UNIVERSITAT JAUME I

ESCOLA SUPERIOR DE TECNOLOGIA I CIÈNCIES EXPERIMENTALS

MASTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

Implementación y evaluación de algoritmos de control basados en eventos en el estándar de programación de control distribuido IEC-61499

TRABAJO FIN DE MASTER

AUTOR/A:

Oscar Miguel Escrig

DIRECTOR/A:

Julio Ariel Romero Pérez

Agradecimientos

En primer lugar, me gustaría expresar mi gratitud a mi familia por todo el apoyo que me han brindado siempre en cualquier situación. En particular, a mis padres, por, entre otras muchas cosas, esperar siempre mí llamada por Skype durante mi etapa de estudios en Francia.

En segundo lugar, me gustaría dar las gracias a mis tutores del proyecto, Julio Romero y Julien Roux, y a mi tutor oficioso Esteban Querol, por la paciencia que han tenido conmigo y por todos los conocimientos que me han aportado a todos los niveles, no solo el puramente académico.

Agradecer también a todos mis amigos por estar siempre a mi lado. En particular a Cristian Ferrer, por la voluminosa ayuda que me ha brindado.

En general, me gustaría agradecer a cada una de las personas que me he cruzado a lo largo de mi vida universitaria, a cada compañero y a cada profesor, por todas las experiencias que he compartido con ellos, porque todas ellas me han llevado hasta donde estoy ahora.

Contenido

1. Memoria

1.1 Objeto

El objetivo de este proyecto de fin de máster es realizar un estudio sobre la implementación de algoritmos de control basado en eventos (a partir de ahora se nombrará por sus siglas, CBE) usando el estándar IEC-61499 con el objetivo de acercar las prestaciones de este tipo de control al control periódico clásico. Con este estudio se pretende obtener algoritmos de CBE más eficientes desde el punto de vista del coste computacional al aprovechar la sinergia de estos con el estándar IEC-61499 para constituir una alternativa sólida a los controladores periódicos y a la implementación clásica de estos.

Este proyecto forma parte del proyecto «DESARROLLO DE ALGORITMOS DE CONTROL PARA SISTEMAS DISTRIBUIDOS IMPLEMENTABLES MEDIANTE ESTÁNDARES INDUSTRIALES» con referencia 15I339 inscrito dentro del Departamento de Ingeniería de Sistemas Industriales y Diseño de la Universitat Jaume I.

Dentro del uso de estándares industriales para el control, evaluaremos e implementaremos controladores basados en eventos con el estándar IEC-61499 desarrollando para ello los circuitos y programas necesarios para el control

1.2 Alcance

No se contempla ni la creación, modificación y validación del lenguaje de modelado, el IEC-61499, necesario para crear la arquitectura de control; ni la creación de las herramientas de software de desarrollo específicas, necesarias para el desarrollo de los experimentos a realizar. A tales efectos se usaran herramientas previamente desarrollados por terceros. Aunque sí se contempla el desarrollo de los elementos de software, desarrollados con las herramientas anteriormente especificadas.

Las arquitecturas de control propuestas no deben tomarse como una metodología estándar de modelización e implementación de estructuras de control bajo la norma IEC-61499 ya que, en primer lugar, si bien se ha utilizado el estándar para el desarrollo de las mismas, el propio estándar no provee de una metodología de implementación, y en segundo lugar, debido a la multitud de casos particulares que pueden darse en los sistemas a controlar y unido a la multitud de terminales que se pueden utilizar para las aplicaciones de esta norma, las modelizaciones desarrolladas no son únicas. Pero sin embargo, pueden ser consideradas como arquitecturas generales para llevar a cabo este tipo de control.

El proyecto contempla la validación de las arquitecturas implementadas, de los sistemas desarrollados y de los controladores evaluados; todos ellos en función de las prestaciones en el control que demuestren, pero no se tendrá en cuenta el consumo energético global ni el de los sistemas a controlar aunque este consumo sea uno de los focos de investigación cuando los algoritmos de CBE están involucrados.

1.3 Antecedentes

1.3.1 Contexto académico del CBE y de la norma IEC-61499 en la actualidad

En la última década se han realizado numerosa investigaciones sobre el control basados en eventos de sistemas continuos. A diferencia de los controladores basados en tiempo usados tradicionalmente, en los que la ejecución del algoritmo de control se realiza a un periodo constante, en los controladores basados en eventos el algoritmo de control se ejecuta sólo tras la ocurrencia de eventos asíncronos que indican cambios significativos en el estado del sistema; un ejemplo típico es el cruce de niveles por parte de la señal de error (diferencia entre la referencia o *set-point* y la salida controlada), (Dormido, et al., 2008). El CBE tiene dos objetivos fundamentales, en primer lugar, reducir la cantidad de información necesaria para realizar el control en bucle cerrado, y en segundo lugar, disminuir el coste computacional medio que requieren los algoritmos de control. Estas características han hecho que los algoritmos de CBE se apliquen cada vez más en los sistemas de control distribuidos para reducir el tráfico de información a través de redes de comunicaciones industriales, por ejemplo entre los nodos sensores, controladores y actuadores. La reducción del tráfico en las redes industriales disminuye la probabilidad de la pérdida o retraso en los datos y por consiguiente mejora el comportamiento general del sistema (Gupta & Chow, 2010). Distintas estrategias de control han sido adaptadas al CBE, incluido los controladores PID, ampliamente usados en la industria, los cuales cuentan actualmente con varias versiones y métodos de ajustes adecuados al CBE como los propuestos en los siguientes artículos: (Årzén, 1999), (Durand & Marchand, 2009b), (Vasyutynskyy & Kabitzsch, 2010b), (Beschi & Visioli, 2013), (Romero, et al., 2011), (Romero, et al., 2012), (Romero, et al., 2014a) y (Romero & Sanchis, 2016a).

A la par de los avances teóricos en el CBE, durante los últimos años se ha estado desarrollando un nuevo estándar para la programación de sistemas de automatización y control distribuidos, conocido como IEC-61499 desarrollados y detallados en: (Zoitl & Lewis, 2013), (IEC, 2012) y (IEC, 2013). Dicho estándar introduce conceptos novedosos respecto a su predecesor, el IEC-61131 (John & Tiegelkamp, 2010) y (IEC, 2013), ampliamente usado en la actualidad en la programación de Autómatas Programables, pero con características que limitan su uso en el diseño de aplicaciones distribuidas y reconfigurables, algo que sí contempla el nuevo estándar. De ahí que el IEC-61499 esté siendo actualmente el centro de atención de numerosas investigaciones para su uso en diferentes problemas de automatización y control industrial. Una de las principales diferencias entre el IEC-61131 y el IEC-61499 es el modelo de ejecución de los programas de control: mientras que el primero se basa en el concepto de ciclo de trabajo (o ciclo de SCAN), el segundo centra su funcionamiento en el uso y manejo de eventos.

Si bien es cierto que el número de investigaciones desarrolladas sobre el uso del IEC-61499 en aplicaciones de control de sistemas de eventos discretos es considerable, no se puede decir lo mismo en el caso de las aplicaciones de control de sistemas continuos, muy limitadas a día de hoy, a pesar de su importancia en el funcionamiento de buena parte de los sistemas industriales. Esta escasez de estudios puede deberse a que se trata de un estándar relativamente nuevo. Concretamente en España las investigaciones sobre el IEC-61499 se limitan a una tesis presentada el año 2016 en la Universidad de Zaragoza (Catalán, 2015), y algunas experiencias aisladas en aplicaciones a escala de laboratorio. A la insuficiente cantidad de investigaciones hay que añadir que en todos los estudios realizados hasta ahora en aplicaciones de control continuo

se han considerado únicamente algoritmos de control con ejecución periódica, o sea, controladores basados en tiempo, como en (Strasser, et al., 2004), (Doukas, et al., 2006), (Hatmetner, et al., 2013) y (Querol, et al., 2016). El uso de estos algoritmos desaprovecha una de las características fundamentales del IEC-61499: la ejecución de los programas se controla íntegramente mediante eventos. Esta característica del IEC-61499 sugiere que una aproximación más conveniente al control de sistemas continuos sería el uso de algoritmos de CBE. Sin embargo, en la actualidad no existes estudios al respecto.

Con este proyecto se pretende iniciar los estudios para la implementación de algoritmos de CBE usando el estándar IEC-61499 con el objetivo de reducir la brecha que existe actualmente entre estas dos tendencias actuales del control automático. Esta unión permitirá obtener algoritmos de CBE más eficientes desde el punto de vista del coste computacional al aprovechar un paradigma de programación basado en la gestión de eventos y al mismo tiempo permitirá introducir en el IEC-61499 estrategias de control de sistemas continuos más acorde con las características del propio estándar (control de ejecución por eventos), en sustitución de los controladores basados en tiempo (con ejecución periódica) que se han venido usando hasta ahora.

Dado que los controladores PID son los más utilizados en aplicaciones industriales, y que actualmente existen varios trabajos en los que se propone distintas versiones de PID basados en eventos, es razonable considerar este tipo de controladores como un buen punto de partida en la investigación propuesta. A continuación se presenta un estudio del estado del arte de los controladores PID basados en eventos.

1.3.2 Estado actual de los controladores PID basados en eventos

En los últimos años se han publicado varios trabajos en los cuales se proponen algoritmos de control PID basados en eventos. Dichos trabajos se han desarrollado en el contexto de los sistemas de control en red, y en ellos se trata de aprovechar las conocidas bondades del control PID a la vez que se reduce el tráfico por la red de comunicaciones que interconecta las distintas unidades de hardware que forman el sistema de control: sensores, controladores, actuadores.

Una de las primeras contribuciones al desarrollo de los controladores PID basados en eventos fue introducida por Årzén en (Årzén, 1999) como una forma de reducir el uso de la CPU de los sistemas de control basados en computador sin afectar de forma significativa el comportamiento del bucle de control. En aquel artículo, Årzén puso de relieve algunas de las consideraciones más importantes que se debía tener en cuenta en los controladores PID basados en eventos. Entre éstas se pueden destacar los errores que se producen en los cálculos de los términos integral y derivativo cuando el tiempo entre muestras se incrementa. Varios trabajos posteriores estuvieron dirigidos a resolver los problemas revelados por Årzén, fundamentalmente el relacionado con el error en el cálculo de error del término integral. Deben ser destacados en este sentido los trabajos publicados por Durand (Durand & Marchand, 2009a) (Durand & Marchand, 2009b) y Vasyutynskyy (Vasyutynskyy & Kabitzsch, 2010a), (Vasyutynskyy & Kabitzsch, 2010b).

Una de las estrategias de generación de eventos que más se ha estudiado en el caso del control PID se basa en transmitir el valor de la señal sólo cuando ésta cruza niveles o umbrales δ. Dicha estrategia se conoce como envío por cruce de niveles SOD (a partir de ahora se nombrará por

sus siglas del inglés: *Send-On-Delta*) y su eficacia en cuanto al control y a la reducción de las comunicaciones ha sido ampliamente contrastada, (Dormido, et al., 2008) y (Ploennings, et al., 2010). Una variante de la estrategia SOD aplicada al control PI ha sido propuesta en (Beschi, et al., 2012). En ella se hace una cuantificación de la señal muestreada en cantidades que son múltiplo de un umbral δ, de forma que la relación entre la entrada y la salida del generador de eventos es simétrica con respecto al origen. Esta estrategia se conoce como envío por cruce de niveles simétricos SSOD (a partir de ahora se nombrará por sus siglas en inglés: *Symmetric-Send-On-Delta*). Algunos resultados importantes en el estudio de los controladores PI basados en SSOD han sido presentados recientemente en (Beschi, et al., 2012) y en (Chacón, et al., 2013). En (Beschi & Visioli, 2013) se ha abordado el ajuste de controladores PI bajo una estrategia de muestreo SSOD para el control de sistemas de primer orden con retardo, de manera que los resultados obtenidos se limitan a este tipo de modelos.

Algunos resultados más recientes se han presentado en (Romero, et al., 2014a), (Romero, et al., 2014b), (Romero & Sanchis, 2016a) y (Romero & Sanchis, 2016b). En (Romero, et al., 2014a) los autores presentan una regla para el ajuste de controladores PID basados en SSOD. La regla se basa en una modificación del método desarrollado en (Sanchis, et al., 2010) para el caso de un muestreo estándar. La principal ventaja de la regla propuesta es que ella misma se da en términos del margen de fase, un parámetro de diseño ampliamente usado en el ajuste de controlador es PID, lo cual facilita su aplicación de forma considerable. La regla no sólo se puede aplicar al ajuste de controladores PI sino que, a diferencia de trabajos anteriores, también se puede aplicar al ajuste de los PID.

Por otra parte en (Romero, et al., 2014b) se estudia el efecto de introducir distintos esquemas de muestreo en un bucle de control PID. En concreto se estudia el SSOD y el cuantificador regular RQ (a partir de ahora se nombrará por sus siglas en inglés: *Regular Quantifier*). A partir del estudio de la función descriptiva se obtuvieron las condiciones que debe cumplir el controlador para evitar los ciclos límites y por tanto las oscilaciones asociadas a ellos, cuando se usan dichas estrategias de muestreo. Para ambos casos se obtienen reglas de ajuste de los controladores las cuales están dadas en términos del margen de fase y margen de ganancia. El hecho de disponer de reglas que se basan en parámetros que indican la robustez en los sistemas continuos, como lo son los márgenes de ganancia y de fase, permite aplicar los métodos de ajuste que existen para los controladores continuos al caso de controladores PID basados en eventos. Las reglas son extremadamente simples de aplicar y pueden ser usadas para el ajuste de controladores PID, PI o PD u otras estructuras de controladores.

