|  |  |  |  |
| --- | --- | --- | --- |
|  | 23/03/2011 |  | |
|  | |  | |
| PROYECTO FINAL, LA PC COMO CONTROLADORA DE PROCESOS.  *SOFTWARE PARA MANEJO DE SISTEMA DE CONTROLDE TAREAS DE ROSNANCIA MAGNETICA NUCLEAR* | | | |
|  |  | |  |
|  |  | | Adrián Gómez, Mariano Volarik, Raúl Monti |

PROYECTO FINAL, LA PC COMO CONTROLADORA DE PROCESOS.

SOFTWARE PARA MANEJO DE SISTEMA DE CONTROL DE TAREAS DE RESONANCIA MAGNÉTICA NUCLEAR

**INTRODUCCION**

rmnController.exe es un programa escrito en C/C++ capaz de controlar un sistema compuesto por un generador de pulsos (pp2), un coversor A/D (modelo AD9852), un sintetizador digital de frecuencias (dds). El fin de este equipo es generar señales con frecuencia y fase programables para un equipo de resonancia magnética nuclear, así como también adquirir los datos posteriores al experimento.

El programador de pulsos dirigirá el funcionamiento del resto de las partes componentes del sistema. El dds generará las señales y el conversor A/D tomará los resultados guardando hasta 64K adquisiciones por canal en sus dos canales de 12bits. La comunicación se logra vía el puerto paralelo, usando la interfaz de hardware disponible para comunicarse con cada una de las 3 placas por separado. El software se usará sobre una PC con sistema operativo Windows XP Sp3.

**El Programador de Pulsos (PP2)**

**2.0 INTRODUCCION:**

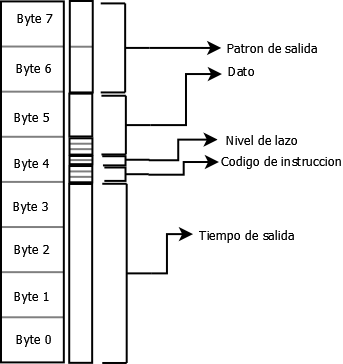
Básicamente, un generador de pulsos esta preparado para generar patrones de salida por un tiempo determinado, ambos programable. Es decir, consta de 16 salidas, cada una de las cuales puede mantener un patrón de salida independiente por un tiempo dado, y era nuestra tarea desarrollar un software de prueba que permita configurar dichos parámetros.

Lo que hace interesante al modulo desarrollado, es que como el PP2 contiene en su interior un microprocesador era necesario traducir de alguna manera los patrones y tiempos a instrucciones que entendiera el PP2, hecho que derivo en el desarrollo de un primitivo pseudo compilador de código.

Toda la información referida a las conexiones internas del dispositivo serán dejadas de lado porque no están estrechamente relacionadas con el desarrollo del estos módulos y además ya se encuentran muy bien documentadas. Información en la cual nos basamos para el desarrollo de este trabajo.

**2.1 FORMATO DE LAS INSTRUCCIONES:**

Las instrucciones con las que trabajo el PP2 son palabras de 64 bits que codifican la siguiente información donde cada uno de los casilleros inferiores representa 1 byte = 8 bits:



**Patrón de salida(16 bits):** Establece la señal que va a generarse en cada una de las 16 salidas que posee el generador.

**Dato(11 bits):** Se lo emplea para almacenar algún dato que necesite la instrucción que esta en ejecución.

**Nivel de lazo(2 bits):** Indica cual es el nivel de lazo que esta en ejecución. Por su capacidad no se permiten más de 4 niveles de lazo anidados.

**Código de instrucción(3 bits):** Contienen el código que identifica a la instrucción que se encuentra en ejecución. Los códigos son:

CONTINUE: (0X01) se utiliza para actualizar y mantener un patrón de salida por el tiempo

programado en el campo Demora. No utiliza el campo de Datos y el campo de Nivel de Lazo. El campo de Patrón de salida contiene el estado deseado de las salidas durante la instrucción.

LAZO: (0X02) se utiliza para repetir el grupo de instrucciones que la preceden. El número de repeticiones se indican en el campo de Datos. El nivel de lazo se indica en el campo de Nivel de Lazo.

