Universidad Nacional de Córdoba
Facultad de Matemática Astronomía
Física y Computación

Trabajo especial de la Licenciatura en
Ciencias de la Computación

Sistema de gestión de experimentos y control de Módulo Digital para Resonancia Cuadrupolar Nuclear

Victor Pablo Bovina Astorga

dirigida por
Ing. Walter Zaninetti

Diciembre 2018

Índice

Agradecimientos

A los profesores.
A mi familia.
A mi novia y su familia.
A mis amigos de la Vieja Escuela.
A FaMAF de punta a punta.
A mis amigos de Ascentio.
A mis amigos de la Bicicleta.
A Dios que es grande.

En memoria de mis abuelos que también dieron sus pasos por la UNC.

Por su paciencia.
Por su perseverancia.
Por su temple.
Por los valores.

Muchas Gracias!

1. Introducción

El área de Resonancia Magnética Cuadrupolar y Nuclear de FaMAF posee en su inventario un Módulo Digital para el estudio de cristales moleculares que se desarrollan en este grupo de investigación, el cual no dispone de una interfaz de uso manual, puesto que está pensado para el uso vía interfaz USB con la PC y actualmente en fuera de uso por la falta de la misma.

En el presente trabajo se aborda el desarrollo de software para el control del Módulo Digital y de manera complementaria una plataforma web que colabore con la gestión de los experimentos vinculados con el uso del mismo.

El objetivo de proveer una plataforma web es la de brindar el acceso a los experimentos y sus resultados de manera remota junto con la capacidad de realizar experiencias de investigación desde la oficina con una ayuda mínima de algún colaborador del área presente cerca de los demás equipos, puesto que siempre hay un miembro por razones de seguridad.

El Módulo Digital consta de tres submódulos:

- Programador de pulsos digitales: tiene la finalidad de ejecutar una secuencia de pulsos para coordinar los submódulos del Módulo Digital.
- Generador digital de señales: genera las señales de estimulación a los núcleos resonantes de la materia a investigar.
- Conversor analógico digital: convierte las señales analógicas de la resonancia a digitales para su posterior análisis.

Cada uno de estos submódulos posee su propia arquitectura y comandos para su manipulación los cuales serán partes claves en el desarrollo del software y la integración de los mismos en un único controlador.

Existen varios tipos de experiencias posibles de ser modeladas con el uso del Módulo digital entre ellas dos:

- Experiencia Promedio: una secuencia iterativa de pulsos de radio frecuencia de propiedades fijas.
- Experiencia Promedio con Pulso variable: una experiencia promedio donde los pulsos de radiofrecuencia varían sus propiedades durante las iteraciones de la misma.

El objetivo es poder modelar ambas aunque con la *Experiencia Promedio* es suficiente para el alcance de este trabajo.

2. Marco de trabajo

En el proceso de investigación del fenómeno físico de *Resonancia Magnética Cuadrupolar y Nuclear* el investigador manipula módulos electrónicos digitales de medición precisos y estables, en algunos casos si intervienen más de uno a la vez, entre ellos sincronizados por medio de interfaces digitales. Estos módulos electrónicos digitales colaboran con la creación del contexto necesario para investigar lo planeado según la necesidad del investigador. En general junto con los módulos electrónicos digitales intervienen otros módulos electrónicos de naturaleza analógica tales como amplificadores operacionales, mezcladores de señales, filtros pasa bajos, entre otros.

La correcta conexión entre los diferentes módulos electrónicos digitales, analógicos y su configuración durante el proceso de experimentación son responsabilidad del investigador y una tarea de suma importancia para el éxito de la experiencia a realizar.

En este marco de responsabilidades del investigador y el módulo digital a ser utilizado es deseable y necesario el desarrollo de mecanismos sencillos de manipulación del mismo y su monitoreo.

2.1. Rol del Software

El rol primario del software en el contexto descrito previamente es el de manipular el módulo digital a través de la PC utilizada por el investigador de forma local o remota.

El rol secundario la administración de usuarios del módulo digital, monitoreo de los experimentos en curso y resultados.

3. Partes de un experimento

Estos son algunos conceptos clave para comprender el contexto y definición de un experimento, los cuales se tuvieron en cuenta al momento de la implementación del sistema.

3.0.1. Radio frecuencia

Es una señal senoidal que tiene como objetivo estimular la muestra presente en el resonador y que puede ser manipulada previamente por otros módulos externos. Tiene los siguientes atributos:

- Frecuencia: 0 a 80 megahercios.
- Fase: 0 a 360 grados.
- Duración: 0 a 16 segundos.

3.0.2. Pulso TTL

Es una señal digital de 5 voltios de amplitud y duración predefinida con la finalidad de trasmitir como entrada a otros módulos digitales o analógicos para sincronizar momentos de relajación y estimulación de las muestras durante la experiencia.

3.0.3. Muestras

Es un grupo de datos obtenidos a cierta frecuencia de muestreo y agrupados de manera contigua en un bloque de longitud 2^n con $n \in \mathbb{N}$.

3.1. Tipo de Experiencia

El tipo de experiencia a realizar esta determinada por el investigador, existen 2 bien diferenciadas:

3.1.1. Experiencia Promedio

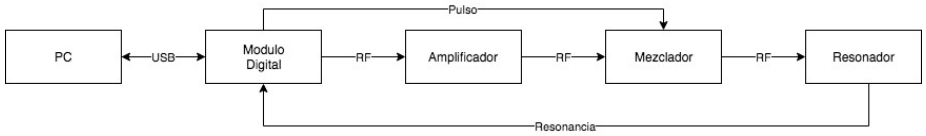
Una experiencia promedio es una que se repite y donde los bloques de muestras ordenadas se suman uno a uno obteniendo una mejor representación los datos para su análisis.

3.1.2. Experiencia Promedio con Pulso variable

Es una experiencia promedio donde la configuración de los pulsos cambia en las sucesivas iteraciones de la misma. Los pulsos cambian su configuración para suprimir interferencias de estimulaciones previas y obtener mediciones mas precisas. Los atributos del pulso que pueden cambiar en las sucesivas iteraciones son la fase y duración.

4. Diagrama de conexiones

Este diagrama representa la interconexión física entre los diferentes módulos digitales y analógicos que forman parte de una experiencia planificada.



.

Figura 1: Diagrama de conexiones

4.1. Mezclador

Los pulsos TTL de salida y la señal de radiofrecuencia son las entradas del mezclador donde se convierten en una sola cuando el pulso TTL está activo en 5 voltios. Esto permite crear pulsos de estimulación y relajación con precisión.

4.2. Amplificador

La señal de RF proveniente del *Mezclador* se amplifican tras los efectos de este que debilitan la señal. La configuración del mismo varía según el experimento a realizar.

4.3. Resonador

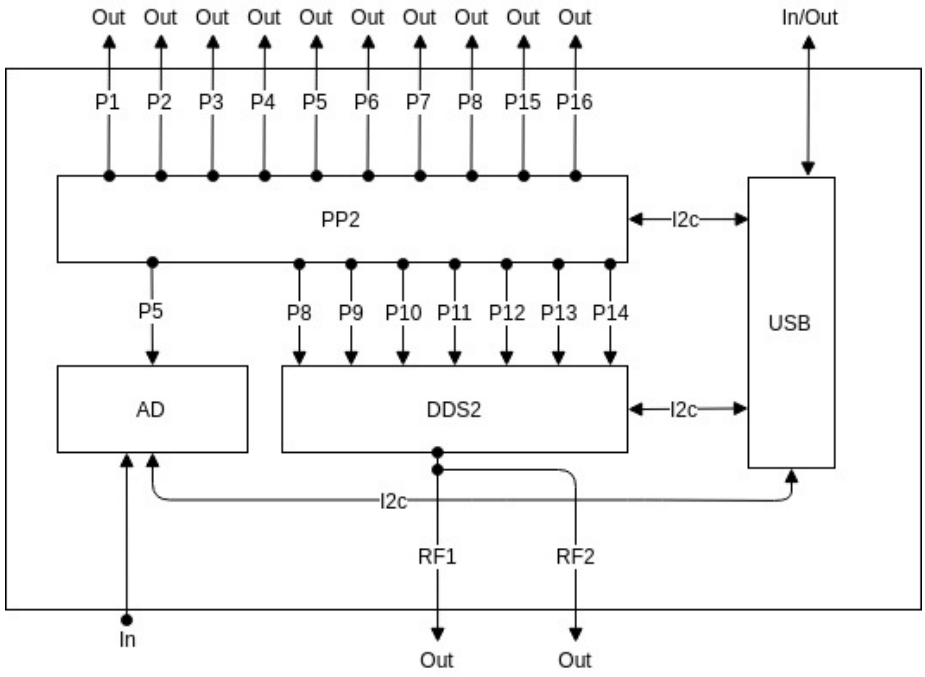
Entre sus partes más relevantes este cuenta con una denominada «malla de acople» la cual tiene como objetivo dejar pasar la señal RF al contenedor de la muestra e impedir que continúen a la etapa de adquisición. La señal de salida de la resonancia es pre-amplificada antes de retornar al *Módulo Digital*

4.4. Configuración alternativa

En algunos casos es posible utilizar el *Módulo Digital* como generador de pulsos TTL con una fuente generadora de señal RF externa. Esto no es habitual pero es posible de realizar.

