Introduction to Embedded Systems – EHB 326E - 10791

Prof. Dr. Müştak Erhan Yalçın

Arş. Gör. Serdar Duran

Homework-4

Yakup Bozkurt

040180048

Call/Return Stack and Interrupts

Stack is a term used for a data structure type in programming. It has a spesific order of the operations which is Last In First Out(LIFO). The CALL/RETURN Stack is a storage for instruction addresses which has the range of 31. Thanks to the CALL/RETURN Stack, we can activate the nested CALL sequences up to 31 levels deep. Also for the activation of the interrupts, there must be a location occupied. Because the CALL/RETURN Stack is used for interrupt actions as well.

A program memory is not necessary fort he stack because there are no instructions to control the stack or the stack pointer due to implementation of the stack as a seperate cyclic buffer.

Fort the handling of asyncronous events, the PicoBlaze micro controller has an optional Interrupt input. These event can be defined as actions that may occur any time of the program. The time of the PicoBlaze microcontroller responding to an interrupt is five clock cycles.

1)

- I created a FIDEx project and do the necessary configurations.

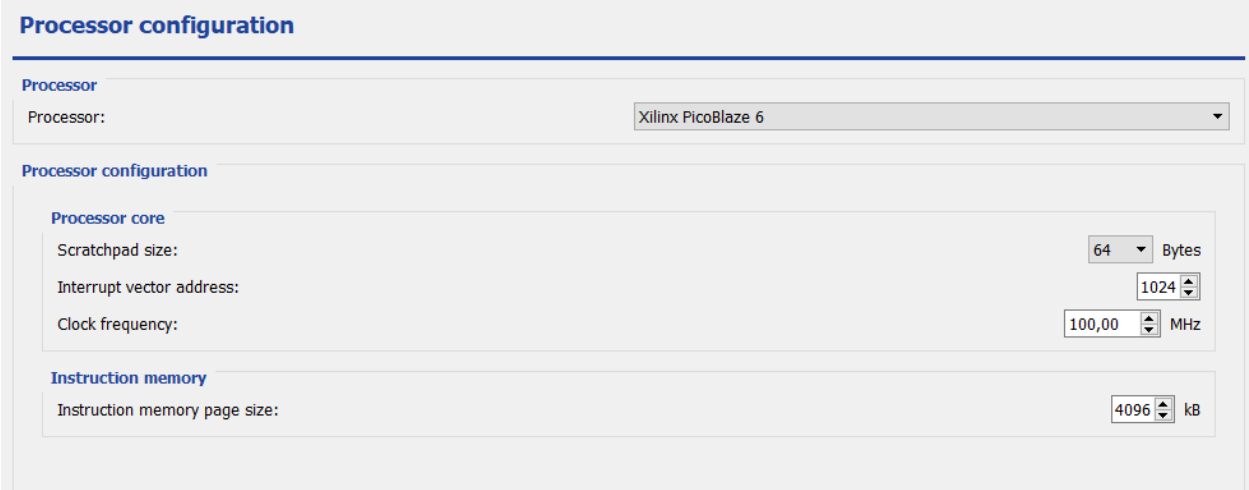


Figure 1

- I wrote the assembly code which has 10 nested interrupts.

```
        #ifDef proc::xPblze6

            #set proc::xPblze6::scrpdSize,          64                  ; [64, 128, 256]
            #set proc::xPblze6::clkFreq,            100000000           ; in Hz

            #set IOdev::BRAM0::en,                  TRUE
            #set IOdev::BRAM0::type,                mem
            #set IOdev::BRAM0::size,                4096

            #set instmem::pageSize,                 4096
            #set instmem::pageCount,                1
            #set instmem::sharedMemLocation,        loMem   ; [ hiMem, loMem ]

            #set IOdev::BRAM0::value,               instMem

            #set IOdev::BRAM0::vhdlEn,              TRUE
            #set IOdev::BRAM0::vhdlEntityName,      "BRAM0"
            #set IOdev::BRAM0::vhdlTmplFile,        "ROM_form.vhd"
            #set IOdev::BRAM0::vhdlTargetFile,      "BRAM0.vhd"
        #endIf

#ORG ADDR, 00
0x000       INT ENABLE
0x001       LOAD s0,0       ;counter for the loop
main:
0x002       LOAD s1,0       ;unimptortant instruction just to get interrupt working
0x003       WRPRT 128,0     ;interrupt activate instruction
0x004       LOAD s1,0       ;unimptortant instruction just to get interrupt working
0x005 end2: jump end2

        #ORG ADDR, 1024 ;1024 is the interrupt vector adress which i specified
0x400 isr:  INT ENABLE
0x401       ADD s0,1
0x402       COMP s0,10      ;a loop for interrupt to get activated ten times
0x403       JUMP Z,end
0x404       WRPRT 128,0     ;interrupt activate instruction
0x405       LOAD s1,0       ;unimptortant instruction just to get interrupt working
0x406 end:  RET
```