Siguiendo otra aproximación a los controladores PID basado eventos, en (Romero, et al., 2012) se propuso un controlador con umbrales adaptables con el que se consiguió reducir de forma considerable la dependencia que existe en los controladores que usan una generación de eventos con umbrales fijos, como es el caso de los muestreos SSOD, entre el número de eventos generados y la magnitud de las perturbaciones. Aunque tradicionalmente los controladores PID basados en eventos han usado esquemas de muestreo por cruce de niveles, siendo el SOD o variantes de este como el SSOD los más usados, algunos trabajos como (Romero, et al., 2012) y (Romero, et al., 2014b) sugieren que el uso de otras formas de muestreo puede ser más convenientes para determinadas aplicaciones. Evidentemente, la propuesta de nuevas estrategias de muestreo implicaría el desarrollo de nuevos métodos de ajuste que se adapten a dichas estrategias aún por concretar.

1.3.3 Estado actual del uso del estándar IEC-61499 para el control de sistemas continuos

Como se ha comentado anteriormente, las investigaciones con respecto al uso del estándar IEC-61499 para el control de sistemas continuos son escasas y están centradas en el uso de controladores periódicos. En las siguientes líneas se mencionan los pocos trabajos que han sido publicados al respecto.

El estudio presentado en (Strasser, et al., 2004) fue uno de los primeros en poner de manifiesto la necesidad de dotar al nuevo estándar de un comportamiento determinista en cuanto a tiempos de ejecución como un requisito fundamental para su uso en aplicaciones de regulación. En (Doukas, et al., 2006) se realizó un estudio experimental de la aplicación del estándar al control PID de un brazo robot, mostrando su viabilidad en este tipo de aplicaciones. Algunos resultados más recientes como los presentados en (Hatmetner, et al., 2013) hacen hincapié en las consideraciones que deben ser tenidas en cuenta al implementar controladores para sistemas continuos bajo el IEC-61499. En (Catalán, 2015) se propone una estrategia de gestión de eventos adaptada al control de máquinas herramientas. Por otra parte, en (Querol, et al., 2016) se presentó un estudio que revela la importancia que tiene la estrategia de gestión de eventos en aplicaciones de control de sistemas continuo bajo el mencionado estándar.

1.4 Normas y referencias

1.4.1 Disposiciones legales y normas aplicadas

Esta sección se dedicará fundamentalmente a explicar la norma IEC-61499. Se tratara de unas explicaciones con carácter introductorio para que el lector pueda tener unas nociones básicas sobre la utilización y contenido para que este pueda interpretar las figuras en relación al estándar que aparecerán a lo largo del proyecto.

Antes de discutir los contenidos de la norma IEC-61499 se debe tener en cuenta que esta no surgió como una nueva metodología de programación, sino que responde a la necesidad de modelar, estructurar e implementar sistemas de control distribuidos DCS (a partir de ahora se nombrará por sus siglas del inglés: *Distributed Control Systems*), que usando la norma IEC-61311 serían más difíciles de implementar, usando como base el Bloque de Funciones.

Para conseguir estos objetivos el estándar se centra en conseguir los siguientes puntos:

- **Portabilidad**: Las herramientas de software pueden aceptar e interpretar correctamente los componentes de software y las configuraciones del sistema creadas por diferentes herramientas de desarrollo.
- **Configurabilidad**: Las herramienta de desarrollo de diferentes fabricantes deben ser capaces de configurar cualquier dispositivo y su software.
- **Interoperabilidad**: Los dispositivos de control pueden trabajar conjuntamente para realizar las funciones requeridas por el sistema independientemente de cual sea su naturaleza u origen, en cuanto al fabricante se refiere.

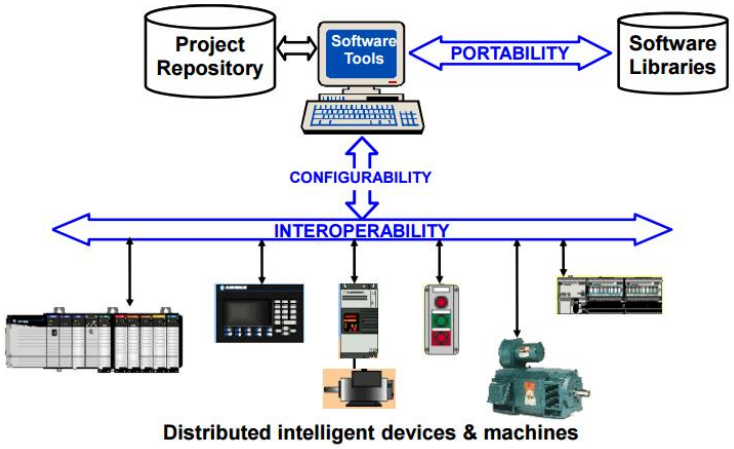


Figura 1: Esquema de los objetivos de la norma IEC-61499 para el DCS (Christensen, 2007)

En la figura 1 se puede ver el impacto que tienen los tres principios comentados anteriormente y por los que se rige la norma sobre los DCS, minimizando los inconvenientes que este tipo de control presenta.

Con respecto a la estructura que presenta el estándar, este se organiza en 4 partes independientes y con diferentes propósitos. Estas 4 partes se sintetizan junto a sus objetivos seguidamente:

- **Parte 1, Arquitectura:** En esta parte se determina la arquitectura de referencia para el desarrollo de la norma. Los conceptos importantes incluidos en esta parte se tratarán más adelante en este proyecto.
- **Parte 2, Requisitos de las herramientas software:** La segunda parte recoge los requisitos de las herramientas de desarrollo para llevar a cabo las arquitecturas expuestas en la primera parte.
- **Parte 3, Tutoriales informativos:** Este punto consta de la información didáctica que presenta las funcionalidades y ejemplos de aplicación de la IEC-61499.
- **Parte 4, Reglas para los perfiles de cumplimiento:** Este punto define las reglas que debe cumplir un sistema para ser compatible con la norma IEC-61499. Estas reglas están relacionadas con los principios de portabilidad, configurabilidad e interoperabilidad descritos anteriormente.

Antes de empezar con la descripción de los elementos de la norma, cabe decir que el estándar se caracteriza por una arquitectura que se compone fundamentalmente por un conjunto de modelos de referencia que representan los principales elementos y reglas que intervienen en un DCS para que este se pueda llevar a cabo correctamente su ejecución. A continuación, pasaremos a describir los diferentes modelos con los que cuenta la norma.

En el centro de la norma se encuentra el concepto de bloque de funciones (se nombrará por las siglas FB del inglés: *Function Block*), los cuales se construyen mediante el modelado o modelo de bloques de funciones (se nombrará por las siglas FBM del inglés: *Function Block Model*), siendo los FB la unidad funcional de software de menor nivel, con su propia estructura de datos que puede ser manipulada por uno o más algoritmos.

En el FB de la norma IEC-61499 confluyen dos tipos de parámetros, los eventos y los datos. La ejecución de un FB viene regulada por la recepción de eventos, pero también dependen de los datos de entrada, de salida y de las variables internas del propio bloque. Un FB puede tanto

recibir como enviar ambos tipos de parámetros, y en concreto, la recepción o envío de un evento se corresponde con el refresco de los datos de entrada o salida asociados con dicho evento. Todo FB pertenece a un tipo (en cuanto a lo que a funcionalidad se refiere, i.e. un tipo de FB que realice una suma), del cual se pueden crear varias instancias cada una con un nombre diferente. En la figura 2 se puede ver una representación de un FB con los diferentes elementos que hemos comentado anteriormente, en particular, se pude ver que la asociación de eventos y datos viene representada por un cuadrado y una línea vertical.

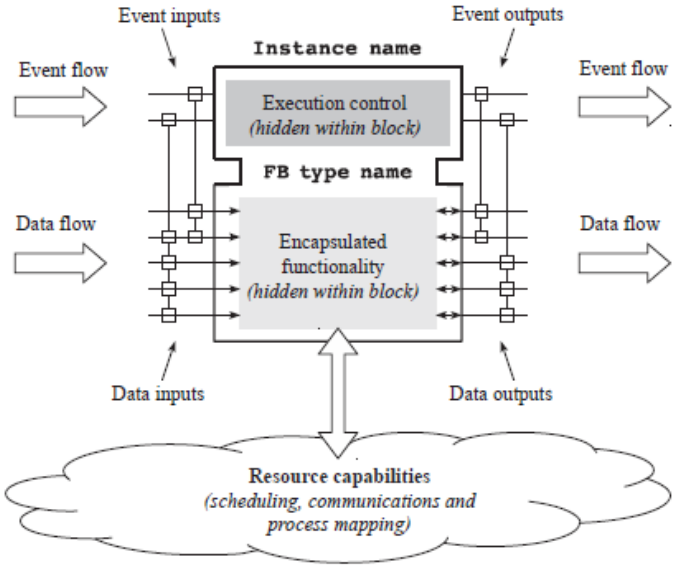


Figura 2: Esquema y componentes de un FB (Zoitl & Lewis, 2013)

En cuanto al comportamiento e interfaz con el resto de elementos de la norma existen tres tipos de FB:

- Los bloques de funciones básicos (se nombrará por las siglas BFB del inglés: *Basic Function Block*). Los BFB se caracterizan por tener una estructura y comportamiento están definidos por medio de un gráfico de control de ejecución (se nombrará por sus siglas ECC del inglés: *Execution Control Chart*).
- Los bloques de funciones compuestos (se nombrará por sus siglas CFB del inglés: *Composite Function Block*). Los CFB se caracterizan por tener su estructura interna compuesta por instancias de otros FB y sus respectivas conexiones, su comportamiento queda definido así en función de FB, la unidad básica de programación de esta norma.
- Los bloques de funciones de interfaz de servicio (se nombrará por sus siglas SIFB del inglés: *Service Interface Function Block*). Los SIFB proveen una interfaz entre la norma y servicios externos como puede ser una comunicación entre FB en diferentes dispositivos (i.e. gestión de la comunicación según el protocolo entre dispositivos).

Tanto los BFB como los SIFB pueden tener variables “globales” dentro del bloque, no siendo esto posible para los CFB (cuyos bloques internos sí pueden tenerlas) ni para las aplicaciones desarrolladas bajo este estándar, siendo esta una de las características peculiares que lo distinguen del estándar anterior.

Con respecto a los BFB, se ha mencionado que se caracterizan por tener una estructura interna caracterizada por un ECC. Un ECC es un gráfico que regula la ejecución y el comportamiento del FB. Los ECC se componen de varios estados, en los que se puede tanto enviar eventos de salida

como ejecutar algoritmos, y a los que se llega por medio de unas transiciones, las cuales se franquean mediante distintas condiciones, las cuales pueden ser simplemente la recepción de eventos o condiciones lógicas con datos. En la figura 3 podemos ver un ECC con las partes que hemos descrito, en azul los estados y las transiciones, en amarillo el algoritmo a ejecutar y en verde los eventos a enviar.

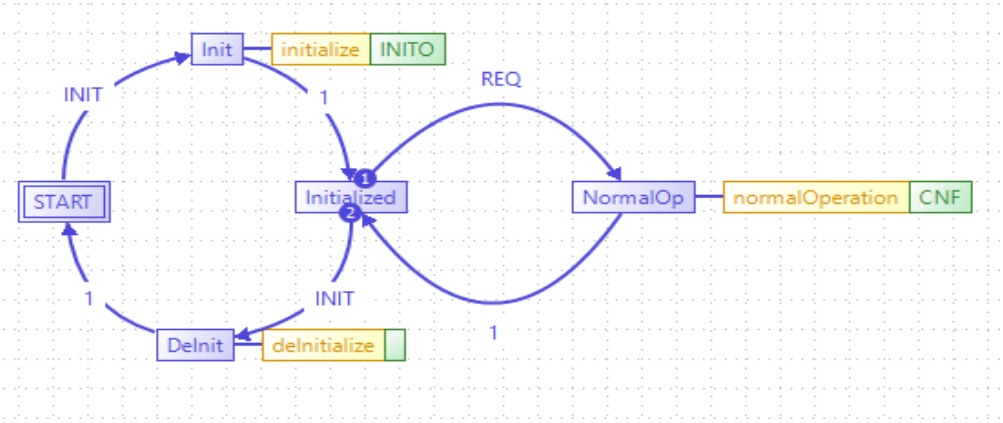


Figura 3: Ejemplo de ECC

Los FB se agrupan en un modelo que configura una aplicación, el modelo de aplicación (a partir de ahora se nombrará por sus siglas AM del inglés: *Application Model*). Una aplicación queda determinada y definida en términos de comportamiento y estructura por los FB que la componen y de la interconexión de eventos y datos de los mismos, siendo esta configuración la que dota de funcionalidad total al programa sin necesitar de variables globales o locales fuera de los FB (a excepción de los bloques que ya hemos explicado anteriormente) ya que el comportamiento del algoritmo debe regularse íntegramente dentro de estos. En la figura 4 podemos ver un ejemplo de aplicación por diferentes FB.

