İSTANBUL TEKNİK ÜNİVERSİTESİ

ELEKTRİK ELEKTRONİK FAKÜLTESİ

INTRODUCTION TO EMBEDDED SYSTEMS

(EHB 326E)

Picoblaze Interrupt Handling Report

Hasan Emre AYDEMİR

1. Explanation of Call/Return Stack and Interrupts

The PicoBlaze processor uses a dedicated hardware call/return stack to store return addresses during both subroutine calls and interrupt operations. When a CALL instruction is executed, the Program Counter (PC) of the next instruction is pushed onto the stack and the processor jumps to the target subroutine. When a RETURN instruction is executed, the last stored address is popped from the stack and execution continues from that location. The same stack mechanism is used for interrupt handling. When an interrupt occurs, PicoBlaze automatically pushes the current return address onto the call/return stack and jumps to the interrupt vector address. The Interrupt Service Routine (ISR) is terminated using the RETURNI instruction, which restores the previously saved Program Counter. Since the call/return stack has a limited depth, excessive nested calls or interrupts may lead to stack overflow, causing loss of return addresses and incorrect program execution.

The same interrupt structure is used for both experiments; only the number of nested interrupts is modified to observe the behavior of the PicoBlaze call/return stack under different conditions. In this part of the assignment, a PicoBlaze assembly program was developed using the Fidex IDE to demonstrate the interrupt mechanism of the PicoBlaze processor and to analyze the stack behavior during nested interrupt operations.

The interrupt signal is generated internally by the PicoBlaze program through an output port. Specifically, bit 7 of the output port (out_port(7)) is asserted to trigger the interrupt input of the processor, allowing the interrupt mechanism to be tested without any external hardware signals. The interrupt vector address is configured as 0xFFF (4095), which corresponds to the highest address of the 4K instruction memory. When an interrupt occurs and the interrupt enable flag is set, the PicoBlaze processor automatically saves the current Program Counter on the call/return stack and jumps to the Interrupt Service Routine (ISR).

Inside the ISR, nested interrupts are intentionally generated to evaluate stack behavior. Two different configurations are implemented. In the first configuration, 10 nested interrupts are generated, which is within the stack capacity of the PicoBlaze processor. In this case, all return addresses are preserved correctly and the Program Counter successfully returns to the original instruction address after all RETURNI operations. In the second configuration, 32 nested interrupts are generated. Since the PicoBlaze processor provides a limited return stack depth (approximately 31 levels shared between CALL and INTERRUPT operations), this configuration exceeds the stack capacity. As a result, stack overflow occurs, causing return address corruption and preventing the Program Counter from restoring the original execution flow.

After the PicoBlaze assembly program was written and verified in the Fidex IDE, it was assembled using the KCPSM6 assembler backend. During this process, the assembler translated the .psm assembly file into a hardware-compatible memory description. As a result, the BRAM0.vhd file was automatically generated. This file represents the PicoBlaze instruction memory and contains the initialized contents of the 4K address space (0x000-0xFFF), with the interrupt vector located at address 0xFFF as required by the assignment.

The generated BRAM0.vhd file is based on Xilinx ROM templates and internally instantiates Block RAM primitives (RAMB36E1) with predefined initialization values corresponding to the assembled PicoBlaze instructions. Once generated, this file was added to the Vivado project as a design source and connected to the PicoBlaze processor (kcpsm6) as the instruction memory. This setup enables accurate behavioral simulation of nested interrupt operations and stack overflow behavior without requiring any physical FPGA hardware.