Figure 2

- I simulated the code and observed the results.

Level	Program counter	Carry flag	Zero flag	Called label
0	3	0	0	ISR
1	404	1	0	ISR
2	404	1	0	ISR
3	404	1	0	ISR
4	404	1	0	ISR
5	404	1	0	ISR
6	404	1	0	ISR
7	404	1	0	ISR
8	404	1	0	ISR
9	404	1	0	ISR

Bank: A

s0 0A s0
s1 00 s1

Figure 3

As we can see in the "*Figure 3*", there are 10 interrupts. After the first one, program counter shows the 404 adress because it is where the interrupts occur in a loop. We can observe that in the "*Figure 2*" line 36. The creation of interrupts stop when s0 reach the value "10".

- I created the BRAM0.vhd file from the assembly code. After that i created a vivado project and add the source files.

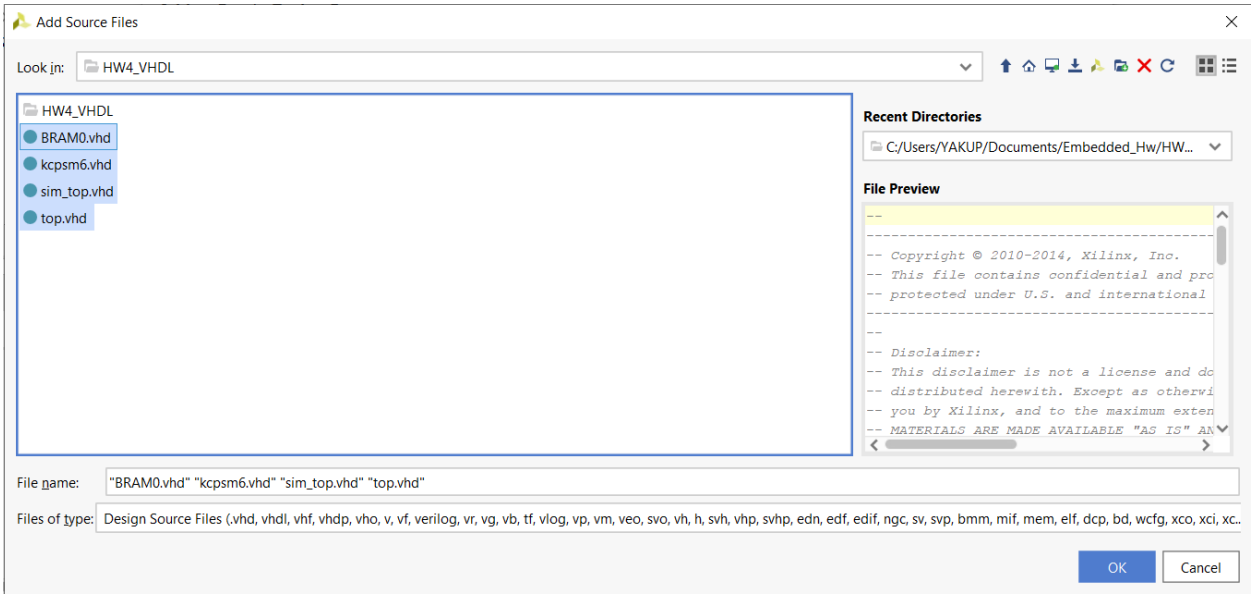


Figure 4

- After the preparation of the vivado project, i changed the interrupt vector as “400” in the top.vhd code. 400 in hexadecimal is 1024 in decimal so i matched the interrupt vector adress in the assembly code and the top.vhd code.

```
                hwbuild => X"00",
        interrupt_vector => X"400",
scratch_pad_memory_size => 64       )
```

- Later, i opened the elaborated design and checked the shematic.

