ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Avenida Professor Luciano Gualberto, travessa 3 nº 158 CEP 05508-900 São Paulo SP
Telefone: (11) 3091-5583 Fax (11) 3091-5294

Departamento de Engenharia de Computação e Sistemas Digitais

PCS 2031
Laboratório de Microprocessadores

Relatório do Projeto Livre
Interrupções de timer usando Qemu e Gnuarm

Leonardo Teixeira *6484294*
Lucas Estevam *6484186*
Salim Skaf *6484015*

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Avenida Professor Luciano Gualberto, travessa 3 nº 158 CEP 05508-900 São Paulo SP
Telefone: (11) 3091-5583 Fax (11) 3091-5294

Departamento de Engenharia de Computação e Sistemas Digitais

1. Introdução

A emulação e entendimento do funcionamento da placa ARM Versatile é de grande interesse atualmente, devido ao crescente número de celulares e outros dispositivos que estão utilizando processadores ARM e placas similares comercialmente. Utilizando o QEMU e o GDB é possível simular e debugar software para essa plataforma, facilitando o desenvolvimento e entendimento da arquitetura. Além disso, a placa emulada pode servir como plataforma para o aprendizado sobre microprocessadores, quando placas físicas não estão disponíveis ou não atendem as necessidades para o que se deseja estudar.

Interrupções de timer é um dos assuntos que são difíceis de serem estudados na placa física, pois não é possível debugar o código de tratamento das interrupções, devido à presença do software Angel, responsável pela comunicação da placa com o debugger. Por isso, é de especial interesse encontrar meios de estudar o assunto sem precisar da placa.

1.1 Objetivo

- Desenvolver um código simples que possibilite demonstrar o funcionamento de interrupções de tempo utilizando a placa Versatile emulada no Qemu.

- Deve ser possível debugar o código utilizando o arm-elf-gdb, executando o programa passo a passo mesmo dentro das rotinas de interrupção, o que não é possível usando a placa física.

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Avenida Professor Luciano Gualberto, travessa 3 nº 158 CEP 05508-900 São Paulo SP
Telefone: (11) 3091-5583 Fax (11) 3091-5294

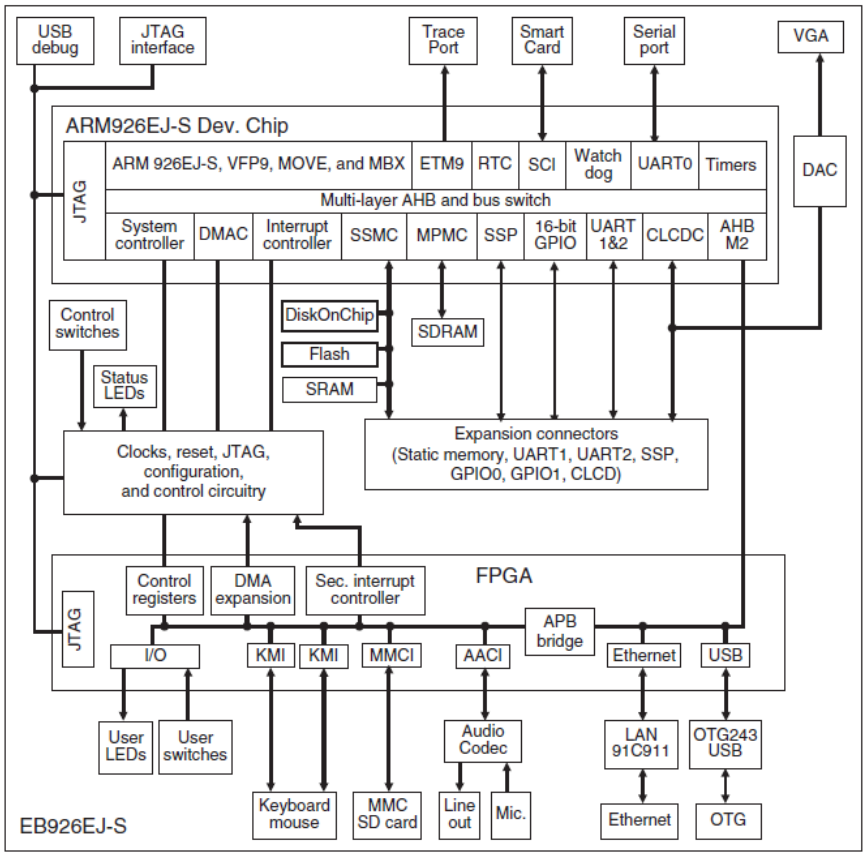
Departamento de Engenharia de Computação e Sistemas Digitais

2. Conceitos

2.1 A placa VersatilePB

A placa emulada VersatilePB é baseada numa placa real que utiliza um processador ARM926EJ-S, e inclui diversos periféricos, servindo como uma boa alternativa para desenvolvimento e aprendizado.