2. Fidex Assembly Code

2.1 Assembly Code – 10 Nested Interrupts

```
;-----------------------------------------
; 10 Nested Interrupts - PicoBlaze (KCPSM6)
;-----------------------------------------
CONSTANT INT_PORT, 01
CONSTANT NEST_MAX, 0A        ; 10 nested
;-----------------------------------------
; Main program at 0x000
;-----------------------------------------
ADDRESS 000
start:
    LOAD s1, 00
    ENABLE INTERRUPT

main_loop:
    ; generate interrupt pulse on out_port(7)
    LOAD s0, 80
    OUTPUT s0, INT_PORT
    LOAD s0, 00
    OUTPUT s0, INT_PORT

    JUMP main_loop
;-----------------------------------------
; ISR body at 0x200
;-----------------------------------------
ADDRESS 200
isr_start:
    ADD s1, 01               ; nest_cnt++

    LOAD s0, NEST_MAX
    COMPARE s1, s0
    JUMP Z, isr_done          ; stop nesting when == max

    ENABLE INTERRUPT          ; allow nesting

    ; trigger another interrupt pulse
    LOAD s0, 80
    OUTPUT s0, INT_PORT
    LOAD s0, 00
    OUTPUT s0, INT_PORT

isr_done:
    RETURNI ENABLE
;-----------------------------------------
; Interrupt Vector MUST be last (0xFFF)
;-----------------------------------------
ADDRESS FFF
    JUMP isr_start
```

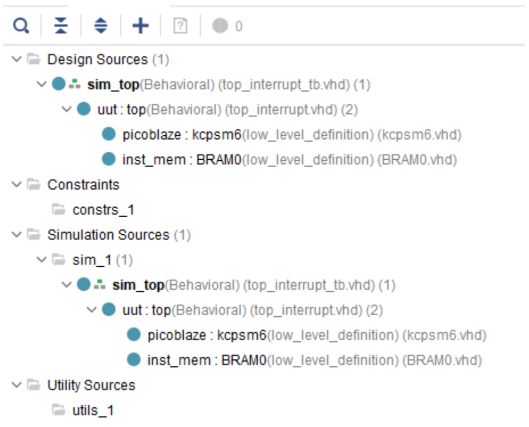
This PicoBlaze program demonstrates nested interrupts generated internally through an I/O port. In the main program (starting at address 0x000), register s1 is cleared and interrupts are enabled, then the program continuously creates an interrupt pulse by writing 0x80 and then 0x00 to INT_PORT (port 0x01), which drives out_port(7) high then low. The interrupt vector is configured at address 0xFFF and redirects execution to isr_start at 0x200. Inside the ISR, s1 is used as a nesting counter and is incremented on each interrupt entry; if s1 reaches NEST_MAX (0x0A = 10), the ISR stops generating further nested interrupts and returns using RETURNI ENABLE. Otherwise, the ISR re-enables interrupts (to allow nesting) and triggers another interrupt pulse through the same I/O port, causing repeated nested entries until 10 levels are reached. This structure allows observing stack growth and shrinkage as the processor pushes and pops return addresses during nested interrupts.

2.2 Assembly Code – 32 Nested Interrupts

```
;=========================================================
; 32 Nested Interrupts - PicoBlaze (KCPSM6)
;=========================================================
CONSTANT INT_PORT, 01
CONSTANT NEST_MAX, 20          ; 32 decimal
; Registers:
; s0 : temp/data
; s1 : nest counter
;-------------------------------
; Main program @ 0x000
;-------------------------------
ADDRESS 000
start:
    LOAD s1, 00
    ENABLE INTERRUPT
main_loop:
    ; Generate first interrupt pulse from main
    LOAD s0, 80
    OUTPUT s0, INT_PORT
    LOAD s0, 00
    OUTPUT s0, INT_PORT

    JUMP main_loop
;-------------------------------
; ISR body @ 0x200
;-------------------------------
ADDRESS 200
isr_start:
    ; Count nested level
    ADD s1, 01

    ; If s1 == NEST_MAX, stop nesting and return
    LOAD s0, NEST_MAX
    COMPARE s1, s0
    JUMP Z, isr_done

    ; Allow nesting, then trigger another interrupt pulse
    ENABLE INTERRUPT

    LOAD s0, 80
    OUTPUT s0, INT_PORT
    LOAD s0, 00
    OUTPUT s0, INT_PORT
isr_done:
    RETURNI ENABLE
;-------------------------------
; Interrupt vector @ 0xFFF  (MUST be last)
;-------------------------------
ADDRESS FFF
    JUMP isr_start
```