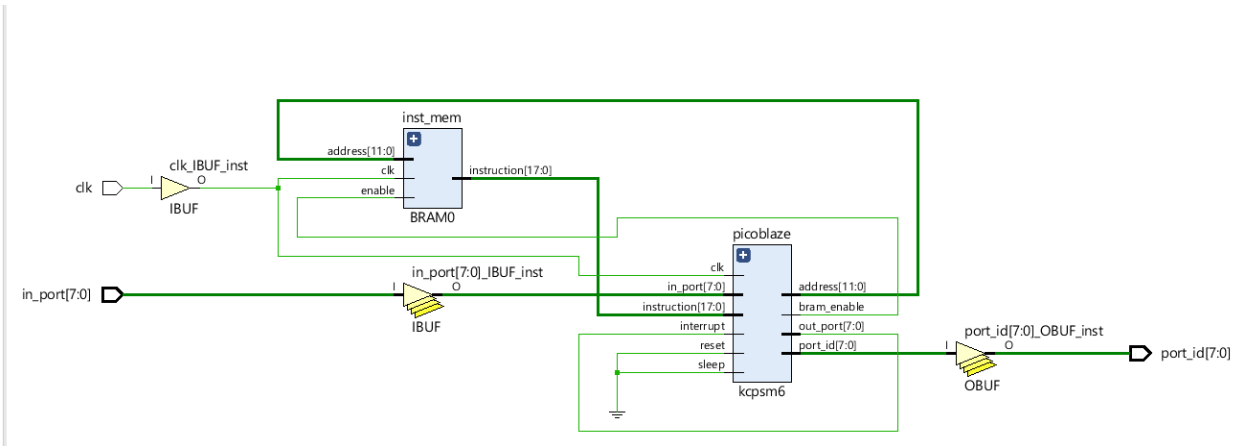


Figure 5

- Finally, i ran the simulation and examine the outcome.