Abaixo temos uma visão geral da estrutura da placa Versatile real:



Avenida Professor Luciano Gualberto, travessa 3 nº 158 CEP 05508-900 São Paulo SP
Telefone: (11) 3091-5583 Fax (11) 3091-5294

Departamento de Engenharia de Computação e Sistemas Digitais

Vale ressaltar que nem todas as características da placa são emuladas pelo Qemu. Para mais informações veja o link para a documentação nas referências.

2.2 O Controlador de interrupções *PrimeCell ® VectoredInterrupt Controller (PL190)*

O controlador de interrupções utilizado pela placa versatile é o PrimeCell VectoredInterruptController (PL190), capaz de gerar tanto FIQs quanto IRQs. Seu funcionamento é razoávelmente simples, bastando programar seus registradores adequadamente.

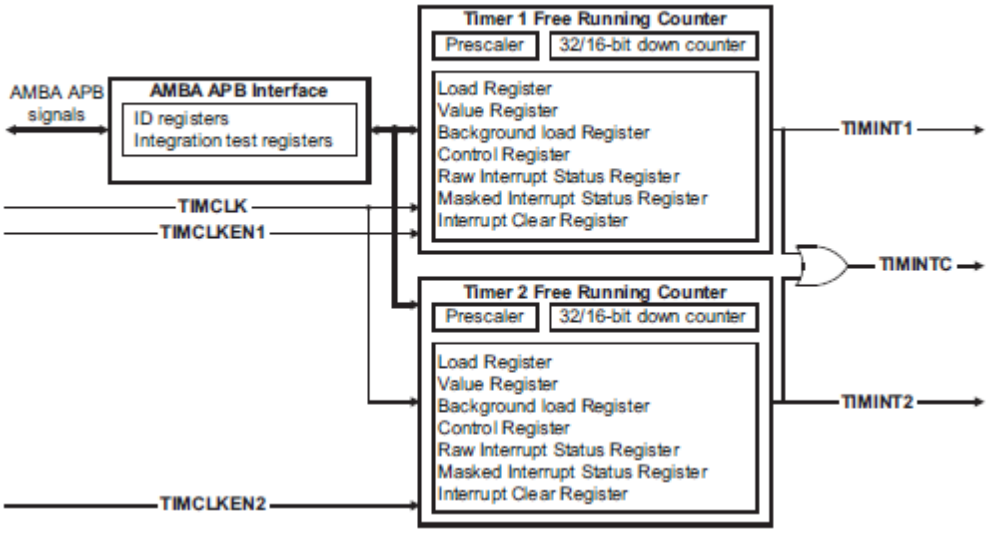
Em especial, os registradores mais importantes são:

- VICIRQSTATUS: Contém o status dos bits de interrupção, permitindo identificar qual dispositivo gerou a interrupção. O bit 4 corresponde a interrupções dos timers 0 e 1, que utilizaremos.
- VICINTSELECT: Permite selecionarquais dispositivos utilizarão IRQs ou FIQs. Um bit setado significa que o dispositivo utilizará FIQs.
- VICINTENABLE: Permite habilitar individualmente as interrupções de cada dispositivo.

Para mais informações veja o link para a documentação nas referências.

2.3 O controladore de timers *ARM Dual-Timer Module (SP804)*

A placa versatile conta com dois módulos de timers ARM Dual-Timer SP804, num total de 4 timers compartilhando dois bits do controlador de interrupções (bits 4 e 5 no caso da versatile).
A figura abaixo mostra a estrutura geral de um módulo SP804:



O controle dos timers é feito através dos registradores a seguir, entre outros:

- TimerControl: Utilizado para configurar o modo de operação do timer, ativá-lo e habitar interrupções.
- TimerValue: Permite ler ou escrever o valor atual do timer. A interrupção é acionada quando este valor chega a 0.
- TimerIntClr: Utilizado para limpar o pedido de interrupção durante a rotina de tratamento. Basta escrever nesse registrador para que o pedido seja encerrado.