This KCPSM6 PicoBlaze program implements the same internally generated interrupt mechanism as the 10-nested case but increases the nesting limit to 32 to observe stack overflow behavior. The main loop at address 0x000 enables interrupts and repeatedly generates an interrupt pulse by writing 0x80 then 0x00 to INT_PORT (0x01), which drives out_port(7) high then low. The interrupt vector at 0xFFF jumps to the ISR located at 0x200. Inside the ISR, register s1 counts the nesting level by incrementing on each interrupt entry; if s1 equals NEST_MAX (0x20 = 32), the ISR stops creating new nested interrupts and returns using RETURNI ENABLE. Otherwise, the ISR re-enables interrupts to allow re-entry and triggers another interrupt pulse via the same I/O port, forcing deeper nesting. Because PicoBlaze's call/return stack has a limited depth (typically around 31 levels), attempting 32 nested interrupts exceeds the stack capacity, which can corrupt return addresses and prevent the Program Counter from reliably returning to the original instruction flow.

3. Vivado Design



```
0
Design Sources (1)
    sim_top(Behavioral) (top_interrupt_tb.vhd) (1)
        uut : top(Behavioral) (top_interrupt.vhd) (2)
            picoblaze : kcpsm6(low_level_definition) (kcpsm6.vhd)
            inst_mem : BRAM0(low_level_definition) (BRAM0.vhd)
Constraints
    constrs_1
Simulation Sources (1)
    sim_1 (1)
        sim_top(Behavioral) (top_interrupt_tb.vhd) (1)
            uut : top(Behavioral) (top_interrupt.vhd) (2)
                picoblaze : kcpsm6(low_level_definition) (kcpsm6.vhd)
                inst_mem : BRAM0(low_level_definition) (BRAM0.vhd)
Utility Sources
    utils_1
```

Figure 1 Vivado Sources

In this stage, the PicoBlaze-based interrupt system was implemented in Vivado for behavioral simulation. No additional HDL code was written. Instead, the design was constructed by integrating the previously generated and provided modules.

The PicoBlaze processor core (kcpsm6.vhd) was used as the central processing unit, while the program memory was implemented using the BRAM0.vhd file generated by the Fidex assembler. These two components were connected inside the provided top level module (top_interrupt.vhd).

The interrupt signal was generated internally by the PicoBlaze program through an output port, where out_port(7) was connected to the interrupt input of the processor. This allowed the interrupt mechanism to be tested entirely in simulation without any external hardware.

For verification purposes, the provided testbench file (top_interrupt_tb.vhd) was used as the simulation top module. All simulations were performed using Vivado's behavioral simulation environment, and relevant internal signals such as the program counter, stack pointer, interrupt signals, and interrupt enable flag were observed through waveform analysis.

4. Simulation Results (Waveform)

4.1 Nested Interrupt Behavior (10 Nested)

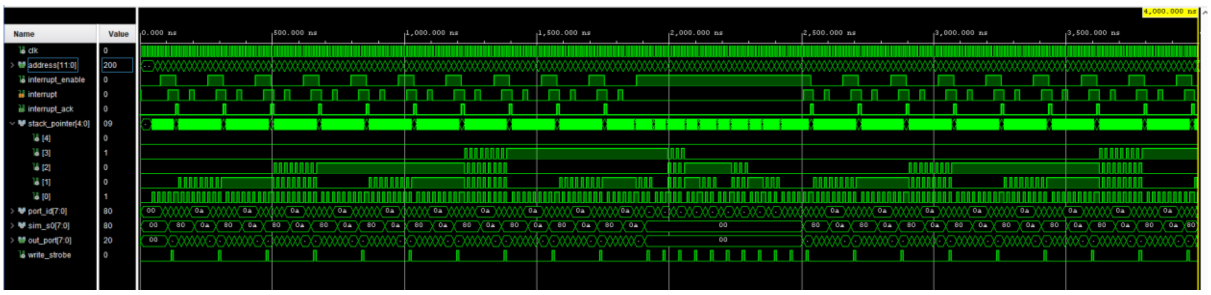


Figure 2 Interrupt enable/acknowledge and stack activity (10 nested)

Figure 2 shows the interrupt enable, interrupt acknowledge and call/return stack behavior during nested interrupts. The interrupt_enable signal remains active while nested interrupts are being generated. The interrupt and interrupt_ack pulses confirm that each interrupt request is correctly acknowledged by the PicoBlaze processor.

