc. Then it is pretty easy to allow the interrupt through, not as complicated as a gic or other interrupt controllers (dont need to do priority stuff here if you dont want) and then allow it into the core.

What I have not figured out is why WFI doesnt release when the interrupt happens.

answered Jul 10 '17 at 17:33

old_timer

44.3k 6 56 113

Note: bad idea to talk to the uart in an ISR, in this case the interrupts are very far apart, it is safe in this implementation, but a bad idea in general. – old_timer Jul 10 '17 at 17:38

this is just a reference of a working example compare it to what you are doing and make your own program... – old_timer Jul 10 '17 at 17:46