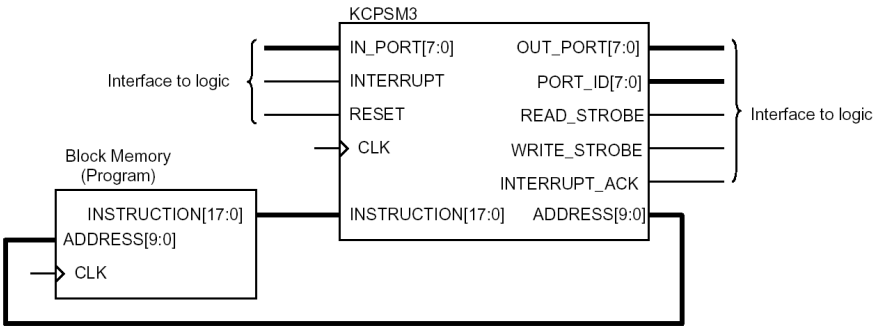
Microcontroladores en FPGAs: Xilinx PicoBlaze

Escuela Politécnica Superior, UAM

Microcontroladores vs. Microprocesadores

- Los microcontroladores (PicoBlaze) son:
  - Pequeños (35-40 CLBs - 96 slices en Spartan3)
  - Se depuran simulando la FPGA
  - Se programan en ensamblador
  - Tienen un rango limitado de aplicaciones
- Los microprocesadores (MicroBlaze) son:
  - Grandes (260 CLBs)
  - No se pueden simular (en un tiempo razonable)
  - Se programan en C
  - Pueden admitir un sistema operativo

Xilinx PicoBlaze

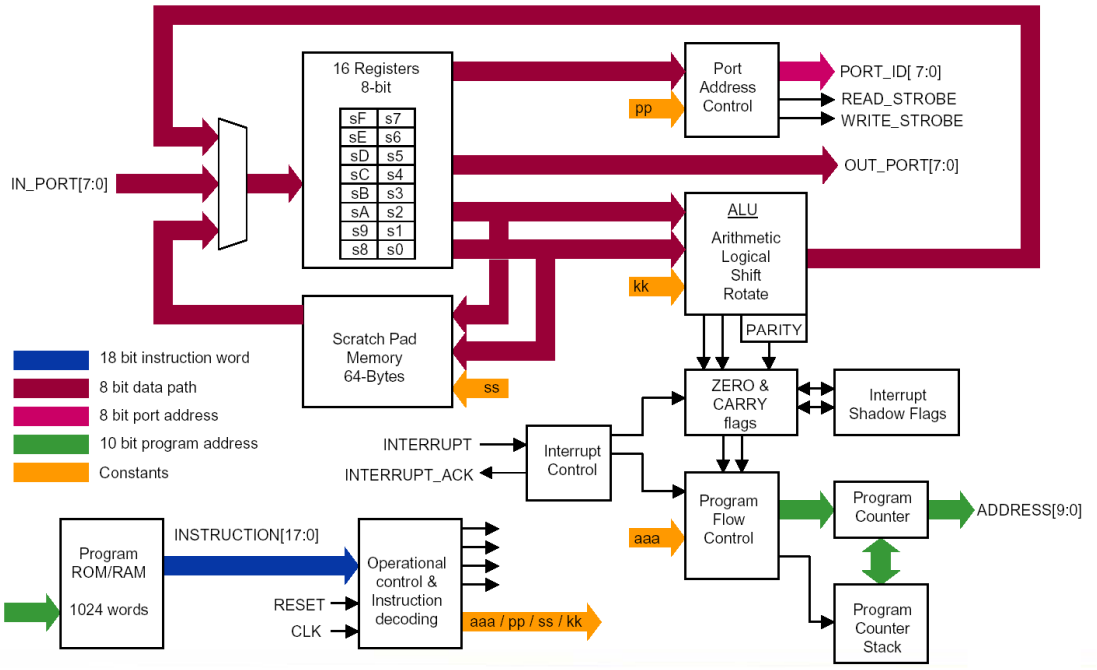


- Es un microcontrolador muy sencillo de 8 bits
- Muy pequeño, 96 slices (menos de un 2,5 % de una XC3S400) y relativamente rápido, de 43 a 66 MIPS
- Memoria de código de 1024 instrucciones en una BRAM

