Comunicación USB para aplicaciones de control con Matlab® / Simulink® y placa ESD-32U

Alfredo Ernesto Puglesi[1], Guido Marcucchi, Pablo Vega, María Susana Bernasconi[1]

[1]Facultad de Ingeniería - Universidad Nacional de Cuyo, Mendoza, Argentina

{apuglesi,sbenasc}@uncu.edu.ar

Abstract. En el Laboratorio de Control de la Facultad de Ingeniería de la UN-Cuyo ha sido de gran ayuda para investigación y desarrollo, el uso de placas de interfaz, aptas para aplicaciones de control en tiempo real. Poseen una sección digital de 8 entradas y 8 salidas, mientras que en la sección analógica poseen 8 entradas y 2 salidas de 8 bits de resolución. Estas placas conseguían su comunicación con la PC a través de un puerto ISA, obsoletos al día de hoy. Por lo que la propuesta fue utilizar placas nacionales de bajo costo con comunicación USB, desarrollando el software de interfaz entre las mismas con los programas MATLAB® / Simulink® que corriendo en tiempo real controlaran plantas pilotos, máquinas y robots, como sus antecesoras conectadas a la PC vía bus ISA. En este trabajo se presentan los principales resultados de lo realizado hasta el momento.

Keywords: control automático, tiempo real, software de interfaz

1 Introducción

Este trabajo describe como se realizaron los programas de comunicación con la placa ESD-32Uy su extensión ESA-U, de fabricación nacional y de bajo costo, para implementar lazos de control en un entorno de Matlab® / Simulink®.

La necesidad surge ante la imposibilidad de adquirir hardware y software que la opción comercial ofrece para trabajar en tiempo real (Real Time Windows Target, RTWT®) por ser de importación y por ende muy costosa, además que dicha aplicación está destinada sólo a determinadas marcas de placas en convenio comercial con Mathworks Inc.™ [1].

Se comenzara a explicar las funcionalidades y características de la placa ESD-32U y su complemento ESA-U. Luego se mostraran las bibliotecas elegidas para manejar

el puerto USB, en Windows 7 con posible extensión a Linux, mencionando los programas que se desarrollaron con el fin de leer y escribir sobre la placa ESD-32U. Más adelante se dará una breve introducción a las MEX-files, para poder entender los programas finales producidos para Matlab® / Simulink®. Finalmente, se expondrá una aplicación de control sobre una planta piloto utilizando las herramientas desarrolladas como producto de este proyecto.

1.1 Placa ESD-32U y su complemento ESA-U

La combinación (Figura1) de la placa ESD-32U y su extensión ESA-U ofrece 40 entradas y 40 salidas digitales, más 8 entradas y 2 salidas analógicas de 12 bits de resolución con conexión USB a PC. El manejo de las señales, entonces, se hace a través del puerto USB de la PC. Esta particularidad es aprovechada para poder implementar control en tiempo real usando Matlab® / Simulink®. Las señales que maneja la tarjeta de interfaz tienen niveles TTL, al igual que los niveles de las señales dentro de la PC.

Al utilizar el puerto USB, la placa es compatible con la gran mayoría de PC´s y Notebooks disponibles hoy en día. Además es posible conectar más de una placa ESD-32U y sus correspondientes extensiones, ya que se puede seleccionarse cada placa con un número de identificación (handler).

