Implementación del Contador de Programa y Registro de Instrucciones de AVR en VHDL

1st Estévez Acosta Alfredo Antonio
Universidad Tecnológica de la Mixteca
MEOSIA
Oaxaca, México
alfredo.estevez.acosta@gmail.com

2nd Castillo Aguilar Imer Francisco
Universidad Tecnológica de la Mixteca
MEOSIA
Oaxaca, México
imercastillo26@gmail.com

3rd Hernández Gómez Hiram Jesús
Universidad Tecnológica de la Mixteca
MEOSIA
Oaxaca, México
hiram1hego25@gmail.com

***Resumen*—En el presente trabajo se ha realizado el modelado, implementación y simulación del contador de programa y el registro de instrucciones propios de la arquitectura del microprocesador AVR. Se hace una selección de todo el set de instrucciones ordenados por código de operación y clasificados en tres grupos (Branch,Load Store y Register-Register). Se diseñan la unidad de control y el camino de datos para el PC e IR, así como una memoria de programa para simular el comportamiento de éstos módulos ante algunas instrucciones pre cargadas.**

***Index Terms*—Contador de Programa, Registro de Instrucciones, Set de Instrucciones AVR.**

I. Introducción

El *Contador de Programa (PC)* es un registro en el que se almacena la dirección de la última instrucción leída, de ésta manera el procesador sabe cuál es la siguiente instrucción que debe ejecutar.[1] El PC en el caso del microcontrolador AVR tiene un tamaño de 16 bits.

Las instrucciones que pueden modificar al contador de programa directamente son:[2]

I-1. Instrucciones Branch:

- **CPSE Rd,Rr**
  Ésta instrucción salta a la siguiente instrucción si Rd y Rr son iguales, de otra forma la siguiente instrucción es ejecutada. Las banderas en el registro de estados (SREG) no son modificadas.
  Existen otras funciones similares como: (BRBS s,k / BRBC s,k / SBRS Rr,b / SBRC Rr,b / SBIS A,b / SBIC A,b) lo que tienen en común es que primero se ejecuta una operación en la ALU antes de decidir que incremento en el PC realizar.
- **RJMP k**
  Ésta instrucción incondicionalmente salta a PC+k+1.
- **RCALL k**
  Es similar a RJMP pero pone PC+1 en la pila y decrementa el apuntador a pila.
- **ICALL**
  Salta a la dirección del registro Z (R31:R30),pone PC+1 en la pila y decrementa el apuntador a la pila.La única diferencia con RCAL es que el nuevo valor de PC viene del registro Z y no del registro de instrucciones.
- **IJMP**
  Salta a la dirección apuntada por el registro Z.
- **RET**
  Regresa de una subrutina, lee 2 bytes de la pila dentro del program counter y continúa desde ahí.La única diferencia con *RETI* es que activa la bandera de interrupción global en SREG.

II. Diseño del camino de datos y unidad de control

Del set de instrucciones presentado anteriormente,vemos que en general se realizan las siguientes operaciones en el PC.

- Incrementar por 1 (Default)
- Incrementar por 2 (Para operaciones de salto como CPSE, SBRS, SBIS)
- Incrementar por k+1 (para RJMP y RCALL)
- Establecer un nuevo valor de 16 bits (Z para ICALL, contenido de la pila para RET y RETI)

A continuación un primer diagrama a bloques simplificado del programa que llamaremos «CtrlPC» (módulo de control del PC).

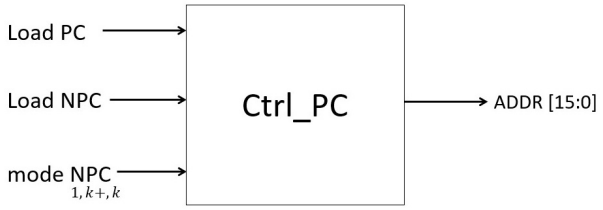

Figura 1: Primer diagrama a bloques del módulo de control del PC

«CtrlPC» maneja tres registros de 16 bits que son: *PC*,*NPC* (Next PC) y *KPC* (Una constante).

