

Introduction to Embedded Systems

EHB326E

Lectures

Prof. Dr. Müştak E. Yalçın

Istanbul Technical University

mustak.yalcin@itu.edu.tr

PICOBLAZE OVERVIEW

The PicoBlaze processor is a compact 8-bit microcontroller core for Xilinx FPGA devices.

Required Reading:

- Xilinx, "PicoBlaze 8-bit Embedded Microcontroller User Guide", https://www.xilinx.com/support/documentation/ip_documentation/ug129.pdf [▶ LINK](#) [▶ PDF](#)
- Pong P. Chu, "FPGA Prototyping by VHDL Examples: Xilinx Spartan -3 Version", Chapter 14, Wiley 2008, [▶ PDF](#)

PicoBlaze Block Diagram

CONTROLLER

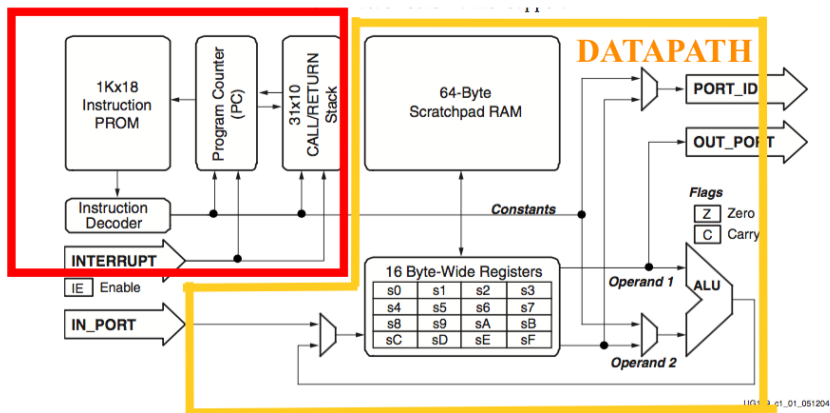
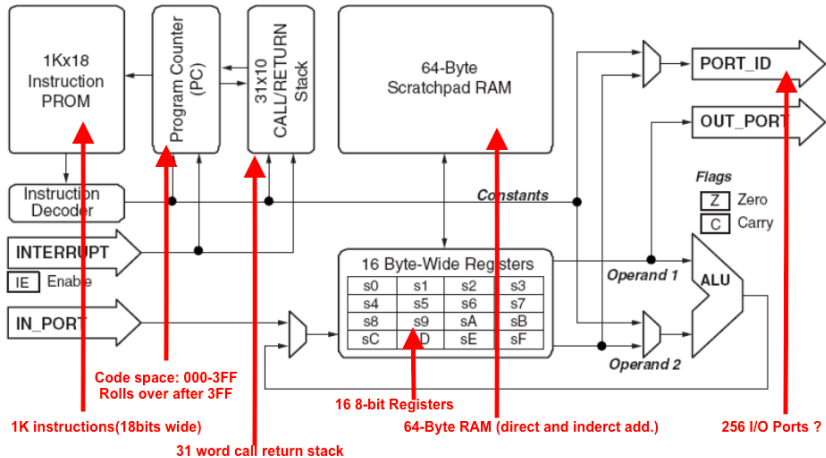


Figure 1-1: PicoBlaze Embedded Microcontroller Block Diagram

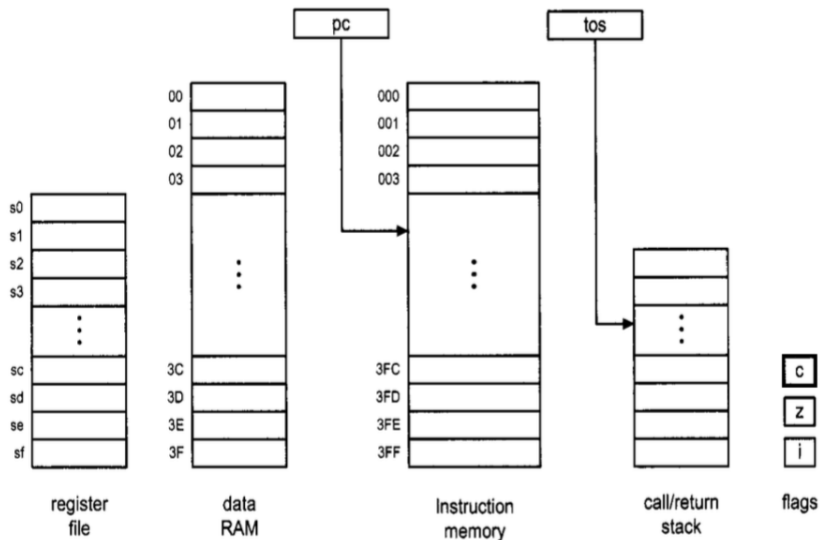
PicoBlaze is a compact 8-bit microcontroller:

- 8-bit data width
- 8-bit ALU with the carry and zero flags
- 16 8-bit general-purpose registers
- 64-byte data memory
- 18-bit instruction width
- 10-bit instruction address, which supports a program up to 1024 instructions
- 31-word call return stack
- 256 input ports and 256 output ports
- 2 clock cycles per instruction
- 5 clock cycles for interrupt handling

PicoBlaze: Basic organization

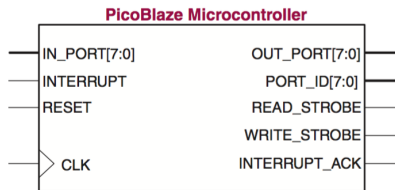


PicoBlaze's Registers



- sX, sY : each representing one of the 16 general-purpose registers, where X and Y take on hexadecimal values from 0 to F.
- pc : program counter.
- tos : top-of-stack pointer of the call/return stack.
- c, z, i : carry, zero, and interrupt flags.
- KK : 8-bit constant value or port id, which is usually expressed as two hexadecimal digits.
- SS : 6-bit constant data memory address, which is usually expressed as two hexadecimal digits.
- AAA : 10-bit constant instruction memory address, which is usually expressed as three hexadecimal digits.

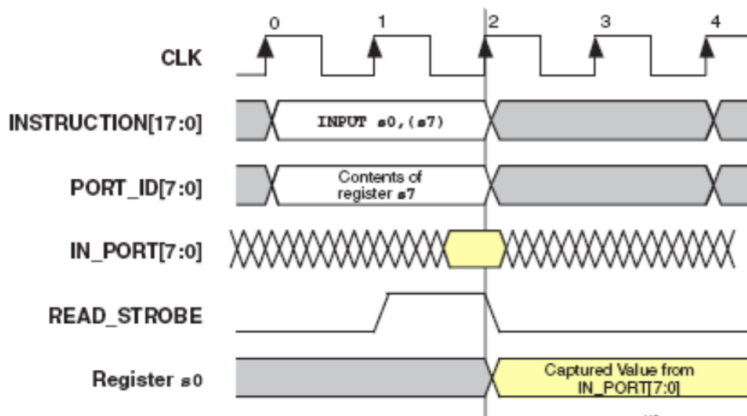
PicoBlaze Interface Connections



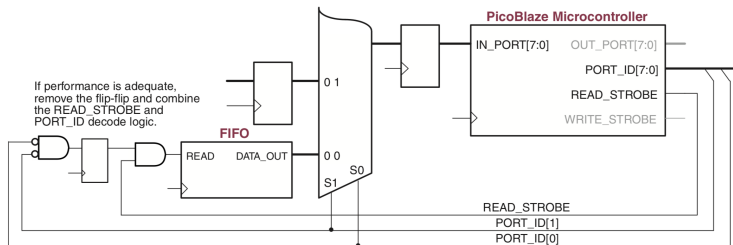
- CLK(input, 1 bit): system clock signal. UG129_c2_01_052004
- RESET (input, 1 bit): reset signal
[PC=0; Flags Cleared; Interrupts disabled; CALL/RETURN Stack reset].
- PORT_ID (output, 8 bits): address of the input or output port.
- IN_PORT (input, 8 bits): input data from I/O peripherals.
- READ_STROBE (output, 1 bit): strobe associated with the Input op.
- OUT_PORT (output, 8 bits): output data to I/O peripherals.
- WRITE_STROBE (output, 1 bit): strobe associated with the Output op.
- INTERRUPT (input, 1 bit): interrupt request from I/O peripherals.
- INTERRUPT_ACK (output, 1 bit): interrupt ackn. to peripherals.

Port Timing during an input instruction (2 clock cycles)!