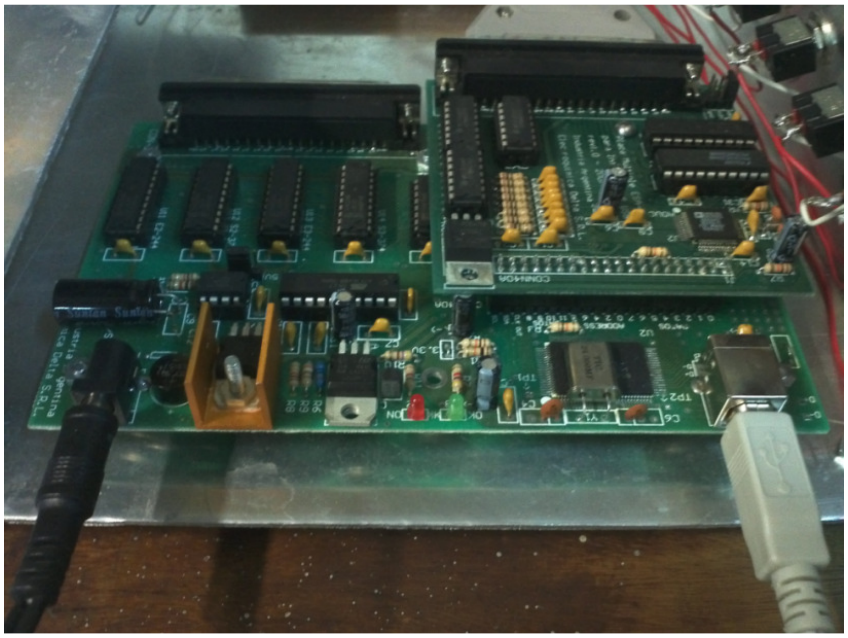

Fig.1. Placa ESD-32U y su extensión ESA-U montada sobre la primera

1.2 Configuración del hardware

Dada la naturaleza de la aplicación, con entradas y salidas analógicas, la planta piloto ensayada se conectó a la placa de extensión ESA-U. La Figura 2, hace énfasis en la configuración de la misma, se aclara que si bien puede manejar 16 entradas / salidas discretas, no se usaron en este caso. La placa ESA-U se conecta a la placa ESD-32U, que oficia de placa madre, y de puente con la PC vía conexión USB, además de proveer la alimentación para ambas.

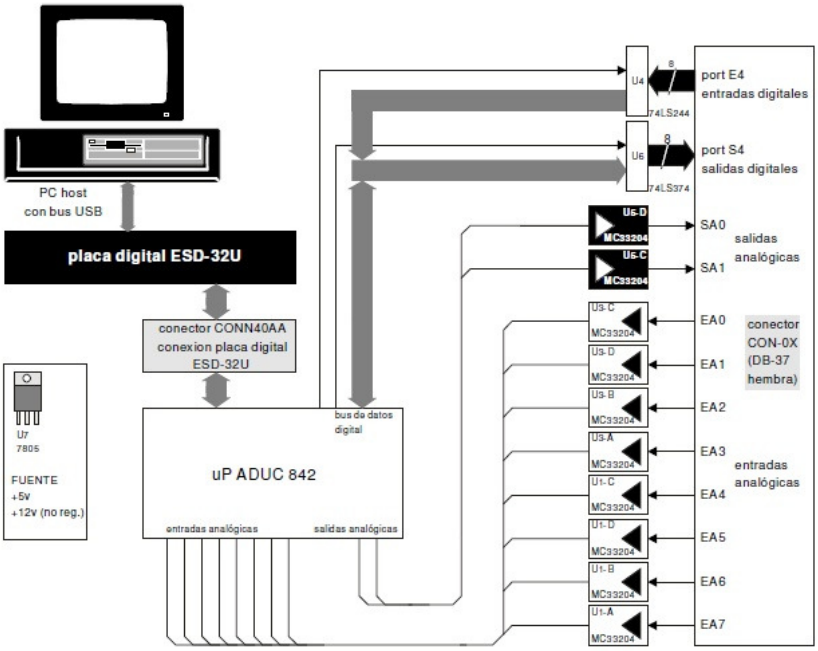

Fig.2. Configuración del hardware, destacando las entradas / salidas de la placa ESA-U

La conexión en la ESA-U desde y hacia campo (planta, máquina, robot), tanto digitales como analógicas se realiza mediante un conector tipo DB-37, siendo la distribución de pines, la mostrada en la Figura 3.

