PicoBlaze (2)

Diseño de Sistemas con FPGA

Patricia Borensztejn

Información

http://www.xilinx.com/ipcenter/processor_central/picoblaze/picoblaze_user_resources.htm

PicoBlaze

clk (I)
Reset (I)
address (O): 10 bits
instruction (I): 18 bits
port_id (O): dirección de E/S : 8 bits
in_port (I): data de E/S : 8 bits
read_strobe (O)
out_port (O): dato de 8bits
write_strobe(O)
interrupt(I): solicitud
interrupt_ack (O): ack

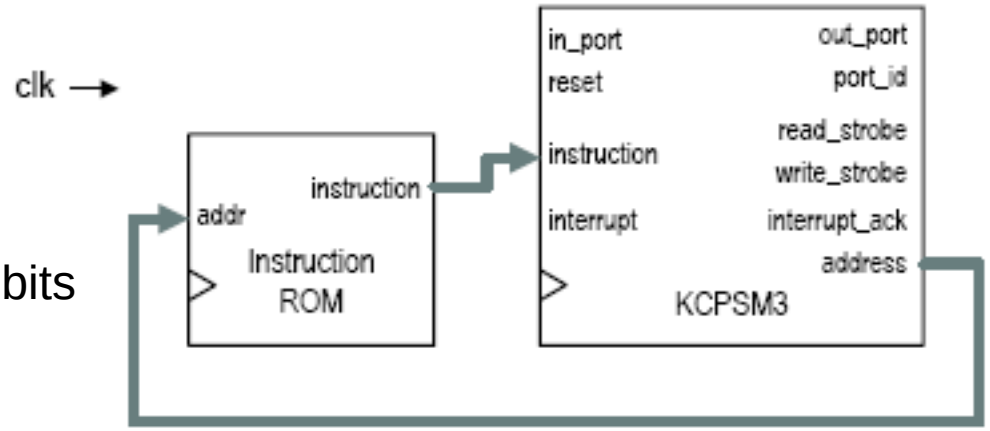


Figure 14.3 Top-level diagram of PicoBlaze.

Constant (K) Coded (C) Programmable State Machine

Ken Chapmann

PicoBlaze: ISA

- 57 instrucciones, cinco formatos.
- Tipos de instrucciones:
  - Lógicas
  - Aritméticas
  - Comparación y Test
  - Shift y Rotate
  - Movimiento de Datos
  - Control de secuencia
  - Interrupciones
- Registros: 16 de 8 bits
- RAM de 64 bytes
- Flags: zero, carry, interrupt
- PC
- TOS (Top of Stack pointer)