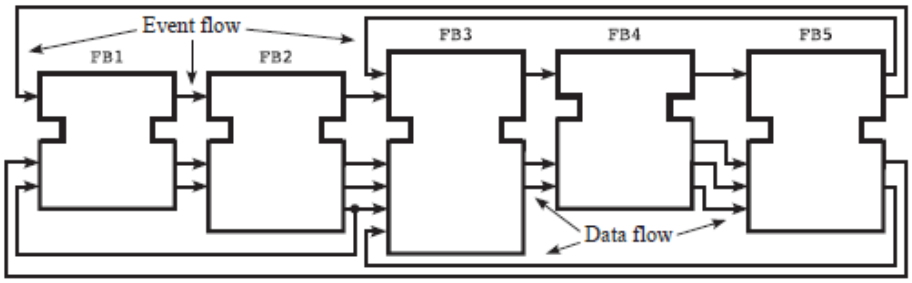


Figura 4: Modelo de aplicación (AM) (Zoitl & Lewis, 2013)

En las aplicaciones de la norma IEC-61499 los eventos son generalmente conexiones de punto a punto, pero, de todas formas, en algunos casos se permiten conexiones de una salida de evento a varias entradas, aunque esta práctica no es deseable y debería ser evitada siempre que fuera posible. Sin embargo, para los datos sí que está permitido conectar un dato de salido a varios de entrada, no siendo esto posible al contrario, es decir, no se puede conectar diferentes datos de distintas salidas a un dato de entrada ya que el FB receptor no dispondrá de la información del origen del dato y no podrá tratar correctamente la información. Cabe destacar que los datos y los eventos pueden tener su origen en diferentes FB pero confluir en otro.

La norma IEC-61499 presenta como ventaja respecto a la norma IEC-61131 justamente que se centra en el AM y no en los recursos que posee el sistema, lo cual repercute en la posibilidad de

poder evaluar el funcionamiento de una aplicación en etapas tempranas del proceso de desarrollo de la aplicación.

Los AM en esta norma pueden desarrollarse en uno o varios dispositivos o recursos que pueden estar conectados entre ellos según una serie de protocolos o redes. El modelo que describe y conecta los dispositivos es el modelo del sistema (a partir de ahora se nombrará por sus siglas SM del inglés: *System Model*). En este modelo se presentan los diferentes dispositivos así como las diferentes redes que pueden interconectarlos, asignando además algunas propiedades concretas en cuanto a la relación del dispositivo con la red (i.e. Asignación de IP's a un dispositivo conectado vía Ethernet). En la figura 5 vemos un esquema de un SM.

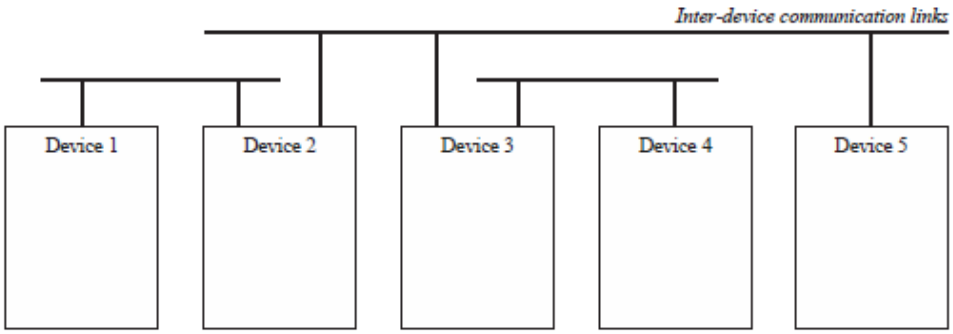


Figura 5: Modelo del sistema (SM) (Zoitl & Lewis, 2013)

Además, cada dispositivo esta modelado por una estructura interna llamada modelo del dispositivo (traducción del original "*Device Model*") que le permite ejecutar las redes de FB de la norma. El objetivo de este modelo es proveer de una infraestructura al dispositivo para soportar uno o más recursos. Los recursos de la norma se definen de forma similar a los de la norma IEC-61131, ya que estos permiten la ejecución de los FB. Sin embargo, a diferencia de la norma anterior, los recursos del nuevo estándar no están ligados necesariamente a una unidad de ejecución, es decir, un recurso es para esta norma una separación lógica dentro de un dispositivo y cada uno proporciona una ejecución y un control de las redes de FB independiente.

La norma trata también en profundidad el modelado de los recursos, sin embargo, para los fines que nos ocupan en este trabajo nos centramos únicamente en el hecho de que es el recurso el que nos proporciona las funciones de programación (en el sentido de "*scheduling*") de la aplicación, siendo esto un factor crucial cuando los requisitos temporales de las aplicaciones son críticos.

Como ya hemos comentado anteriormente, uno de los objetivos fundamentales de la norma es facilitar la implementación de aplicaciones distribuidas. Para este fin la norma cuenta con el modelo de distribución, el cual conecta la aplicación o partes de la aplicación (modelada en el AM), la cual a priori es independiente del sistema o sistemas que la ejecutan, con el modelo del sistema (SM). Este modelado trata, por tanto, de asignar partes de la aplicación a diferentes dispositivos y recursos y de configurar los dispositivos que ejecutan la aplicación.

Según la norma IEC-61499 las aplicaciones distribuidas deben distribuirse a nivel de la unidad fundamental de programación de la norma, es decir, el FB. De lo cual se desprende, que una aplicación distribuida para la norma consiste en una red de FB los cuales funcionan en diferentes dispositivos. En la figura 6 podemos ver un ejemplo de un modelo de distribución.

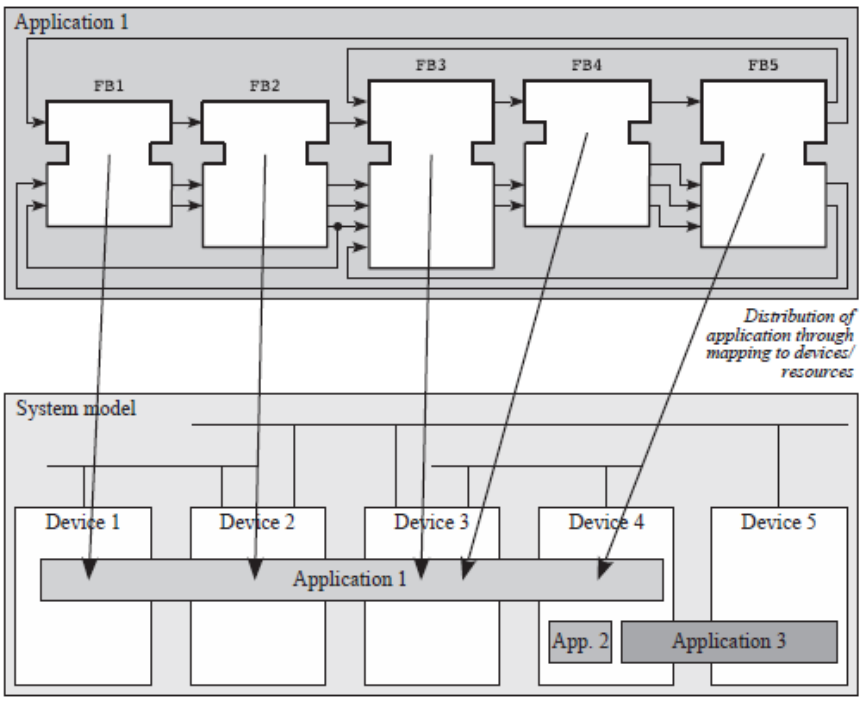


Figura 6: Ejemplo de modelo de distribución (Zoitl & Lewis, 2013)

Una particularidad del SM y del modelo de distribución del estándar es que permiten la ejecución de más de una aplicación en cada dispositivo, pudiendo además, asignar más de una aplicación a un mismo recurso. Para ello, en la concepción del modelo del dispositivo, el cual no vamos a detallar porque resultaría excesivo para una explicación introductoria de la norma, se ha previsto que se pueda cargar y borrar las aplicaciones de un dispositivo sin perturbar el funcionamiento de las otras aplicaciones en curso.

Hay que tener en cuenta para el desarrollo de aplicaciones distribuidas que se debe prestar atención a las características del recurso y del dispositivo, ya que el rendimiento de la aplicación dependerá del recurso donde esta haya sido mapeada y de las comunicaciones entre los recursos.

Aunque no se vaya a usar en el proyecto, se debe mencionar que, en las aplicaciones distribuidas, las conexiones entre FB situados en diferentes recursos se puede decir que se cortan y que además no se pueden añadir FB en el AM que proporcionen una configuración específica de la comunicación entre dispositivos porque el AM tiene que ser independiente de los sistemas que lo ejecutan. Para solventar estos problemas, la norma permite añadir FB de configuraciones específicas de los sistemas dentro de los recursos, de esta forma la aplicación se mantiene independiente de los sistemas y se asegura la funcionalidad de la aplicación.

Para finalizar con la introducción a la norma, se debe tratar uno de los objetivos principales en su concepción, facilitar el desarrollo de aplicaciones reconfigurables. Se entiende por reconfigurabilidad a la capacidad de un algoritmo de cambiar su comportamiento, estructura y/o los datos que maneja, sin necesidad de la intervención del programador, es decir, hacer este cambio dinámicamente. En este punto se debe hacer una distinción entre lo que se considera programación reconfigurable y paramétrica, la clave consiste en que en la programación paramétrica todo está programado antes de la ejecución y esta depende de unos parámetros, por el contrario, en la programación reconfigurable lo que se programa son los mecanismos que permiten al programa auto-reprogramarse a medida que se desarrolla la ejecución.

Para las aplicaciones reconfigurables, se proveen una serie de FB que permiten gestionar la instanciación, destrucción e interconexión de otros FB y los datos que manejan, iniciar la ejecución de FB como parte de aplicaciones distribuidas (i.e. Activar partes de la aplicación que normalmente no se activan), cambiar el estado en el que se encuentran los FB y proveer servicios a demandas por parte de las redes de comunicación.

1.4.2 Programas utilizados

Abarcamos en este apartado a todos los programas que se han utilizado para el desarrollo del proyecto en su faceta más técnica con el fin de completar las diferentes etapas que han constituido este proyecto:

4DIAC: Es el principal programa utilizado en todas las etapas del proyecto. Con este programa se realiza tanto la programación de aplicaciones como de los elementos que las constituyen, todo ello bajo el paradigma de programación de la norma IEC-61499.

Figura 7: Logotipo de 4DIAC

CMake: Programa encargado de generar los proyectos que van a proporcionar la estructura para la ejecución de las diferentes aplicaciones desarrolladas a lo largo de este proyecto.

Figura 8: Logotipo del programa CMake

Eclipse-neon: Programa para el desarrollo de partes específicas de las aplicaciones así como para el desarrollo de la estructura de ejecución de las mismas.

Figura 9: Logotipo de Eclipse-neon

EAGLE: con este programa se ha desarrollado el esquemático de las tarjetas de circuito impreso (a partir de ahora PCB, siglas en inglés de “Printed Circuit Board”) así como el ruteo de las mismas.

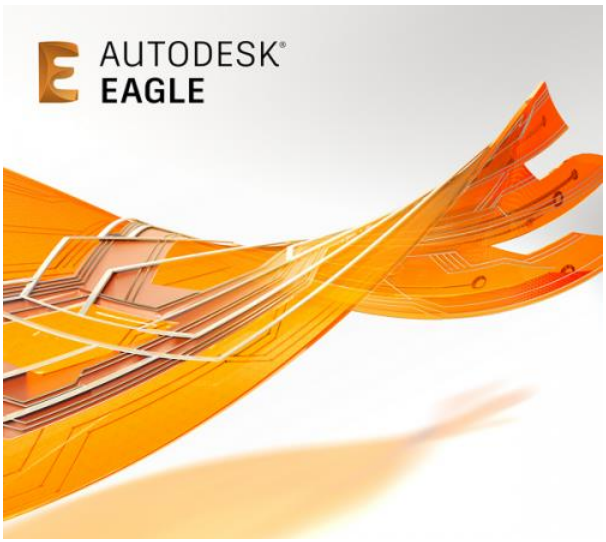


Figura 10: Logotipo de EAGLE

PuTTY: con este programa open-source se ha podido acceder a la tarjeta BeagleBone Black (a partir de ahora BBB) para lanzar las aplicaciones y obtener datos sobre dicha tarjeta.

Figura 11: Logotipo de PuTTy

Filezilla: con este programa se ha podido recuperar los archivos que son resultado de las aplicaciones para su posterior representación e interpretación.

Figura 12: Logotipo de Filezilla

Matlab: con este programa se han representado los datos obtenidos de las ejecuciones de las diferentes aplicaciones desarrolladas y se han realizado diferentes comparativas entre modelos teóricos y experimentales de los diferentes controladores desarrollados e implementados.

Figura 13: Logotipo de Matlab

1.4.3 Plan de gestión de calidad aplicado durante la redacción del Proyecto

Al tratarse de un proyecto cuya naturaleza es de carácter investigador, no se considera necesario implementar estrictamente estándares de calidad en la redacción como el ISO 9001 u otros métodos, eso sí, teniendo siempre estos como referencia.

1.4.4 Bibliografía

Årzén, K. E., 1999. A simple Event-based PID Controller. *Proceedings of 14th World Congress of IFAC,* Volume Q, pp. 423-428.

Åström, K. & Hägglund, T., 2004. Revisiting the Ziegler–Nichols step response method for PID control. *Journal of Process Control,* 14(6), pp. 635-650.

Beschi, M., Dormido, S., Sanchez, J. & Visioli, A., 2012. Characterization of symmetric send-on-delta PI controllers. *Journal of Process Control.*

Beschi, M. & Visioli, A., 2013. Tuning of symmetric send-on-delta proportional-integral controllers. *IET Control Theory and Applications.*

Catalán, C., 2015. *Modelos y plataforma IEC 61499 adaptados al control distribuido de máquinas herramienta en sistemas de fabricación ágil.* Zaragoza: s.n.

Chacón, J. et al., 2013. Characterization of limit cycles for self-regulating and integral processes with PI control and send-on-delta sampling. *Journal of Process Control.*

Christensen, J. H., 2007. *www.holobloc.com.* [Online]
Available at: http://www.holobloc.com/papers/kitara07.pdf
[Accessed June 2017].