Se pude ver que el módulo propuesto tiene tres entradas de control:
loadPC para copiar NCP dentro del PC, **loadNPC** para copiar el resultado de la suma o del valor en KPC, y **modeNPC** para controlar el sumador: mode "1": NPC :=

NPC + 1, mode “k+”: NPC := NPC + k, mode “k”: NPC := KPC.

II-1. Realizando las 4 operaciones básicas del PC : Para realizar las operaciones listadas anteriormente se propone realizar las siguientes acciones:

- *Incrementar por 1:* Se consigue estableciendo NCP=1.
- *Incrementar por 2:*Es lo mismo que incrementar por 1 pero se realiza 2 veces en las primeras transiciones de la máquina de estados.
- *Incrementar por k+1:*Se consigue haciendo k incrementos en la fase de ejecución. NPC se colocará igual a k+.
- *Establecer un nuevo valor de 16 bits:*Se coloca el valor de 16 bits en KPC y el modo NPC=k.

II-2. Considerando al registro de instrucciones IR : Sólo se tiene que hacer una modificación al diagrama propuesto anteriormente, esto con el fin de que en el estado en el que se ejecuta IR pueda estar disponible la palabra de bits de PC+1 en la salida de la memoria de programa. Con ésta modificación ya no se requeriría a NPC. El registro de instrucción IR con la entrada de control **loadIR**

El diagrama a bloques quedaría como se muestra a continuación:

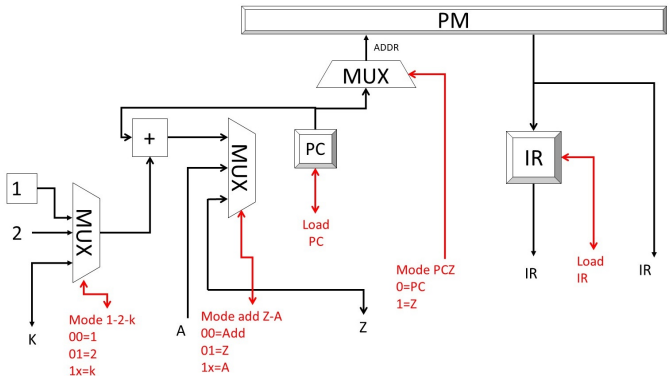

Figura 2: Diagrama a bloques del módulo de control del PC

La dirección del programa viene del registro Z o del registro del PC dependiendo de la señal de control *modePCZ*.

El registro del PC se carga sólo si loadPC está activado y obtiene su valor ya sea del registro Z, del sumador o de la entrada A (dirección), dependiendo del valor de la entrada de control **modeAddZA**.

El sumador agrega uno de los 3 valores: *1,2,k* al PC dependiendo de la entrada de control de 2 bits **mode12k**

III. Desarrollo e implementación en VHDL

III-1. Módulo de control del PC e IR y camino de datos : En el módulo «CtrlFetch» se implementarán los registros PC e IR así como sus entradas,salidas y señales de control de las que previamente se ha hablado.

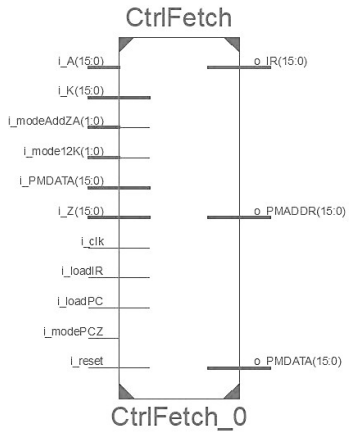

Figura 3: Módulo de control del PC e IR

El código VHDL que modela el comportamiento de la figura 3 se muestra a continuación, se incluye el camino de datos correspondiente:

```
entity CtrlFetch is
port (
    i_clk:          in std_logic;
    i_reset:        in std_logic;
    -- Control de las entradas al PC
    i_mode12K:      in std_logic_vector(1 downto 0);
    i_modeAddZA:    in std_logic_vector(1 downto 0);
    i_modePCZ:      in std_logic;
    i_loadPC:       in std_logic;
    i_loadIR:       in std_logic;
    -- Camino de datos
    i_K:            in unsigned(15 downto 0);
    i_A:            in unsigned(15 downto 0);
    i_Z:            in unsigned(15 downto 0);
    o_IR:           out unsigned(15 downto 0);
    o_PMDATA:       out unsigned(15 downto 0);
    -- Interfaz con la memoria
    o_PMADDR:       out unsigned(15 downto 0);
    i_PMDATA:       in unsigned(15 downto 0)
);
end CtrlFetch;
```