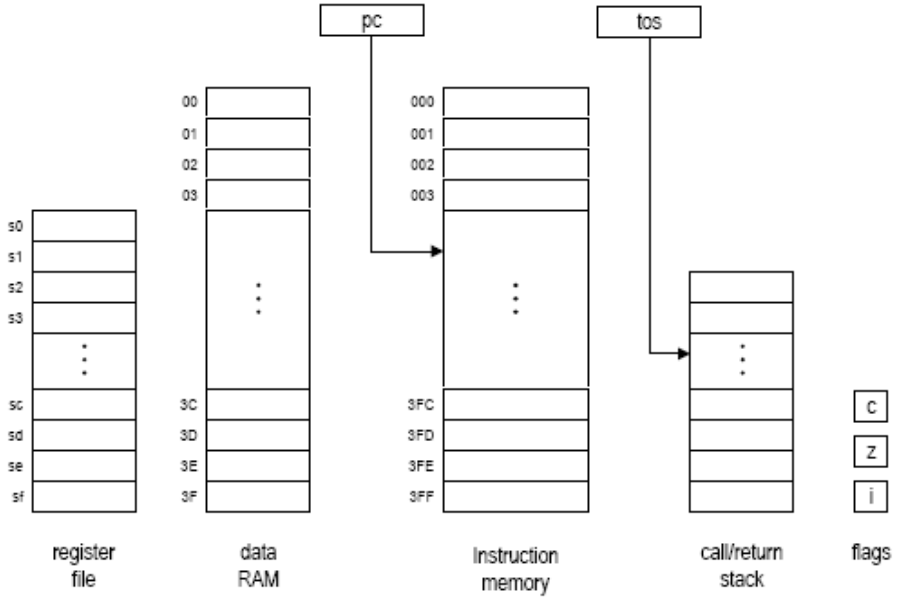


Figure 14.5 PicoBlaze programming model.

Formatos de Instrucción

- registro-registro
  op sx,sy ; sx← sx op sy
  Ejemplo: add s1,s3
- registro-constante (kk es una constante de 8 bits)
  - op sx,kk ; sx← sx op kk

  Ejemplo: cmp s1, 0F
- registro solo (shift o rotate)
  op sx ; sx← op sx
  Ejemplo: sl1 s3 (shift izquierda 1 posición entrando un 1 en LSB)
- una dirección (jump y call)
  - op AAA ; si (condición) PC←AAA (AAA es una dirección de 10 bits)

  Ejemplo: call AAA
- cero operando (misceláneas)
  - op

  Ejemplo: ret

Lógicas

- Todas las operaciones lógicas ponen el flag de acarreo (C) a cero.
- EL flag Z refleja el resultado de la operación
- Hay 3 operaciones lógicas: AND, OR y XOR en sus modos reg-reg y reg-cte

Aritméticas

- Los flag Z y C reflejan el resultado de la operación
- Hay 8 operaciones: ADD, ADDC, SUB, SUBC en sus modos reg-reg y reg-cte

Comparación y Test

- comp: comparan dos registros o bien reg-cte y setean los flags Z y C
- test: Realiza un AND lógico.
  - t ← sX and sY
  - si t=0 entonces z ← 1 sino z ← 0
  - c ← t(7) xor t(6) ….. xor t(0)
  - (lo mismo cuando sY es una constante)