Dormido, S., Sanchez, J. & Kofman, E., 2008. Muestreo, control y comunicacion basados en eventos. *Revista Iberoamericana de Automática e Informática Industrial (RIAI).*

Doukas, G. S., Koveos, Y. & Thramboulidis, K. C., 2006. *Using the Function Block Model for Robotic Arm Motion Control.* Ancona, s.n.

Durand, S. & Marchand, N., 2009a. *An Event-Based PID Controller With Low Computational Cost.* s.l., s.n.

Durand, S. & Marchand, N., 2009b. *Further results on event-based PID Controller.* Budapest, s.n.

Gupta, R. A. & Chow, M., 2010. Networked Control System: Overview and Research Trends. *IEEE Transactions on Industrial Electronics,* pp. 2527-2535.

Hatmetner, R., Schitter, G., Voigt, A. & Zoitl, A., 2013. *Implementation guidelines for closed loop control algorithms on PLCs.* s.l., s.n.

IEC, I. E. C., 2012. *Function blocks - Part 2: Software tool requirements.* s.l.:s.n.

IEC, I. E. C., 2013. *Programmable controllers - Part 3: Programming languages.* s.l.:s.n.

John, K.-H. & Tiegelkamp, M., 2010. IEC 61131-3: programming industrial automation systems: concepts and programming languages, requirements for programming systems, decision-making aids. *Springer Science & Business Media.*

Ploennings, J., Vasyutynskyy, V. & Kabitzsch, K., 2010. Comparative Study of Energy-Efficient Sampling Approaches for Wireless Control Networks. *Industrial Informatics, IEEE Transactions,* 6(3), pp. 416-424.

Querol, E., Romero, J. A., Sanchis, R. & Serrano, J., 2016. *Evaluation of closed loop control applications using differente event management strategies under IEC 61499.* Krakow, s.n.

Romero, J., Pascual, N., Penarrocha, I. & Sanchis, R., 2012. *Event-based PI controller with adaptative thresholds.* s.l., s.n.

Romero, J., Penarrocha, I. & Sanchis, R., 2014b. *Ajuste de controladores PID basados en eventos por cuantificación y cruce de niveles.* Valencia, s.n.

Romero, J. & Sanchis, R., 2016a. A new method for tuning PI controllers with symmetric send-on-delta sampling strategy. *ISA Transactions,* Volume 64, pp. 161-173.

Romero, J. & Sanchis, R., 2016b. *Analysis of a simple rule for tuning SSOD based PIDs.* Krakow, s.n.

Romero, J., Sanchis, R. & Balaguer, P., 2011. P, PI and PID auto-tuning procedure based on simplified single parameter optimization. *Journal of Process Control 21*, pp. 840-851.

Romero, J., Sanchis, R. & Penarrocha, I., 2014a. A simple rule for tuning Event-Based PID controllers with Symmetric Send-on-Delta sampling strategy. *Emerging Technology and Factory Automation (ETFA),* pp. 16-19.

Sanchis, R., Romero, J. & Balaguer, P., 2010. *Tuning of PID controllers based on simplifierd single parameter optimisation.* s.l., s.n.

Strasser, T., Zoitl, A. & Auinger, F., 2004. *Development, implementation and use of an IEC-61499 function block library for embedded closed loop control.* Berlin, s.n.

Vasyutynskyy, V. & Kabitzsch, K., 2010a. *Time contraints in PID controls with send-on-delta sampling.* s.l., s.n.

Vasyutynskyy, V. & Kabitzsch, K., 2010b. A comparative study of PID control Algorithms adapted to send-on-delta sampling. *Industrial Electronics (ISIE), 2010 IEEE International Symposium,* pp. 3373-3379.

Ziegler, J. & Nichols, N., 1942. Optimum settings for automatic controllers. *trans. ASME,* 64(11).

Zoitl, A., 2008. *Real-time Execution for IEC-61499.* s.l.:ISA.

Zoitl, A. & Lewis, R., 2013. *Modeling Control Systems Using IEC-61499.* s.l.:The Institution of Engineering and Technology.

1.5 Definiciones y abreviaturas

1.5.1 Definiciones

A

Anti-windup	Algoritmo implementado para evitar el *windup* de los controladores PID, efecto que acumula mucho error en el término integral entorpeciendo el control

B

BeagleBone Black	Tarjeta de hardware libre que usaremos para el desarrollo de los experimentos que se realizarán en este proyecto.

C

Controlador PID	Un controlador PID es un mecanismo de control por realimentación que se caracteriza por tener tres términos: el proporcional, que contribuye a la acción de control proporcionalmente en función del error; el integral, que contribuye con los errores pasados; y el derivativo, el cual contribuye prediciendo los errores futuros.
Ciclos de SCAN	Es una secuencia de operación que se da en los autómatas de manera repetitiva una vez entra en modo RUN (ejecución normal). Se caracteriza por tener tres fases: lectura de estado de entradas, ejecución del programa de usuario y actualización del estado de las salidas.
Capa (en electrónica)	Circuito impreso el cual se diseña para conectarse sobre una tarjeta electrónica.

R

Runtime	Software diseñado para soportar la ejecución de otros programas en un dispositivo.
RLC	Circuito de filtrado pasivo compuesto por una resistencia, una bobina y un condensador.

S

Sistemas de tiempo real	Son sistemas que interactúan con un entorno con dinámica conocida en relación con las entradas, salidas y restricciones temporales para funcionar correctamente.

Schedulling	Traducido como planificador, en el contexto de los sistemas operativos multitarea y multiproceso, es el proceso de ordenar, controlar y optimizar el trabajo y las cargas de trabajo, es decir, consiste en repartir el tiempo disponible del procesador entre los procesos que están disponibles para su ejecución.
T	
Thread	También llamado hilo de ejecución, hebra o subproceso, es una secuencia muy pequeña de tareas encadenadas que puede ser ejecutada por un sistema operativo. Los diferentes *threads* que pueden coexistir comparten una serie de recursos, en nuestro caso el procesador es el recurso principal.

1.5.2 Abreviaturas

A			
ADC	*Analog to Digital Converter*	AM	*Application Model*
B			
BFB	*Basic Function Block*		
C			
CBE	Control/Controlador Basado en Eventos	CFB	*Composite Function Block*
CPU	*Central Processing Unit*		
D			
DCS	*Distributed Control Systems*		
E			
ECC	*Execution Control Chart*		
F			
FB	*Function Block*	FBDK	*Function Block Development Kit*
FBM	*Function Block Model*	FSR	*Full Scale Range*
I			
IAE	*Integral Accumulated Error*	IDE	*Integrated Development Environment*
IEEE	*Institute of Electrical and Electronics Engineers*		
L			
LSB	*Least Significant Bit*		
P			
PCB	*Printed Circuit Board*	PID	Ver controlador PID
PID	*Process IDentifier*	PLC	*Programmable Logic Controller*
POSIX	*Portable Operative System Interface with base uniX*	PWM	*Pulse Width Modulation*

R

RQ	*Regular Quantifier*	RTOS	*Real-Time Operative System*

S

SCADA	*Supervisory Control And Data Acquisition*	SSH	*Secure SHell*
SIFB	*Service Interface Function Block*	SOD	*Send-On-Delta*
SSOD	*Symmetric-Send-On-Delta*		

X

XML	*eXtensible Markup Language*

1.6 Requisitos de diseño

En este proyecto se pretende implementar y evaluar el comportamiento de diferentes algoritmos de control basados en eventos, comparándolos con los controladores periódicos, bajo el entorno que proporciona la norma IEC-61499, para este fin, se necesita diseñar un método para poder evaluar los controladores, implementar los controladores y crear los modelos necesarios para su ejecución y asegurar su correcto funcionamiento. Una visión general del proyecto la encontramos en la figura 14.

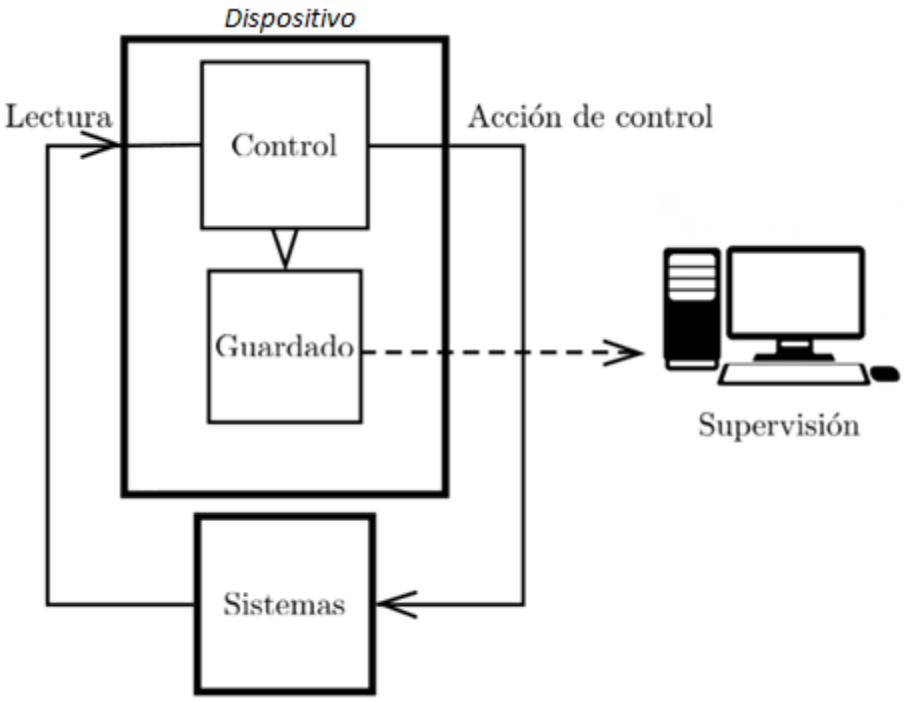


Figura 14: Esquema de las componentes generales del proyecto

En cuanto a los controladores a implementar, se pretende desarrollar y probar el funcionamiento en concreto de los controladores PID de Årzén, detallado en (Årzén, 1999), y de Durand, detallado en (Durand & Marchand, 2009b), (en adelante controladores de Årzén y de Durand) ambos controladores periódicos basados en eventos (los trataremos más adelante), y compararlos con el controlador periódico. Esta comparación se realizará con una tarjeta que

soporte las aplicaciones de la norma IEC-61499, sobre la cual se montarán unos sistemas electrónicos, que habrá que construir para realizar los experimentos, además habrá que crear una tarjeta para generar eventos de forma externa, la cual usaremos para experimentar no solo con la generación de eventos intrínseca si no también, con la extrínseca, para así ahorrarnos tiempo de cálculo de la CPU. Los experimentos se realizarán usando una modelización utilizando la norma IEC-61499. Una vez realizadas las experiencias de control, se realizará una evaluación de su desempeño y se realizará una caracterización temporal en cuanto al consumo de los algoritmos.

Por lo tanto, en cuanto a los requisitos del proyecto los podemos dividir en tres categorías, atendiendo a las necesidades anteriormente nombradas, según su naturaleza, las cuales son requisitos electrónicos, requisitos de programación y requisitos del control.

1.6.1 Requisitos electrónicos

Con respecto a los sistemas a controlar se pretende, por una parte que sean sistemas electrónicos, ya que estos son flexibles, es decir, se puede configurar fácilmente su dinámica; tiene un coste muy asequible, son sistemas que presentan una gran comodidad en la manipulación y nos ofrecen la posibilidad de operar con varios sistemas, tanto iguales como distintos entre ellos. Estos se deben montar en forma de capa sobre una tarjeta capaz de ejecutar aplicaciones basadas en la norma, con la que se realizará el control de los mismos, y que habrá que seleccionar. Al desarrollar los sistemas en forma de capa se facilita la conexión de los sistemas con la tarjeta de control. Por otra parte, se requiere que los sistemas sean lo más flexibles posibles en cuanto a la dinámica, es decir, que se pueda ajustar la dinámica de los mismos con facilidad.

Además, es deseable que los sistemas a controlar sean, como mínimo, de segundo orden subamortiguados para poder realizar experiencias de control con sistemas más completos y complejos, recordamos que en el artículo (Beschi & Visioli, 2013) las experiencias de control se limitaban a sistemas de primer orden con retardo.

Adicionalmente, se requiere también que en el PCB a desarrollar se puedan montar al menos cuatro circuitos para realizar también experimentos de control con más de un bucle de control. La consecuencia directa que se desprende de esto último es que los circuitos contengan el menor número de componentes posible ya que, previsiblemente, el espacio sobre la tarjeta a seleccionar será reducido.

Por otra parte, se requiere de la creación de un circuito electrónico que permita una detección y generación de eventos de tipo SSOD, de forma que se puedan generar y detectar los eventos externamente, es decir, sin utilizar cálculos de la CPU para este fin.

1.6.2 Requisitos de programación

En cuanto a los requisitos de programación, en este proyecto se deben desarrollar los diferentes algoritmos a utilizar en forma de FB con un software indicado para desarrollar aplicaciones y bloques de la norma IEC-61499, el cual se deberá seleccionar. En particular, se deben

implementar los algoritmos de control periódicos, los algoritmos de CBE y demás FB que sean necesarios para poder realizar el control de los sistemas. En cada caso resultara conveniente utilizar un tipo u otro de FB a criterio del desarrollador. También se debe modelar a su vez las diferentes aplicaciones en las que estos FB se utilizan.

Además, se deben introducir las modificaciones necesarias para asegurar la ejecución en tiempo real del *runtime* de la aplicación, en caso de que la ejecución en el sistema operativo por defecto no fuera suficiente para suplir las necesidades del control.