Más sobre PicoBlaze

- Proporcionado por Xilinx como una netlist Edif o código VHDL
  - Se puede usar con todas las FPGAs de Xilinx.
  - Descrito en la XAPP213, por Ken Chapman
  - Diseño muy cuidado para que sea muy pequeño
- 16 registros de propósito general
- 64 bytes scratch pad memory
- 256 puertos de entrada salida
- ALU de 8 bits con:
  - Suma, resta, operaciones lógicas y rotaciones/desplazamientos
  - Flags de carry y zero
- Entrada de interrupción

Arquitectura de PicoBlaze



Juego de instrucciones

'X' and 'Y' refer to the definition of the storage registers 's' in the range 0 to F.
'kk' represents a constant value in the range 00 to FF.
'aaa' represents an address in the range 000 to 3FF.
'pp' represents a port address in the range 00 to FF.
'ss' represents an internal storage address in the range 00 to 3F.

Program Control Group

```
JUMP aaa
JUMP Z,aaa
JUMP NZ,aaa
JUMP C,aaa
JUMP NC,aaa

CALL aaa
CALL Z,aaa
CALL NZ,aaa
CALL C,aaa
CALL NC,aaa

RETURN
RETURN Z
RETURN NZ
RETURN C
RETURN NC
```

Note that call and return supports up to a stack depth of 31.

Arithmetic Group

```
ADD sX,kk
ADDCY sX,kk
SUB sX,kk
SUBCY sX,kk
COMPARE sX,kk

ADD sX,sY
ADDCY sX,sY
SUB sX,sY
SUBCY sX,sY
COMPARE sX,sY
```

Interrupt Group

```
RETURNI ENABLE
RETURNI DISABLE

ENABLE INTERRUPT
DISABLE INTERRUPT
```

Logical Group

```
LOAD sX,kk
AND sX,kk
OR sX,kk
XOR sX,kk
TEST sX,kk

LOAD sX,sY
AND sX,sY
OR sX,sY
XOR sX,sY
TEST sX,sY
```

Storage Group

```
STORE sX,ss
STORE sX,(sY)
FETCH sX,ss
FETCH sX,(sY)
```

Shift and Rotate Group

```
SR0 sX
SR1 sX
SRX sX
SRA sX
RR sX

SL0 sX
SL1 sX
SLX sX
SLA sX
RL sX
```

Input/Output Group

```
INPUT sX,pp
INPUT sX,(sY)
OUTPUT sX,pp
OUTPUT sX,(sY)
```

Como usarlo desde VHDL

```
component kcpsm3 is port (
                address : out std_logic_vector(9 downto 0);
                instruction : in std_logic_vector(17 downto 0);
                port_id : out std_logic_vector(7 downto 0);
                write_strobe : out std_logic;
                out_port : out std_logic_vector(7 downto 0);
                read_strobe : out std_logic;
                in_port : in std_logic_vector(7 downto 0);
                interrupt : in std_logic;
                interrupt_ack : out std_logic;
                reset : in std_logic;
                clk : in std_logic);
end component;

component myProgramRom is port
(
        instruction :           out std_logic_vector(17 downto 0);
        address :               in std_logic_vector(9 downto 0);
        clk :                   in std_logic
);
end component;
```