INPUT sX, (sY) (IN sX, sY)	Read value on input port location pointed to by register sY into register sX	PORT_ID ← sY sX ← IN_PORT	-	-
-------------------------------	--	------------------------------	---	---

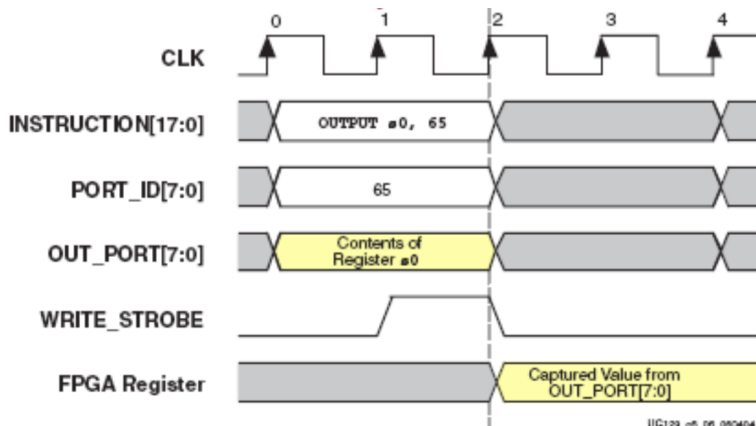


READ_STROBE Indicates a Successful INPUT Operation



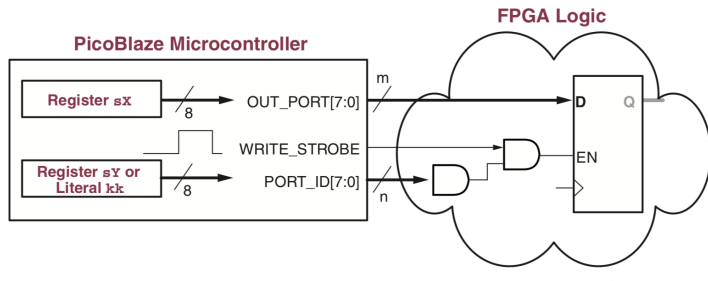
Port Timing during an output instruction (2 clock cycles)!

OUTPUT sX, pp (OUT sX, pp)	Write register sX to output port location pp	PORT_ID ← pp OUT_PORT ← sX	-	-
-------------------------------	--	-------------------------------	---	---



PicoBlaze's I/O: WRITE_STROBE

WRITE_STROBE : data on the OUT_PORT port is valid and ready for capture.



Instruction format

- op sX,sY:register-register format. $sX \leftarrow sX \text{ op } sY$
- op sX,KK:register-constant format. $sX \leftarrow sX \text{ op } KK$
- op sX: single-register format. $sX \leftarrow \text{op } sX$
- op AAA: single-address format (jump and call operations).

- Arithmetic / Logical Instruction

- Logical instructions,

- and sX,sY; // $sX \leftarrow sX \text{ AND } sY$, $c \leftarrow 0$

- xor sX,KK; // $sX \leftarrow sX \text{ XOR } KK$, $c \leftarrow 0$

- Arithmetic instructions

- add sX,sY; // $sX \leftarrow sX + sY$

- addc sX,KK; // $sX \leftarrow sX + KK + c$

- Compare and test instructions

- comp sX,sY; // if ($sX==sY$) $z \leftarrow 1$ else $z \leftarrow 0$;

- if ($sY>sX$) $c \leftarrow 1$ else $c \leftarrow 0$

- test sX,sY; // $t \leftarrow sX \text{ AND } sY$; if $t=0$ $z \leftarrow 1$ else $z \leftarrow 0$;

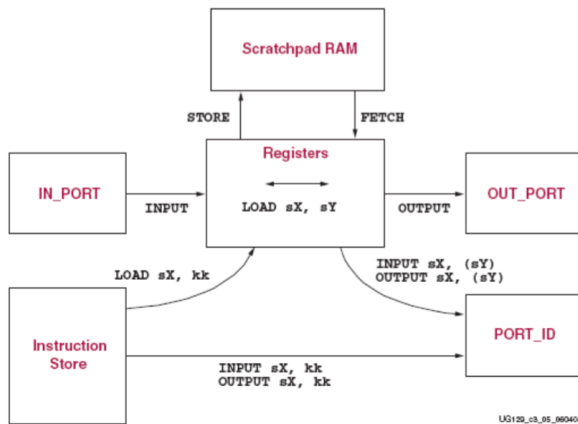
- $c \leftarrow t(7)\text{XOR } t(6) \dots\text{XOR } t(0)$