1.6.3 Requisitos del control

En cuanto a los requisitos de control, se debe evaluar el desempeño de los algoritmos de control implementados en forma de FB así como de las modelizaciones realizadas y corregirlas en caso de que presenten inconvenientes desde el punto de vista del control.

Las evaluaciones a realizar estarán enfocadas a evaluar el control de un sistema con un controlador periódico, con los diferentes controladores basados en eventos y con un controlador con generación externa de eventos. Además habrá que evaluar el comportamiento del control cuando hay más de un bucle a controlar. En el caso multi-sistema, se deberán tener en cuenta también las diferentes modelizaciones posibles para el control de varios sistemas.

1.7 Análisis de soluciones

Para la realización de este proyecto se deben aplicar conjuntamente conocimientos relativos a tres áreas diferentes, por un lado la electrónica, la programación y la automatización. Como ya hemos vistos en el apartado anterior tenemos requisitos que cumplir en lo relativo a estas tres áreas, por lo tanto, vamos a dividir el análisis de soluciones igualmente en estas tres áreas.

1.7.1 Análisis de soluciones electrónicas

Para la realización del proyecto es necesaria la selección de una tarjeta que soporte las aplicaciones desarrolladas bajo la norma IEC-61499, así como la construcción de dos circuitos electrónicos, el primero debe contener los sistemas a controlar y el segundo un método de generación de eventos, para lo cual se han analizado diferentes alternativas.

1.7.1.1 Selección del dispositivo de control

Como características esenciales del dispositivo de control, se encuentran, en primer lugar, la capacidad de soportar la ejecución del *runtime* de la norma IEC-61499, y en segundo lugar debe contener el mayor número de entradas analógicas y salidas para una señal PWM a fin de utilizar estas entradas y salidas para interactuar con los sistemas a controlar.

Además, se debe poder operar fácilmente con esta tarjeta en términos de conexión y obtención de datos, así como de carga de aplicaciones y *runtimes* para agilizar de esta forma los experimentos.

A la hora de seleccionar tarjetas para esta aplicación se han evaluado diferentes alternativas, pero de entre todas ellas, dos tarjetas destacan del resto, las cuales son la tarjeta Raspberry Pi 3 y la tarjeta BeagleBone Black.

Ambas tarjeas presentan un gran parecido en cuanto a las características técnicas del hardware montado en ellas, como se puede ver en la tabla 1.

	BeagleBone Black	**Raspberry Pi 3**
Precio (€)	40	30
Procesador	1GHz TI Sitara AM3359 ARM Cortex A8	4× ARM Cortex-A53, 1.2GHz
RAM	512 MB DDR3L @ 400 MHz	1GB LPDDR2
Almacenamiento	2 GB on-board eMMC, microSD	microSD
Sistemas operativos	Angstrom, Ubuntu, Android, ArchLinux, Gentoo, Minix, RISC OS…	Raspbian, Ubuntu, Android, ArchLinux, FreeBSD, Fedora, RISC OS…
Consumo	210-460 mA a 5V	150-350 mA a 5V
Pines	65 Pines	40 Pines
Periféricos	1 USB Host, 1 Mini-USB Client, 1 10/100 Mbps Ethernet	2 USB Hosts, 1 Micro-USB Power, 1 10/100 Mbps Ethernet, RPi camera connector
PWM	8	4 (iguales 2 a 2)
Entradas analógicas	7	0

Tabla 1: Características hardware de las tarjetas

Pero sin embargo, para la finalidad de este proyecto vemos claramente, como ilustra esta misma tabla, que la tarjeta que debemos escoger es la BeagleBone Black, ya que esta tarjeta puede proporcionarnos hasta 8 salidas PWM y 7 entradas analógicas frente a las 4 salidas PWM de la Raspberry pi 3 (las cuales además son iguales dos a dos) y a la ausencia de entradas analógicas.

1.7.1.2 Tarjeta con los sistemas a controlar

En primer lugar, para el diseño de la tarjeta con los sistemas a controlar, varios circuitos nos podrían dar una ecuación de transferencia de un sistema de orden dos o superior. Las dos alternativas a la solución final que se estudiaron fueron, al igual que la solución finalmente adoptada, filtros de paso bajo, en concreto se estudiaron el circuito de filtrado pasivo RLC y el circuito de filtrado activo a base de celdas de Sallen-Key, este último muy similar a la solución final, que es un filtro activo a base de celdas de Rauch.

Con respecto al circuito RLC, es el circuito más simple ya que solo consta de tres componentes como su nombre indica, una resistencia, una bobina y un condensador. En la figura 15 se puede ver el circuito de filtrado RLC.

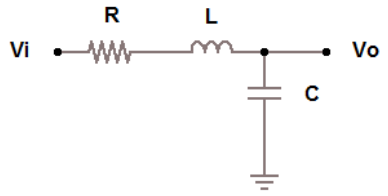


Figura 15: Filtro RLC paso bajo

Obtenemos la ecuación de transferencia de este circuito que resulta en:

$$H(s) = \frac{\frac{1}{LC}}{s^2 + \frac{R}{L}s + \frac{1}{LC}}$$

Comparando con la ecuación estándar de los sistemas de segundo orden:

$$H(s) = \frac{K\omega_n^2}{s^2 + 2\delta\omega_n s + \omega_n^2}$$

Obtenemos los parámetros δ y ω_n:

$$\omega_n = \frac{1}{\sqrt{LC}}$$

$$\delta = \frac{R}{2}\sqrt{\frac{C}{L}}$$

Los sistemas que queremos obtener deben ser subamortiguados, para ello el factor de amortiguamiento δ debe ser menor que la unidad; y los tiempos de establecimiento deben ser inferiores a un segundo. Por lo tanto, operando con las ecuaciones para el caso límite:

$$1 = \frac{R}{2}\sqrt{\frac{C}{L}};\ 4L = R^2C$$

$$t_s = 1 = \frac{4}{\delta\omega_n}; 1 = \frac{8L}{R}; R = 8L$$

Igualando:

$$4L = (8L)^2C; 4L = 64L^2C; \frac{1}{16L} = C$$

Con estas ecuaciones para los valores típicos de la bobina (de entre el orden de µH y mH) los valores que tendríamos que escoger tanto para la resistencia como para el condensador podrían ser un poco extraños en tanto en cuanto al hecho de que no son valores estándar o son valores muy pequeños para la resistencia o muy grandes para el condensador (para L=10 mH, R =80 mΩ y C=6,25 F), y, por lo tanto, desestimamos esta solución.

Por otra parte, tenemos el circuito de filtrado activo de paso bajo a base de celdas de Sallen-Key, el cual consta de un amplificador operacional, dos resistencias y dos condensadores. En la figura 16 podemos ver el montaje del filtro de Sallen-Key en configuración de paso bajo.

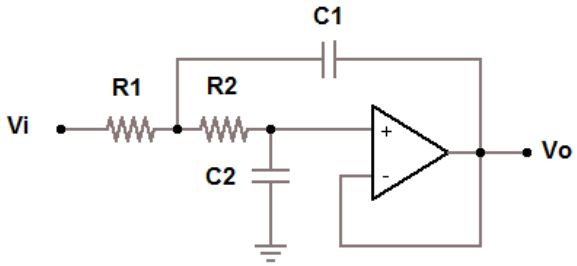


Figura 16: Circuito del filtro activo paso bajo de segundo orden de Sallen-Key

Analizando este circuito tenemos que su ecuación de transferencia es:

$$H(s) = \frac{\frac{1}{R_1 R_2 C_1 C_2}}{s^2 + \frac{R_1 + R_2}{R_1 R_2 C_2} s + \frac{1}{R_1 R_2 C_1 C_2}}$$

Igualando con la ecuación estándar de los sistemas de segundo orden obtenemos:

$$\omega_n = \frac{1}{\sqrt{R_1 R_2 C_1 C_2}}$$

$$\delta = \frac{R_1 + R_2}{2} \sqrt{\frac{C_1}{R_1 R_2 C_2}}$$

Haciendo iguales R_1 y R_2 a un valor R las expresiones quedan de la siguiente forma:

$$\omega_n = \frac{1}{R\sqrt{C_1 C_2}}$$

$$\delta = \sqrt{\frac{C_1}{C_2}}$$

Y en cuanto al tiempo de establecimiento de los sistemas tenemos que:

$$t_s = \frac{4}{\delta \omega_n} = 4RC_2$$

Realmente, no existe mucha diferencia entre las ecuaciones de este circuito con el circuito de filtrado de paso bajo a base de celdas de Rauch, el cual viene representado en la figura 17.

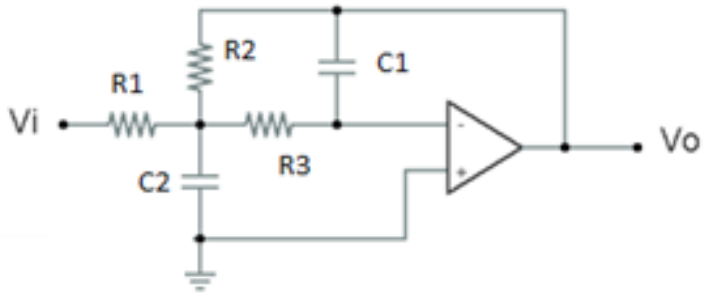


Figura 17: Circuito del filtro activo paso bajo de segundo orden de Rauch

El cual presenta la siguiente ecuación de transferencia:

$$H(s) = -\frac{\frac{1}{R_1 R_3 C_1 C_2}}{s^2 + \frac{R_2 + R_3 + \frac{R_2 R_3}{R_1}}{R_2 R_3 C_2} s + \frac{1}{R_2 R_3 C_1 C_2}}$$

Igualando con la ecuación estándar de los sistemas de segundo orden obtenemos:

$$\omega_n = \frac{1}{\sqrt{R_2 R_3 C_1 C_2}}$$

$$\delta = \frac{1}{2}\left(R_2 + R_3 + \frac{R_2 R_3}{R_1}\right)\sqrt{\frac{C_1}{R_2 R_3 C_2}}$$

Haciendo $R_1 = R_2 = R_3 = R$ obtenemos:

$$\omega_n = \frac{1}{R\sqrt{C_1 C_2}}$$

$$\delta = \frac{3}{2}\sqrt{\frac{C_1}{C_2}}$$

Y en cuanto al tiempo de establecimiento tenemos que:

$$t_s = \frac{8}{3} \cdot R \cdot C_2$$

Como podemos ver no existe mucha diferencia entre el circuito con celdas de Sallen-Key y el circuito con celdas de Rauch en cuanto a las ecuaciones que se utilizarían para caracterizar los sistemas. Aunque el circuito de Rauch presente el inconveniente de que tenemos que adaptarlo porque devuelve una respuesta invertida, lo escogeremos por delante del circuito de Sallen-Key porque esta adaptación es sencilla y no complica el montaje de la placa PCB, y porque de cara a futuros cambios este circuito tiene un componente más con el que poder ajustar otro tipo de sistemas.

1.7.1.3 Tarjeta para la generación de extrínseca de eventos

Para la tarjeta para la generación externa, o extrínseca, de eventos se pretende construir un circuito el cual permita detectar eventos según el método de muestreo SSOD, el cual tiene una forma similar a una conversión por parte de un ADC pero con unos umbrales de histéresis de valor δ iguales a los umbrales de detección. Un ejemplo de señal muestreada con esta metodología sería la presentada en la figura 18.

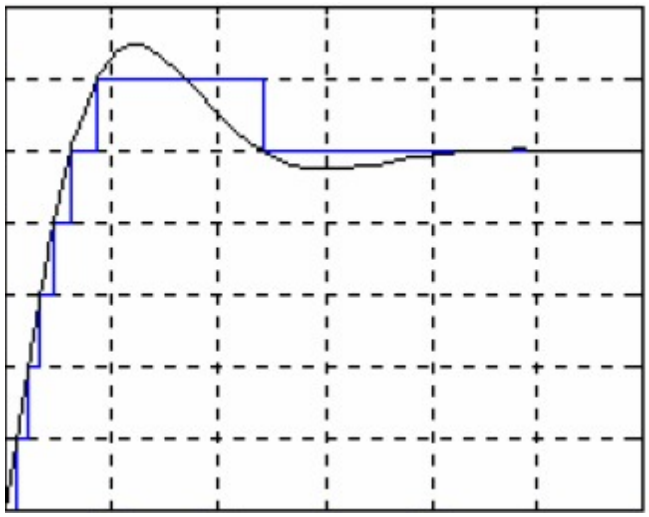


Figura 18: Ejemplo de señal muestreada con una metodología SSOD

Para implementar el circuito lógico asincrónico que debe enviar la señal digital a la tarjeta, convertimos la señal que hay que medir en un conjunto de señales digitales con un ADC, de ese modo, podemos utilizar el cambio de los bits resultantes de la conversión como la entrada del circuito de detección, y modificándolos podemos personalizar nuestra delta de detección

En una primera aproximación se examinó la posibilidad de realizar la modelización del circuito por el método de Huffman. Tanto en la modelización final como en esta vía explorada, el circuito comienza con la conversión anteriormente mencionada con el ADC. Empezaremos por ilustrar en la figura 19 como cambian los bits con los que realizaremos el circuito, que serán el bit menos significativo (LSB) y el inmediatamente superior.