Ensamblador para PicoBlaze

- Xilinx ofrece un sencillo ensamblador KCPSM3.EXE
- Los registros de llaman s0, s1, s2 ... sE y sF
- Los comentarios empiezan tras el punto y coma ;
- Los números se escriben en hexadecimal
- Definiendo constantes y nombre de registros:

```
            CONSTANT light_port, 03      ;light sensor port
            CONSTANT light_sensor, 01    ;bit0 is light sensor
            CONSTANT temp_sensor, 40     ;temperature sensor port
            NAMEREG sF, light_count_msb ;16-bit light pulse counter
            NAMEREG sE, light_count_lsb
            NAMEREG sD, new_temp         ;current temperature
            CONSTANT peak_temp, 2E       ;peak temperature memory
light_test: INPUT s1, light_port         ;test for light
            TEST s1, light_sensor
            JUMP Z, temp_test            ;jump if no light
            ADD light_count_lsb, 01      ;increment counter
            ADDCY light_count_msb, 00
 temp_test: INPUT new_temp, temp_sensor ;read temperature
            FETCH s2, peak_temp
            COMPARE s2, new_temp         ;compare with peak value
            JUMP NC, light_test          ;new value is smaller
            STORE new_temp, peak_temp    ;write new peak value
            JUMP light_test
```

Ensamblador para PicoBlaze (2)

- Para definir etiquetas:

```
wait_loop1: INPUT s0, event_port        ;read input
            AND s0, 01                  ;test bit0
            JUMP Z, wait_loop1          ;wait for an event
```

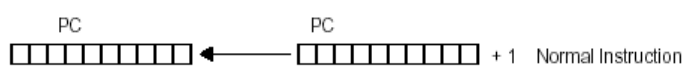
- Se puede especificar también la dirección origen del código (ORG)

```
0B                          ;Subroutines
A0                          ADDRESS A0                ;increment counter
A0  4A01      inc_count16:  ADD sA, 01
A1  5B00                    ADDCY sB, 00
A2  8080                    RETURN
```

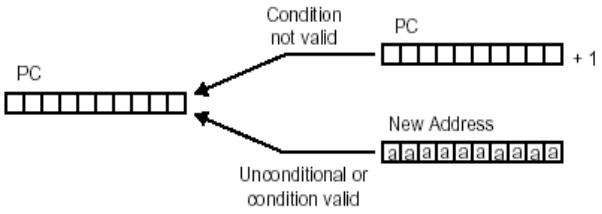

Algunas Instrucciones (1)

JUMP

Under normal conditions, the program counter (PC) increments to point to the next instruction. The address space is fixed to 1024 locations (000 to 3FF hex) and therefore the program counter is 10 bits wide. It is worth noting that the top of memory is 3FF hex and will increment to 000.



The JUMP instruction may be used to modify this sequence by specifying a new address. However, the JUMP instruction may be conditional. A conditional JUMP will only be performed if a test performed on either the ZERO flag or CARRY flag is valid. The JUMP instruction has no effect on the status of the flags.

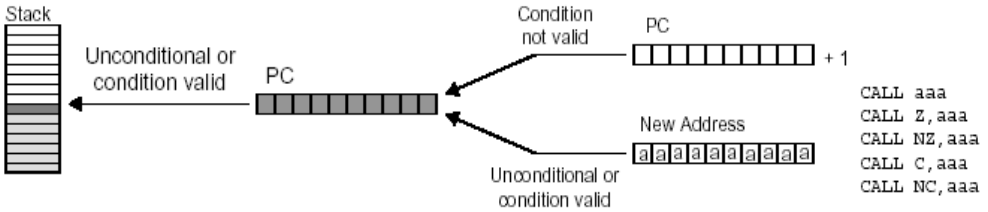



```
JUMP aaa
JUMP Z,aaa
JUMP NZ,aaa
JUMP C,aaa
JUMP NC,aaa
```

Algunas Instrucciones (2)

CALL

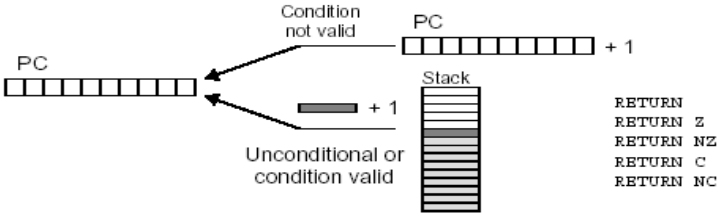
The CALL instruction is similar in operation to the JUMP instruction in that it will modify the normal program execution sequence by specifying a new address. The CALL instruction may also be conditional. In addition to supplying a new address, the CALL instruction also causes the current program counter (PC) value to be pushed onto the program counter stack. The CALL instruction has no effect on the status of the flags.