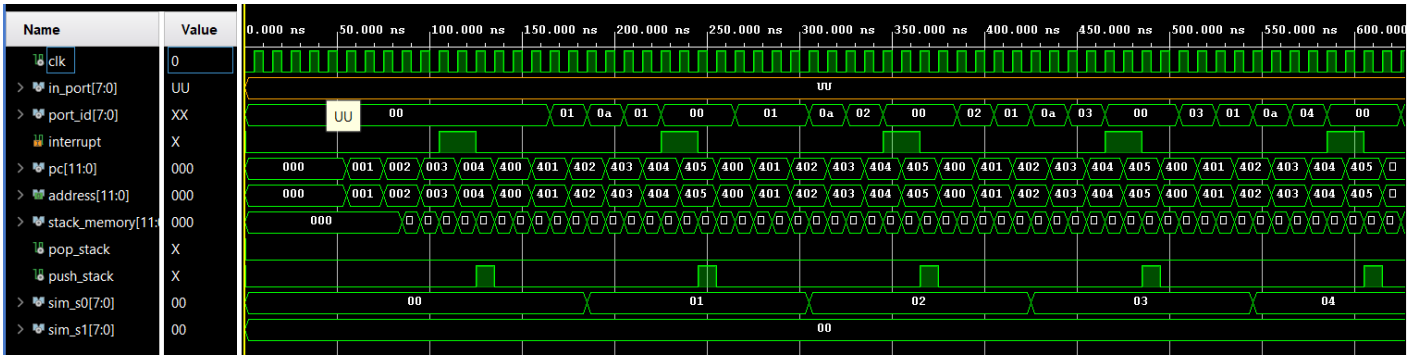


Figure 6

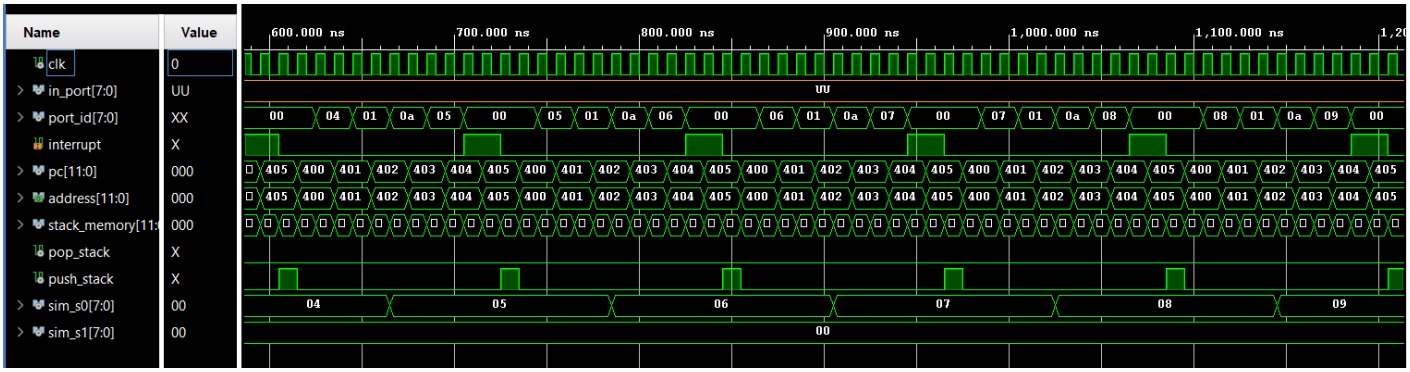


Figure 7

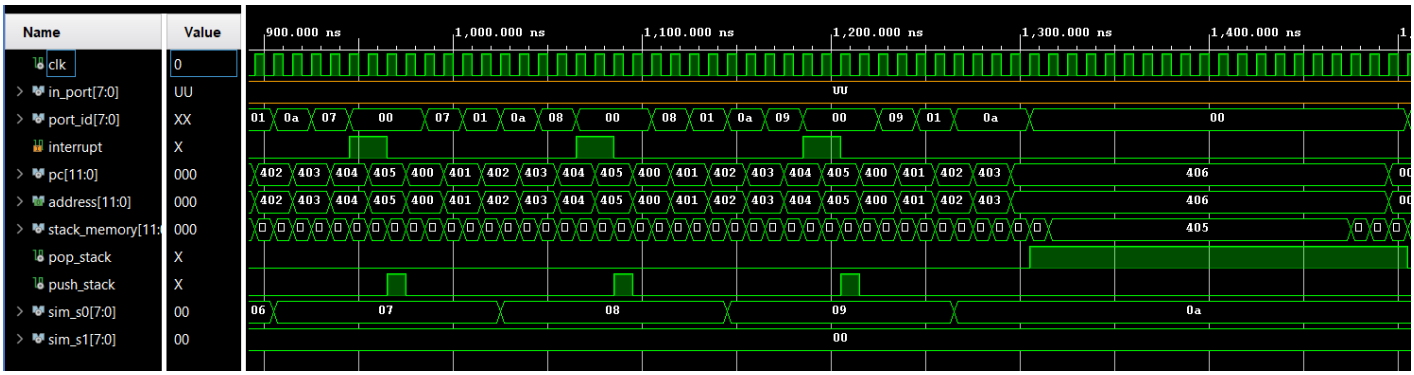
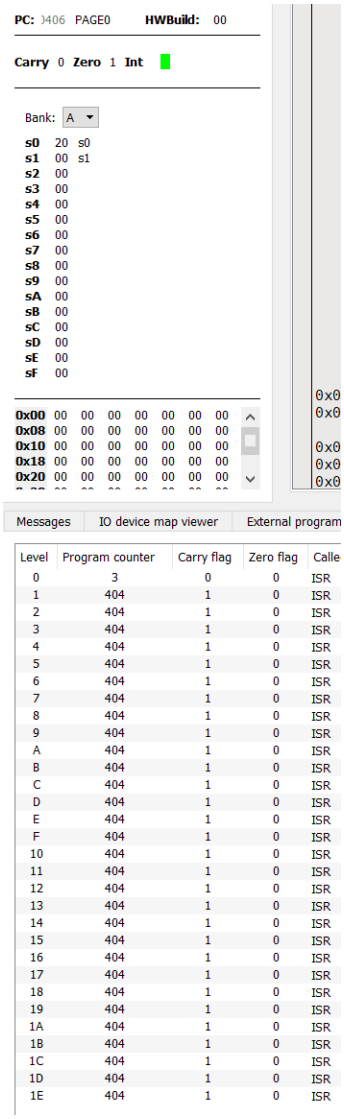


Figure 8

As we can see in the "*Figure 6*", first interrupt occur when the program counter is at 3. After that it always come when the program counter is at 404 just like we observed in the FIDEx program. After the interrupt we can see the push stack is getting loaded. So every time an interrupt occurs, push stack increase. After the push stacks and interrupts, pop stack comes step in and interrupts get erased one by one. We can observe that in the "*Figure 8*". We can also see that s0 register increase one by one until it reaches 10. After that the increase and interrupts stop.