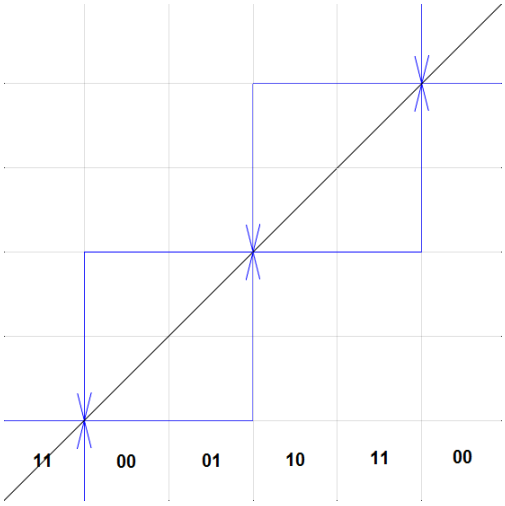


Figura 19: Bits y señal para la modelización con el método de Huffman

En la figura 19 podemos ver cómo cambian los bits (escritos como B_1B_0 en la imagen) en función de la entrada y cuál es la conversión del muestreo SSOD en azul. Estudiando esta figura podemos identificar dos tipos de estados:

- Estados tipo A: Estamos desde el valor 00 hasta el 11. De este tipo de estados se sale si estando en 11 vamos directamente a 00 o si estando en 00 vamos a 11 directamente.
- Estados tipo B: Estamos desde el valor 10 hasta el 01. De este tipo de estados se sale si estando en 10 vamos directamente a 01 o si estando en 01 vamos directamente a 10.

En la imagen 20 tenemos a la izquierda los estados del grupo A, que decidimos que darán como salida el valor lógico 0 y a la derecha tenemos los estados del grupo B que darán como salida 1.

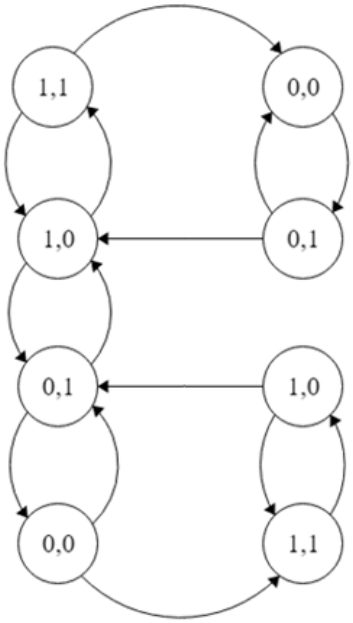


Figura 20: Grafo de estados primitivos del generador de eventos externos

Tenemos ocho estados posibles que numeramos del 1 al 8 comenzando por el 00 del grupo A en sentido horario. A partir de este grafo se construye la tabla de transiciones primitivas (Tabla 2). Siguiendo con la metodología de Huffman se debería fusionar los estados que nos conduzcan a los mismos estados con el fin de reducir el circuito resultante, sin embargo, nos demos cuenta que si hacemos esta fusión interna de estados podemos introducir transiciones que son falsas, y por lo tanto, el sistema no funcionaría como queremos.

(B1, B0) / **Estados**	**(0,0)**	**(0,1)**	**(1,1)**	**(1,0)**	**S**
1	①	2	8	-	0
2	1	②	-	3	0
3	-	2	4	③	0
4	5	-	④	3	0
5	⑤	6	8	-	1
6	5	⑥	-	3	1
7	-	2	8	⑦	1
8	5	-	⑧	7	1

Tabla 2: Tabla de transiciones primitivas

Pasamos por tanto a la siguiente etapa, es decir, a realizar la codificación interna de los estados, como tenemos 8 estados posibles utilizaremos 3 variables internas para codificarlos las cuales llamaremos Y1, Y2 y Y3. Se representa la codificación de los estados en la Tabla 3.

(B1, B0) / Y_1, Y_2, Y_3	(0,0)	(0,1)	(1,1)	(1,0)	S
000	000	001	100	-	0
001	000	001	-	011	0
011	-	001	010	011	0
010	110	-	010	011	0
110	110	111	100	-	1
111	110	111	-	011	1
101	-	001	100	101	1
100	110	-	100	101	1

Tabla 3: Codificación de los estados internos

A partir de esta tabla se puede obtener la tabla de verdad para cada uno de los estados internos y para la salida y agrupar para pasar a la expresión algebraica del circuito lógico a obtener. Podemos ver las tablas de verdad y las agrupaciones en las tablas 4, 5, 6 y 7.

(B1, B0) / Y_1, Y_2, Y_3	(0,0)	(0,1)	(1,1)	(1,0)
000	0	0	1	-
001	0	0	-	0
011	-	0	0	0
010	1	-	0	0
110	1	1	1	-
111	1	1	-	0
101	-	0	1	1
100	1	-	1	1

Tabla 4: Tabla de verdad para Y_1

(B1, B0) / Y_1, Y_2, Y_3	(0,0)	(0,1)	(1,1)	(1,0)
000	0	0	0	-
001	0	0	-	1
011	-	0	1	1
010	1	-	1	1
110	1	1	0	-
111	1	1	-	1
101	-	0	0	0
100	1	-	0	0

Tabla 5: Tabla de verdad para Y_2

(B1, B0) / Y_1, Y_2, Y_3	(0,0)	(0,1)	(1,1)	(1,0)
000	0	1	0	-
001	0	1	-	1
011	-	1	0	1
010	0	-	0	1
110	0	1	0	-
111	0	1	-	1
101	-	1	0	1
100	0	-	0	1

Tabla 6: Tabla de verdad para Y3

(B1, B0) / Y_1, Y_2, Y_3	S
000	0
001	0
011	0
010	0
110	1
111	1
101	1
100	1

Tabla 7: Tabla de verdad para la salida S

Con estas agrupaciones obtenemos las ecuaciones algebraicas que definen el comportamiento de la generación de eventos:

$$Y_1 = B_1\,B_0\,\overline{Y_1}\,\overline{Y_2}\,\overline{Y_3} + Y_2\,\overline{Y_3}\,\overline{B_0}\,\overline{B_1} + Y_1\,Y_2\,\overline{B_1} + B_0\,Y_1\,Y_2\,\overline{Y_3} + Y_1\,\overline{Y_2}\,B_1 + Y_1\,\overline{Y_2}\,\overline{Y_3}\,\overline{B_0}$$

$$Y_2 = \overline{Y_1}\,Y_3\,B_1\,\overline{B_0} + \overline{Y_1}\,Y_2\,B_1 + Y_2\,\overline{Y_3}\,\overline{B_1}\,\overline{B_0} + Y_1\,Y_2\,\overline{B_1} + Y_1\,Y_2\,Y_3\,\overline{B_0} + Y_1\,\overline{Y_2}\,\overline{Y_3}\,\overline{B_0}\,\overline{B_1}$$

$$Y_3 = \overline{Y_2}\,\overline{Y_1}\,\overline{B_1}\,B_0 + \overline{Y_1}\,Y_2\,Y_3\,\overline{B_1}\,B_0 + \overline{Y_1}\,Y_3\,B_1\,\overline{B_0} + \overline{Y_1}\,Y_2\,\overline{Y_3}\,B_1\,\overline{B_0} + Y_1\,Y_2\,\overline{B_1}\,B_0 +$$
$$+ Y_1\,\overline{Y_2}\,Y_3\,\overline{B_1}\,B_0 + Y_1\,Y_3\,B_1\,\overline{B_0} + Y_1\,\overline{Y_2}\,\overline{Y_3}\,B_1\,\overline{B_0}$$

$$S = Y_1$$

A la vista de las ecuaciones nos podemos hacer una idea de todas las puertas lógicas que se deberían utilizar (y por lo tanto del tamaño del circuito), decidimos por tanto no utilizar el circuito resultante de esta modelización y utilizar otro más sencillo.

Además, esta modelización por el método de Huffman asume que se puso un cero en el sitio de cada transición no posible (marcado con un guion) y puede darse el caso de que el ADC haga conversiones de bits no consecutivos, es decir, si en una conversión traduce por ejemplo al valor 3 y en la siguiente 5, podemos tener un problema porque tendríamos un cambio no previsto en la codificación interna de la máquina de estados. Este problema aparece también en el circuito que se escogió finalmente pero como la velocidad de conversión es superior a la dinámica de los sistemas a muestrear no encontramos ningún problema.

La alternativa desarrollada a este circuito, la cual se detalla en el apartado de resultados, consiste en un circuito cuyas entradas son también dos bits de un ADC, pero la detección no se realiza como en el circuito anterior íntegramente con puertas lógicas si no que se utilizan también biestables para el almacenamiento de estados, consiguiendo de esta forma, reducir de forma significativa las dimensiones del circuito resultante y por tanto su impresión en forma de PCB.

1.7.2 Análisis de soluciones de programación

1.7.2.1 Selección de la herramienta de desarrollo

En primer lugar, se debe determinar la herramienta de programación que se va a usar para el desarrollo e implementación del estándar IEC-61499. Existen varias herramientas para el desarrollo: FBDK, 4DIAC, nxtStudio e ISaGRAF, siendo las dos primeras herramientas *opensource*. Pasaremos a describir las herramientas y tomaremos una decisión sobre la herramienta a utilizar.

La primera herramienta, FBDK, fue uno de los primeros entornos que aparecieron para crear modelos de acuerdo al estándar IEC-61499, desarrollada por uno de los creadores del propio estándar.

En cuanto a las funcionalidades de la herramienta, ésta presenta las funciones básicas necesarias para crear modelos completos según la norma. Presenta un interfaz muy simple con una única perspectiva que permite definir todos los modelos de FB existentes. Una vez creados los FB, la herramienta incorpora una función de testeo para validar su comportamiento de forma gráfica. De igual manera se pueden crear aplicaciones de control uniendo FB y sistemas de control distribuido en su conjunto.

Uno de sus puntos fuertes es su cumplimiento de la parte 2 de la norma, que define el formato neutro en XML para el intercambio de ficheros entre aplicaciones de desarrollo, de forma que la herramienta puede, tanto generar los FB en formato neutro, como interpretar los mismos creados por otras aplicaciones que cumplen el estándar. Además, FBDK incluye ciertas librerías útiles para el desarrollo de aplicaciones. Finalmente, este IDE destaca por su portabilidad entre sistemas, al estar programado íntegramente en Java se puede ejecutar sobre cualquier plataforma que posea un Java *runtime*.

En contrapartida, uno de los puntos débiles de esta opción es precisamente su simplicidad. El entorno gráfico (Figura 21) es demasiado simple, puede incluso resultar poco amigable para el usuario. Junto con esto, la herramienta presenta ciertos fallos, algunos especialmente importantes como la imposibilidad de testear un bloque tras su modificación sin reiniciar la herramienta.

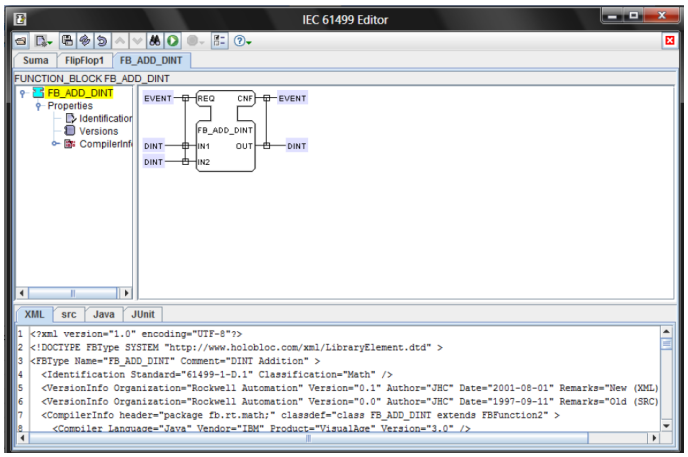


Figura 21: Entorno gráfico de FBDK

La segunda herramienta, 4DIAC (*Framework for Distributed Iindustrial Automation and Control*) nace en 2007 con el objetivo de crear una herramienta de código abierto para modelar e implementar sistemas de control distribuidos. El resultado ha sido el desarrollo de una herramienta *opensource*, basada en el entorno de desarrollo Eclipse, que abarca todas las fases del desarrollo de un DCS, desde su modelado y su validación hasta su implementación.

4DIAC-IDE presenta ciertas ventajas respecto a otras soluciones, tanto en sus funcionalidades respecto a la norma, como en el propio funcionamiento de la aplicación. Por una parte, este IDE contempla las funcionalidades necesarias para modelar sistemas según el IEC-61499, como la creación de FB, la importación y exportación de los mismos en formato neutro o el modelado de sistemas. Además, permite testear de forma gráfica tanto los FB, como las aplicaciones y los sistemas de control en su conjunto, y la lectura/escritura de datos de forma remota. También contempla la creación de escenarios de validación, mediante una secuencia de test, para comprobar de forma automática los bloques. Añadido a estas funcionalidades, 4DIAC se caracteriza por su énfasis en la apertura, así desde este entorno se pueden configurar y utilizar

directamente diversos *runtimes* como FORTE o FBRT. Finalmente, 4DIAC también proporciona unas amplias librerías de FB, que contemplan todo tipo de situaciones, desde bloques para la utilización de las redes de comunicación a librerías matemáticas

Por otra parte, en cuanto a su funcionamiento, 4DIAC-IDE cuenta con una interfaz gráfica muy potente (figura 22), creado a partir de Eclipse, basado en el uso de perspectivas. A este interfaz además, se le pueden agregar complementos, en forma de Plug-in, por lo que se puede personalizar al uso y potenciar, si fuera necesario, por el usuario final. El entorno, al igual que FBDK, está desarrollado en Java por lo que se puede ejecutar en cualquier dispositivo compatible con el mismo. El desarrollo, tanto de la herramienta como de las librerías, se realizan por un equipo central dedicado a ello, pero toda la comunidad puede proponer sus propias mejoras y correcciones para que se incluyan en el desarrollo principal, siendo la comunidad una pieza clave en el desarrollo del proyecto.