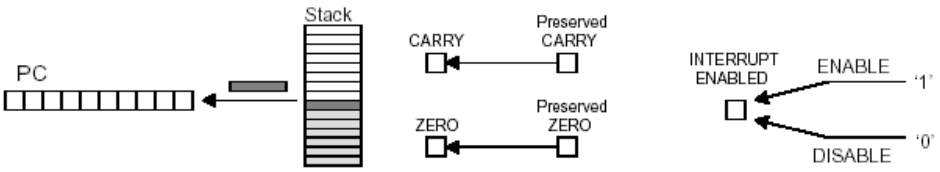
The program counter stack supports a depth of 31 address values. This enables nested 'CALL' sequences to a depth of 31 levels to be performed. However, the stack will also be used during an interrupt operation and hence at least one of these levels should be reserved when interrupts are enabled. The stack is implemented as a separate cyclic buffer. When the stack becomes full, it simply overwrites the oldest value. Hence it is not necessary to reset the stack pointer when performing a software reset. This also explains why there are no instructions to control the stack and why no other memory needs to be reserved or provided for the stack.

Algunas Instrucciones (3)
RETURN y RETURNI

The RETURN instruction is the complement to the CALL instruction. The RETURN instruction may also be conditional. In this case the new program counter (PC) value will be formed internally by incrementing the last value on the program address stack. This ensures that the program will execute the instruction following the CALL instruction which resulted in the subroutine. The RETURN instruction has no effect on the status of the flags.



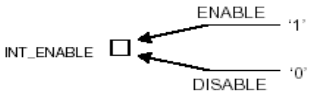
The RETURNI instruction is a special variation of the RETURN instruction which should be used to conclude an interrupt service routine. The RETURNI is unconditional and therefore will always load the program counter (PC) with the last address on the program counter stack (the address is not incremented in this case since the instruction at the address stored will need to be executed). The RETURNI instruction restores the flags to the condition they were in at the point of interrupt. The RETURNI also determines the future ability of interrupts using ENABLE and DISABLE as an operand.



Algunas Instrucciones (4)

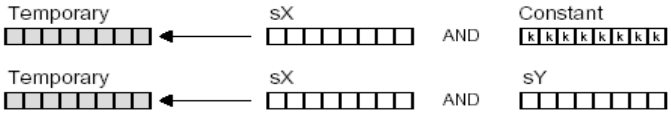
ENABLE/DISABLE INTERRUPT

These instructions are used to set and reset the INT_ENABLE flag. Before using ENABLE INTERRUPT a suitable interrupt routine must be associated with the interrupt address vector (located at address 3FF). Interrupts should never be enabled whilst performing an interrupt service routine.



TEST

The TEST instruction performs a bit-wise logical 'AND' operation between two operands. Unlike the 'AND' instruction, the result of the operation is discarded and only the flags are affected. The ZERO flag is set if all bits of the temporary result are low. The CARRY flag is used to indicate the **ODD PARITY** of the temporary result. Parity checks typically involve a test of all bits, i.e. if the contents of 's5' = 3D (00111101), the execution of TEST s5,FF will set the CARRY flag indicating ODD parity. Bit testing is typically used to isolate a single bit. For example TEST s5,04 will test bit2 of the 's5' register which would set the CARRY flag if the bit is high (reset if the bit is low) and set the ZERO flag if the bit is low (reset if the bit is high).



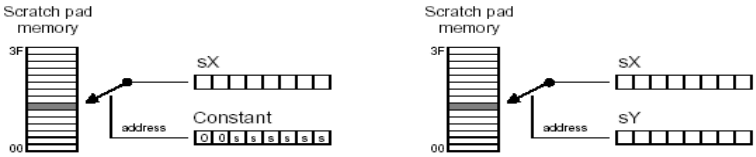
CARRY ? Set if there are an odd number of bits set to '1' in the temporary result.

ZERO ? Set if all bits of temporary result are zero. Reset in all other cases.