Shift y Rotaciones

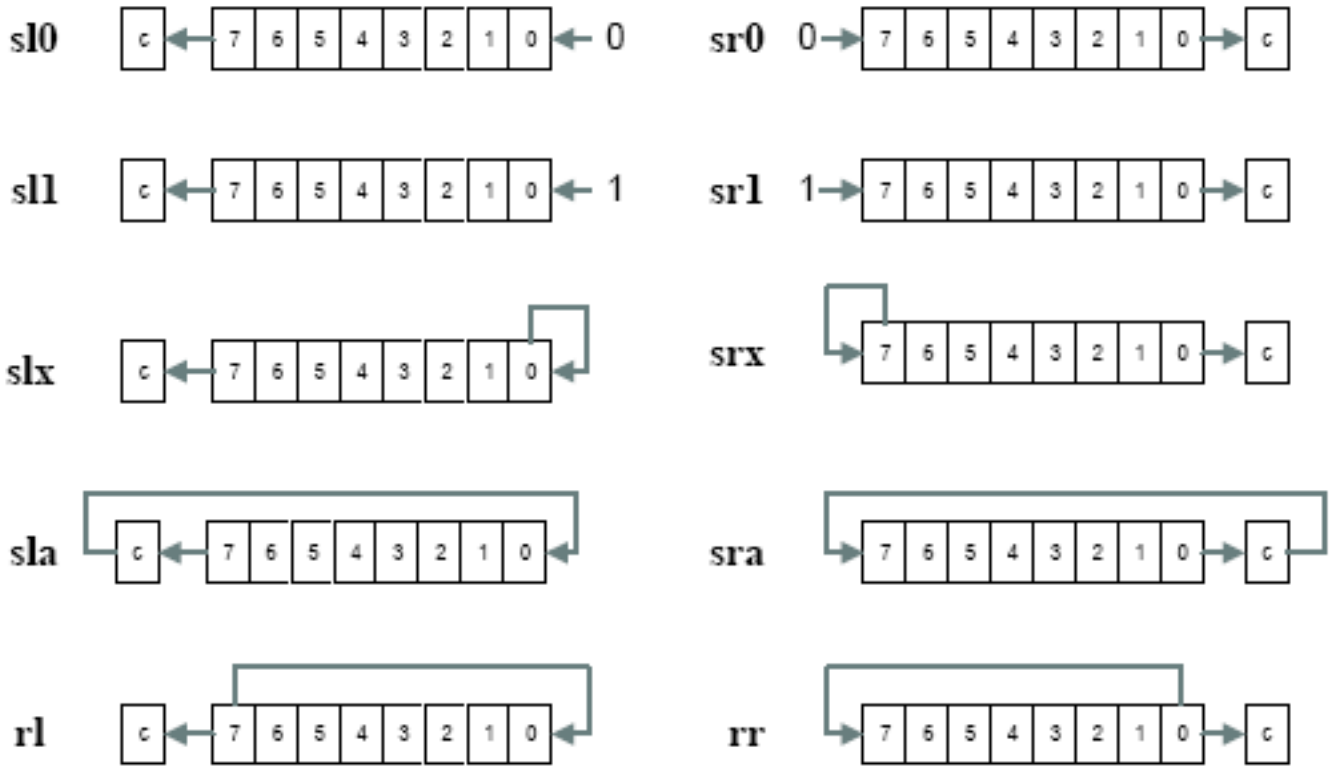


Figure 14.6 Illustration of shift and rotate instructions.

Movimiento de Datos

- PicoBlaze es un RISC ➔ todas las operaciones de la ALU son entre registros (o reg-cte).
- Hay instrucciones específicas para mover datos entre registros, entre registro y memoria y entre registro y puertos de E/S.
  - Entre registros: load

  Ejemplo: load sX,sY
  load sX, KK
  - Entre registro y RAM de datos: fetch y store

  Ejemplo: fetch sX, (sY)
  fetch sX, SS
  store sX, (sY)
  store sX, SS
  - Entre registro y puerto de E/S: input y output

  Ejemplo: input sX, (sY); port_id←sY; sX←port_id
  input sX, kk;
  output sX, (sY)
  output sX,kk

Flujo de Programa

- Saltos condicionales (C y Z) e incondicionales

  Ejemplo:
  ```
  jump AAA
  jump c, AAA
  jump nc, AAA
  jump z, AAA
  jump nz, AAA
  ```

- Llamadas a subrutina

  Ejemplo:
  ```
  call AAA
  call c, AAA
  call z, AAA
  call nz, AAA
  ```

- Retornos de subrutina

  Ejemplo:
  ```
  return
  return c
  ```

Interrupciones

- Las interrupciones son iniciadas por pedidos externos de servicio. Cuando están habilitadas y hay una petición, el procesador completa la ejecución de la instrucción en curso, salva el PC de la siguiente instrucción, preserva el C y Z, desabilita interrupciones y carga el PC con la dirección 3FF, que es la dirección inicial de la rutina de atención.
- Instrucciones para retornar de las interrupciones.
  - returni disable : vuelve y deja desabilitadas las interrupciones
  - returni enable : vuelve y deja habilitadas las interrupciones
  - Enable interrupt
  - Disable interrupt

Ensamblador KCPSM3

- Directivas
  - address
    - Indica que el código que sigue a continuación debe estar puesto en la dirección especificada de la memoria ROM

      Ejemplo: **address** 3FF
  - namereg
    - Da un nombre simbólico a un reg

      Ejemplo: **namereg** s5, index
  - constant
    - Le da un nombre simbólico a una constante

      Ejemplo: **constant** max, F0
- Convenciones
  - Se usa “:” después de una dirección simbólica
  - Se usa “;” antes de un comentario
  - Se usa HH para el valor de una constante, donde H es un dígito Hexadecimal.