Shows the behavior of the PicoBlaze call/return stack during 10 nested interrupts. The stack_pointer signal increases step-by-step from 0 to 10 as each interrupt occurs and a return address is pushed onto the stack. After the last nested interrupt, the stack_pointer decreases from 10 back to 0 as each RETURNI instruction pops a return address from the stack. This proves that the PicoBlaze stack correctly stores and restores the Program Counter for 10 nested interrupt levels.

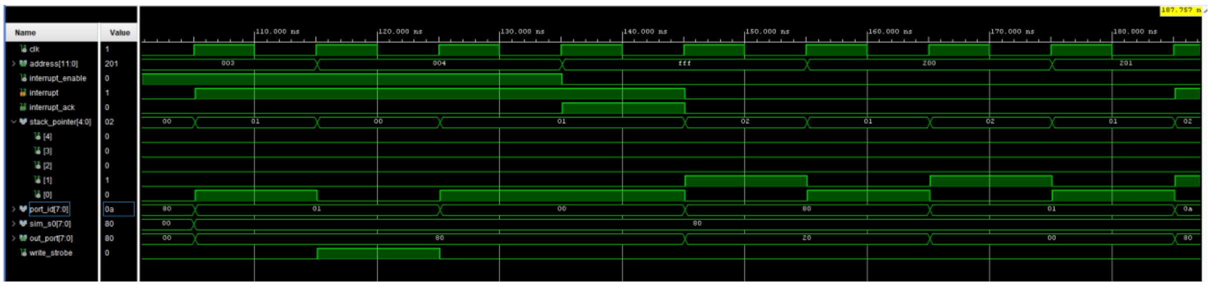


Figure 3 Interrupt vector jump to address 0xFFF

Figure 3 demonstrates the interrupt vector operation of the PicoBlaze processor. When an interrupt request occurs, the Program Counter (address) jumps from the main program addresses (0x003 and 0x004) directly to the interrupt vector address **0xFFF**. After this jump, the processor continues execution at the Interrupt Service Routine entry address **0x200**. The stack_pointer increases during the interrupt entry, proving that the return address is pushed onto the call/return stack. This confirms that the interrupt vector is correctly configured to 0xFFF.

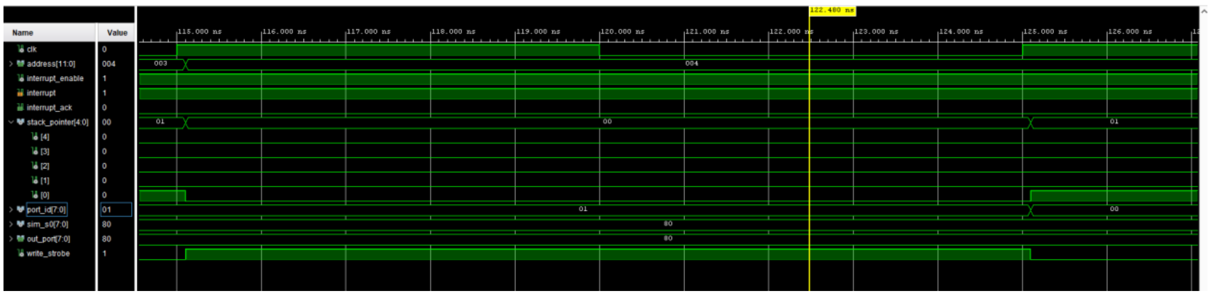


Figure 4 ***I/O port generated interrupt during 10 nested operation***

Figure 4 shows that the interrupt signal is generated internally by the PicoBlaze processor using an I/O port. While the write_strobe signal is active, the processor writes data to the I/O address port_id = 01. At this moment, the value **0x80** is written to out_port(7), setting bit 7 to logic ‘1’ and creating a short interrupt pulse. Immediately after, the value is returned to **0x00**, completing the interrupt pulse. The interrupt signal becomes active at the same time, proving that the interrupt is generated by software through the I/O interface.