Algunas Instrucciones (5)

STORE

The STORE instruction enables the contents of any register to be transferred to the 64-byte internal scratch pad memory. The storage address (in the range 00 to 3F) can be defined by a constant value or indirectly as the contents of any other register. The Flags are not affected by this operation.



Each STORE instruction must specify the source register as 's' followed by a hexadecimal digit. It must then specify the storage address using a register value in a similar way or specify a 6-bit constant storage address using 2 hexadecimal digits. The assembler supports register naming and constant labels to simplify the process. Although the assembler will reject constants greater than 3F, it is the responsibility of the programmer to ensure that the value of 'sY' is within the address range.

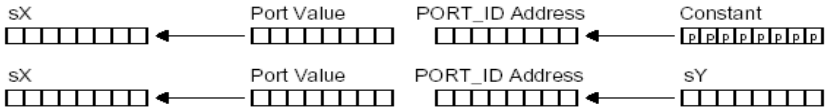
FETCH

The FETCH instruction enables data held in the 64-byte internal scratch pad memory to be transferred any of the internal registers. The storage address (in the range 00 to 3F) can be defined by a constant value or indirectly as the contents of any other register. The Flags are not affected by this operation.

Algunas Instrucciones (5)

INPUT

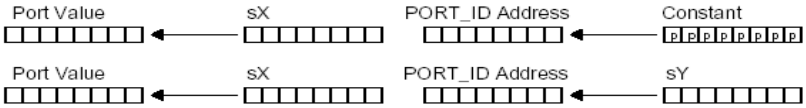
The INPUT instruction enables data values external to KCPSM3 to be transferred into any one of the internal registers. The port address (in the range 00 to FF) can be defined by a constant value or indirectly as the contents of any other register. The Flags are not affected by this operation.



The user interface logic is required to decode the PORT_ID port address value and supply the correct data to the IN_PORT. The READ_STROBE is set during an input operation (see 'READ and WRITE STROBES'), but it is not always necessary for the interface logic to decode this strobe. However, it can be useful for determining when data has been read, such as when reading a FIFO buffer (see 'Design of Input Ports').

OUTPUT

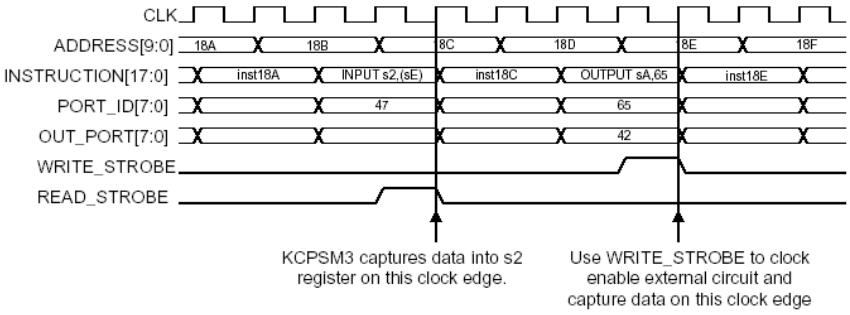
The OUTPUT instruction enables the contents of any register to be transferred to logic external to KCPSM3. The port address (in the range 00 to FF) can be defined by a constant value or indirectly as the contents of any other register. The Flags are not affected by this operation.



The user interface logic is required to decode the PORT_ID port address value and capture the data provided on the OUT_PORT. The WRITE_STROBE is set during an output operation (see 'READ and WRITE STROBES'), and should be used to clock enable the capture register or write enable a RAM (see 'Design of Output Ports').