El campo de Patrón de salida contiene el estado deseado de las salidas durante la instrucción.

RETL: (0X03) se utiliza para indicar la finalización del número o nivel de lazo de una instrucción de Lazo. El campo de Datos se utiliza para indicar la dirección de la instrucción de Lazo correspondiente. El campo de Patrón de salida contiene el estado deseado de las salidas durante la instrucción.

FIN:(0X07)Se utiliza para finalizar un programa de secuencias de pulsos.

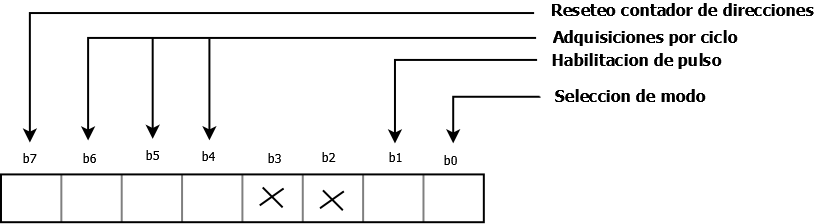
**Tiempo de espera(32 bits):** aquí se almacena el tiempo durante el cual se mantendrá el patrón en las salidas del generador.

Habiendo visto la estructura de las instrucciones que debemos escribir, estamos en condiciones de analizar las direcciones de registro con que cuenta el generador de pulsos.

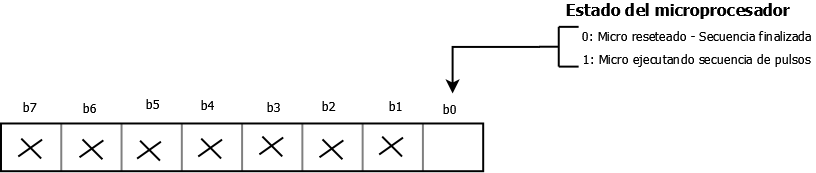
**2.2 REGISTROS:**

Para poder configurar el PP2, nuestro modulo necesitara acceder a distintos registros internos del mismo, donde cada uno de los bits escritos allí tiene vital importancia. Por ella, y como serán empleados más adelante, veamos cuales son estos registros.

**Registro de comando(0x50):** Al escribir en esta dirección estamos accediendo al registro de control, donde la palabra ingresada contiene la siguiente información:



**Registro de estado (0x50):** Cuando estamos leyendo en esta dirección, lo que estamos haciendo es acceder al registro interno de estado, donde cada uno de los bits corresponde a los siguientes datos:

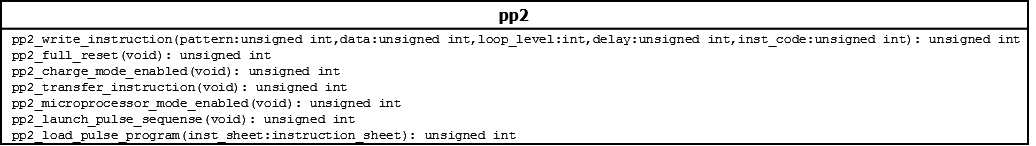


**Registro de instrucción(0x51):** En este registro se van a cargar las instrucciones obtenidas por nuestro modulo. Como las mismas son de 64 bits y el registro es de solo 8, es necesario cargar los 64 bits de a un byte comenzando por el menos significativo.

**Registro de transferencia(0x52):** Este registro se emplea solamente para transferir la instrucción cargada a la memoria RAM del dispositivo. El dato que se escriba es irrelevante.

Habiendo comprendido en líneas generales el comportamiento de este dispositivo, estamos en condiciones de analiza la estructura y funcionamiento del modulo

**2.3 ESTRUCTURA DEL MODULO:**



En este módulo encontramos todas las funciones que nos permiten trabajar sobre el PP2. Todo desde habilitar la carga del programa hasta la misma carga de instrucción por instrucción y el lanzamiento de la secuencia de pulsos cargados, se logra a partir de las funciones exportadas desde este módulo.

