

## LAB7.1: 基于 KCPSM3 的中断功能测试与信号监控

### 1. 实验目的

1. 了解 FPGA 的开发部流程。
2. 熟悉 Spartan-3E 开发套件。
3. 熟悉 PicoBlaze 8 位控制器。

### 2. 实验环境

- **硬件:** Xilinx FPGA 开发板 (仿真环境)
- **软件:**
  - Xilinx ISE 14.7 (综合与实现)
  - ModelSim 10.4 (波形仿真)
- **代码库:**
  - kcpsm3.vhd (KCPSM3 微控制器)
  - int\_test.vhd (程序 ROM)
  - kcpsm3\_int\_test.vhd (顶层模块)
  - test\_bench.vhd (测试平台)

### 3. 实验内容与步骤

根据参考书第七章的 LAB1 内容, 创建工程, 添加相应 HDL 源文件, 编译设计, 功能仿真后, 在此成果的基础上, 对原有 test\_bench.vhd 修改, 使之显示出信号:

- address: 处理器输出的指令地址
- instruction: ROM 返回的指令数据
- interrupt: 中断请求信号
- interrupt\_ack: 中断确认信号
- write\_strobe: 写使能信号

最后对工程中的源码一一研读, 对相应代码加入注释。

#### 3.1 实验需求分析

- **功能需求:** 通过外部中断事件触发 KCPSM3 的中断服务程序, 验证中断响应逻辑。
- **监控需求:** 在波形仿真中观察以下信号:
  - address: 处理器输出的指令地址。
  - instruction: ROM 返回的指令数据。
  - interrupt: 中断请求信号。
  - interrupt\_ack: 中断确认信号。
  - write\_strobe: 写使能信号。

#### 3.2 代码修改与调试

##### 3.2.1 修改 kcpsm3\_int\_test.vhd

- **步骤 1: 添加输出端口**

在实体中声明新增的监控信号:

```
entity kcpsm3_int_test is
```

```
Port (
```

```
-- 原有端口
```

```
counter          : out std_logic_vector(7 downto 0);
```

```
waveforms        : out std_logic_vector(7 downto 0);
```

```
interrupt_event   : in  std_logic;
```

```
clk              : in  std_logic;
```

```

        -- 新增监控信号
        address      : out std_logic_vector(9 downto 0);
        instruction   : out std_logic_vector(17 downto 0);
        interrupt     : out std_logic;
        interrupt_ack  : out std_logic;
        write_strobe  : out std_logic
    );
end kcpsm3_int_test;

```

- **步骤 2: 重命名内部信号**  
为避免命名冲突，将内部信号重命名并映射到输出端口：

architecture Behavioral of kcpsm3\_int\_test is

```

    signal internal_address      : std_logic_vector(9 downto 0);
    signal internal_instruction : std_logic_vector(17 downto 0);
    signal internal_interrupt    : std_logic := '0';
    signal internal_intr_ack     : std_logic;
    signal internal_wr_strobe    : std_logic;
begin
    -- 映射到实体端口
    address      <= internal_address;
    instruction   <= internal_instruction;
    interrupt     <= internal_interrupt;
    interrupt_ack <= internal_intr_ack;
    write_strobe  <= internal_wr_strobe;
end Behavioral;

```

### 3.2.2 修改 test\_bench.vhd

- **步骤 1: 更新组件声明**  
在测试平台中声明新增的端口：

```

COMPONENT kcpsm3_int_test
    Port (
        -- 原有端口
        counter      : out std_logic_vector(7 downto 0);
        waveforms    : out std_logic_vector(7 downto 0);
        interrupt_event : in  std_logic;
        clk           : in  std_logic;
        -- 新增端口
        address      : out std_logic_vector(9 downto 0);
        instruction   : out std_logic_vector(17 downto 0);
        interrupt     : out std_logic;
        interrupt_ack  : out std_logic;
        write_strobe  : out std_logic
    );
END COMPONENT;

```

- **步骤 2: 连接新增信号**  
在实例化模块时完成端口映射：

```

uut: kcpsm3_int_test
port map (
    counter          => counter,
    waveforms        => waveforms,
    interrupt_event   => interrupt_event,
    clk               => clk,
    address           => address,
    instruction       => instruction,
    interrupt         => interrupt,
    interrupt_ack     => interrupt_ack,
    write_strobe      => write_strobe
);

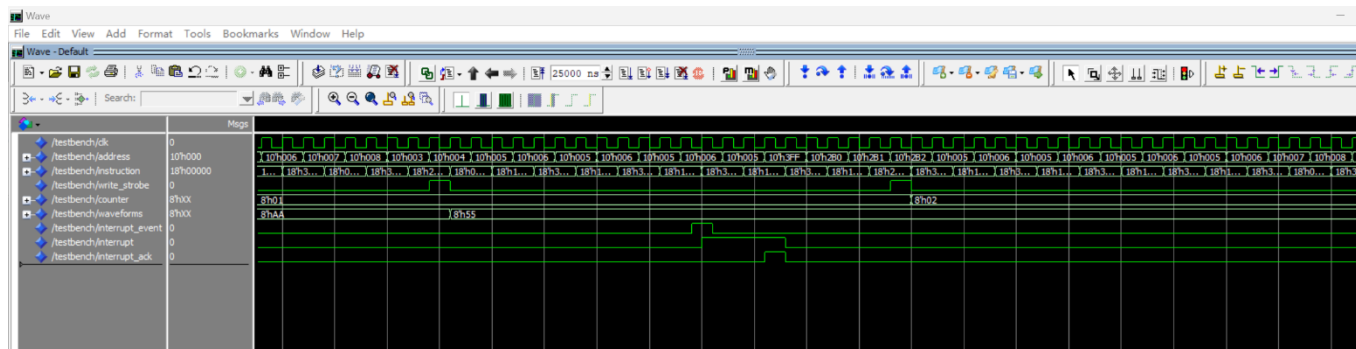
```