Para mais informações veja o link para a documentação nas referências.

2.4 Utilização do QEMU e arm-elf-gdb

O **QEMU** é um emulador de processadores capaz de simular diversas arquiteturas. Em especial utilizaremos o qemu-system-arm, que emula diversas máquinas arm.

Para compilar e instalar o qemu no Ubunto basta executar o comando abaixo:

```
>sudo apt-get install qemu
```

Existem diversas opções de utilização do qemu-system-arm. Nesse projeto

Avenida Professor Luciano Gualberto, travessa 3 nº 158 CEP 05508-900 São Paulo SP
Telefone: (11) 3091-5583 Fax (11) 3091-5294

Departamento de Engenharia de Computação e Sistemas Digitais

utilizaremos o comando a seguir:

```
> qemu-system-arm -M versatilepb -m 128M -nographic -serial /dev/null -kernel irq.elf -s -S
```

- -M versatilepb: Indica que queremos simular a placa versatilepb.
- -m 128M: Indica o tamanho da memória da placa sendo simulada.
- -nographic: Indica que não é necessário simular uma interface gráfica.
- -serial /dev/null: Ignora a porta serial, apontando-a para /dev/null
- -kernel irq.elf: Indica o kernel que desenvolvemos, que deve ser carregado na memória.
- -s -S: indica que o simulador deverá escutar a porta padrão (1234) e esperar pela conexão de um debugger antes de iniciar a simulação.

Para debugar e verificar o funcionamento do código desenvolvido, utilizamos o arm-elf-gdb. Rodamos e apontamos ele para a placa sendo emulada pelo qemu com os seguintes comandos:

```
>arm-elf-gdb irq.elf
> (gdb) target remote :1234
```

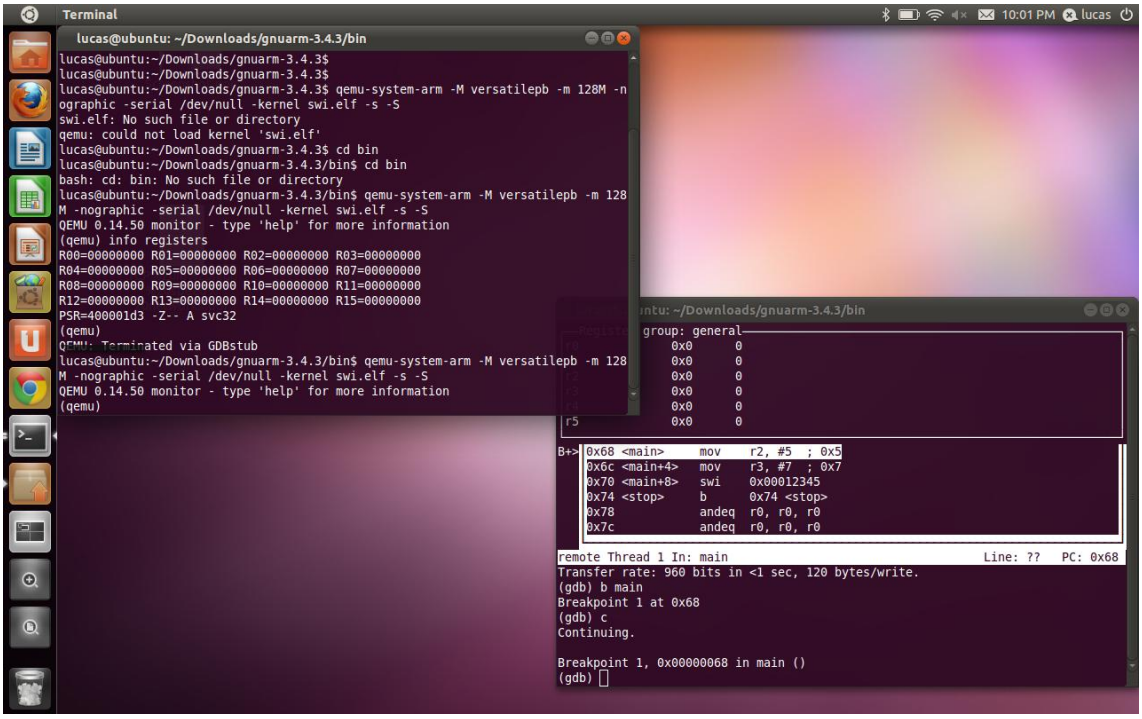
Feito isso, podemos colocar breakpoints, executar passo a passo e utilizar todas os outros recursos do gdb normalmente.