5. Descripción del módulo digital

El siguiente diagrama de bloques muestra la configuración interna del aparato junto a sus características técnicas.



.

Figura 2: Diagrama de bloques del módulo digital

5.1. Salidas de pulsos TTL

El usuario dispone de las salidas P1, P2, P3, P4, P6, P7, P15, P16. Las salidas P5, P8 si bien están disponibles para su utilización tienen una finalidad especial en el funcionamiento interno del módulo que abordaremos más adelante.

5.2. Salidas RF

Las salidas radiofrecuencia RF1 y RF2 son gemelas con un valor pico a pico de 200 milivoltios.

5.3. Entradas AD

Las entradas del conversor analógico digital son AD1 y AD2 correspondientes a los canales A y B respectivamente. Admiten hasta 1.5 voltios de entrada.

6. Submódulo PP2

El programador de pulsos digitales PP2 es un microprocesador diseñado a medida con instrucciones que generan un *patrón de salida*. Un *patrón de salida* es una combinación de 16 pulsos TTL en paralelo por un periodo de tiempo determinado. Un programa ejecutable por el PP2 será una secuencia de no más de 512 instrucciones y la ejecución derivará en una *secuencia de patrones de salida* de duración determinada.

6.1. Instrucciones

El PP2 tiene 4 instrucciones básicas: *Continue*, *Retl*, *Loop* y *End* cada una es una secuencia de 64 bits con la siguiente estructura:

Patrón de salida	Dato	Nivel de lazo	Código de instrucción	Duración
16 bits	11 bits	2 bits	3 bits	32 bits

Cuadro 1: Estructura de las instrucciones.

6.1.1. Duración de una instrucción

Cada instrucción requiere de 4 pulsos de reloj ($4*40ns = 160ns$) y la duración mínima es 2 pulsos ($2 * 40ns = 80ns$). Por lo tanto el pulso de valle mínimo es de $160ns + 80ns = 240ns$.

6.1.2. Continue

Mantiene una combinación de 16 pulsos de salida por un tiempo determinado.

- patrón de salida: es la combinación de 16 pulsos de salida.
- dato: no utilizado.
- nivel de lazo: no utilizado.
- código de instrucción: 0x01.
- duración: duración de la instrucción.

6.1.3. Loop

Marca el apertura de un bloque de repetición.

- patrón de salida: es la combinación de 16 pulsos de salida.
- dato: contador de repeticiones.

- nivel de lazo: nivel de profundidad de anidamiento entre lazos.
- código de instrucción: 0x02.
- duración: duración de la instrucción.

6.1.4. Retl

Marca de cierre de un bloque de repetición.

- patrón de salida: es la combinación de 16 pulsos de salida.
- dato: dirección de la instrucción Loop de inicio.
- nivel de lazo: no utilizado.
- código de instrucción: 0x03.
- duración: duración de la instrucción.

6.1.5. End

Finaliza la secuencia de pulsos.

- patrón de salida: es la combinación de 16 pulsos de salida.
- dato: no utilizado.
- nivel de lazo: no utilizado.
- código de instrucción: 0x07.
- duración: duración de la instrucción.

6.2. Registros del PP2

El PP2 cuenta con los siguientes registros de 8 bits:

Dirección	Descripción	Modo
0x50	comando	escritura
0x51	carga	escritura
0x52	señal	escritura

Cuadro 2: Estructura de las instrucciones.

6.2.1. Carga de un programa en el PP2.

Algoritmo 1 Carga de una programa en el PP2

```
1:  procedimiento UploadProgram()
2:      // Reset
3:      write(0x50, 0x02)
4:      // Modo carga
5:      write(0x50, 0x03)
6:      // Continue 0x55AA 4
7:      write(0x51, 0x04)
8:      write(0x51, 0x00)
9:      write(0x51, 0x00)
10:     write(0x51, 0x00)
11:     write(0x51, 0x01)
12:     write(0x51, 0x00)
13:     write(0x51, 0xAA)
14:     write(0x51, 0x55)
15:     write(0x52, 0x00)
16:     // End
17:     write(0x51, 0x00)
18:     write(0x51, 0x00)
19:     write(0x51, 0x00)
20:     write(0x51, 0x00)
21:     write(0x51, 0x00)
22:     write(0x51, 0x00)
23:     write(0x51, 0x00)
24:     write(0x51, 0x00)
25:     write(0x52, 0x00)
```

6.2.2. Ejecución de un programa.

Algoritmo 2 Ejecución de un programa.

```
1:  procedimiento ExcuteProgram()
2:      // Modo microprocesador
3:      write(0x50, 0x00)
4:      // señal de ejecución
5:      write(0x08)
```

7. Submódulo DDS2

El submódulo DDS2 es un generador de señales digitales con un rango de frecuencia de 0 a 80 Mhz. El usuario puede almacenar hasta 2 frecuencias y un total de 16 fases para combinarlos en un experimento. Existe una relación entre el PP2 y el DDS2, puesto que este tiene como entrada los pulsos 8,9,10,11,12,13,14 para su manipulación externa por aquel durante la ejecución de un programa.

7.1. Configuración inicial, activación y desactivación.

Dirección	B7	B6	B5	B4	B3	B2	B1	B0	Hex
1D	0	0	0	1	1	1	1	1	0x17
1E	0	1	0	0	0	1	0	0	0x44
1F	0	0	0	0	0	0	1	0	0x02
20	0	0	0	0	0	0	0	0	0x00

Cuadro 3: Reset

Dirección	B7	B6	B5	B4	B3	B2	B1	B0	Hex
1D	0	0	0	1	0	0	0	0	0x10
1E	0	1	0	0	0	1	0	0	0x44
1F	0	0	0	0	0	0	1	0	0x02
20	0	0	0	0	0	0	0	0	0x00

Cuadro 4: Activación

Dirección	B7	B6	B5	B4	B3	B2	B1	B0	Hex
1D	0	0	0	1	1	1	1	1	0x17
1E	0	1	0	0	0	1	0	0	0x44
1F	0	0	0	0	0	0	1	0	0x02
20	0	0	0	0	0	0	0	0	0x00

Cuadro 5: Desactivación

7.2. Registros del DDS2

Dirección	Descripción	Modo
0x70	direccionamiento	Escritura
0x71	modo	Escritura
0x72	reset	Escritura
0x73	test	Lectura
0x74	señal de escritura	Escritura
0x75	direccionamiento	Escritura
0x76	señal de transferencia	Escritura
0x77	test	Lectura
0x78	señal de escritura	Escritura

Cuadro 6: Registros internos del DDS2.

7.3. Fases

El DDS2 tiene disponible 32 posiciones de memoria 8 bits para el almacenamiento de fases con un bus de datos de 8 bits. La fase es un valor de 14 bits, los 2 bits restantes del MSB son descartados. El MSB de una fase debe almacenarse siempre en una dirección par de la memoria, luego el LSB de la misma en el valor impar contiguo siguiente. Antes de almacenar el valor entero de la fase F debemos convertirlo a su equivalente en unidades de 45 de la siguiente manera:

$$F_h \in \mathbb{N} \tag{1}$$

$$F_l \in \mathbb{N} \tag{2}$$

$$F \in \mathbb{N} \tag{3}$$

$$F_h = (45 * F)/256 \tag{4}$$

$$F_l = (45 * F) - (F_h * 256) \tag{5}$$

Direcciones de RAM disponibles	Modo
0x00 0x02 0x04 0x06 0x08 0x0A 0x0C 0x0E	0x02
0x10 0x12 0x14 0x016 0x18 0x1A 0x1C 0x1E	0x02

Cuadro 7: Direcciones de Fase.

7.3.1. Algoritmo base de almacenamiento de fases

Algoritmo 3 Almacenamiento de una fase en dirección 0x00

1: **procedimiento** SAVEPHASE(F)
2: // MSB y LSB de valor de fase
3: $F_h = (45 * F)/256$
4: $F_l = (45 * F) - (F_h * 256)$
5: // direcciones contiguas
6: $dir_h \leftarrow 0x00$
7: $dir_l \leftarrow dir_h + 1$
8: // Modo carga de fases
9: $write(0x71, 0x02)$
10: // MSB
11: $write(0x70, dir_h)$
12: $write(0x74, f_h)$
13: // LSB
14: $write(0x70, dir_l)$
15: $write(0x74, f_l)$
16: // Modo PC
17: $write(0x71, 0x00)$

7.3.2. Direccionamiento de fases

Los 16 valores de fase son direccionados con 4 pulsos correspondientes a los pulsos 11,12,13,14. Con el pulso 9 se activa la carga de fases y con el pulso 10 se transfiere la fase direccionada al registro de trabajo. El tiempo entre el pulso 9 y 10 debe ser menor a 100 nanosegundos.

7.4. Frecuencias

El DDS2 tiene disponible 12 registros internos de frecuencia de 8 bits, cada frecuencia ocupa 6 registros, por lo tanto se pueden almacenar 2 frecuencias. Las frecuencias disponibles van de 0 a 80 MHz y el valor que se almacena en los registros en representación se obtiene con la siguiente cálculo:

$$clock = 2 \times 10^6 \tag{6}$$

$$frec \in \mathbb{N} \wedge 0 < frec < clock \tag{7}$$

$$valor = frec \times (2^{48} - 1)/clock \tag{8}$$

Nro. Frecuencia	Registros	Modo
1	0x04 0x05 0x06 0x07 0x08 0x09	0x00
2	0x0A 0x0B 0x0C 0x0D 0x0E 0x0F	0x00

Cuadro 8: Registros de Frecuencia.