- For the final part, i arranged the assembly code to stop the interrupts after 32 nested interrupts.

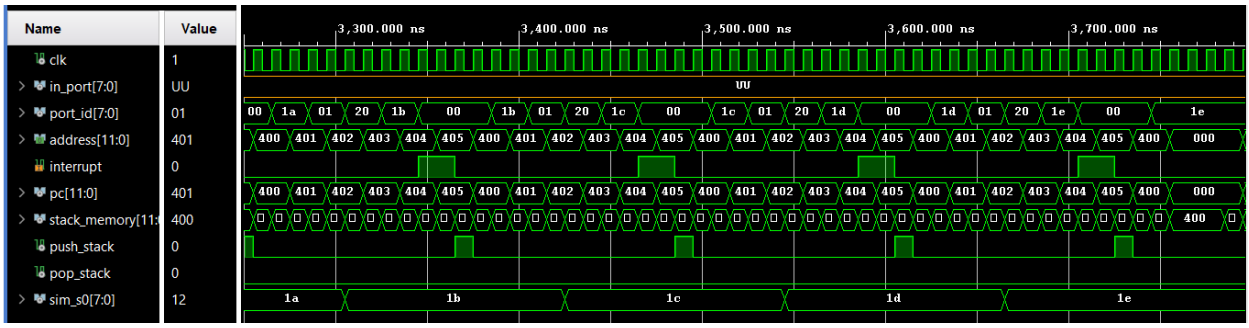
```
        21
        22 #ORG ADDR, 00
0x000   23         INT ENABLE
0x001   24         LOAD s0,0        ;counter for the loop
        25 main:
0x002   26         LOAD s1,0        ;unimptortant instruction just to get interrupt working
0x003   27         WRPRT 128,0      ;interrupt activate instruction
0x004   28         LOAD s1,0        ;unimptortant instruction just to get interrupt working
0x005   29 end2:   jump end2
        30
        31         #ORG ADDR, 1024 ;1024 is the interrupt vector adress which i specified
0x400   32 isr:    INT ENABLE
0x401   33         ADD s0,1
0x402   34         COMP s0,32       ;a loop for interrupt to get activated 32 times
0x403   35         JUMP Z,end
0x404   36         WRPRT 128,0      ;interrupt activate instruction
0x405   37         LOAD s1,0        ;unimptortant instruction just to get interrupt working
0x406   38 end:    RET
```



Level	Program counter	Carry flag	Zero flag	Calle
0	3	0	0	ISR
1	404	1	0	ISR
2	404	1	0	ISR
3	404	1	0	ISR
4	404	1	0	ISR
5	404	1	0	ISR
6	404	1	0	ISR
7	404	1	0	ISR
8	404	1	0	ISR
9	404	1	0	ISR
A	404	1	0	ISR
B	404	1	0	ISR
C	404	1	0	ISR
D	404	1	0	ISR
E	404	1	0	ISR
F	404	1	0	ISR
10	404	1	0	ISR
11	404	1	0	ISR
12	404	1	0	ISR
13	404	1	0	ISR
14	404	1	0	ISR
15	404	1	0	ISR
16	404	1	0	ISR
17	404	1	0	ISR
18	404	1	0	ISR
19	404	1	0	ISR
1A	404	1	0	ISR
1B	404	1	0	ISR
1C	404	1	0	ISR
1D	404	1	0	ISR
1E	404	1	0	ISR

s0 gets increased until there is 31 interrupts. 32 is not possible because the stack memory is full.

- After that, i created the BRAM0.vhd and added to my vivado project and ran the simulation.



We can see the similarity from the first part of the simulation this time interrupts occur until s0 got the 32 value.