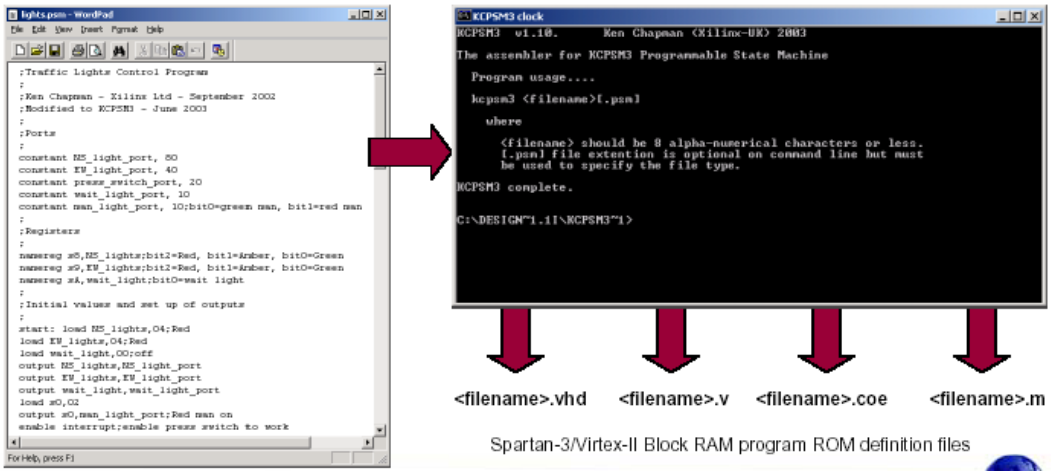
READ and WRITE STROBES

These pulses are used by external circuits to confirm input and output operations. In the waveforms below, it is assumed that the content of register sE is 47, and the content of register sA is 42.

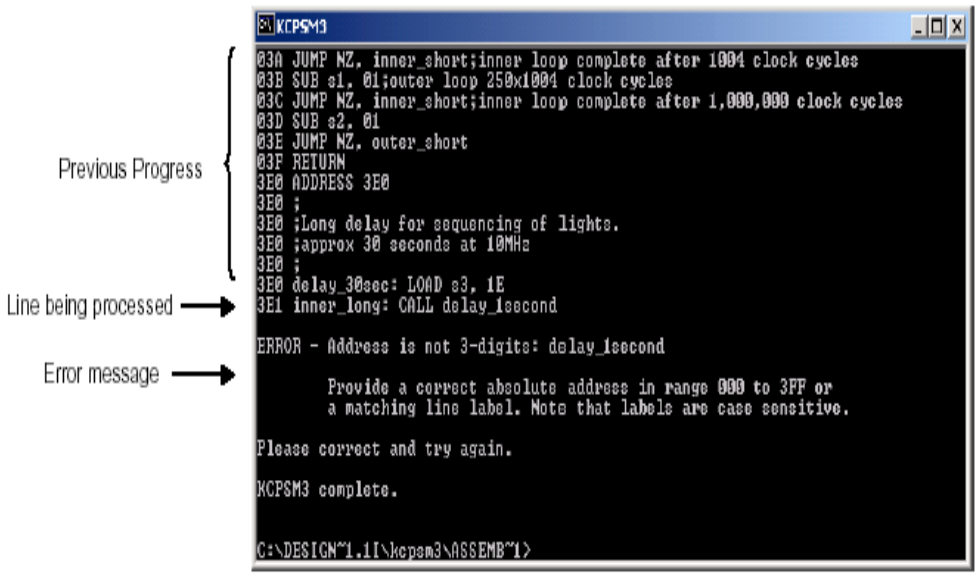


PORT_ID[7:0] is valid for 2 clock cycles providing additional time for external decoding logic and enabling the connection of synchronous RAM. The WRITE_STROBE is provided on the second clock cycle to confirm an active write by KCPSM3. In most cases, the READ_STROBE will not be utilised by the external decoding logic, but again occurs in the second cycle and indicates the actual clock edge on which data is read into the specified register.