Figura 4: Definiendo camino de datos y señales de control para CtrlFetch

Éste módulo contiene a los dos registros (s_PC, s_IR) y unas señales asíncronas internas auxiliares.

```
architecture Behavioral of CtrlFetch is

-- Registros PC e IR
signal s_PC: unsigned(15 downto 0);
signal s_IR: unsigned(15 downto 0);

-- Señales internas asíncronas
signal s_adderInput1: unsigned(15 downto 0);
signal s_adderOutput: unsigned(15 downto 0);
signal s_pcInput: unsigned(15 downto 0);
signal s_addr: unsigned(15 downto 0);

begin
```

Figura 5: Se definen los registros PC e IR

Se incluye un sumador para agregar 1,2 o k según se determine por *i_mode12K*.

```
-- Sumador con multiplexor para el segundo operando
-- (1, 2, K)
s_adderOutput <= s_adderInput1 + s_PC;
s_adderInput1 <=
   "0000000000000001" when i_mode12K = "00" else
   "0000000000000010" when i_mode12K = "01" else
   i_K;
```

Figura 6: Sumador

La entrada al multiplexor provee el nuevo valor al PC (Z,A, o la salida del sumador).

```
-- Entradas del multiplexor para el PC
s_pcInput <=
   i_Z when i_modeAddZA = "10" else
   i_A when i_modeAddZA = "11" else
   s_adderOutput;
```

Figura 7: Multiplexor

Se crea un sencillo multiplexor que determina la dirección de entrada a la memoria de programa (PM). Ésta puede ser lo que tenga almacenado PC o también el registro Z.

```
-- Multiplexor para la dirección del PM
s_addr <=
   s_PC when i_modePCZ = '0' else
   i_Z;
```

Figura 8: Multiplexor para la entrada al PM

Se crea de igual manera la interfaz de la memoria del programa y hace que el registro IR esté disponible en la salida del módulo.

```
-- Interfaz PM
o_PMADDR <= s_addr;
o_PMDATA <= i_PMDATA;

o_IR <= s_IR;
```

Figura 9: Memoria de programa (PM)

Por último, se crea un proceso síncrono para el PC e IR con un reset síncrono.

```
-- Carga sincrona del PC e IR
process (i_clk)
begin
   if rising_edge(i_clk) then
      if i_reset = '1' then
         s_PC <= (others => '0');
         s_IR <= (others => '0');
      else
         if i_loadPC = '1' then
            s_PC <= s_pcInput;
         end if;
         if i_loadIR = '1' then
            s_IR <= i_PMDATA;
         end if;
      end if;
   end if;
end process;
```

Figura 10: Proceso síncrono de carga y reset del PC e IR

III-2. Memoria de Programa *:* Se implementa una pequeña memoria de programa para probar el PC y el IR, en ésta memoria de programa se almacenan instrucciones. Es una memoria de sólo lectura y siempre habilitada. En cada flanco de subida adquiere una dirección en la entrada y provee de un dato a la salida.

```
architecture Behavioral of ProgramMemory is

type PROGMEM is array(7 downto 0) of unsigned(15 downto 0);

signal s_pm: PROGMEM := (
x"0000", x"0000", x"0000", -- sin usar
x"9409", -- ijmp
x"cfff", -- rjmp .-2
x"cfff", -- rjmp .-2
x"fe00", -- sbrs r0, 0
x"c003"  -- rjmp .+6
);

begin

process (i_clk)
begin
   if rising_edge(i_clk) then
      o_data <= s_pm(to_integer(i_addr));
   end if;
end process;

end Behavioral;
```

Figura 11: Memoria de Programa

III-3. Máquina de estados *:* Se necesita una FSM para controlar las cargas y el tipo de instrucción que se ejecutará[3], en éste caso se basa el

diseño una máquina de estados tipo Moore, dónde las salidas dependen sólo del estado actual, éstas generan las señales de control para las tres instrucciones del programa de prueba.