**2.4 FUNCIONAMIENTO GENERAL DEL MÓDULO:**

Los siguientes pasos son los habituales para cargar y lanzar el programa de pulsos:

1. Se resetea y habilita la carga del pp2 usando las funciones pp2\_full\_reset(void) y pp2\_charge\_mode\_enable(void).
2. Una vez completada la estructura instruction\_sheet del modulo de mismo nombre, se la pasa como argumento a la función pp2\_load\_pulse\_program(inst\_sheet: instruction\_sheet) el cual se encarga de armar las instrucciones para el pp2 y enviarlas con la ayuda de las funciones pp2\_write\_instruction(…) y pp2\_transfer\_instruction(…).
3. En este punto las instrucciones ya fueron cargadas en el pp2 por lo que solo queda a la función pp2\_microprocessor\_mode\_enable() seguida de pp2\_launch\_pulse\_sequence() para que se dispare la secuencia de pulsos

**Sintetizador Digital de Señales (DDS2**)

**3.1 INTRODUCCIÓN**:

Como ya habíamos mencionado, el DDS2 es uno de los dispositivos que integran el componente sobre el cual trabajamos. Su tarea no es mas que la de generar una onda sinusoidal, con fase y frecuencia configurada por el usuario, de manera digital. En este caso particular, el DDS2 con el que trabajamos tenia la capacidad de ser configurado con 2 frecuencias y una RAM para fases, que una vez cargada, se puede direccionar la fase deseada con un procedimiento que se explicara mas adelante.

Ahora bien, para configurar y manejar el DDS, se necesitaba trabajar empleando *words*, de manera que nuestro programa debe efectuar de manera automática la conversión entre fases y frecuencias ingresadas por el usuario y tales palabras, y en esta tarea es que se encontraba el mayor reto de este modulo, ya que las demás tareas resultaron bastante sencillas.

Cálculos realizados:

Para calcular la frecuencia se usa la siguiente formula:

Vfrec = (f\*2^32)/(50\*10^6)

Donde f es la frecuencia deseada en Hz, mientras que Vfrec es el valor resultante. Además f debe satisfacer que 12300Hz < f < 20MHz

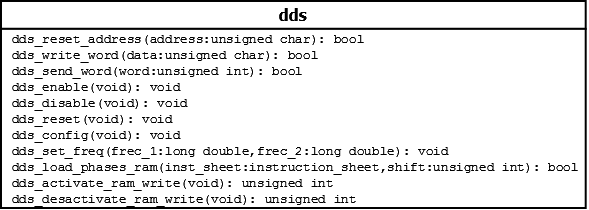
Para el casi de las fases, se debía realizar un cálculo similar, donde para obtener el valor a cargar en el DDS se necesita efectuar:

Vfase = (360/4096)\*Fgrad = ((2\*Pi)/4096)\*Fgrad

Donde f es el valor de la fase deseada y Vfase es la palabra a cargar en el DDS.

**3.2 MODO DE USO:**

Modulo:



* dds\_config();

Se realizan todas las configuraciones previas a la carga de los parámetros ingresados por el usuario

* dds\_set\_freq(frec1, frec2);

Se cargan las frecuencias ingresadas por el usuario realizando de forma automática los cálculos antes mencionados.

* dds\_enable();

Activa el dispositivo

* instruction\_sheet\_config\_instructions(inst\_sheet, delay\_sheet);

Se le da valores a los delays de los pulsos y fases dentro de la estructura inst\_sheet de acuerdo a los datos contenidos en el archivo con ruta indicada por delay\_sheet.

* dds\_load\_phases\_ram(inst\_sheet, shift);

Procede a la carga efectiva de las fases en la RAM de fases con los valores contenidos en inst\_sheet comenzado con un desplazamiento de fase dado por shift (inicialmente en 0)

**Conversor Analógico-Digital (A/D**)

**4.1 INTRODUCCIÓN:**

El conversor Analógico Digital es la cara más visible del software, debido a que se presentan los datos al usuario de una manera más amigable que los demás módulos del programa. Con el fin de abstraer el programa final los mas posible de la plataforma de hardware, las funciones de control de este circuito se realizan mediante la interfaz provista por el modulo AD que mas adelante procedemos a explicar.