7.4.1. Algoritmo base de almacenamiento de frecuencias

Algoritmo 4 Almacenamiento de una frecuencia de trabajo 1.

procedimiento SAVEFREQUENCY(F)
    // Conversión de la frecuencia deseada al valor
    $clock \leftarrow 2 \times 10^6$
    $value = frec \times (2^{48} - 1)/clock$
    // Modo PC
    $write(0x71, 0x00)$
    // primero MSB hasta LSB
    // Byte 5
    $write(0x75, 0x04)$
    $write(0x78, value_5)$
    // Byte 4
    $write(0x75, 0x05)$
    $write(0x78, value_4)$
    // Byte 3
    $write(0x75, 0x06)$
    $write(0x78, value_3)$
    // Byte 2
    $write(0x75, 0x07)$
    $write(0x78, value_2)$
    // Byte 1
    $write(0x75, 0x08)$
    $write(0x78, value_1)$
    // Byte 0
    $write(0x75, 0x09)$
    $write(0x78, value_0)$
    // Actualización registro de trabajo
    $write(0x76, 0x00)$

7.4.2. Direccionamiento de frecuencias

Se admiten hasta 2 frecuencias de trabajo, se seleccionan con el pulso 8.

8. Submódulo AD

El submódulo conversor analógico digital AD tiene 2 canales de adquisición en paralelo con una resolución de 12 bits por canal, una frecuencia de muestreo máxima de 10 Mz y capacidad de almacenamiento en bloques de 1KB, 2KB, 4KB, 8KB, 16KB, 32KB, 64KB, 128KB.

Dirección	Descripción	Modo
0x0B	comando y control	lectura/escritura
0x0C	muestreo	escritura
0x08	canal B	lectura
0x09	canal AB	lectura
0x0A	canal A	lectura

Cuadro 9: Registros del AD

8.1. Registro de comando

Bit	Descripción
0	modo
1	modo
2	-
3	-
4	bloque
5	bloque
6	bloque
7	reset

Cuadro 10: Registro de comando.

8.2. Registro de datos

Dirección	Descripción
0x08	canal B
0x09	canal AB
0x0A	canal A

Cuadro 11: Registros de datos.

8.3. Configuración de un ciclo de adquisición

Un ciclo de adquisición requiere la configuración del intervalo de muestreo de acuerdo a la siguiente fórmula:

$$period = 100ns \tag{9}$$

$$interval \in \mathbb{N} \land 0 < interval < 25500 \tag{10}$$

$$n = interval/period \tag{11}$$

$$delta = 255 - n \tag{12}$$

$$\tag{13}$$

8.3.1. Configuración bloque de 1KB y muestreo 1 microsegundo

Algoritmo 5 Configuración 1KB 1 microsegundo

```
1:  procedimiento SETUP()
2:
3:      config ← 0x00
4:      config ← config ∨ 0x02
5:      config ← config ∨ 0x80
6:      config ← config ∧ 0xFE
7:      write(0x0B, config)
8:
9:      config ← 0x00
10:     config ← config ∨ 0x02
11:     config ← config ∧ 0x7F
12:     config ← config ∨ 0x01
13:     write(0x0B, config)
14:
15:     period ← 100
16:     samples = 1000/period
17:     delta = 255 − samples
18:     write(0x0C, delta)
```

8.4. Extracción de los datos de la memoria interna

El bus de datos de la memoria del AD es de 8 bits y las muestras de 12 bits, siendo necesarias 3 lecturas de 8 bits y una operación de partición para obtener la muestra final de los canales A y B.

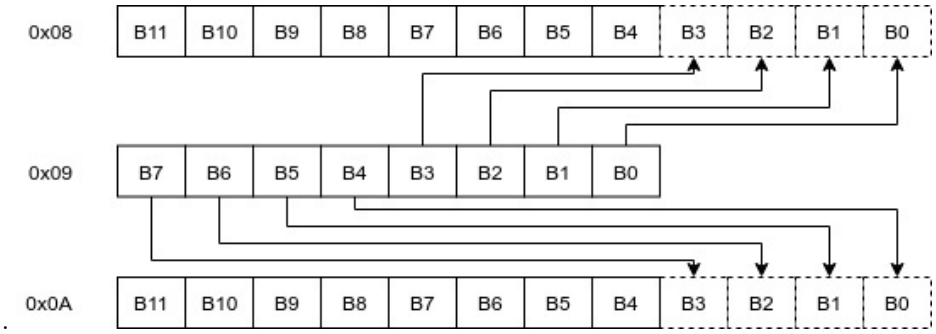


Figura 3: Buffers del submódulo AD

Para la extracción de los datos es necesaria esta serie de pasos:

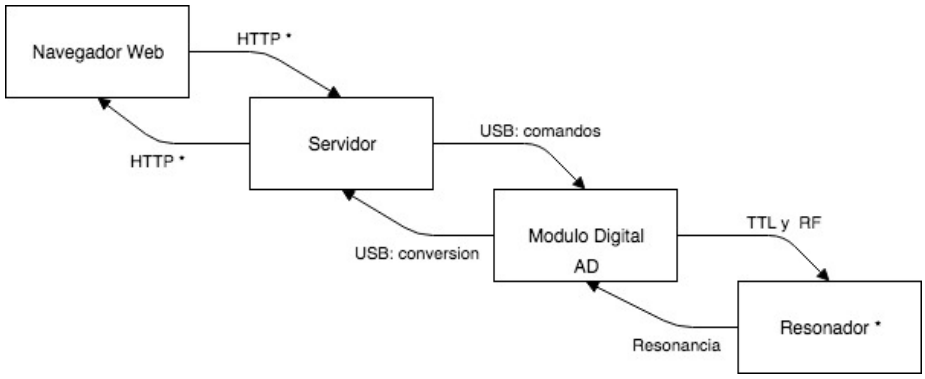
- modo PC, deshabilitada la adquisición y reset contador de direcciones
- modo PC, deshabilitada la adquisición y contador de direcciones en modo normal
- lectura del registro 0x08 del canal B
- modo PC, deshabilitada la adquisición y reset contador de direcciones
- modo PC, deshabilitada la adquisición y contador de direcciones en modo normal
- lectura del registro 0x0A del canal A
- modo PC, deshabilitada la adquisición y reset contador de direcciones
- modo PC, deshabilitada la adquisición y contador de direcciones en modo normal
- lectura del registro 0x09 con nibbles del canal A y B.
- armado de los datos de los respectivos canales.

Algoritmo 6 Extracción de datos del AD

```
1:  procedimiento GetDataFromAD()
2:      config ← 0x00
3:      config ← config ∨ 0x02
4:      config ← config ∨ 0x80
5:      config ← config ∧ 0xFE
6:      write(0x0B, config)
7:      config ← 0x00
8:      config ← config ∨ 0x02
9:      config ← config ∧ 0x7F
10:     config ← config ∧ 0xFE
11:     write(0x0B, config)
12:     A ← read(0x08)
13:
14:     config ← 0x00
15:     config ← config ∨ 0x02
16:     config ← config ∨ 0x80
17:     config ← config ∧ 0xFE
18:     write(0x0B, config)
19:     config ← 0x00
20:     config ← config ∨ 0x02
21:     config ← config ∧ 0x7F
22:     config ← config ∧ 0xFE
23:     write(0x0B, config)
24:     B ← read(0x0A)
25:
26:     config ← 0x00
27:     config ← config ∨ 0x02
28:     config ← config ∨ 0x80
29:     config ← config ∧ 0xFE
30:     write(0x0B, config)
31:     config ← 0x00
32:     config ← config ∨ 0x02
33:     config ← config ∧ 0x7F
34:     config ← config ∧ 0xFE
35:     write(0x0B, config)
36:     AB ← read(0x09)
37:
38:     sample_A ← join_buffers(A, AB)
39:     sample_B ← join_buffers(B, AB)
```

9. Visión general del sistema

Este diagrama representa la interacción entre los diferentes elementos de hardware y software en una experiencia planificada.



.

Figura 4: Visión General del sistema

9.1. Navegador web

El navegador web renderiza la *Interfaz Gráfica* solicitada al *Servidor*, esta provee al usuario final el control remoto del sistema.

9.2. Servidor

El *Servidor* provee servicios REST solicitados por la *Interfaz Gráfica* durante la vida de la sesión del usuario. Estos servicios hacen llamadas al controlador del *Módulo Digital* vía USB.

9.3. Módulo Digital

El *Módulo Digital* procesa los mensajes del controlador via USB y ejecuta el microcódigo del mismo para la configuración y ejecución de las secuencias de pulsos.

9.4. Resonador

El *Resonador* recibe los pulsos provenientes del *Módulo Digital* generando una señal de resonancia enviada al *Módulo Digital* para su conversión digital.