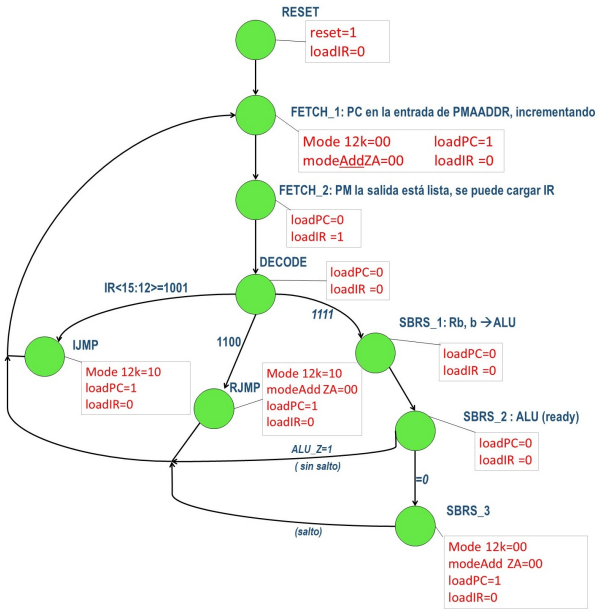

Figura 12: Máquina de estados

IV. Resultados

IV-1. Circuito implementado en VHDL : A continuación se muestra el diseño esquemático implementado en VHDL correspondiente al comino de datos y unidad de control para la ejecución del Contador de Programa (PC) y Registro de Intrucciones (IR).

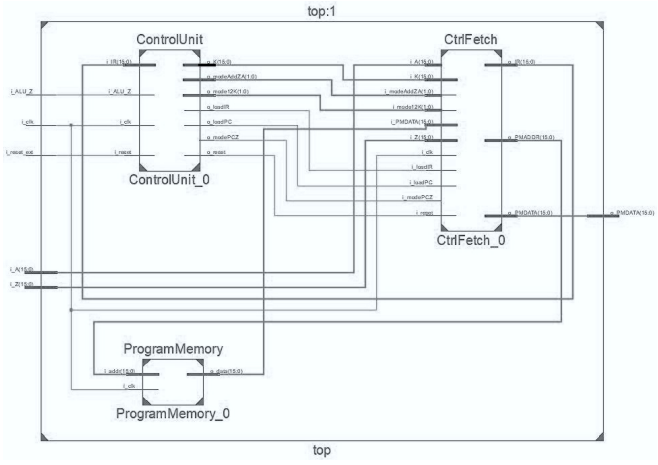

Figura 13: Máquina de estados

IV-2. Simulación de las instrucciones propuestas : Recordemos que el programa a ejecutar realizará las siguientes instrucciones:

```
ijmp
rjmp .-2
rjmp .-2
sbrs r0, 0
rjmp .+6
```

Para éstas tres diferentes instrucciones únicamente se necesita conocer un código de operación de 4 bits[4]:

- 1001… para IJMP,
- 1100… para RJMP, y
- 1111… para SBRS.

En la máquina de estados al incicio se colocan todos los registros en cero. Después pasan a los estados de búsqueda de instrucción (FETCH).

El primer estado (FETCH_1) pasa la dirección a la memoria de programa, el segundo estado(FETCH_2) espera por la instrucción a la salida de la memoria de programa lista para ser cargada en el registro de instrucción IR.

A continuación sigue la decodificación dónde se interpreta a IR.

Dependiendo del código de operación de 4 bits se ejecuta la correspondiente instrucción.