**4.2 CARACTERÍSTICAS GENERALES:**

Las principales características del Conversor Analógico-Digital utilizado son:

* El conversor AD utilizado es del tipo Flash. Debido a su velocidad y al tipo de señal

digitalizada no es necesario utilizar Sample and Hold en la cadena de adquisición, ya que el circuito se basta por si mismo para realizar la conversión.

* La resolución del conversor es de 12 bits por canal. Y consta de 2 canales.
* Adicionalmente, el sistema de adquisición tiene incorporada una memoria interna de 64K por

canal, con lo cual es posible almacenar hasta 128K de muestras sucesivas.

* La señal que marca el inicial de adquisición es disparada por el PP2. El AD posee dos modos

distintos de funcionamiento:

Continuo: En este modo, el conversor recibe únicamente la señal de inicio de conversión, y adquiere en un único barrido todas las muestras.

Modulado: Cuando se configura el AD de esta manera, solo adquiere mientras el PP2 le provea de una señal de adquisición. Con este método, es posible realizar adquisiciones discontinuas, brindando una mayor flexibilidad al sistema.

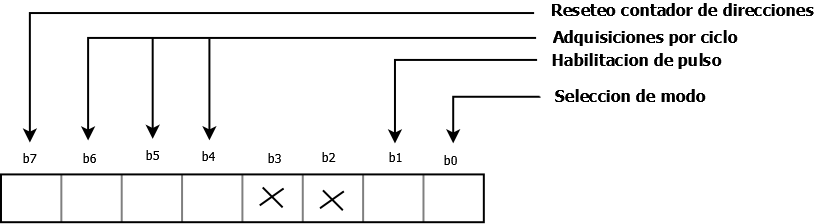
**4.3 FUNCIONAMIENTO INTERNO: HARDWARE**

A nivel interno, el funcionamiento del circuito es bastante sencillo: por un lado, tenemos el conversor A/D Flash de 2 canales, que es en definitiva el encargado de hacer la conversión. Sin embargo, con el fin de hacer el proceso mas confiable (en un SO multitarea es posible que el rendimiento no sea optimo) este circuito utiliza (como se menciono antes) una memoria interna. Tiene además un contador interno que apunta a algún lugar de esta memoria. Cuando se ejecuta la operación leer este contador se incrementa automáticamente, por lo que al llamar nuevamente a la operación leer el resultado será distinto. Debido a las características del conversor Flash, no hay necesidad de poner un Sample and Hold ni un filtro antialias.

**4.4 PROGRAMACIÓN:**

Internamente, el conversor A/D posee un conjunto de registros que controlan el funcionamiento del sistema. Estos son los siguientes:

**Registro de control:** Es el que configura casi todo el funcionamiento del conversor. Siendo de 8 bits, las opciones de posible configuración son las siguientes:



**b0 – selección de modo:** Con este bit se define el modo de funcionamiento del AD

0: modo PC: En este modo la PC tiene acceso a la memoria buffer del 12x2 FLASH y el 12x2 FLASH queda inhibido de realizar adquisiciones.

1: modo AD: En este modo cuando el A/D es disparado se completa un ciclo de adquisición, salvando los datos en la memoria buffer.

**b1 – habilitación de pulso:** Con este bit se selecciona el modo en que se adquieren las muestras una vez disparado el conversor.

0: modo modulado: EL ancho del pulso determina cuantas adquisiciones se realizan. Para completar una adquisición se necesita un ciclo de reloj. La duración del pulso debe calcularse en función del numero de adquisiciones por segundo (Modo modulado).

1: modo continuo: El circuito esta siempre habilitado, hasta completar las adquisiciones.

**b6, b5, b4: adquisiciones por ciclo:** aquí definimos la cantidad de datos que deseamos adquirir

000: 0,5K adquisiciones

001 : 1K adquisiciones

010: 2K adquisiciones

011 : 4K adquisiciones

100: 8K adquisiciones

101 : 16K adquisiciones

110: 32K adquisiciones

111 64K adquisiciones

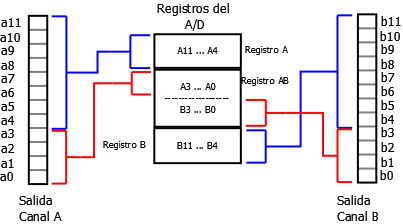
**b7 – reseteo del contador de direcciones**: Como se explico antes, el conversor mantiene un puntero interno a la dirección a la que ingreso por ultima vez. Por esto, antes de iniciar una nueva adquisición hay que poner este contador en 0, aunque es preciso no dejar el bit encendido, ya que de otra manera la adquisición no se guardara.