10. Definición de un experimento

Para representar la definición de un experimento se eligió notación JSON, por el soporte built-in por parte de los lenguajes usados para desarrollar el sistema, Python y Javascript, también por la implementación de servicios REST necesarios en el *Servidor*[2]. Haciendo uso de aritmética podemos deducir que la representación de un experimento en JSON es liviana puesto que el número de instrucciones de un programa P será de $N < 512$ y para un archivo de texto de 512 líneas y 80 columnas de caracteres de 1 Byte tiene un peso aproximado de 327.68 Kilobytes. La validación del esquema JSON de un experimento, es llevada a cabo por un módulo programado dentro del sistema para tal fin. Cabe destacar que los tipos de datos presentes en la especificación de un experimento son soportados sin pérdida de precisión por parte de la notación JSON.[1]

```
{
  "cpoints": [
    {
      "lsb": "00000011",
      "freq_unit": "hz",
      "t_unit": "ns",
      "type": "C",
      "msb": "00000011",
      "time": "10",
      "phase": "0",
      "freq": "100",
      "data": "0",
      "id": 1535931832181
    }
  ],
  "settings": {
    "a_times": "3",
    "a_name": "exp 1",
    "a_channel": "3",
    "a_description": "descr exp 1",
    "a_freq": "100",
    "a_msb": "00000000",
    "a_freq_unit": "hz",
    "a_ts_unit": "ns",
    "a_lsb": "00000000",
    "a_ts": "100",
    "a_bloq": "1",
    "a_phase": "0"
  },
  "execute": true
}
```

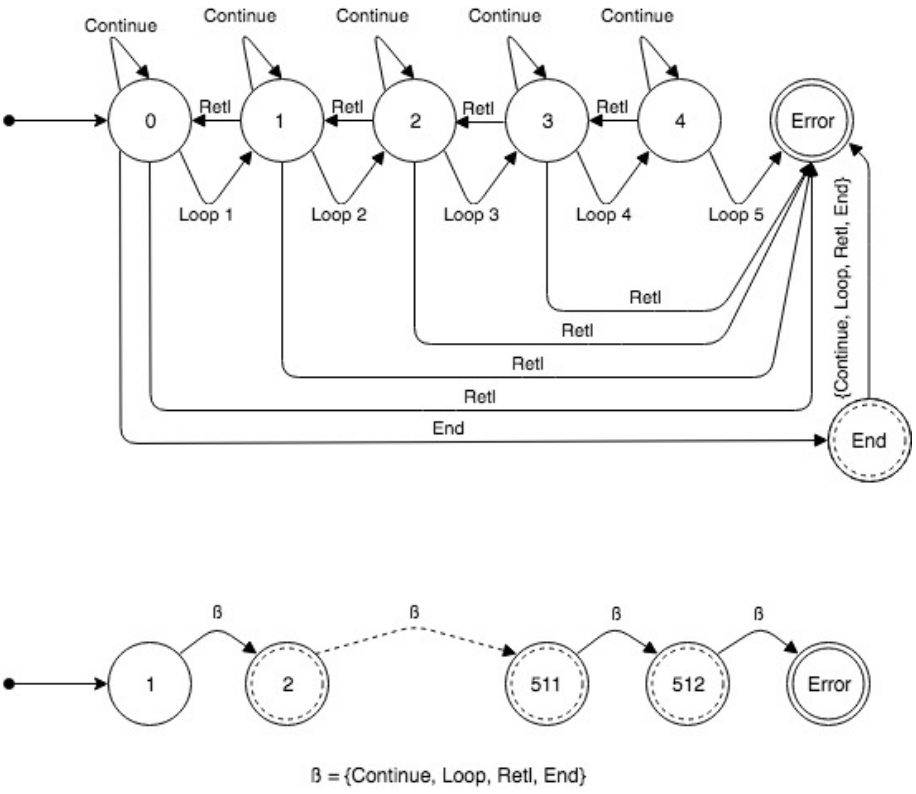
11. Algoritmo de traducción de experimentos

Un programa almacenable y ejecutable en el PP2 es una secuencia de instrucciones $I_1...I_N$ con $1 \leq N \leq 512$.
Donde las instrucciones disponibles son:

- *Continue*
- *Loop*
- *Retl*
- *End*

No todos los programas escritos con estas instrucciones son admitidos como válidos para el PP2, por esto tenemos las siguientes reglas:

- Regla 1: la instrucción *End* siempre es la última.
- Regla 2: la instrucción *End* aparece sólo una vez.
- Regla 3: la instrucción *End* siempre está presente.
- Regla 4: un bucle siempre inicia con instrucción *Loop* y finaliza con *Retl*.
- Regla 5: puede haber hasta 3 instrucciones *Loop* anidados en un *Loop*.
- Regla 6: la longitud del programa no puede ser mayor a 512 instrucciones.



.

Figura 5: Los programas aceptados por el PP2.

El algoritmo desapila instrucciones de P y según el tipo se valida de manera distinta. La variable L cuenta el número de Loops abiertos al momento, si es mayor a 4 finaliza en error caso contrario se agrega la instrucción a la secuencia S y se aplica recursión. El único caso donde no hay recursividad es cuando la instrucción desapilada es *End* donde si hay mas intrucciones por desapilar de P ó el contador L es no esta en 0 termina en error caso contrario se agrega *End* a la secuencia S y se valida al final que S contenga al menos una instrucción, la última sea *End* y que no supere la longitud de 512 instrucciones, el limite de almacenamiento del $PP2$.

Algoritmo 7 Algoritmo de traducción base

```
1: procedimiento TRANSLATE(P, L=0, S=[])
2:
3:     assert(len(S) ≤ 512)
4:     ins ← pop(P)
5:     if ins = Continue then
6:         S.append(ins)
7:         Translate(P, L, S)
8:     if ins = Retl then
9:         if L > 0 then
10:            S.append(ins)
11:            L ← L − 1
12:            Translate(P, L, S)
13:        else
14:            return error
15:    if ins = Loop then
16:        if L < 4 then
17:            S.append(ins)
18:            L ← L + 1
19:            Translate(P, L, S)
20:        else
21:            return error
22:    if ins = End then
23:        if len(P) = 0 ∧ L = 0 then
24:            S.append(ins)
25:        else
26:            return error
27:
28:    assert(1 ≤ len(S) ≤ 512)
29:
30:    assert(last(S) = End)
31:    return S
```

12. Ejecución de un experimento

Un modo de capturar fallos es simular la ejecución de un experimento, evitando la interacción vía USB, con el módulo digital. En modo simulado el experimento se ejecuta en un entorno limitado con el objetivo de detectar tres situaciones no deseadas:

- Una excepción de software no capturada.
- Parámetros fuera de rango para alguno de los submódulos.
- Un error en la traducción del experimento a un programa ejecutable en el PP2.

Un experimento simulado se ejecuta el hilo principal, puesto que la demora es baja. De manera alternativa el modo simulado puede verse también como un test de integración entre las unidades de software desarrolladas. Si el experimento simulado tiene éxito entonces se procede con la ejecución efectiva del experimento que se desprende del hilo principal y en el cual debemos tener algunos detalles en cuenta:

- Puede durar horas.
- Puede ser interrumpido por el usuario en cualquier momento.
- Debe estar sincronizado con la base de datos.

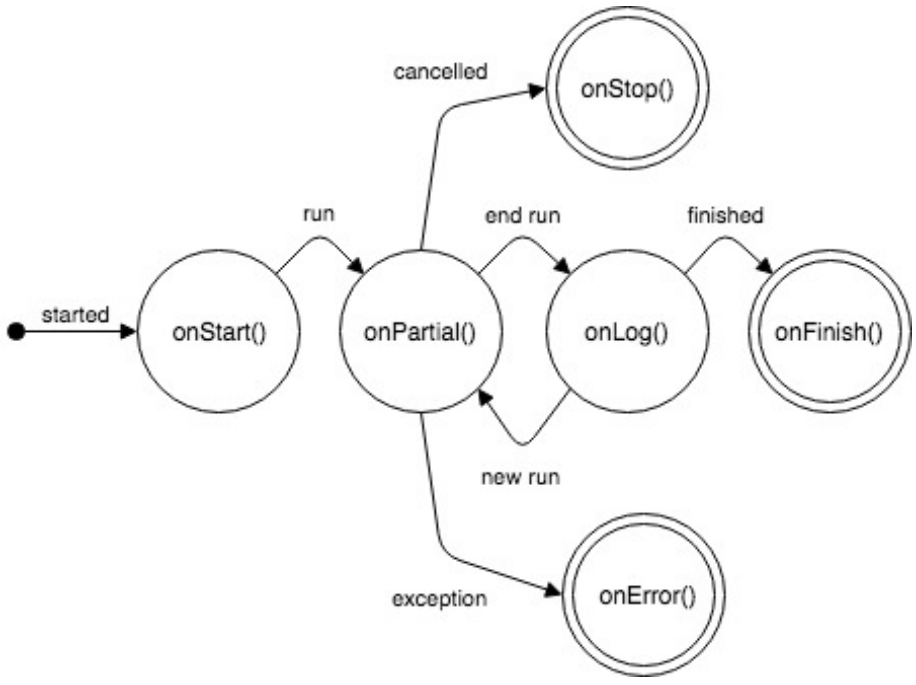
13. Hilo de ejecución de un experimento

La duración de un experimento puede ser de horas, ésta característica involucra tres problemáticas a resolver:

- Evitar time-out del lado del cliente web.
- Desacoplar la atención de una petición HTTP de usuario de la ejecución de un experimento.
- Lograr una experiencia de usuario agradable en lo que respecta a performance.