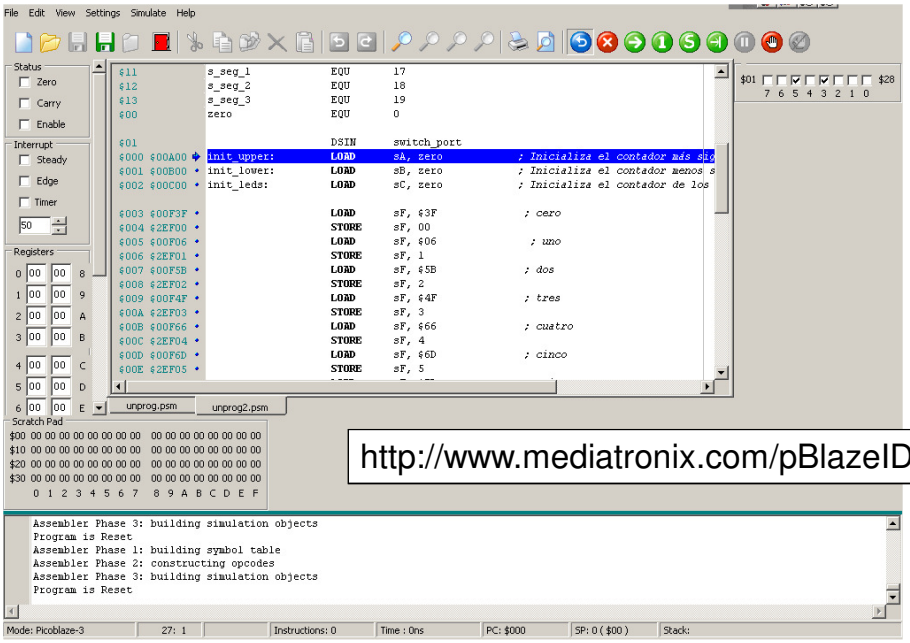
Utilizando KCPSM3.EXE



Utilizando KCPSM3.EXE: Errores



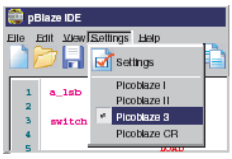
Un debugger: pBlazeIDE

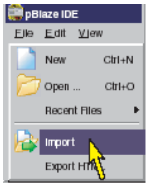


http://www.mediatronix.com/pBlazeIDE.htm

Debugger pBlazeIDE

- Se debe configurar el tipo de picoblaze que se utilizará
- Lamentablemente la sintaxis no es exactamente igual entre Pico Blaze y pBlaze IDE.





KCMPSM Source Code

```
                CONSTANT myconstant, A5

                NAMEREG s0, count16_lsb
                NAMEREG s1, count16_msb

                ADDRESS 000

            main:
; initialize 16-bit counter, enable interrupts
                LOAD count16_lsb, myconstant
                ENABLE INTERRUPT

            loop:
; continuously increment 16-bit counter
                CALL increment_count
                JUMP loop
        end_main:

increment_count:
; add 1 to LSB of 16-bit counter
                ADD count16_lsb, 01
; only add one to MSB if carry generated by LSB
                ADDCY count16_msb, 00
                RETURN

            isr:
; decrement 16-bit counter by one on interrupt
; subtract 1 from LSB of 16-bit counter
                SUB count16_lsb, 01
; only subtract one from MSB if borrow
;   generated by LSB
                SUBCY count16_msb, 00
                RETURNI ENABLE

; interrupt vector is always in last memory location
                ADDRESS 3FF
; jump to interrupt service routing (ISR)
                JUMP isr
```

Code Imported/Converted Into pBlaze IDE

```
myconstant      EQU     $A5

count16_lsb     EQU     s0
count16_msb     EQU     s1

                ORG     0

main:
; initialize 16-bit counter, enable interrupts
                LOAD    count16_lsb, myconstant
                EINT

loop:
; continuously increment 16-bit counter
                CALL    increment_count
                JUMP    loop
end_main:

increment_count:
; add 1 to LSB of 16-bit counter
                ADD     count16_lsb, 1
; only add one to MSB if carry generated by LSB
                ADDC    count16_msb, 0
                RET

isr:
; decrement 16-bit counter by one on interrupt
; subtract 1 from LSB of 16-bit counter
                SUB     count16_lsb, 1
; only subtract one from MSB if borrow
; generated by LSB
                SUBC    count16_msb, 0
                RETI    ENABLE

; interrupt vector is always in last memory location
                ORG     $3FF
; jump to interrupt service routing (ISR)
                JUMP    isr
```