Abaixo temos uma imagem de duas telas de terminal, uma simulando a placa com o qemu-system-arm e outra rodando o arm-elf-gdb conectado à placa:

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Avenida Professor Luciano Gualberto, travessa 3 nº 158 CEP 05508-900 São Paulo SP
Telefone: (11) 3091-5583 Fax (11) 3091-5294

Departamento de Engenharia de Computação e Sistemas Digitais



2.2 Linker Script

O script abaixo é usado durante a compilação do código para especificar que as instruções na região text devem ser alocadas à partir do endereço 0x0 pelo linker. Isso é necessário para que o vetor de interrupções que definiremos fique na posição correta da memória.

```
ENTRY(_start)
SECTIONS
{
  . = 0x0;
  .text : { * (.text); }
}
```

O comando abaixo indica que o arm-elf-ld deve utilizar o linker script definido no arquivo irqld.ld:

```
> ./arm-elf-ld -T irqld.ld -o irq.elf irq.o
```

3. Procedimentos

3.1 Definicao do vetor de interrupções

A arquitetura arm especifica a estrutura do vetor de interrupções, que deve ficar na posição 0x0 da memória, como já mencionado. Abaixo temos o trecho de código que define esse vetor. A posição 0x18 corresponde às interrupções do tipo IRQ, que utilizaremos para o timer.

```
.global _start
.text
_start:
 b _Reset                          @posição 0x00 - Reset
 ldr pc, _undefined_instruction    @posição 0x04 - Intrução não-definida
 ldr pc, _software_interrupt       @posição 0x08 - Interrupção de Software
 ldr pc, _prefetch_abort           @posição 0x0C - Prefetch Abort
 ldr pc, _data_abort               @posição 0x10 - Data Abort
 ldr pc, _not_used                 @posição 0x14 - Não utilizado
 ldr pc, _irq                      @posição 0x18 - Interrupção (IRQ)
 ldr pc, _fiq                      @posição 0x1C - Interrupção(FIQ)

_undefined_instruction: .word undefined_instruction
_software_interrupt:    .word software_interrupt
_prefetch_abort:    .word prefetch_abort
_data_abort:        .word data_abort
_not_used:      .word not_used
```

```
_irq:       .word irq
_fiq:       .word fiq
```

3.2 Definição de constantes

Após a definição do vetor de interrupções, definimos as constantes que utilizaremos, especificamente os endereços dos registradores de controle de interrupções e de timers:

```
INTPND:             .word 0x10140000 @Interrupt status register
INTSEL:             .word 0x1014000C @interrupt select register( 0 = irq, 1 = fiq)
INTEN:        .word 0x10140010 @interrupt enable register
TIMER0L:      .word 0x101E2000 @Timer 0 load register
TIMER0V:      .word 0x101E2004 @Timer 0 value registers
TIMER0C:      .word 0x101E2008 @timer 0 control register
TIMER0X:      .word 0x101E200c @timer 0 interrupt clear register
```