- Shift and rotate instructions

- sll0 sX; // $sX \leftarrow sX(0\dots6) \& 0$; $c \leftarrow 0$

Instruction Set

- Data-transfer Instructions



- Data-transfer Instructions

```
load sX, KK; // sX ← KK (immediate add.)
```

```
load sX, sY; // sX ← sY (register direct add.)
```

```
fetch sX, (sY); // sX ← RAM(SS) (register indirect)
```

```
fetch sX, SS; // sX ← RAM(sY) (register indirect)
```

```
input sX, KK; // port_id ← KK, sX ← in_port
```

```
output sX, (sY); // port_id ← sY, out_port ← sX
```

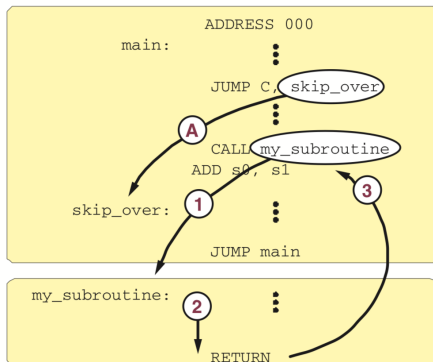

Instruction Set

- Branch instructions: Program flow control instructions

jump does not affect CALL/RETURN Stack and does not affect CARRY and ZERO

call push the current PC to stack and load PC with label then executes until RETURN

ret pop the stack and increment the value then load into PC.



Instruction Set

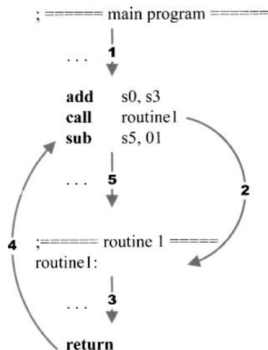
- Branch instructions: Program flow control instructions

`jump AAA; //pc ← AAA (unconditional jump)`

`jumpc AAA; //if (c=1) pc= ← AAA else pc=pc+1; (conditional)`

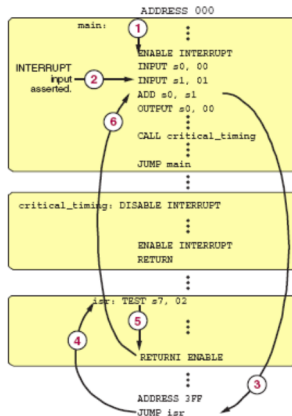
`call AAA; //tos=tos+1; STACK[tos] ← pc; pc ← AAA`

`ret c; //if(c) pc ← 1+STACK[tos]; tos=tos-1; else pc=pc+1`



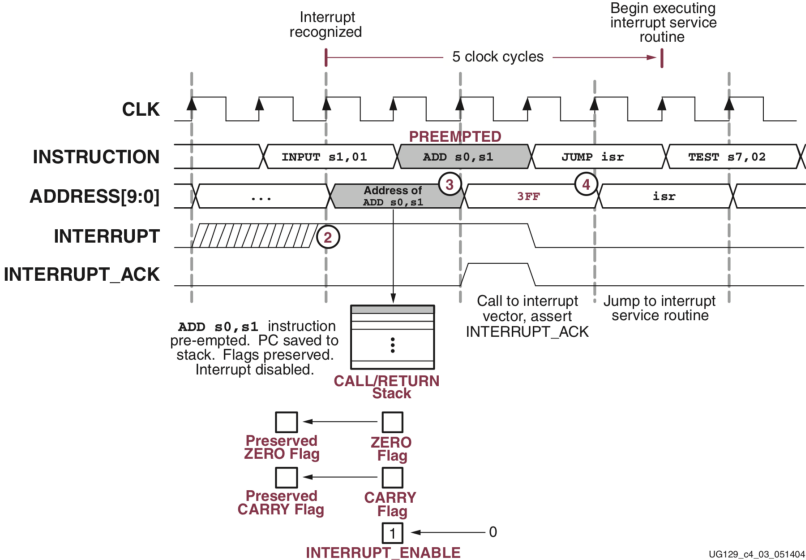
Instruction Set

- Branch instructions: Interrupt related instructions



!!!Interrupt signal must be applied at least 2 clock cycles !!