0: Funcionamiento normal

1: contador permanentemente reseteado

**Registro de estado:** La principal función de este registro es la de indicar, mediante el bit b0, si ya ha finalizado un ciclo de adquisición y/o lectura de la memoria del buffer.

**Registro de datos:** Como ya se ha mencionado antes, la resolución es de 12 bits. Por ello, los valores obtenidos del A/D se obtienen mediante 3 registros:



Registro de intervalo de muestreo: Con este registro podemos seleccionar el intervalo entre dos muestras consecutivas, con lo cual podemos deducir la cantidad de muestras por segundo. Notemos que este registro es de 8 bits, por lo cual solo hay 255 posibles valores distintos (el registro debe ser distinto de 0). El periodo de la frecuencia base es de 100 nSeg., por lo cual para obtener una muestra cada 1 uSec utilizamos la siguiente formula:

N = 1uSeg/100nSeg

= 1uSeg/0,1uSeg

= 10

RM = 255 – 10

= 245

**COMPORTAMIENTO GENERAL Y FORMA DE USO:**

El programa funciona a través de una interfaz gráfica la cual asegura que los datos sean ingresados correctamente. Por lo tanto para usarlo es necesario simplemente seguir los pasos que esta indica. Sin embargo es necesario tener en cuenta los siguientes puntos:

1- Se pide como dato de entrada el programa de pulsos. Este deberá ser un archivo .txt redactado de acuerdo al apartado \_.

2- La repetición del experimento más de una vez (usando el campo “Número de repeticiones”) provoca que se pise el archivo que guarda los resultados de cada adquisición. Por lo que solo quedará guardada la adquisición correspondiente a la ejecución final.

Internamente una ejecución normal del programa sigue los siguientes pasos:

1- Se lee y revisa la hoja de pulsos comprobando su correctitud sintáctica. Se usa para ello la función parse() perteneciente al módulo “parser”. Esta función devuelve una estructura 'instruction\_sheet' con toda la información obtenida de la hoja de pulsos.

2- Se agrega la información de tiempos de cada pulso y delay a la instruction\_sheet previamente creada. Esta información viene dada a partir de un archivo redactado según el apartado \_ y debe corresponder a la hoja de pulsos parseada en el punto 1. Para todo esto se usa la función instruction\_sheet\_config\_instructions() del módulo 'instruction\_sheet'.

3- Se configura y habilita el dds. Para ello es necesario saber cuáles son las frecuencias de trabajo deseadas para el dds. Si bien el dds tiene la posibilidad de cargar 2 frecuencias distintas, el programa solo admite la utilización de la frecuencia número 1. Todo esto se hace utilizando las funciones dds\_config(), dds\_set\_freq() y dds\_enable() todas del módulo 'dds'.

4- Carga las fases en la RAM de fases para su posterior utilización desde el dds. La información de las fases se toma a partir de la estructura instruction\_sheet. Se usa para esto la función dds\_load\_phases\_ram() del módulo 'dds'.

5- Se carga el programa de pulsos en el pp2 usando la estructura instruction\_sheet a partir de la función pp2\_load\_program().

6- Se configura el A/D usando la función ad\_config() del módulo 'ad' . Para ello se requieren los siguientes datos: cantidad de muestras por segundo; tamaño en K de la cantidad de adquisiciones; el modo en que se desea adquirir (solo está disponible en este programa el modo continuo).

7- Se dispara el programador de pulsos. Para ello se usan las funciones pp2\_microprocessor\_mode\_enabled() y pp2\_launch\_pulse\_sequence() ambas pertenecientes al módulo 'pp2'.