Modelando el proceso de ejecución con un hilo separado del principal es una solución válida para resolver el problema y teniendo en cuenta los siguientes:

- Seguimiento del estado del hilo.
- Control sobre el número de hilos creados.
- Gestión de los datos durante la ejecución del hilo.



.

Figura 6: Estados de un hilo de experimento.

14. Flujo de ejecución de un experimento

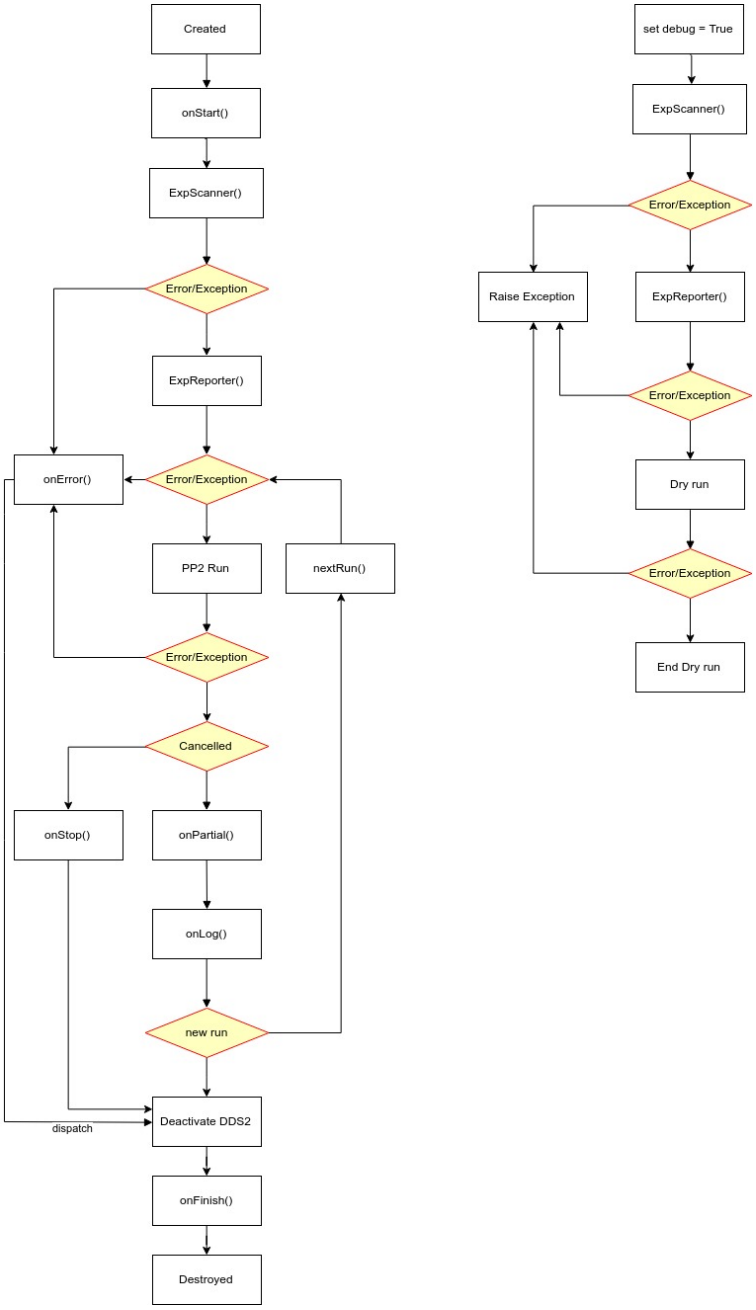


Figura 7: Flujo de ejecución de un experimento.

15. Requisitos del sistema

Estos son los requerimientos generales del sistema a desarrollar:

- Permitir a más de un usuario utilizar el sistema.
- Permitir a usuarios la gestión de experimentos.
- Modelar un experimento y su ejecución.
- Proveer acceso remoto al sistema.

De estos requerimientos generales se desprenden los siguientes específicos:

- El usuario tiene una sesión asignada al ingreso del sistema y revocada a la salida del mismo.
- Varios usuarios pueden acceder al sistema al mismo tiempo.
- El usuario puede ver si hay un experimento en ejecución y el detalle del mismo.
- El usuario tiene un espacio de trabajo donde puede crear/ver/actualizar/eliminar sus experimentos.
- El usuario puede cancelar el experimento en ejecución.
- El usuario puede solicitar un reporte parcial de un experimento en ejecución.
- El usuario puede solicitar el reporte final de un experimento finalizada la ejecución.
- El usuario puede verificar el estado de un experimento.
- El usuario es notificado antes de la ejecución de un experimento en las siguientes situaciones:
  - Salida voluntaria por error:
    - No es una secuencia ejecutable por el PP2.
    - Ya existe algún experimento ejecutándose.
    - Algún parámetro de configuración es inválido en algún submodulo PP2, DDS2, AD, USB.
    - Falló en conexión vía USB.
  - Salida involuntaria por error:
    - Hubo alguna excepción de sistema no controlada.
    - No se pudo establecer conexión con el servidor.
    - Un experimento finalizado no actualizó su estado impidiendo la ejecución de otro.

- Un experimento tiene una secuencia definida con sus parámetros.
- Un experimento tiene una marca de tiempo de creación/actualización.
- Un experimento eliminado no es recuperable.
- Un experimento tiene un autor que es el usuario que lo creo.
- Un experimento tiene estado *Created* cuando está almacenado en base de datos.
- Un experimento es solo visible para el usuario que lo creo.
- Un experimento tiene un título.
- Un experimento tiene una descripción.
- Un experimento no tiene un historial de edición asociado.
- Un resultado es el producto de la ejecución de un experimento.
- Un resultado es parcial cuando la ejecución del experimento asociado no finalizó.
- Un resultado es final cuando la ejecución del experimento asociado finalizó.
- Un resultado tiene un único experimento asociado.
- Un reporte parcial es el grupo de datos de un resultado parcial.
- Un reporte final es el grupo de un resultado final.
- Un reporte contiene:
  - Log de la ejecución.
  - Datos del AD en formato CSV.
  - Experimento ejecutado.
- El sistema tiene servicios REST para:
  - Crear/Ver/Editar/Eliminar experimentos.
  - Iniciar/Cancelar ejecución de un experimento.
  - Cancelar la ejecución de todos los experimentos.
  - Descargar reportes parciales y finales.
  - Proveer autenticación a todos los servicios.
- El sistema tiene una interfaz de usuario web.

16. Planificación del desarrollo

De acuerdo con los requerimientos descritos se propone implementar el sistema con:

- Framework backend: Django 1.11
- Lenguaje backend: Pythton 2.7
- Frameworks frontend: ReactJs 16.0.0[6]/ NodeJs 8.11.3 / NPM 5.6.0
- Lenguaje frontend: Javascript ECM6.
- Control de versión de código: Git (GitHub)[15] .

Las tecnologías elegidas son ampliamente soportadas por la comunidad de desarrolladores, de código abierto y soporte en PC. Con el objetivo de asegurar que lo primero que se implemente sea en relación a los requerimientos de control del Módulo Digital, luego proveer una manipulación remota del mismo y finalmente una interfaz de usuario para la gestión de los experimentos, se organizó el desarrollo en el siguiente orden:

- 1) Módulo Digital.
- 2) Servidor.
- 3) Interfaz Gráfica Web.

El mapeo entre los requerimientos y el orden propuesto genera el siguiente esquema de tareas a realizar.

- Módulo Digital
  - Configuración del proyecto y entornos.
  - Implementación control AD.
  - Implementación control DDS2.
  - Implementación control PP2.
  - Implementación abstracción *Experimento*.
  - Implementación de interfaz USB.
  - Implementación de un programa principal.
  - Gestión de logs.
  - Integración con *Servidor*.
- Servidor
  - Configuración del proyecto y entornos.
  - Servicios REST para la edición de experimentos.
  - Servicios REST para control de ejecución de experimentos.

- Servicios REST para la generación de reportes.
- Modelado de base de datos.
- Configuración de sesiones.
- Configuración de acceso a los servicios REST.
- Gestión de logs.
- Integración con Interfaz WEB.
- Integración con Módulo Digital.

- Interfaz WEB
  - Configuración de proyecto y entornos.
  - Implementación de transiciones.
  - Componente de edición de experimentos.
  - Componente de control de ejecución de experimentos.
  - Componente de login/logout.
  - Componente de visualización de errores.
  - Componente de definición de experimento.
  - Implementación de control de estados.
  - Integración con Servidor.

17. Planificación de Testing

La planificación de testing del sistema tiene varios enfoques en respuesta a diferentes objetivos:

- Brindar una experiencia predecible al usuario final.
- Lograr una integración coherente entre módulos.
- Aumentar cobertura de requerimientos de usuario y sistema.
- Facilitar la comprensión del comportamiento esperado del sistema.

Por unidad existen las siguientes validaciones:

- Módulo Digital
  - Entradas de configuración para el PP2, DDS2 y AD.
  - Entradas de programa ejecutable por el PP2.
  - Existencia de conexión vía interfaz USB durante la ejecución de experimentos.
  - Salidas de señales RF.
  - Salidas de pulsos TTL.
  - Input de conversión de datos del AD.
  - Salida de reportes de los canales A y B.
- Servidor
  - Autenticación obligatoria en los servicios.
  - Ejecución secuencial de los experimentos.
  - Aislamiento de espacio de trabajo de usuarios.
  - Coherencia en los estados de los experimentos.
  - Generación de reportes.
  - Gestión de procesos de ejecución de experimentos.
- Interfaz de usuario web
  - Visualización coherente de elementos y estilos.
  - Tiempos cortos de carga de las vistas.
  - Visualización de alertas de usuario.
  - Transiciones entre vistas.