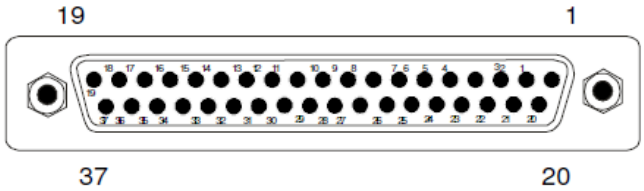

Fig. 3. Conector DB-37

2 Requerimientos del Sistema y Drivers

2.1 Sistema Operativo

Los sistemas operativos soportados por los drivers originales son los siguientes: Windows 98SE, Windows Me, Windows 2000, Windows XP SP1, Windows XP SP2, Windows Server 2003. En este caso se amplió el uso de la placa a otros sistemas operativos más actuales, estos deben soportar HID 1.1 o superior. El HID es un tipo de protocolo de comunicación USB frecuentemente usado para ratón, teclado o joystick, que son dispositivos de interface humana.

2.2 Drivers

La placa se conecta como un dispositivo HID 1.1 al sistema. En el CD que viene con ella se provee una biblioteca llamada "edeltausb.dll" que maneja la conexión entre el driver y el sistema tipo Windows, pero la misma no es adecuada para sistemas operativos más modernos.

Para que se adapte a otros sistemas operativos se tuvo que manejar el HID directamente, para ello se utilizó el paquete de bibliotecas "hidapi" que permitieron escribir los programas de comunicación HID con relativa facilidad, además de ser open source. Para este proyecto se descargó la biblioteca hidapi de:

https://github.com/signal11/hidapi/downloads

2.3 Clase HID - Introducción

El USB es una arquitectura de comunicaciones que le da a la PC la capacidad de conectarse con una variedad de dispositivos utilizando solo un cable de cuatro líneas. El USB, de hecho, establece la comunicación mediante dos líneas a 1.5 Mb/s o 12 Mb/s. El protocolo USB puede configurar dispositivos en el arranque del sistema o cuando son conectados cuando el mismo se encuentra funcionando. Estos dispositivos pueden clasificarse en clases. Cada clase define un comportamiento común, así como protocolos para dispositivos con funciones similares. En nuestro caso particular, será una clase para placas de entradas y salidas de señales digitales y analógicas. Los objetivos principales de la clase HID son los siguientes:

- Ser tan compacta como sea posible a fin de ahorrar memoria en el dispositivo.
- Permitir al software ignorar información desconocida.
- Ser extensible y robusta.
- Soportar anidamiento y colecciones.

2.4 Report Descriptor del Dispositivo

El dispositivo envía a la PC un Report Descriptor que contiene información sobre el mismo. Para poder realizar la comunicación con la placa de adquisición de datos

ESD-U y su ampliación ESA-U, esta información es esencial ya que sin ella no se podría saber la cantidad de datos que son enviados por el dispositivo cuando realiza la adquisición de datos, ni tampoco la forma en la que se debe escribir en los puertos de salida.

Una vez que se conoce el tamaño de la información enviada y recibida por la placa ESA-U, se requiere un trabajo de ingeniería inversa. Esto es así dado que si bien la cantidad de bytes enviados y recibidos es conocida, no se tiene información precisa de la manera en la que cada byte de información se relaciona con los puertos de entrada y salida de la placa.

Para obtener el Report Descriptor existen varios métodos, por supuesto el más apropiado es buscar en el manual del dispositivo, pero dado que dicha información no está disponible se utilizo un método alternativo. Precisamente, para conseguir la información del dispositivo se utilizaron comandos desde la consola, accediendo así a casi toda la información necesaria sobre la placa de adquisición de datos para poder utilizarla desde la PC.

Del device descriptor encontrado de esta manera se puede recopilar información sobre el número de identificación el fabricante, del producto (Vendor ID y Product ID), el largo en bytes de la información transmitida al leer el dispositivo y el largo en bytes de la información enviada al escribir (Report Count), todo esto junto con otros datos del dispositivo.

2.5 Funciones de prueba

La biblioteca HIDAPI es multiplataforma lo cual da la posibilidad de emplearla para este proyecto en Windows 7 (64 y 32 bit) y si fuere necesario en Linux (Ubuntu 64 bit) sin muchos inconvenientes. Sabiendo el tamaño de los buffers de entrada y salida (31 y 20 bytes, respectivamente), se utilizó un programa de prueba para encontrar la correspondencia entre los bytes de los buffers y de la placa. Por supuesto, esta información debe ser provista por el fabricante pero si no fuera así, como ya se dijo, es necesario un trabajo de ingeniería inversa. Luego de un proceso de prueba y error, se encontraron la forma en la que los buffers deben ser escritos o leídos. Para el buffer de escritura, la estructura es como se destaca en la Tabla1:

Tabla 1

Posición en buffer	**Función**
Buff[0]	Report ID, si hay un solo report el Report ID debe ser 0x00
Buff[1]	Byte de control, dejar en 0x00 para funcionamiento normal
Buff[2]	Byte de control, dejar en 0x00 para funcionamiento normal
Buff[3]	Byte de control, dejar en 0x00 para funcionamiento normal
Buff[4]	Byte de control, dejar en 0x00 para funcionamiento normal
Buff[5]	Byte de control, función “hold”. Dejar en 0x00 para funcionamiento normal
Buff[6]	Byte de control, función “hold”. Dejar en 0x00 para funcionamiento normal
Buff[7]	Byte de control, función “hold”. Dejar en 0x00 para funcionamiento

	normal
Buff[8]	Byte de control, función “hold”. Dejar en 0x00 para funcionamiento normal
Buff[9]	Salida digital S0
Buff[10]	Salida digital S1
Buff[11]	Salida digital S2
Buff[12]	Salida digital S3
Buff[13]	Byte de control de la salida S4. Dejar en 0xff para escribir
Buff[14]	Salida digital S4
Buff[15]	Byte de control de la salida S4. Dejar en 0xff para poder escribir
Buff[16]	Permite o no, escribir en la salida analógica S0 y S1 (valores posibles: 00, 01, 10,11). Dejar en 0x03(11 en binario), para poder escribir
Buff[17]	Salida analógica S0(primer byte)
Buff[18]	Salida analógica S0(medio byte final)
Buff[19]	Salida analógica S1(primer byte)
Buff[20]	Salida analógica S1(medio byte final)

Ahora la Tabla 2 muestra la estructura de la escritura:

Tabla 2

Posición del buffer	**Descripción**
Buffer[0]	Byte con los estados de los 8 bits de la entrada digital 0
Buffer[1]	Byte con los estados de los 8 bits de la entrada digital 1
Buffer[2]	Byte con los estados de los 8 bits de la entrada digital 2
Buffer[3]	Byte con los estados de los 8 bits de la entrada digital 3
Buffer[4]	Byte con los estados de los 8 bits de la salida digital 0
Buffer[5]	Byte con los estados de los 8 bits de la salida digital 1
Buffer[6]	Byte con los estados de los 8 bits de la salida digital 2
Buffer[7]	Byte con los estados de los 8 bits de la salida digital 3
Buffer[8]	Byte con los estados de los 8 bits de la entrada digital 4
Buffer[9]	Byte con los estados de los 8 bits de la salida digital 4
Buffer[10]	Primer byte con el valor de la entrada analógica 0
Buffer[11]	Segundo byte con el valor de la entrada analógica 0 (primeros 4 bits)
Buffer[12]	Primer byte con el valor de la entrada analógica 1
Buffer[13]	Segundo byte con el valor de la entrada analógica 1 (primeros 4 bits)
Buffer[14]	Primer byte con el valor de la entrada analógica 2
Buffer[15]	Segundo byte con el valor de la entrada analógica 2 (primeros 4 bits)
Buffer[16]	Primer byte con el valor de la entrada analógica 3
Buffer[17]	Segundo byte con el valor de la entrada analógica 3 (primeros 4 bits)
Buffer[18]	Primer byte con el valor de la entrada analógica 4
Buffer[19]	Segundo byte con el valor de la entrada analógica 4 (primeros 4 bits)
Buffer[20]	Primer byte con el valor de la entrada analógica 5
Buffer[21]	Segundo byte con el valor de la entrada analógica 5 (primeros 4 bits)
Buffer[22]	Primer byte con el valor de la entrada analógica 6
Buffer[23]	Segundo byte con el valor de la entrada analógica 6 (primeros 4 bits)
Buffer[24]	Primer byte con el valor de la entrada analógica 7
Buffer[25]	Segundo byte con el valor de la entrada analógica 7 (primeros 4 bits)
Buffer[26]	Primer byte con el valor de la salida analógica 0