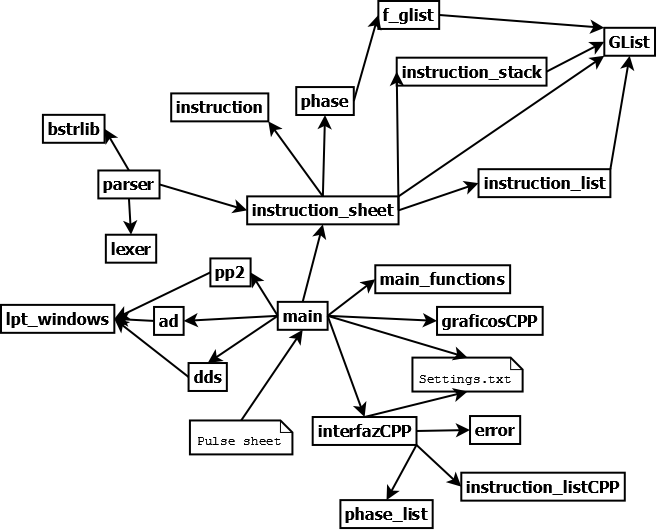
8- Se espera el resultado de la adquisición usando la función adquiere() perteneciente al módulo 'ad' y opcionalmente se los guarda en el archivo de salida especificado usando la función ad\_to\_file(). Este último paso es obligatorio si se desea poder graficar más adelante los resultados ya que el programa está diseñado para comunicarse con la interfaz y el graficador a través de los archivos que estos generan.

9- Se llama al programa "Graficador.exe" el cual interpreta el archivo de salida del A/D y lo grafica. (Notar que esto permite colocar en el directorio cualquier programa que sepa interpretar los datos guardados en el archivo de salida del A/D siempre y cuando se le dé el nombre "Graficador.exe").

10- Termina el programa.

**GUÍA DE MÓDULOS**

**6.1 DIAGRAMA GENERAL DE MÓDULOS:**



**6.2 LISTADO DE MÓDULOS:**

**Módulo 'ad':**

Aquí encontramos todas las funciones necesarias para configurar habilitar el conversor, como así también aquellas para esperar la adquisición y guardar los resultados.

**Módulo 'dds':**

Este módulo contiene funciones necesarias para la configuración y control tanto del dds como de la RAM de fases. Entre ellas se encuentra la función encargada de escribir las fases en la RAM.

**Módulo 'f\_glist':**

Módulo con una lista circular de flotantes construido sobre las listas enlazadas de la librería Glib. Para usos varios en otros módulos.

**Módulo 'instruction':**

La estructura aquí definida abstrae el concepto de instrucción que se da en cada línea de la hoja de pulsos para traducirla a una instrucción legible por el pp2.

**Modulo 'instruction\_sheet':**

Su estructura representa la abstracción completa de la hoja de pulsos y de tiempos. A partir de esta estructura se logra cargar el programa en el pp2. Sus funciones sirven específicamente para este propósito.

**Módulo 'interface':**

Se encarga de proporcionar funciones que sirvan de interfaz entre la GUI y el programa.

**Módulo 'lpt\_windows':**

Contiene funciones para la utilización del puerto paralelo en Windows Xp. Exporta funciones para comunicarse con las placas del pp2, dds y A/D.

**Módulo 'main\_functions':**

Contiene funciones que sincronizan la ejecución de los distintos procesos que se pueden llevar a cabo con el programa. Por el momento son solo 2, la primera (main\_parse\_pulse\_program()) ejecuta un parseo sobre la hoja de pulsos y crea archivos de interfaz para usarlos desde la GUI (ver apartado \_), la segunda (main\_run()) corre el programa entero según los parámetros obtenidos desde la interfaz.

**Modulo 'parser':**

La única función que exporta se llama 'parse'. Esta realiza el control sintáctico sobre la hoja de pulsos y rellena, con la información obtenida de esta, la estructura instruction\_sheet utilizada para cargar el pp2.

**Módulo 'phase':**

Módulo que abstrae el concepto de fases, la cuales son cargadas desde el programa de pulsos.

**Módulo 'pp2':**

Entre sus funciones más importantes encontramos las que nos permiten cargar el pp2 y correr el programa de pulsos.