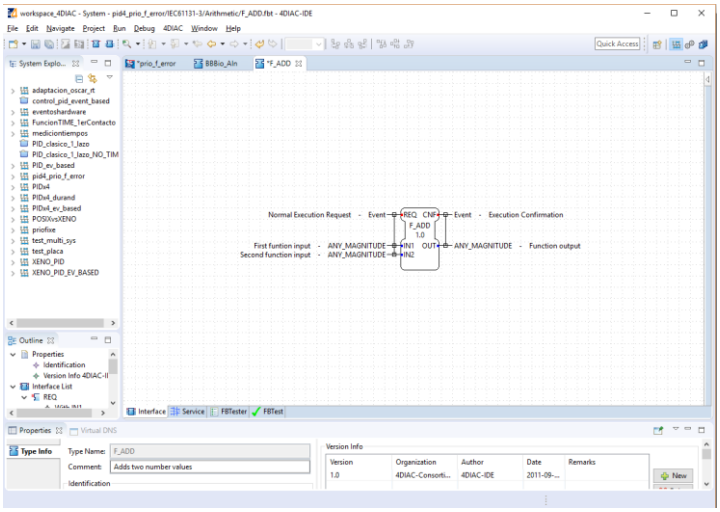


Figura 22: Entorno gráfico de 4DIAC

El único punto negativo de este entorno sería la imposibilidad de crear nuevos tipos de datos, ya que, sólo permite la utilización de los datos definidos en el estándar IEC 61131-3, lo que limita el desarrollo de bloques complejos o el encapsulamiento de información.

La siguiente herramienta, nxtStudio, es un IDE comercial creado por la compañía NXT Control. Al igual que 4DIAC, esta herramienta está concebida para abarcar todas las etapas de un proyecto de control distribuido, desde su modelado hasta su implementación. Es una de las primeras herramientas comerciales, junto con ISaGRAF, en ofrecer una solución comercial enfocada directamente al mundo industrial (figura 23).

Entre las principales funcionalidades que presenta este entorno, tenemos su capacidad para crear, simular y depurar modelos de control completos, permitiendo generar e importar FB. Además, también permite la creación de nuevos tipos datos y extender las funcionalidades del IEC-61499. Concretamente, esta herramienta permite la creación de entornos gráficos para los sistemas y facilita la integración de los controles creados en sistemas SCADA.

Por otra parte, esta herramienta permite, y está preparada para, configurar directamente equipos industriales como los PLC que cumplan con el estándar. A estos efectos, cuenta con un PLC simulado en el que se puede probar y depurar el control antes de ejecutarlo en un dispositivo real. Por último, una de las ventajas más notables de este IDE son sus librerías, nxtStudio incluye un gran número de librerías con bloques prefabricados pensados para poder

ser usados en situaciones de control típicas directamente. Estas librerías, además de incluir los propios FB, también tienen una representación gráfica, directamente vinculada con los bloques, para incluirla directamente en el entorno gráfico de control de la aplicación.

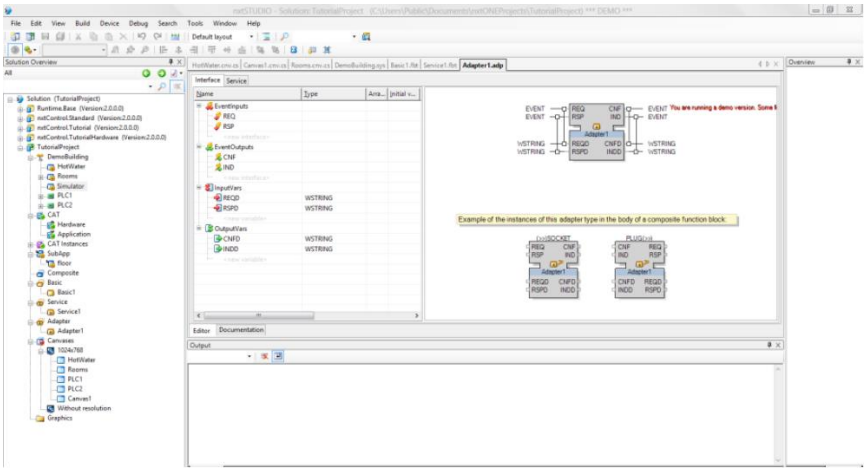


Figura 23: Entorno gráfico de nxtStudio

Esta herramienta no presenta prácticamente ninguna desventaja, es la referencia en el desarrollo industrial de aplicaciones con el IEC-61499. Sin embargo, su uso queda restringido a proyectos puramente industriales debido a su elevado coste, con lo que su compra puede resultar inviable para la mayoría de proyectos de investigación.

Finalmente, ISaGRAF es la otra herramienta comercial existente para el desarrollo de modelos basado en el IEC-61499. Esta herramienta ha sido desarrollada por la compañía ICS Triplex, perteneciente a Rockwell Automation. Su producto contempla todas las etapas del desarrollo de un sistema de control distribuido, y es especialmente efectiva en la integración ente DCS y sistemas de control convencionales.

Su principal ventaja es su compatibilidad con el IEC-61131, de forma que se puede usar el mismo entorno para crear, tanto sistemas distribuidos como, sistemas convencionales sobre PLC. Incorpora además, prácticas comunes en la programación con la 61131 a la 61499, lo que supone una ventaja destacable para los ingenieros de control acostumbrados a trabajar con el estándar 61131. En cuanto a su interfaz gráfico, presenta un interfaz potente basado en perspectivas que integra todas las etapas del desarrollo en un único producto. De igual forma que nxtStudio y 4DIAC, incluye librerías prediseñadas para facilitar la creación de aplicaciones de control.

El principal, y muy grave, inconveniente de esta herramienta es su hermetismo. A diferencia de las anteriores, ISaGRAF no permite la importación y exportación de FB y no cumple la naturaleza abierta del estándar. Por tanto, si bien es una herramienta que permite crear sistemas distribuidos, no se recomienda su utilización si realmente se quiere crear un sistema de control de acuerdo a los principios del IEC-61499.

De entre las cuatro alternativas, la primera a descartar es precisamente ISaGRAF por las razones anteriormente mencionadas. La segunda opción que se declinó fue nxtStudio, aunque si bien es la mejor de entre las opciones presentadas, el coste de la licencia es muy elevado. Finalmente entre 4DIAC y FBDK se opta por 4DIAC por las ventajas gráficas y por la ausencia de errores importantes que presenta FBDK.

1.7.2.2 Entorno de ejecución de las aplicaciones

Este apartado del análisis de alternativas en la programación viene motivado por un fallo detectado en la ejecución de las aplicaciones desarrolladas, por ello utilizaremos como ejemplo diferentes modelizaciones y experimentos realizados.

En primer lugar, y como se ha comentado en la descripción de la norma IEC-61499, solamente la modelización de una aplicación no asegura en absoluto que obtendremos la ejecución que se desea. Esto es debido a que la planificación de la ejecución de la aplicación es responsabilidad del recurso o recursos que ejecutan la aplicación.

Como ya hemos comentado anteriormente disponemos de una tarjeta BeagleBone Black, la cual tiene como sistema operativo Debian, que está basado en Linux, y el cual no nos ofrece funcionalidades de ejecución en tiempo real. En términos de las normas escritas por el IEEE se enmarcan dentro de los sistemas operativos de tipo POSIX, y dentro de estos, el sistema operativo de la BBB no contempla ninguna extensión para la ejecución en tiempo real.

Esto equivale a decir que todos los recursos que podamos crear dentro de la tarjeta BBB si usamos Debian no ofrecerán planificaciones de ejecución en tiempo real, y, por lo tanto, que no podremos controlar correctamente la ejecución de los FB dentro de las aplicaciones.

En según cuales aplicaciones lo anteriormente descrito podría no constituir un problema siempre y cuando los tiempos de ejecución no sean críticos, pero a medida que vamos descendiendo en la escala de tiempo necesaria para la ejecución de las aplicaciones nos vamos a encontrar problemas en la ejecución derivados de esta planificación no controlada.

Para ilustrar este hecho vamos a ejecutar la misma aplicación en un entorno de POSIX no-RTOS (el que lleva por defecto la BeagleBone Black) y en un entorno de tiempo real. La aplicación será simple, mucho más que un controlador, consta de un bucle que incrementa el valor de una variable que se guarda, todo ello con un período fijo. También cuenta con un FB para detener la aplicación pasado un tiempo. En un primer experimento fijaremos el tiempo de ejecución cíclica a 100 ms. Luego, reduciremos el período a un valor más pequeño, 1 ms. La figura 24 muestra la estructura y la configuración de FB en el experimento:

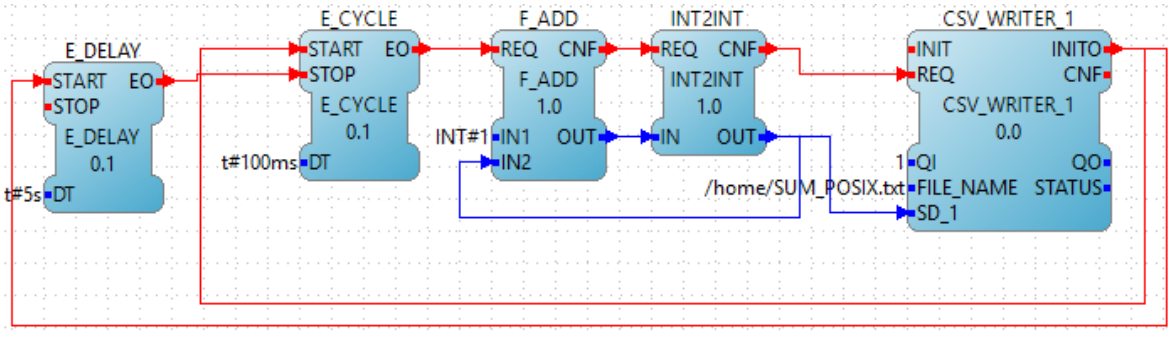


Figura 24: Montaje para la comparativa en la ejecución entre un RTOS y un no-RTOS

Utilizando un período de 100 ms vemos que no nos encontramos con ningún problema de ejecución, cómo podemos ver en la figura 25, el sistema operativo que no ofrece características de tiempo real puede sumar sucesivamente sin ningún problema hasta el final de la aplicación.

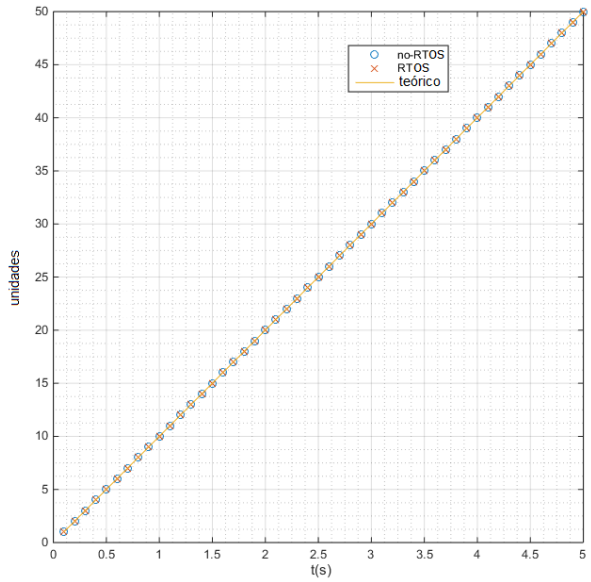


Figura 25: Test entre un RTOS y un no-RTOS con periodo 100 ms

Sin embargo, podemos ver que si reducimos el período de ejecución a un valor más crítico, a 1 ms esta vez, que es un tiempo suficiente para ejecutar los FB en tiempo real, vemos en la figura 26 que el sistema con tiempo real puede ejecutarle sin ningún problema, pero en el log de ejecución que da la misma aplicación sin tiempo real nos aparece un mensaje que dice: "*Event queue is full, event dropped!*", es decir, que se pierden eventos y no podemos asegurar correctamente el funcionamiento de la aplicación cuando los tiempos de ejecución se vuelven críticos. Podemos concluir pues, que el sistema operativo que se debe utilizar para las aplicaciones que van a desarrollar en este proyecto deben soportar las características de tiempo real, por esta razón se utilizará Xenomai como sistema operativo.

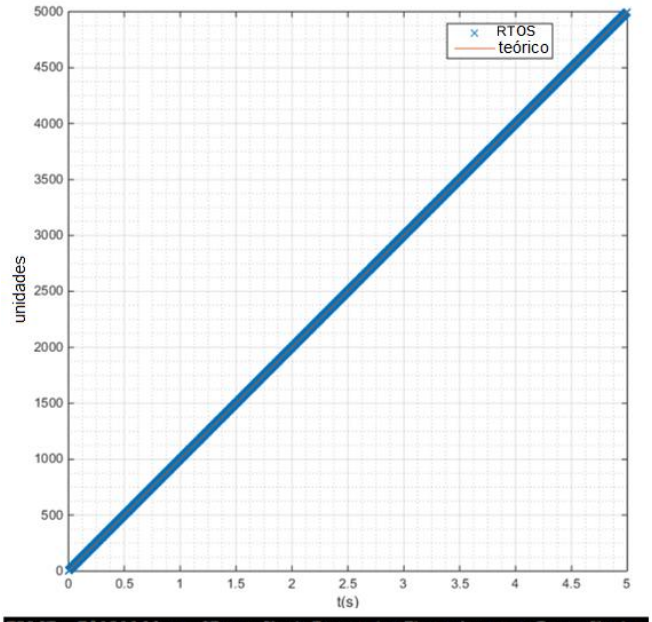


Figura 26: Test entre un RTOS y un no-RTOS con periodo 1 ms

1.8 Resultados finales

1.8.1 Desarrollo de las tarjetas electrónicas

1.8.1.1 Tarjeta con los sistemas a controlar

Tal y como se ha presentado en los antecedentes de este proyecto no existen demasiadas experiencias con sistemas reales con un orden superior a uno, por tanto, para cumplir con esa finalidad se ha creado una capa, que por medio del cambio de los componentes electrónicos, sea capaz de contener diferentes tipos de sistemas.