Ensamblador KCPSM3

- “Uno jamás se salva del lenguaje ensamblador…”

```
; this is a demo segment
            test        s0, 80              ; test del MSB de s0
            jump        z, clr_s1           ; si MSB no está activado, ir a clr_s1
            load        s1, FF              ; sino, cargar s1 con todos unos
            jump        despues
clr_s1:
            load        s1, 01              ; cargar s1 con 0000 0001
despues:
```

Ensamblador KCPSM3

```
; trabajamos con el bit 1 del byte

constant        SET_MASK, 02        ; mask=0000 0010
constant        CLR_MASK, FD        ; mask=1111 1101
constant        TOG_MASK, 02        ; mask=0000 0010

or      s0, SET_MASK                ; set 2nd LSB de s0
and     s0, CLR_MASK                ; clr 2nd LSB de s0
xor     s0, TOG:MASK                ; toggle 2nd LSB de s0
```

- Las operaciones de bits son útiles para controlar actividades de E/S, o bien flags como Z.
- También podemos trabajar con grupos de bits, por ejemplo con un nibble.

```
; trabajamos con el nibble inferior del byte

constant        SET_MASK, 0F        ; mask=0000 1111
constant        CLR_MASK, F0        ; mask=1111 0000
constant        TOG_MASK, 02        ; mask=0000 1111

or      s0, SET_MASK                ; set nibble bajo de s0
and     s0, CLR_MASK                ; clr nibble bajo  de s0
xor     s0, TOG:MASK                ; toggle nibble bajo de s0
```

Ensamblador

- Un problema común a los microcontroladores de 8 bits, es que si queremos trabajar con datos de 16 bits (o más) tenemos que propagar información entre dos instrucciones.
- Vamos a hacer un programa que opere con operandos de 24 bits

```
namereg    s0, x0
namereg    s1, x1
namereg    s2, x2
namereg    s3, y0
namereg    s4, y1
namereg    s5, y2

; add : {x2, x1, x0} + {y2,y1,y0}
add        x0,y0        ; suma bytes menos significativos.
addcy      x1,y1        ; suma con carry de los del medio
addcy      x2,y2        ; suma con carry de los del medio


; shift {x2,x1,x0} via carry
sl0        x0           ; 0 al LSB de x0, MSB de x0 va al carry
sla        x1           ; carry al LSB de x1, MSB de x1 al carry
sla        x2           ; carry al LSB de x2, MSB de x2 al carry
```

Estructuras de control

```
If (s0==s1) {
        /* then statements  */
}
else    {
        /* else statements  */
}
```

```
        compare     s0, s1
        jump        nz, else_branch
        ; código para then branch
        …
        jump        if_done
else_branch:
        ; código para else branch
        …
if_done:
```

```
for (i=MAX, i=0, i-1) {
        /* loop body statements */
}
```

```
namereg     s0, i
constant    MAX, …

        load        i, MAX
loop_body:
        ; código para loop body
        …
        sub         i, 01
        jump        nz, loop_body
        ; código que sigue al loop
        ….
```

Estructuras de control

```
switch (s0) {
        case valor1:
                /* case valor1 statements */
                break;
        case valor2:
                /* case valor2 statements */
                break;
        case valor3:
                /* case valor3 statements */
                break;
        default:
                /* default statements */
        }
```

```
                constant    valor1, …
                constant    valor2, …
                constant    valor3, …

                compare     s0, valor1
                jump        nz, caso_2
                ; código para caso_1
                …
                jump        case_done
caso_2:
                compare     s0, valor2
                jump        nz, caso_3
                ; código para caso_2
                …
                jump        case_done
caso_3:
                compare     s0, valor3
                jump        nz, default
                ; código para caso_3
                …
                jump        case_done
default:
                ; código para default
                …
case_done:
```