Buffer[27]	Segundo byte con el valor de la salida analógica 0 (primeros 4 bits)
Buffer[28]	Primer byte con el valor de la salida analógica 1
Buffer[29]	Segundo byte con el valor de la salida analógica 1 (primeros 4 bits)

3 Matlab® – Introducción

Con el fin de poder utilizar diferentes herramientas de control a través de la placa ESD-32U y su ampliación ESA-U, se implementó un programa en Matlab® basado en la biblioteca "Hidapi". Luego este programa es integrado en un bloque de Simulink® para formar parte de un lazo de control entre una variadísima gama de estrategias de control posibles y que este potente software dispone.

3.1 MEX-files

Es posible llamar desde MATLAB® a funciones programadas en C, C++ y en Fortran como si fueran funciones propias de MATLAB®. Los programas en C y Fortran que se pueden llamar desde MATLAB® son las MEX-files o archivos MEX. Los MEX-files son subrutinas ligadas dinámicamente que el interpretador de MATLAB® puede automáticamente cargar y ejecutar. MEX proviene de las palabras "Matlab® EXecutable" [2]. Las funciones MEX tienen varias aplicaciones:

- Evitan tener que rescribir en MATLAB® funciones que ya han sido escritas en C o Fortran.
- Por motivos de eficiencia puede ser interesante rescribir en C o Fortran las funciones críticas de una aplicación determinada.

En particular, se necesita un compilador soportado por MATLAB®, en este caso se utilizó Microsoft Visual C++ 2010 Express o Microsoft Visual Studio 2010 [3]. También es necesario tener el paquete SDK (Software Development Kit) de Microsoft correspondiente. Se puede buscar la lista de compiladores soportados y compatibles en la página web de Mathworks Inc.™, que dependen también de que versión de MATLAB® se esté usando, en este caso la R2010a. Si se tiene hecho un programa independiente en C, se debe usar una función main(). MATLAB® se comunica con la MEX-file usando una gateway routine (rutina de puerta de entrada). Por otro lado, la función de MATLAB® que crea una rutina se llama mexfunction. MATLAB® guarda los arreglos en una variable de tipo mxArray. Se utiliza mxArray en el programa en C, para pasar datos de MATLAB® hacia y desde la MEX-file. El documento MATLAB® C and Fortran API [4], es la referencia de las funciones para trabajar con los mxArray. Esta API tiene varias bibliotecas. Las funciones en la biblioteca mx crean y manipulan MATLAB® arrays. Estas funciones pertenecen a la categoría denominada MX Array Manipulation. Entonces, de manera general, para crear un binary MEX-file, se necesita:

- Ensamblar las funciones del proyecto y las funciones de MATLAB® API en uno o más archivos fuentes en C.
- Escribir una *gateway funtion* en uno de los archivos fuentes.
- Usar la función mex de MATLAB®, que es una secuencia de comandos de compilación, para compilar una binary MEX-file.
- Usar la binary MEX-file en MATLAB® como cualquier otra M-file o función, producto final de este Proyecto.

3.2 Implementación en MATLAB®

Utilizando las capacidades de las funciones MEX de Matlab® como una interfaz con el par de placas ESD-32U y ESA-U, las capacidades de estas últimas se ven ampliadas, dado que permite llevar a cabo el control de procesos en tiempo real implementando algoritmos complejos.

El programa utilizado está escrito en C++, usa la función Gateway de MATLAB® y está dividido en cuatro funciones básicas: Leer puertos analógicos, leer puertos digitales, escribir puertos analógicos y escribir puertos digitales. Se puede acceder a cada una de estas funciones desde el espacio de trabajo o "workspace" de MATLAB® mediante los siguientes comandos:

- Leer puertos analógicos: EDHID("read", "puerto"), en donde "puerto" hace referencia al puerto que se quiere leer. Los puertos analógicos que pueden leerse son: AE0, AE1, AE2, AE3, AE4, AE5, AE6, AE7, AS0 y AS1. La salida de la función es el valor en milivoltios, en una rango de 0 a 5000 mV, existente en el puerto.
- Leer puertos digitales: EDHID("read", "puerto" ,bit), en donde "puerto hace referencia al puerto que se quiere leer. Los puertos digitales que pueden ser leídos son: DE0, DE1, DE2, DE3, DE4, DS0, DS1, DS2, DS3 y DS4. El parámetro bit sirve para indicar el bit específico del puerto que se quiere leer, dado que cada puerto tiene 8 bits, el numero *bit* puede ir desde el 0 al 7. La salida de la función es el estado del bit (1 o 0)
- Escribir puertos analógicos: EDHID("write", "puerto", valor), en donde "puerto" hace referencia al puerto que se quiere escribir. Los puertos que pueden ser escritos son: AS0 y AS1. El parámetro "*valor"* sirve para indicar el valor que se quiere escribir en la salida, en milivoltios, con un valor máximo 5000 mV y un valor mínimo de 0 mV. La función no entrega ningún dato como salida.
- Escribir puertos digitales: EDHID("write","puerto", bit, valor), en donde "puerto" hace referencia al puerto que se quiere escribir. Los puertos que pueden ser escritos son: DS0, DS1, DS2, DS3, DS4. El parámetro *bit* hace referencia al bit específico que se quiere escribir, el numero *bit* puede ir desde 0 a 7. Finalmente, el parámetro *valor* se refiere al estado que se desea escribir: 0 o 1.

Razones de espacio limitan una mayor descripción de lo realizado sobre este particular, no obstante mayores detalles se encuentran en [2], Capítulo 4 "Creating C/C++ Language MEX-Files".

4 Aplicación de control en tiempo real con Simulink®

4.1 Planta Piloto a controlar

Para probar el programa en un lazo de control se utilizó la versión MEX-file EDHID en Simulink® sobre una Planta para el control de nivel de una columna, disponible en el Laboratorio de Control, Figura 4.

Fig. 4. Planta Piloto para el control del nivel.

Esta planta consiste en dos recipientes y dos bombas, una para llenar y otra para vaciar el recipiente controlado (derecha). Cuenta con un indicador de caudal y un controlador, afectado al nivel que ejecuta el conocido algoritmo Proporcional + Integral + Derivativo (PID), ambos instrumentos basados en µP y de factura comercial. El elemento de acción final del lazo es una válvula de control tipo globo (modulante); las señales de caudal y de nivel provienen de sendos transmisores con salida en 4- 20 mA.

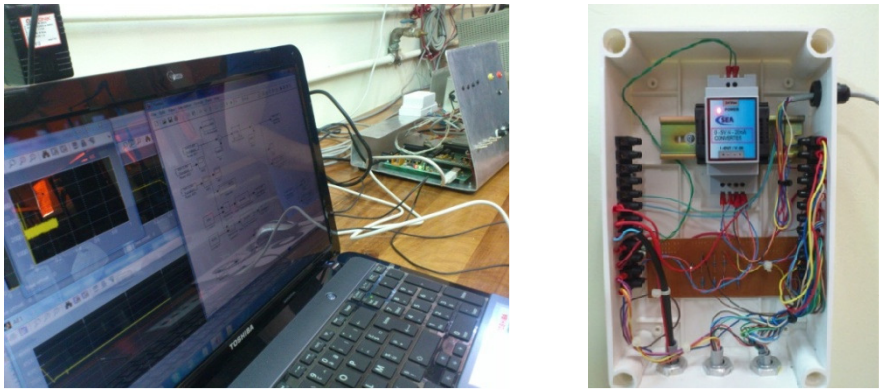

Fig 5. Notebook, placas ESD-32U y ESA-U y acondiconador de señales

A su vez la Figura 5 muestra a la izquierda una notebook y hacia atrás las placa ESD-32U y su extensión ESA-U, a la derecha el acondicionador de señales para com-

patibilizar los niveles TTL de las placas con el régimen en 4-20 mA de entradas / salidas de la planta. La Figura 6 corresponde al control implementado mediante diagramas de bloques en el entorno Simulink®, que dispone de vastas utilidades para cambiar y desarrollar variadas estrategias de control, como las que se mencionan, en otras, en las Conclusiones. Siendo así, una excelente herramienta para aplicaciones de control en tiempo real, y más factible, como ya se apuntó, que la opción comercial de Mathworks Inc.