The results show that PicoBlaze correctly handles 10 nested interrupts using its call/return stack and successfully returns the Program Counter to the original instruction address. The interrupt signal is generated internally using an I/O port and the interrupt vector address 0xFFF operates as expected.

4.3 Nested Interrupt Behavior (32 Nested)

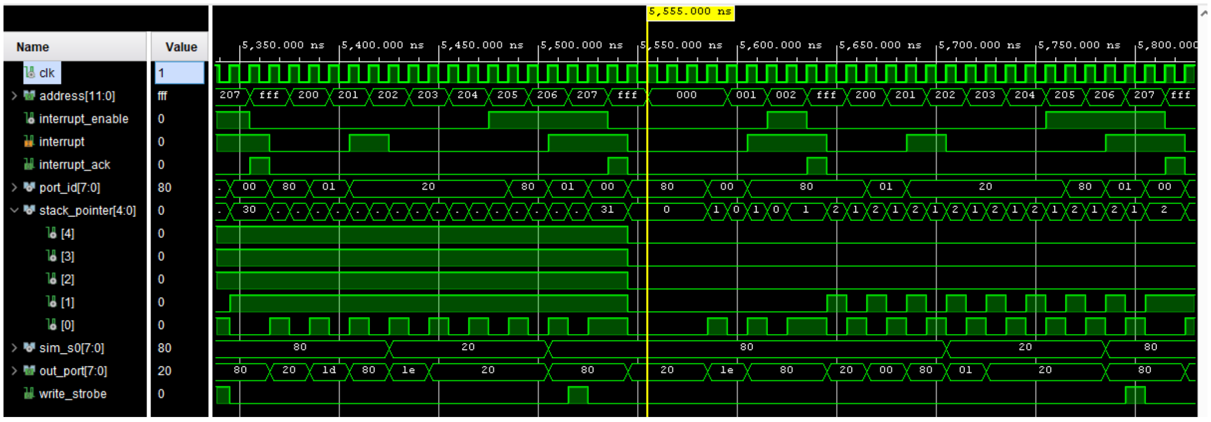


Figure 5 Stack overflow during 32 nested interrupts

Figure 5 shows the behavior of the PicoBlaze call/return stack when 32 nested interrupts are generated. The stack_pointer increases until it reaches its maximum capacity 31, and then overflows. After the overflow, the stack_pointer can no longer return to zero correctly. At the same time, the Program Counter (address) fails to restore the original instruction address and instead jumps to unexpected program locations. This proves that PicoBlaze has a limited stack depth and that 32 nested interrupts cause stack overflow.

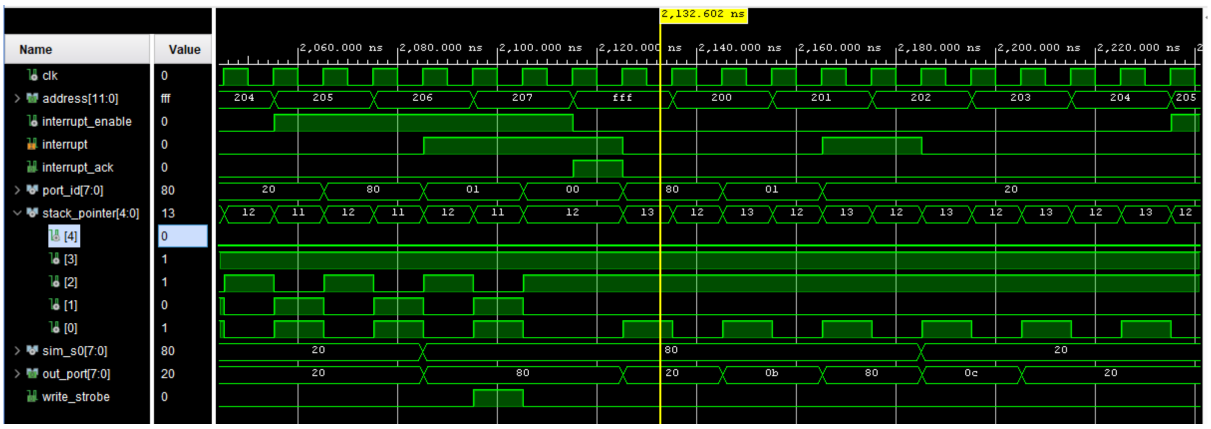


Figure 6 Corrupted stack and incorrect PC return (32 nested)