Apartados

**7.1 REDACCIÓN DE LA HOJA DE PULSOS:**

Ejemplo:

phases:

ph1 = 90 90

ph2 = 0 0

program:

loop 2 times phase\_shift

loop 2 times

d5

p1(ph1)

p2(ph2)

d3

endloop

endloop

acquire

Como vemos en el ejemplo, la hoja de pulsos consiste en dos partes claramente separadas. Una de estas partes trata de la declaración de fases: comienza con la palabra 'phases:' y por cada fase que se desea tener se debe escribir una línea con las siguientes características: debe empezar con la palabra phX donde X es un número que representa la fase. Luego, separados por espacio, deben ir todos los valores que se pretende pueda tomar esa fase.

La otra parte es la declaración del programa. Como vemos comienza con la palabra 'program:'. Por cada pulso que se desee enviar se debe tener una línea con la letra p seguida inmediatamente por el número de identificación del pulso y entre paréntesis la fase que se desea que el pulso de salida tenga. Esta fase debe ser una de las declaradas más arriba. Por otro lado por cada delay que se desea es suficiente con colocar una línea nueva comenzando con la letra d seguida inmediatamente del número de identificación para el delay.

Si se quieren lazos se envuelven las órdenes que se desean repetir entre las instrucciones 'loop X times' y 'endloop' donde X un número entero representando la cantidad de veces que se desea repetir las instrucciones envueltas. Si se desea que la fase rote en cada repetición es suficiente con agregar la opción 'phases\_shift' a la derecha de la palabra times separada por un espacio, al momento de abrir el lazo.

Por último en caso de requerir una adquisición se debe simplemente escribir una línea con la palabra 'acquire'.

Las instrucciones, a excepción de los lazos, se cargaran de manera secuencial para su posterior ejecución de acuerdo a como se hayan escrito en la hoja de pulsos.

**7.2 REDACCIÓN DE LA HOJA DE TIEMPOS:**

Ejemplo:

p1:200

d3:150

p32:400

d1:100

Como vemos simplemente se coloca la letra p (para pulso) o d (para delay) seguida del identificador y denotar la instrucción. Luego se colocan ':' seguido del numero en unidades de tiempo (cada unidad de tiempo equivale a 40ns) durante las cuales se desea mantener el pulso o el delay. No hay un orden obligatorio para cada línea, pero deben estar presentes todas las instrucciones nombradas en la hoja de pulso.

**7.3 FORMATO DE LOS ARCHIVOS FASES.DAT E INSTRUCTION.DAT:**

Estos archivos son creados por la función 'main\_parse\_pulse\_program(char \*pulse\_sheet, char \*fases\_sheet, char \*instr\_sheet)' del módulo 'main\_functions'.

Una vez ejecutada esta función, suponiendo que la hoja de pulsos esta correctamente escrita, se generarán en el directorio del ejecutable dos archivos con los nombres especificados por los últimos dos parámetros entregados.

El primero de estos archivos contiene información acerca de las fases. Más específicamente estará compuesto por sucesivas líneas de la forma:

'id de fase' 'valor de fase 1' 'valor de fase 2' ... 'valor de fase N'

Tendremos una línea por cada declaración de fase de la hoja de pulsos. En la línea el primer número es el número de identificación de la fase y los sucesivos números serán los distintos valores de fase posible para esta fase. Cada número estará separado de los demás por un espacio y cada línea excepto la última estará separada de la que sigue por un caracter salto de línea. El fin de este archivo es otorgar una representación conveniente de las fases para poder utilizarla desde una interfaz gráfica.

Con el mismo fin se construye el segundo archivo, este con información acerca de las instrucciones. Sus líneas tendrán el siguiente formato:

'id' 'data' 'number' 'shift'

El primer número indica de qué tipo de instrucción se trata. El segundo campo varía de acuerdo al tipo de instrucción. El tercer número indica el número de instrucción, y el último el grado de corrimiento de fase si es que se trata de una instrucción de pulso.

* 1. **MÓDULOS CON SUS RESPECTIVAS OPERACIONES EXPORTADAS Y ESTRUCTURA INTERNA:**