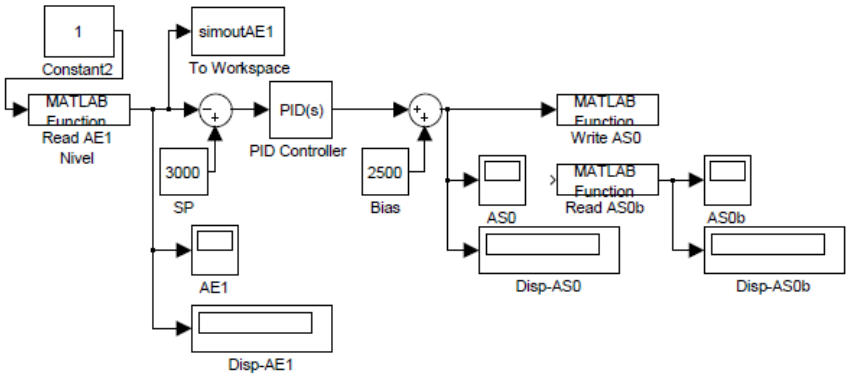

Fig. 6. Diagrama en Simulink®

Como se ve, se usa un control tipo PID, que está dentro de las funciones de biblioteca de Simulink®, la Figura 7 muestra el bloque con los valores de sintonía usados.

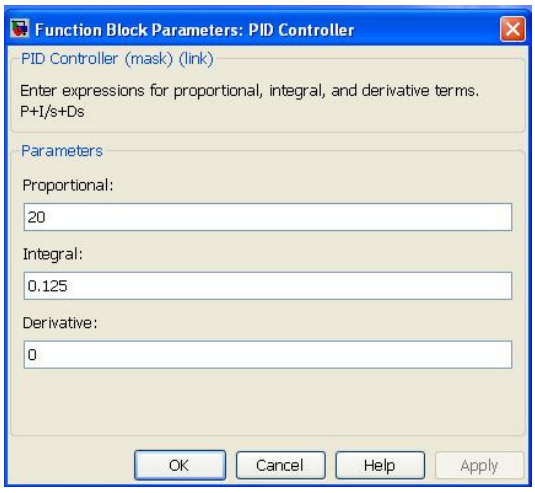

Fig. 7 – Controlador PID - Parámetros

Notar que la entrada es el bloque denominado "MATLAB® Function - Read AE1Nivel", que lee el puerto analógico de la entrada 1 que está conectado al transmisor de nivel del recipiente derecho. Esta lectura va al espacio de trabajo de

MATLAB® por medio del bloque “simoutAE1 - To workspace” que genera un archivo con los valores del nivel en toda la corrida, además es graficada en un “Scope” para observar en línea su evolución y por supuesto esta señal también se dirige hacia el bloque de control PID luego de restarse con el valor deseado o set point, que es el bloque identificado como “SP”, siendo importante destacar, que se puede cambiar en línea con el programa ejecutándose, al igual que en los controladores comerciales.

A su vez, la salida del “PID controller”, escribe en el bloque “MATLAB® Function – Write AS0” y de allí a la válvula de control. Esta secuencia se ejecuta de manera iterativa hasta finalizar el tiempo de ejecución en tiempo real, que se puede especificar a discreción.

A continuación y través de la Figura 8, se pueden ver las funciones de lectura y escritura en los bloques de “MATLAB® Function”