Subrutinas

```
; rutina mult_soft
; función: multiplicación unsigned de 8 bits usando el algoritmo de “shift and add”
; input registers:
;               s3: multiplicando
;               s4: multiplicador
;output registers:
;               s5: parte alta del producto
;               s6: parte baja del producto
;registro temporal
;               s0: I

namereg     s0, I

mult_soft:
            load        s5, 00              ;clear s5
            load        I, 08               ; inicializa loop index
mult_loop:
            sr0         s4                  ; shift LSB al carry
            jump        nc, shift_prod      ; LSB is 0
            add         s5, s3              ; LSB is 1
shift_prod:
            sra         s5                  ; shift upper byte right
                                            ; carry to MSB, LSB to carry
            sra         s6                  ; shift lower byte right
                                            ; LSB of s5 to MSB of s6
            sub         i, 01               ; dec loop index
            jump        nz, mult_loop       ; rep until i=0
            return
```

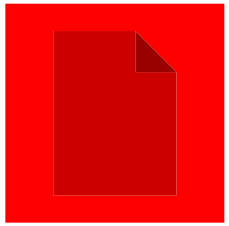
Programa Ejemplo

- Programa que lee dos entradas, a y b, de los switches de entrada (a y b son de 4 bits), calcula a^2+b^2 y muestra el resultado en los LEDS (8).

```
                call        clear_data_ram
forever:
                call        read_switch
                call        square
                call        write_led
                jump        forever
```

programa.psm

- Miramos el código



~patricia\Pico\programa.psm

(aquí tengo el código y el ensamblador)

Ensamblando

- Bajar el ensamblador llamado KCSPM3 de la página de xilinx.
- Ensamblar invocando desde una pantalla de DOS.
  - kcpsm3 programa.psm