Dados los diferentes enfoques son necesarias se eligieron las siguientes para probar:

- *Postman* para simular comunicación HTTP con servicios REST.

- *Arduino* para visualizar los pulsos TTL via *Serial Plotter*.
- *Osciloscopio* para medir señales RF generadas en los experimentos.
- *Generador de señales* para simular adquisición de datos del AD.
- *Google Chrome* y herramientas para validar estados de la interfaz gráfica.
- *Mock del servidor* escrito en *Flask-Python* para simular servicios REST.

18. Arquitectura del Módulo Digital

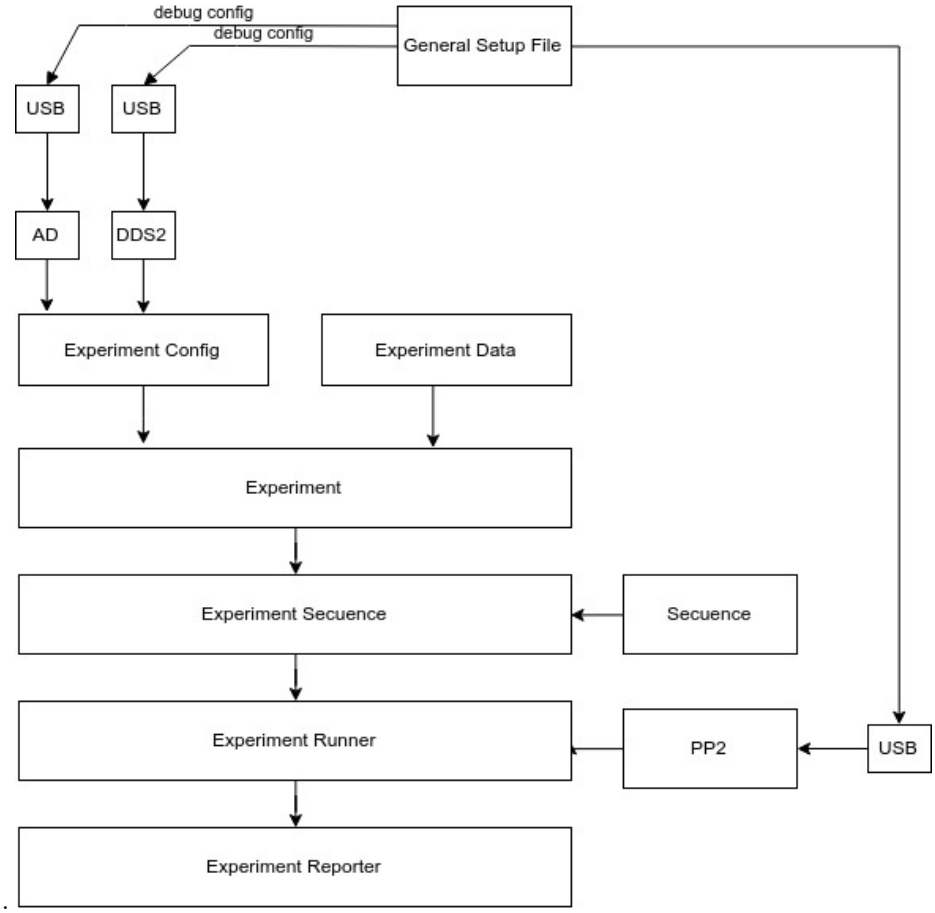


Figura 8: Arquitectura Módulo Digital

Entre las clases principales dentro del *Módulo Digital* se destaca *Experiment Reporter* que es el último eslabón de una cadena de especializaciones la cual implementa el método «Next» de las clases de Python, en consecuencia a la naturaleza iterable de los experimentos. Una instancia de esta clase tiene un comportamiento diferenciado según se trate de un experimento simulado ó no simulado, dictaminado por el estado de la propiedad *simulation-mode* definida en *General Setup File*.

19. Arquitectura del Servidor

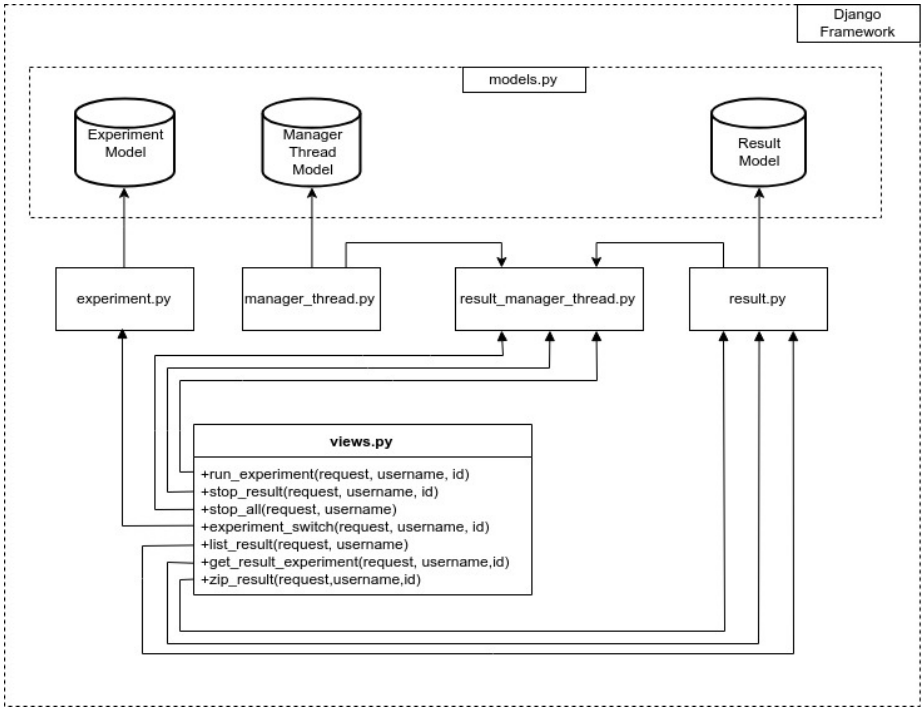
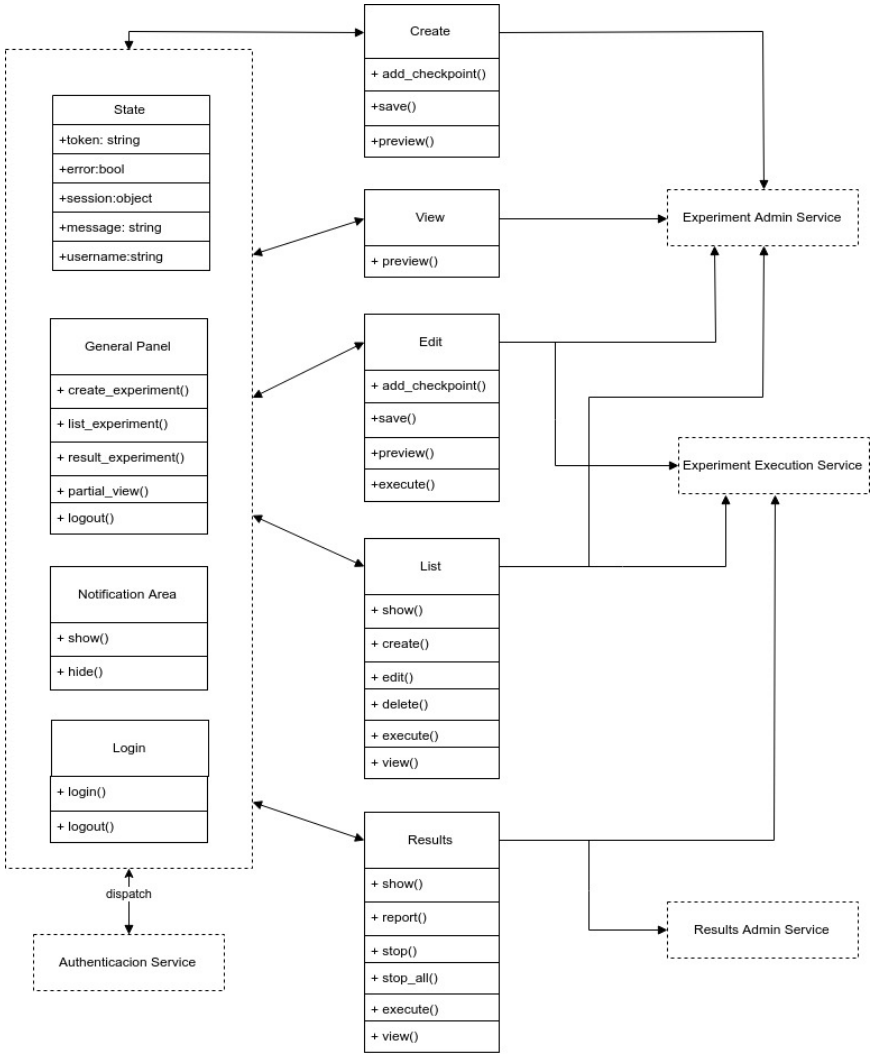


Figura 9: Arquitectura del Servidor

El *Servidor* cumple con la estructura que propone *Django Framework* siguiendo el patrón MVC[3]. En este proyecto existen 3 modelos básicos que abstraen el experimento, resultados y ejecución del mismo para el seguimiento del estado de un experimento en curso. Cada modelo tiene una abstracción intermedia con la finalidad de ofrecer las operaciones sobre los estos cumpliendo los requerimientos de usuario para ser invocados por los servicios REST implementados en las vistas.