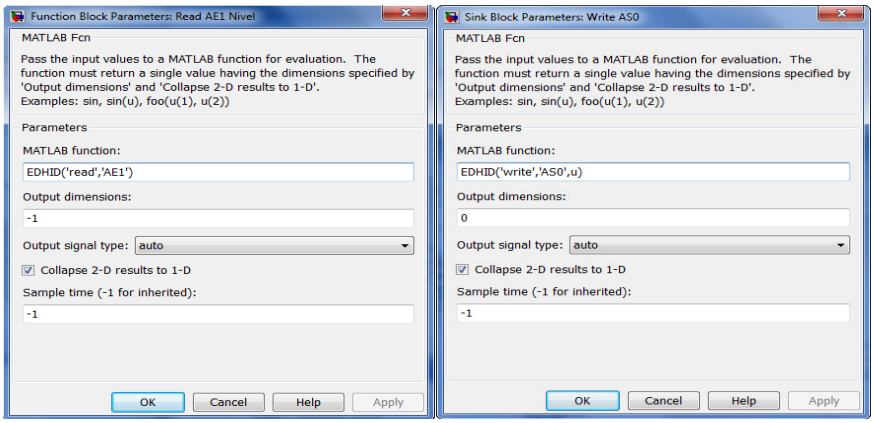

Fig. 8. Bloque de lectura a través del puerto AE1 y de escritura con el puerto AS0.

Basta especificar los parámetros para que realice la lectura del transmisor de nivel a través del 'read' para indicar lectura y 'AE1' para indicar que puerto se debe leer. En forma similar para enviar una salida a la válvula de control mediante 'write' sobre el puerto 'AS0'.

4.2 Resultados

La Figura 9 expone una corrida en tiempo real de 9 minutos, observándose un excelente desempeño, equivalente a los controladores basados en µP y de factura comercial, esto es así si lo comparamos con el desempeño del controlador Yokogawa® que como ya se dijo, puede controlar en forma alternativa la misma planta . Los valores están en el rango de 0 a 5000 mV equivalentes a una variación del nivel del 0 al 100%.

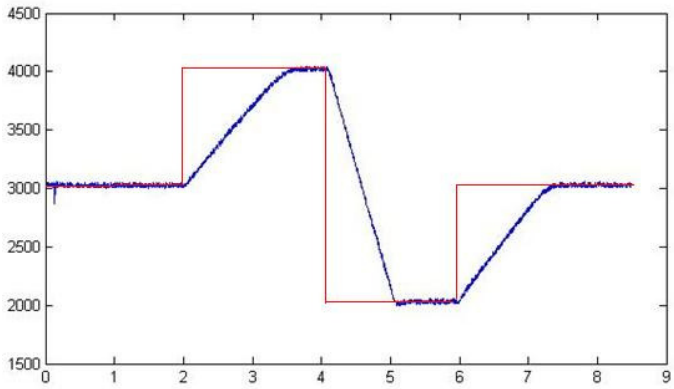

Fig.9. Corrida de 9 minutos – en rojo el SP y en azul el nivel controlado

Se observa que el sistema tiene una buena respuesta al control PID, con un error de estado estacionario aceptable de 50 mV (1% del total del nivel).

5 Conclusiones

Al momento, por falta de tiempo, no se ha determinado la magnitud del retardo en la lectura y escritura de datos, que conllevan la medición del tiempo de latencia y manejo de interrupciones. Se sabe por los resultados obtenidos con este trabajo, que el desempeño es aceptable para el control de procesos capacitivos, y por lo tanto lentos, pero también se sabe que el cuello de botella estará sobre el control de servomecanismos y por lo tanto será imprescindible realizar las mediciones apuntadas y aconsejadas, para saber hasta dónde llegan las prestaciones del desarrollo en tiempo real. Finalmente lo presentado es una de las tantas posibilidades que ofrece este Proyecto, ya que Simulink® permite implementar otras estrategias de control, como y a manera de ejemplo: controladores difusos entrenados por redes neuronales artificiales, tanto para control realimentado como de acción precalculada [5]. La diversidad de opciones es muy amplia y hacen posible la investigación y desarrollos de control en tiempo real sobre plantas y equipos, la mayoría de las veces no disponibles a nivel comercial.

Referencias

1. http://www.mathworks.com/products/rtwt/supported/index.html
2. The MathWorks, Inc ™:MATLAB® External Interfaces – R2012a, (2012)
3. Arrioja Landa Cosio, N.: C# - Guía Total del Programador, Manuales User, (2012)
4. The MathWorks, Inc ™:MATLAB® C and Fortran API Reference (2009)
5. Puglesi, A., Bernasconi, M.S.: Control de acción precalculada con Lógica Difusa. XIV Reunión de Trabajo en Procesamiento de la Información y Control – RPIC 2011- Universidad Nacional de Entre Ríos. Oro Verde, Entre Ríos, Argentina (2011)