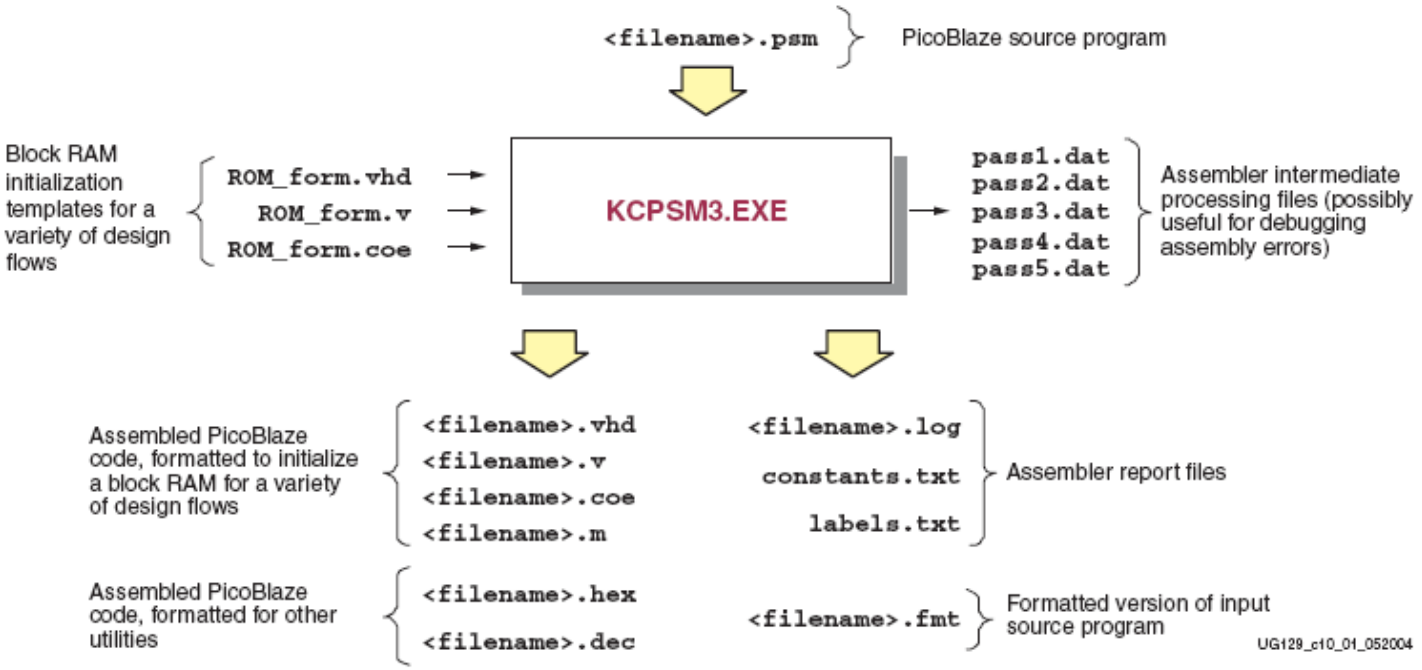


Figure 10-1: **KCPSM3 Assembler Files**

Ensamblando

- Armé un directorio en: C:~PATRICIA\PICO
- Copié :
  - KCPSM3.exe
  - ROM_form.vhd
  - ROM_form.v
  - ROM_form.coe
  - y programa.psm
- Ejecuté bajo DOS
  - kcpsm3 programa.psm
  - Y obtuvimos, entre otras cosas:

PROGRAMA.V

Simulación

- Se simula con PBlazeIDE. Bajarlo de

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

- El picoblaze y el PBlazeIDE los tengo en:

  Patricia\materia FPGA\xilinx\KCPSM3

Síntesis con PicoBlaze

- Cuando ya tenemos simulado el programa de picoblaze y tenemos también el archivo HDL para la ROM (programa.v) hay que combinar el programa con el procesador sintetizando todo para la FPGA.
- Como el procesador PicoBlaze no tiene interface con E/S, hay que construirla.
- En nuestra aplicación, necesitamos definir los switches como entrada y los leds como salida.

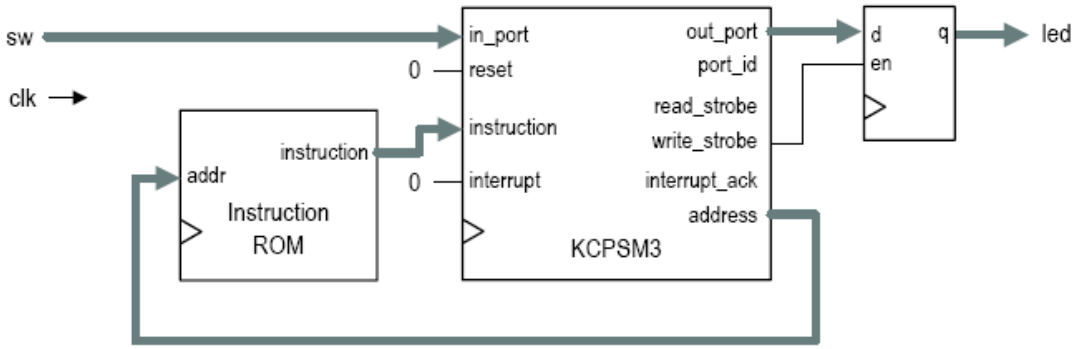
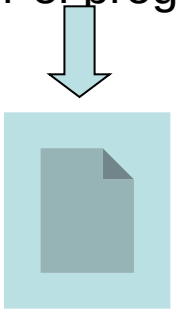


Figure 15.3 PicoBlaze with a simple I/O interface.

Abrir el programa



Y ahora…

- Lanzar el IDE…. Compilar, y si esta todo bien… generar el archivo de configuración y cargarlo en la FPGA…

- Y… probarlo…
- A=3 B=4 Leds= 9+16=25= 011001
- A=7 B=2 Leds= 49+4=53=110101