Figure 6 demonstrates the interrupt vector operation of the PicoBlaze processor under 32 nested interrupt conditions. When an interrupt request occurs, the Program Counter (address) still jumps to the interrupt vector address 0xFFF and then to the ISR entry address 0x200. However, unlike the 10-nested case, the stack_pointer does not return to zero after the nested interrupts. Instead, it oscillates between corrupted values (such as 12 and 13), indicating that the call/return stack has overflowed. As a result, the Program Counter fails to restore the original program flow, proving that the stack depth limit has been exceeded.

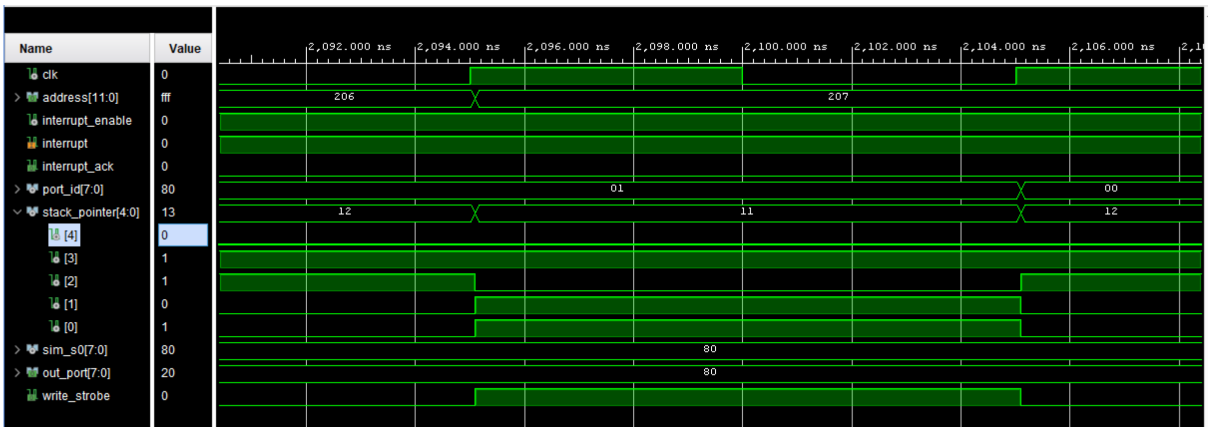


Figure 7 I/O port generated interrupt during 32 nested operation

Figure 7 shows that the interrupt signal continues to be generated internally by the PicoBlaze processor using an I/O port during the 32 nested interrupt test. While the write_strobe signal is active, the processor writes data to the I/O address port_id = 01. At this moment, the value **0x80** is written to out_port(7), setting bit-7 to logic ‘1’ and creating a short interrupt pulse. Immediately after, the value returns to **0x00**, completing the interrupt pulse. The interrupt signal becomes active at the same time, confirming that the interrupt is still generated by software through the PicoBlaze I/O interface even under overflow conditions.

In conclusion:

In this study, the interrupt and call/return stack behavior of the PicoBlaze processor was experimentally investigated. The interrupt signal was successfully generated internally by using an I/O port, and the interrupt vector address was correctly configured to 0xFFF. For the 10 nested interrupt case, the stack_pointer increased from 0 to 10 and then returned to 0, proving that the PicoBlaze call/return stack correctly preserved and restored the Program Counter. However, in the 32 nested interrupt case, the stack depth limit was exceeded, causing stack overflow. As a result, the stack_pointer could not return to zero and the Program Counter failed to restore the original program flow. These results confirm both the correct operation of nested interrupts within the stack limits and the overflow behavior when the stack capacity is exceeded.