3.3 Tratamento das Interrupções

Em seguida, temos as rotinas de tratamento de interrupções. Além da rotina de tratamento IRQ, incluímos a rotina de tratamento de interrupções de software, para ilustrar que o código também pode ser usado para estudá-las.

```
_Reset
        bl  main
        b   .

undefined_instruction:
        b   .

software_interrupt:
        b   do_software_interrupt @vai para o handler de interrupções de software
```

```
prefetch_abort:
        b   .

data_abort:
        b   .

not_used:
        b   .

irq:
        b   do_irq_interrupt @vai para o handler de interrupções IRQ

fiq:
        b   .

do_software_interrupt:      @Rotina de Interrupçãode software
 add r1, r2, r3          @r1 = r2 + r3
 mov pc, r14                    @volta p/ o endereço armazenado em r14

do_irq_interrupt: @Rotina de interrupções IRQ

        STMFDsp!, {r0 - r3, LR}       @Empilha os registradores

        LDR     r0, INTPND @Carrega o registrador de status de interrupção
        LDR     r0, [r0]

        TST     r0, #0x0010 @verifica se é uma interupção de timer
        BNE     handler_timer  @vai para o rotina de tratamento da interupção de timer

        LDMFD   sp!, {r0 - r3,lr}       @retorna
        mov pc, r14
```

3.4 Tratamento da interrupção de timer

Em seguida, temos a rotina de tratamento da interrupção de timer. A primeira coisa que ela deve fazer é limpar o pedido de interrupção, escrevendo no registrador

Avenida Professor Luciano Gualberto, travessa 3 nº 158 CEP 05508-900 São Paulo SP
Telefone: (11) 3091-5583 Fax (11) 3091-5294

Departamento de Engenharia de Computação e Sistemas Digitais

Interrupt Clear do timer, para evitar que o processador permaneça em loop na interrupção. Feito isso, podemos executar o código que quisermos na interrupção.

```
handler_timer:

        LDR r0, TIMER0X
        MOV r1, #0x0
        STR r1, [r0] @Escreve no registrador TIMER0X para limpar o pedido de
interrupção

        @ Inserir código que sera executado na interrupção de timer aqui
(chaveamento de processos,  ou alternar LED por exemplo)

        LDMFD   sp!, {r0 - r3,lr}
        mov pc, r14 @retorna
```

3.5 Rotina de inicialização do timer

Abaixo temos a rotina responsável por ativar as interrupções de timer e inicializá-lo da forma desejada. Inicialmente é preciso ativar as interrupções IRQ limpando o bit 7 do CPSR. Em seguida, setamos o quarto bit do registrador INTEN pra habilitar as interrupções do timer. Por fim, escrevemos no registrador TIMER0C para ativar o timer e escolher seu modo de operação. Podemos então escrever no TIMER0V para setar o valor atual do timer, se assim desejarmos.

```
timer_init:
        mrs     r0, cpsr
        bic     r0,r0,#0x80
        msr     cpsr_c,r0 @enabling interrupts in the cpsr
        LDR r0, INTEN
        LDR r1,=0x10 @bit 4 for timer 0 interrupt enable
        STR r1,[r0]
        LDR r0, TIMER0C
        LDR r1, [r0]
        MOV r1, #0xA0 @enable timer module
        STR r1, [r0]
```

```
        LDR r0, TIMER0V
        MOV r1, #0xff @setting timer value
        STR r1,[r0]
        mov pc, lr
```

3.6 Programa principal

Para testarmos o funcionamento correto do código acima, escrevemos um programa simples que chama a rotina de inicialização das interrupções e em seguida passa a rodar em loop infinito, permitindo que verifiquemos quando ocorrem as interrupções.

```
main:
        bl timer_init @initialize interrupts and timer 0
stop:   b stop
```

4. Conclusão

Com o código exibido acima, fomos capazes de simular a ocorrência de interrupções de timer corretamente no Qemu, podendo inclusive debugar utilizando o arm-elf-gdb e executar passo a passo as rotinas de tratamento de interrupções.
Acreditamos que o Qemu utilizado dessa forma pode ser de grande utilidade didática para o entendimento de interrupções em microprocessadores.

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Avenida Professor Luciano Gualberto, travessa 3 nº 158 CEP 05508-900 São Paulo SP
Telefone: (11) 3091-5583 Fax (11) 3091-5294

Departamento de Engenharia de Computação e Sistemas Digitais

5. Bibliografia

Versatile Application Baseboard for ARM926EJ-S User Guide
http://infocenter.arm.com/help/topic/com.arm.doc.dui0225d/DUI0225D_versatile_application_baseboard_arm926ej_s_ug.pdf

ARM Dual-Timer Module (SP804) Technical Reference Manual
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0271d/DDI0271.pdf

PrimeCell ® VectoredInterrupt Controller (PL190) Technical Reference Manual
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0181e/DDI0181.pdf