El resultado que esperamos tener es el siguiente:

- **Primer paso** Se decodifica la instrucción *RJMP* la cual agrega *k* al PC, por tanto se ajusta al PC a 0004 (+1+3). Figura 14
- **Segundo paso** En el siguiente estado se lee la instrucción *IJMP* (9409) la cual regresa al PC a la dirección 0001.Figura 15
- **Tercer paso** En ésta dirección para el siguiente estado se realiza un salto para cargar la instrucción *SBRS* (fe00). Figura 16
- **Cuarto paso** Y por último en el siguiente estado se regresa a la instrucción *RJMP* (cfff).Figura 16

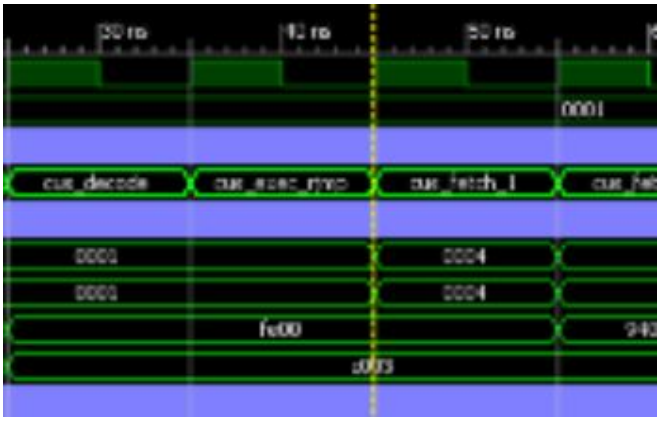

Figura 14: Se decodifica RJMP (c003) y se actualiza PC con 4

Figura 15: Se decodifica IJMP (9409) y se actualiza PC con 1

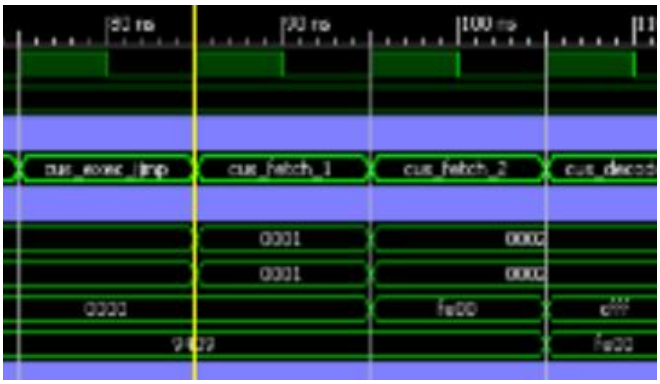

Figura 16: Se decodifica SBRS (fe00) y después RJMP(cfff)

Conclusiones

Se presentó el modelado del camino de datos y la unidad de control para el contador de programa (PC) y el registro de instrucciones (IR). Se hizo uso de una pequeña lista de instrucciones almacenada en una memoria de programa para así poder visualizar por medio de la simulación como las instrucciones son ejecutadas en cada fase de la máquina de estados, como cada instrucción repercute en el valor de la dirección del PC, PM e IR.

Referencias

[1] shehata2004design, title=Design and implementation of FPGA based and microcontroller based current relay, author=Shehata, Khaled and Bahaa, Ahmed and Morad, Karim and Sharaf, Ahmed, booktitle=Proceedings. The 16th International Conference on Microelectronics, 2004. ICM 2004., pages=783–786, year=2004, organization=IEEE

[2] gal2011fpga, title=FPGA implementation of 8-bit RISC microcontroller for embedded systems, author=Gal, Ryszard and Golda, Adam and Frankiewicz, Maciej and Kos, Andrzej, booktitle=Proceedings of the 18th International Conference Mixed Design of Integrated Circuits and Systems-MIXDES 2011, pages=323–328, year=2011, organization=IEEE

[3] hajduk2014fpga, title=An FPGA embedded microcontroller, author=Hajduk, Zbigniew, journal=Microprocessors and Microsystems, volume=38, number=1, pages=1–8, year=2014, publisher=Elsevier

[4] zhihua2007design, title=Design and Implementation of UART Based on FPGA [J], author=Zhihua, MENG, journal=Electronic Engineer, volume=4, year=2007