20. Arquitectura de Interfaz Gráfica



.

Figura 10: Arquitectura de la Interfaz Gráfica.

La *Interfaz Gráfica* está implementada bajo directrices de ReactJs[12], donde cada vista es una clase dinámica con su propio estado interno[9] . El objeto destacado y trasversal en todas las vistas es *State*[10] el cual contiene información requerida por el *Servidor* tal como las cookies de usuario ó como también el despliegue de notificaciones ante una respuesta de error a alguno de los servicios REST.

21. Integración de los submódulos

La integración de los módulos del sistema se implementa a diferentes niveles.

21.1. Repositorio

Un submódulo git es un repositorio dentro de otro que lo incluye, con su propio historial de cambios. Cuando hay un cambio en un submódulo el contenedor lo contemplará siempre y cuando lo agregue al historial de este [8]. El repositorio afectado por esta metodología es el del *Servidor* incluyendo al *Módulo Digital* y a la *Interfaz Gráfica*.

El árbol de directorios del repositorio queda así:

```
Server
├── build
│   └── static
├── error
├── experiments
│   └── DigitalModule
├── log
├── login
├── out
├── ReactUI
└── settings
```

21.2. Código

El *Servidor* provee el middleware *ManagerThread* para poder interactuar con el *Módulo Digital* y con el modelo de la base de datos correspondiente según el estado de la ejecución del experimento en curso. Para integrar el *Servidor* con la *Interfaz Gráfica* el soporte es built-in via configuración *Django* el cual provee autenticación de los servicios siempre y cuando esté presente el CSRF-Token en el archivo principal de la *Interfaz Gráfica*[4] junto con la cookie de sesión correspondiente en cada petición HTTP[5].

22. Casos de Prueba

De acuerdo con los enfoques generales y por unidad se diseñaron los siguientes casos de prueba:

22.1. Módulo Digital

- Experimento con configuración inválida del AD.
- Experimento con configuración inválida del DDS2.
- Experimento con secuencia inválida para el PP2.
- Experimento con un pulso largo.
- Experimento con un pulso corto.
- Experimento con pulsos largos y cortos intercalados.
- Experimento promedio con un pulso.
- Experimento promedio con un pulso variable en duración.
- Experimento promedio con un pulso variable en fase.
- Experimento con un pulso dentro de un *Loop*.
- Experimento con un pulso dentro de un *Loop* de nivel 4.
- Experimento con un *Retl* sin *Loop* asociado.
- Experimento con un *Loop* sin *Retl* asociado.
- Experimento con un *Loop* de nivel 5.

22.2. Servidor

- Solicitud de servicio sin token de autenticación.
- Solicitud de autenticación con usuario no registrado.
- Solicitud de autenticación con usuario registrado pero contraseña inválida.
- Solicitud de fin de sesión.
- Solicitud de creación de un experimento.
- Solicitud de edición de un experimento.
- Solicitud de edición de un experimento inexistente.
- Solicitud de edición de un experimento en ejecución.
- Solicitud de eliminación de un experimento.

- Solicitud de eliminación de un experimento inexistente.
- Solicitud de eliminación de un experimento en ejecución.
- Solicitud de ejecución de un experimento habiendo uno en ejecución.
- Solicitud de ejecución de un experimento inexistente.
- Solicitud de ejecución de un experimento.
- Solicitud de cancelación de una ejecución en curso.
- Solicitud de reporte durante la ejecución.
- Solicitud de reporte finalizada la ejecución.
- Solicitud de reporte de un experimento que nunca se ejecutó.
- Solicitud de reporte de una ejecución cancelada.
- Solicitud de reporte de una ejecución fallida.

22.3. Interfaz Gráfica

- Ingreso al sitio con usuario y contraseña válidos.
- Ingreso al sitio con usuario ó contraseña inválidos.
- Salida del sitio con botón login.
- Crear un experimento con sus variantes.
- Visualización de lista de experimentos creados.
- Filtrado de lista de experimentos creados.
- Ver un experimento seleccionado de la lista.
- Edición de un experimento seleccionado de la lista.
- Eliminación de un experimento seleccionado de la lista.
- Ejecución de un experimento seleccionado de la lista.
- Visualización de lista de resultados.
- Visualización de detalles de un resultado.
- Visualización de menú de gestion de experimentos.
- Visualización de menú de gestion de resultados.
- Visualización de menú de navegación.
- Visualización de alertas.

23. Errores conocidos

Estos son los errores conocidos resultado de la ejecución de los casos de pruebas diseñados.

- si el sistema es interrumpido por un apagado imprevisto durante una ejecución es posible que un estado de la base de datos sea inconsistente. Esto se debe a que no está implementado un comportamiento para tal situación.
- las alertas deben ser cerradas para obtener nuevas si las hubiese. El apilado de alertas no está contemplado, puesto que pueden provenir de diferentes acciones de usuarios y no hay una política de requerimiento definida en estos casos.
- si el sistema experimenta una interrupción no hay un mecanismo definido para apagar el *Módulo Digital* y es requerida la acción humana.

24. Limitaciones del sistema

Existen limitaciones de diversos tipos:

- Plataforma
  - Driver y Dll USB disponible para windows únicamente.
  - Administración de filesystem restrictiva.
  - Soporte librerías de matemática y gráficos limitada.
  - Celery y RabbitMQ no son soportados en Windows.[16]
- Módulo Digital
  - Requiere la implementación de un manager para su utilización.
- Server
  - Gestión del proceso del servidor requiere un sysadmin siempre.
  - Gestión de usuarios es manual vía interfaz Django, no auto administrable.
  - Seguimiento de recursos de sistema es manual, no hay política de balance de carga.
- Interfaz Gráfica
  - Jquery no es soportado de manera directa por ReactJS[7] .

25. Mejoras a futuro

- Módulo digital
  - Interfaces de servicio.
  - Independencia como programa de consola.
  - Test automáticos.
  - Integración continua.
- Server
  - Cola de experimentos.
  - Soporte a otros Módulos Digitales u Aparatos.
  - Post-procesado de datos en tiempo real.
  - Test automáticos.
  - Integración continua.
- Interfaz Gráfica
  - Soporte a teléfonos móviles.
  - Visualización de gráficos.
  - Tests automáticos.
  - Vista de manipulación de resultados.
  - Vista de gestión de perfil.
  - Tests automáticos.
  - Integración continua.

26. Métricas de desarrollo

Se presentan algunas métricas tales como total de líneas de código y commits de los proyectos obtenidos con los comandos:

`$ git ls-files | xargs wc -l , $ git log --oneline | wc -l y $cloc`

respectivamente.

- Módulo Digital (57 Commits)

```
  2 .gitignore
674 LICENSE
133 Microchip/mpusbapi.dll
 84 Microchip/mpusbapi.h
  0 __init__.py
 50 dry_run.py
 55 main.py
 49 main_logger.py
131 source/Usb.py
  0 source/__init__.py
193 source/ad.py
228 source/dds2.py
 27 source/experiment2.py
 17 source/experiment_config.py
 53 source/experiment_data.py
 47 source/experiment_reporter.py
 23 source/experiment_runner.py
271 source/experiment_scanner.py
125 source/experiment_secuence.py
108 source/pp2.py
 59 source/secuence.py
  7 source/setup.py
263 tests.py
2599 total
```

- Interfaz Gráfica (27 Commits)

```
  24 .gitignore
2164 README.md
  24 package.json
  12 public/favicon.ico
  45 public/index.html
  15 public/manifest.json
  35 src/crear_experimento/action_panel/index.js
  34 src/crear_experimento/experiment_admin/index.js
  43 src/crear_experimento/experiment_view/index.css
  33 src/crear_experimento/experiment_view/index.js
```

```
142 src/crear_experimento/experiment_view/modal_generic.js
 82 src/crear_experimento/general_settings/index.js
146 src/crear_experimento/index.js
 39 src/editar_experimento/action_panel/index.js
 33 src/editar_experimento/experiment_admin/index.js
 43 src/editar_experimento/experiment_view/index.css
 34 src/editar_experimento/experiment_view/index.js
142 src/editar_experimento/experiment_view/modal_generic.js
 82 src/editar_experimento/general_settings/index.js
194 src/editar_experimento/index.js
  5 src/index.css
160 src/index.js
 48 src/intermediate_server/data.js
150 src/lista_de_experimentos/action_panel/index.js
 44 src/lista_de_experimentos/action_panel/view.js
 42 src/lista_de_experimentos/index.js
 82 src/lista_de_experimentos/table_experiments/index.js
  7 src/logo.svg
 23 src/monitor_de_estado/index.js
  4 src/notification_area/notification.css
 27 src/notification_area/notification.js
 13 src/panel_general/index.css
107 src/panel_general/index.js
  7 src/panel_general/logo.svg
 10 src/procesamiento_de_datos/index.js
108 src/registerServiceWorker.js
139 src/resultado_de_experimentos/action_panel/index.js
 44 src/resultado_de_experimentos/action_panel/view.js
 42 src/resultado_de_experimentos/index.js
 82 src/resultado_de_experimentos/table_experiments/index.js
 64 src/sesion_de_inicio/index.js
 39 src/ver_experimento/action_panel/index.js
 33 src/ver_experimento/experiment_admin/index.js
 43 src/ver_experimento/experiment_view/index.css
 34 src/ver_experimento/experiment_view/index.js
140 src/ver_experimento/experiment_view/modal_generic.js
 82 src/ver_experimento/general_settings/index.js
193 src/ver_experimento/index.js
 39 src/ver_resultado/action_panel/index.js
 33 src/ver_resultado/experiment_admin/index.js
 43 src/ver_resultado/experiment_view/index.css
 34 src/ver_resultado/experiment_view/index.js
140 src/ver_resultado/experiment_view/modal_generic.js
 82 src/ver_resultado/general_settings/index.js
193 src/ver_resultado/index.js
 74 src/vista_parcial/index.js
```

```
5776 total
```