### 3.3 错误与解决

- **错误 1:** Undefined symbol 'internal\_instr'  
原因: 未正确声明重命名的内部信号。  
解决: 在架构中声明 internal\_instruction 并映射到实体端口。
- **错误 2:** Object write\_strobe of mode OUT can not be read  
原因: 直接读取输出端口 write\_strobe。  
解决: 通过内部信号 internal\_wr\_strobe 中转。

## 4. 实验结果与分析

### 4.1 波形仿真结果



- **关键信号行为:**
  1. **中断触发:** 在时钟周期 30、67、183 时, interrupt\_event 置高, 触发 interrupt 信号——interrupt<='1' (clk 上升沿触发)。
  2. **中断响应:** KCPSM3 响应中断后, interrupt\_ack 短暂置高 (当 ack 信号为 '1' 后, interrupt <='0')。
  3. **地址与指令:** address 随程序执行递增, instruction 显示 ROM 中的指令数据。
  4. **写使能:** write\_strobe 在数据写入外设时置高, 当 port\_id (1) = '1' 时: waveforms <= out\_put, port\_id (2) = '1' 时: counter <= out\_put。
  5. **ADDRESS 信号:** 3FF 是中断向量地址, 2B0 是中断处理函数的地址

### 4.2 功能验证

- **中断功能:** 中断服务程序正确执行, counter 和 waveforms 按预期更新。

- **信号监控：**所有新增信号在波形中可见，逻辑行为符合设计预期。

## 5. 实验总结

### 1. 成果：

- 成功验证 KCPSM3 的中断机制。
- 实现关键信号的波形监控，提升调试效率。

### 2. 经验：

- 模块化设计中需严格管理信号命名与映射。
- 测试平台需与设计实体保持端口一致性。

## 附录：相应汇编代码及其注释

# Interrupt Service Routine

In the assembler log file for the example, it can be seen that the interrupt service routine has been forced to compile at address '2B0', and that the waveform generation is located in the base addresses. This makes it easier to observe the interrupt in action in the operation waveforms. This program is supplied as 'int\_test.psm' for you to assemble yourself.


```

000                                ;Interrupt example
000                                ;
000                                CONSTANT waveform_port, 02                ;bit0 will be data
000                                CONSTANT counter_port, 04
000                                CONSTANT pattern_10101010, AA
000                                NAMEREG sA, interrupt_counter
000                                ;
000 00A00    start: LOAD interrupt_counter[sA], 00                        ;reset interrupt counter
001 002AA    LOAD s2, pattern_10101010[AA]                            ;initial output condition
002 3C001    ENABLE INTERRUPT
003                                ;
003 2C202    drive_wave: OUTPUT s2, waveform_port[02]
004 00007    LOAD s0, 07
005 1C001    loop: SUB s0, 01
006 35405    JUMP NZ, loop[005]
007 0E2FF    XOR s2, FF
008 34003    JUMP drive_wave[003]
009                                ;
009 2B0      ADDRESS 2B0
009 18A01    int_routine: ADD interrupt_counter[sA], 01                ;increment counter
010 2CA04    OUTPUT interrupt_counter[sA], counter_port[04]
011 38001    RETURNI ENABLE
012 2B3      ;
013 3FF      ADDRESS 3FF
014 342B0    JUMP int_routine[2B0]
  
```

Main program delay loop where most time is spent

Interrupt Service Routine (located at address 2B0 onwards)

Interrupt vector set at address 3FF and causing JUMP to service routine



KCPSM3 Manual 60

```

000                                ;Interrupt example
000                                ;
000                                CONSTANT waveform_port, 02                ;定义波形输出端口 bit0 为数据
000                                CONSTANT counter_port, 04                ; 定义计数器输出端口
000                                CONSTANT pattern_10101010, AA            ; 定义波形模式 10101010
000                                NAMEREG sA, interrupt_counter            ; 将 寄存器 sA 命名 为
                                interrupt_counter
000                                ;
000 00A00    start: LOAD interrupt_counter, 00                        ;复位 interrupt counter 为 0
001 002AA    LOAD s2, pattern_10101010[AA]                            ;初始 s2 为波形模式 10101010
002 3C001    ENABLE INTERRUPT                                          ; 启用中断
003                                ;
003 2C202    drive_wave: OUTPUT s2, waveform_port[02]                ; 输出 s2 的值到波形端口
004 00007    LOAD s0, 07                                              ;delay size
005 1C001    loop: SUB s0, 01                                          ;delay loop
006 35405    JUMP NZ, loop[005] ; s0 不为 0 则跳回 loop
  
```

007 0E2FF	XOR s2, FF	;反转 waveform
008 34003	JUMP drive_wave[003]	
009	;	
2B0	ADDRESS 2B0 ; 设置中断处理程序的地址	
2B0 18A01	int_routine: ADD interrupt_counter[sA], 01	;increment counter
2B1 2CA04	OUTPUT interrupt_counter[sA], counter_port[04]	
2B2 38001	RETURNI ENABLE ; 返回并启用中断	
2B3	;	
3FF	ADDRESS 3FF	;设置中断向量的地址
3FF 342B0	JUMP int_routine[2B0]	