Instruction Set



UG129_c4_03_051404

- Branch instructions: Interrupt related instructions

```
eint; // enable interrupt
```

```
dint; // disable interrupt
```

```
reti disable; // return the interrupt and disable
```

```
    interrupt, pc  $\leftarrow$  STACK[tos]; tos=tos-1; c,
```

```
    z  $\leftarrow$  preserved c, z
```

Read :Pong P. Chu, "FPGA Prototyping by VHDL Examples: Xilinx Spartan -3 Version", Chapter 14, Wiley 2008

PicoBlaze Instruction Set simulator:

Download (1.1MB): <http://web.itu.edu.tr/yalcinmust/EHB432/PBTool02a.zip>

The screenshot shows the PicoBlaze IDE simulator interface with several components labeled:

- Status flags:** Zero, Carry, Enable.
- Interrupt control for simulation:** Steady, Edge, Timer.
- Data registers:** Registers 0-7 with values.
- Status window:** Assembler Show, Program is Reset.
- Simulation control buttons:** Run, Stop, Step, etc.
- Switches:** \$S00-\$S27.
- LEDs:** \$S01-\$S05.
- mailbox:** \$M00-\$M05.
- Port ID Number:** \$P0.
- Port Value:** \$P0.
- Constant declaration:** \$SWZ, \$DSOFT, \$DSZO.
- Register aliasing:** input_value, LED_output.
- Defined start address:** \$S000.
- Instruction address and code:** \$S001-\$S007.
- Syntax-highlighted assembly code:** IN, CALL, OFF, JMPD.
- Code coverage indicator:** Green arrow pointing to instruction \$S001.
- Next instruction to be executed:** \$S004.
- Breakpoint set at this instruction:** \$S007.
- Scratchpad RAM display:** \$R00-\$R0F.
- Execution time at specified clock frequency:** 95 ns.
- Current Stack Pointer:** \$P1 (\$S01).
- Stack values:** \$S04.
- Cursor row and column position:** \$S001, \$S001.
- Number of instructions already executed up to current code position:** 4.
- Current Program Counter:** \$P0 (\$S006).

```
; 24-bit addition: {x2,x1,x0}+{y2,y1,y0}
PIN00 DSIN $00
POUT00 DSOUT $00
...
PIN05 DSIN $05
POUT05 DSOUT $05

x0 EQU $00
x1 EQU $01
x2 EQU $02
y0 EQU $03
y1 EQU $04
y2 EQU $05
in s0,x0
...
in s5,y2
add s0,s3 ;add least significant bytes
addc s1,s4 ;add middle bytes with carry
addc s2,s5 ;add most sig. bytes with carry
out s2, x0
out s1, x1
out s0, x2
```

```
; bit manipulation
PIN00 DSIN $00
POUT00 DSOUT $00
PIN01 DSIN $01
POUT01 DSOUT $01
PIN02 DSIN $02
POUT02 DSOUT $02

set_mask equ $02
clr_mask equ $fd
tog_mask equ $02

in s0,$00;
or s0,set_mask ;set 2nd LSB to 1
out s0, $00
in s1,$01
and s1,clr_mask ;clear 2nd LSB to 0
out s1,$01
in s2,$02
xor s2,tog_mask ;toggle 2nd LSB
out s2,$02
```

```

if (s0==s1){

/* then-branch statements */

else {

/* else-branch statements */

}

```

```

switch (s0) {

    case value1:

        /* case value1 statements */

        break;
    case value2:

        /* case value2 statements */

        break;
    case value3:

        /* case value3 statements */

        break;
    default:

        /* default statements */
}

```

```

compare s0, s1
jump nz, else-branch
    ;code for then branch
    ...
    jump if-done
else-branch :
    ;code for else branch
    ...
if-done:
    ;code following if statement
    ...

```

```

constant value1 , ...
constant value2, ...

compare s0, value1
jump nz, case-2
;code for case 1
...
jump case-done
case-2 :
compare s0, value2
jump nz, case-3
;code for case 2
...
jump case-done
case-3 :
compare S0, value3
jump default
;code for case 3
jump case-done
default:
;code for default case
...
case-done :

```



```
for (i=MAX;i>0;i--) {  
  ...  
}
```

```
for (i=MAX;i>=0;i--) {  
  ...  
}
```

```
nop:  
  load sX,sX
```

```
  load s0, MAX  
for_loop:  
  ...  
  sub s0,1  
  jump nz, for_loop
```

```
  load s0, MAX  
for_loop:  
  ...  
  sub s0,1  
  jump nc, for_loop
```