- Servidor (33 commits):

```
   0 experiments/__init__.py
   6 experiments/admin.py
   8 experiments/apps.py
  49 experiments/experiment.py
  45 experiments/manager_thread.py
   0 experiments/migrations/__init__.py
  34 experiments/models.py
 195 experiments/result.py
 102 experiments/result_manager_thread.py
   6 experiments/tests.py
  13 experiments/urls.py
 139 experiments/views.py
   0 login/__init__.py
   6 login/admin.py
   8 login/apps.py
   0 login/migrations/__init__.py
   6 login/models.py
   6 login/tests.py
   8 login/urls.py
  28 login/views.py
  22 manage.py
 853 total
```

A continuación las métricas provistas por el comando «cloc» sobre el repositorio del *Servidor* el cual contiene por integración al *Módulo Digital* y a la *Interfaz Gráfica*.

```
  98 text files.
  81 unique files.
  33 files ignored.

github.com/AlDanial/cloc v 1.74  T=0.46 s (162.3 files/s, 16302.6 lines/s)
-------------------------------------------------------------------------------
Language          files          blank        comment           code
-------------------------------------------------------------------------------
JavaScript           30            302             54           2226
Python               35            460            234           1860
Markdown              2            676              0           1490
CSS                   4              8              0             57
JSON                  2              0              0             39
C/C++ Header          1             15             38             31
HTML                  1              3             21             21
-------------------------------------------------------------------------------
SUM:                 75           1464            347           5724
-------------------------------------------------------------------------------
```

27. Conclusión

Dado el desarrollo realizado, los problemas encontrados y los objetivos finales en relación al control del módulo digital con la PC via un servidor web, podemos concluir que:

- El sistema es capaz del controlar remotamente el *Módulo Digital*.
- La administración de experimentos es efectiva.
- Actualmente la implementación se ejecuta en Windows pero para dar soporte a nuevas funcionalidades tales como graficación el número de bibliotecas de acceso libre predominante da unicamente soporte a Linux.
- El desarrollo esta hecho en Python 2.7 pero la última version de Django 2.x requiere Python 3.x.
- El soporte USB por parte de Microchip es únicamente a Windows, esto es un cuello de botella a resolver por medio del desarrollo de una implementación del driver con soporte a Linux o por medio de un cambio de arquitectura en el sistema desarrollado.

La factibilidad de las 4 primeras propuestas es alta, puesto que el proyecto está definido en 3 partes bien diferenciadas y desacopladas, aunque es posible que sea necesario sumar funcionalidad al *Módulo Digital* para tal fin, puesto que ahora sería un cliente HTTP del *Servidor* y no un subproceso como se lo define en la arquitectura original.

28. Acrónimos

- DDS2: Generador digital de señales.
- AD: Conversor analógico digital.
- PP2: Programador de pulsos digitales.
- RF: Radio frecuencia.
- TTL: Transistor–transistor logic.
- HTTP: Hypertext Transfer Protocol.

29. Anexo A: Comandos del PP2

30. Anexo B: Comandos del DDS2

El DDS2 cuenta con la capacidad para combinar 16 valores de fase y 2 de frecuencias. Los valores de fase ocupan 2 bytes cada una mientras que cada valor de frecuencia 6 bytes. Las fases son almacenadas en una memoria RAM de 32 posiciciones donde el MSB se almacena en las direcciones pares y el LSB en la direccion impar contigua. Las frecuencias se almacenan en registros internos del DDS2 (12 registros en total). Para el almacenamiento de fases y frecuencias hacemos uso del byte de comando el cual es escrito en el registro interno 0x71 del DDS2.

B0	B1	B2	B3	B4	B5	B6	B7
Modo	Carga de fase	Selector de fases	X	X	X	X	X
0 PC 1 DDS2	0 deshabilitada 1 habilitada	0 deshabilitado 1 habilitado	X X	X X	X X	X X	X X

Cuadro 12: Byte de comando.

Para almacenar las fases debemos habilitar el modo PC, carga de fases y deshabilitar el selector fases. Para almacenar frecuencias debemos habilitar el modo PC, carga de fases y deshabilitar el selector de fases.

Una vez terminada la carga de fases y frecuencias debemos habilitar en modo DDS2 para ser manipulado externamente via pulsos.

- comando reset:

  `['a'][0x72][0x00]`

- comando control:

  `['b'][0x71][byte de control]`

- comando almacenamiento de byte de frecuencia:

  `['k'][0x75][registro interno][0x78][byte freq]`

- comando almacenamiento de una fase:

  `['w'][0x70][dirección][0x74][MSB][0x70][dirección+1][0x74][LSB]`

- comando udlck:

  `['u'][0x76][0x00]`

- comando estado:

  `['g'][0x73]`

31. Anexo C: Comandos del AD

El conversor analogico digital tiene 2 modos de trabajo bien definidos, modo PC y modo AD. Los modos son configurados con el bit cero, del byte de control en 0 y 1 respectivamente. En modo PC se pueden realizar 2 acciones principales: la configuracion general y la extraccion de los datos. En modo AD, el conversor se pone en espera del pulso 5 para realizar la conversion de los datos para una posterior extraccion por parte del usuario.

El buffer de adquisicion es variable en bloques de $64 * 16 * 2^i$ bytes con $0 \leq i \leq 7$, la eleccion del bloque se hace con los bits 4,5,6 del byte de control. Se debe reiniciar el contador de direcciones de buffer antes de una conversion a su valor por defecto, esto se hace por medio del bit 7 del btye de control poniendolo sucesivamente en 1 y 0.

Existen 2 formas de adquisicion, por pulsacion y continuo, en este trabajo utilizamos el continuo por medio del bit 1 del byte de control en 0.

Los bits 3 y 4 del byte de control son sin cuidado y su valor es puesto en cero.

Es posible determinar el estado de finalizacion de la conversion por medio del comando de estado, el cual responde con un byte donde el valor del primer bit indicara en cero el fin de la adquisicion.

La configuracion del intervalo de muestreo se hara almacenando en el registro 0xC el byte de represetacion del tiempo entre muestra y muestra.

La obtencion de los datos del buffer de adquisicion requiere el reset del contador (bit 7 en 1 y 0 sucesivamente) antes de la ejecucion del comando de obtencion de datos, el cual tiene como parametro la direccion del buffer y se obtiene como respuesta un arreglo contiguo de bytes acorde al numero de bloques configurado en el registro de control.

- comando de control:

  `['r'][0x0B][byte de control]`

- comando de estado:

  `['S'][0x0B][intervalo]`

- comando de configuracion de intervalo:

  `['t'][0x0C][byte ts]`

- comando de extraccion de adquisisciones:

  `['B'][0xO9|0x0A|0x0B]`

Referencias

[1] JSON notation -https://stackoverflow.com/questions/35709595/why-would-you-use-a-string-in-json-to-represent-a-decimal-number

[2] Introducing JSON - https://www.json.org

[3] Writing your first Django app - https://docs.djangoproject.com/en/1.11/intro/tutorial01/

[4] Cross Site Request Forgery protection - https://docs.djangoproject.com/en/1.11/ref/csrf/

[5] Using cookie-based sessions - https://docs.djangoproject.com/en/1.11/topics/http/sessions/#cookie-session-backend

[6] Create React App - https://github.com/facebook/create-react-app

[7] ReactJS JQuery Integration - https://reactjs.org/docs/integrating-with-other-libraries.html#integrating-with-dom-manipulation-plugins

[8] Git Submodules basic explanation - https://gist.github.com/gitaarik/8735255

[9] Components and Props - https://reactjs.org/docs/components-and-props.html

[10] State and Lifecycle - https://reactjs.org/docs/state-and-lifecycle.html

[11] Lifting State Up - https://reactjs.org/docs/lifting-state-up.html

[12] Thinking in React - https://reactjs.org/docs/thinking-in-react.html

[13] React Router DOM - https://github.com/ReactTraining/react-router

[14] AxiosJS - https://github.com/axios/axios

[15] GitHub Home - https://github.com

[16] Does Celery support Windows? - http://docs.celeryproject.org/en/latest/faq.html#does-celery-support-windows

«Los abajo firmantes, miembros del Tribunal de Evaluación de tesis, damos Fe que el presente ejemplar impreso, se corresponde con el aprobado por éste Tribunal»