Como el espacio del que se dispone encima de la tarjeta no es demasiado grande se decidió utilizar un circuito electrónico bien estudiado y caracterizado como lo es un filtro activo con una sola celda de Rauch en su configuración en paso bajo, de haber dispuesto de más espacio se hubieran podido encadenar las celdas para aumentar el orden de los sistemas.

Como se ha expuesto en el apartado anterior, se ha escogido este tipo de circuito porque cumple con los requisitos especificados en cuanto al orden de los sistemas y porque se pueden lograr diferentes configuraciones en función de la colocación y de la magnitud de los componentes pasivos.

El montaje utilizado es el mostrado en el apartado anterior en la figura 18, pero para adecuarlo a las características de la BeagleBone Black se le han aplicado dos modificaciones. En primer lugar, se le ha introducido un *offset* para que con la señal PWM aplicada obtengamos siempre valores positivos de la señal de salida, y más en concreto para que para un valor del 50% del ciclo de trabajo de la PWM se obtenga una salida que se corresponda con la mitad de la escala de detección. En segundo lugar, se ha introducido un ajuste para que el valor máximo de la salida del circuito se corresponda con el valor máximo que puede leer la BBB.

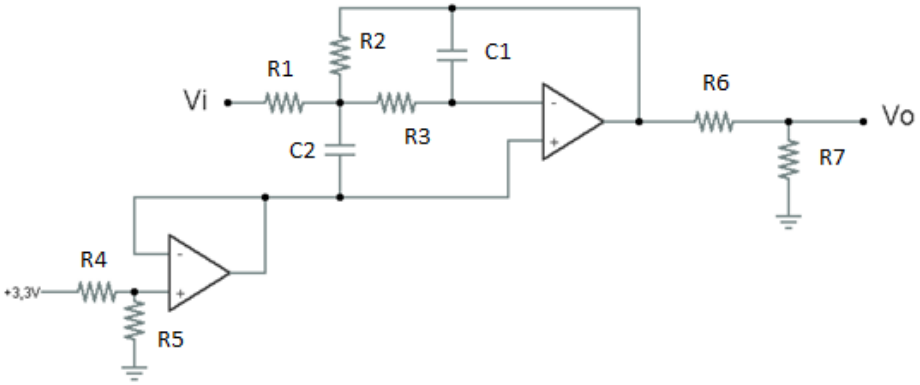


Figura 27: Circuito finalmente implementado

En la figura 27 podemos ver las tres partes que hay bien diferenciadas. En primer lugar, el amplificador operacional de la parte inferior junto con las resistencias R_4 y R_5 forman una referencia de tensión para la etapa siguiente, con esto se centraliza la salida como se dijo antes. En segundo lugar, el otro operacional con las resistencias R_1, R_2 y R_3 y con los condensadores C_1 y C_2 forman el filtro paso-bajo. Finalmente, con las resistencias R_6 y R_7 hacemos un puente divisor

para adaptar la ganancia del sistema a los niveles de tensión requeridos por la tarjeta para la entrada analógica.

En este punto podemos dar ya algunos valores para las resistencias:

- R_4 y R_5 las dos con valor 1 kΩ Para tener una tensión de referencia de 1,65 V.
- R_6 y R_7 con valores de 1,5 kΩ y 1,8 kΩ. Respectivamente para hacer el puente divisor. Construimos el puente divisor con estos valores porque el rango de la salida del circuito es de 0 a 3,3 V y el valor máximo admitido para la entrada analógica de la tarjeta es de 1,8 V (para evitar quemar la tarjeta añadimos también diodos para evitar los picos de tensión aunque no los hemos añadido en el esquema)

En cuanto a la elección del resto de componentes hemos visto en el apartado anterior las ecuaciones que rigen el comportamiento del sistema en cuanto a su dinámica en función de los valores de resistencias y condensadores.

Para facilitar la elección de estos componentes se ha representado gráficamente la variación de la dinámica, en particular, el valor del amortiguamiento δ, del sistema en función de los componentes en la figura 28; la variación del tiempo de establecimiento de sistemas subamortiguados (δ<1) en función de los componentes del circuito en la figura 29. Complementariamente también se ha añadido la variación del tiempo de establecimiento para sistemas sobreamortiguados (δ>1) en la figura 30, aunque no vamos a escoger este tipo de sistemas por sus similitudes con los sistemas de primer orden, cuya ecuación para el tiempo de establecimiento es:

$$t_s = 12 \cdot R \cdot C_1$$

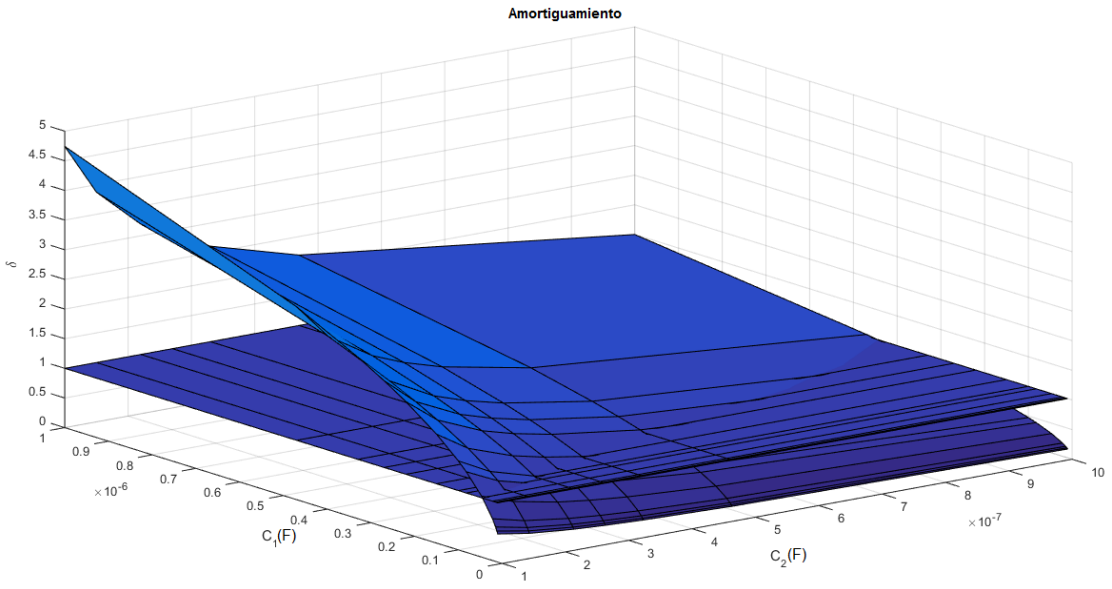


Figura 28: Amortiguamiento en función de los condensadores

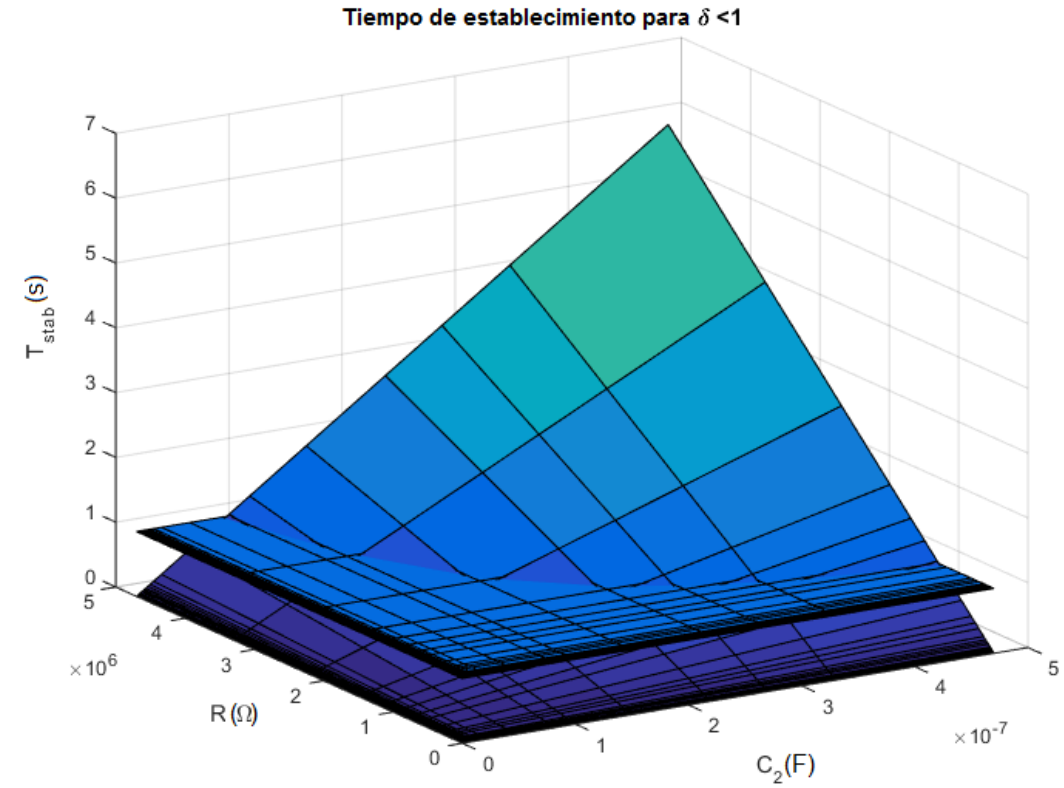


Figura 29: Tiempo de establecimiento para δ<1

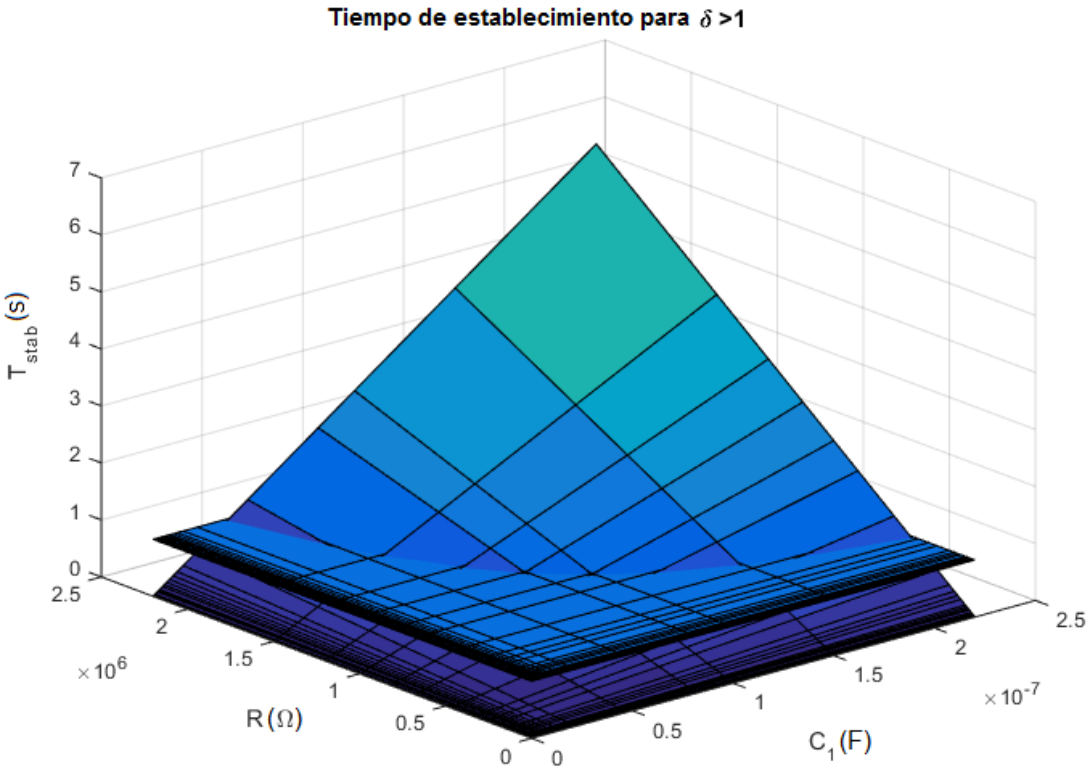


Figura 30: Tiempo de establecimiento para δ>1

Todas las gráficas han sido cortadas por un plano a la altura de 1 porque en el caso del amortiguamiento es la frontera entra un sistema subamortiguado, un sistema con

amortiguamiento crítico y un sistema sobreamortiguado. Y en el caso de los tiempos de establecimiento porque no se quiere sistemas con dinámica más lenta que 1 segundo.

Se ha decido montar cuatro sistemas en la misma capa para simular diferentes tipos de sistemas simultáneamente (en el Planos P.001 se puede ver el esquemático final y en el Anexo 2.2.1.1 la placa del PCB). Como se ha comentado anteriormente decidimos no experimentar con sistemas con un coeficiente de amortización superior a la unidad porque este tipo de sistemas se parecen considerablemente a los sistemas de primer orden, y queremos tratar justamente las oscilaciones producidas por coeficientes de amortiguamiento inferiores a la unidad.

Con este fin se han elegido diferentes sistemas para realizar los experimentos, cuyas características vienen detalladas en la tabla 8.

Sistema	Componentes	F. transferencia	Polos	Amort. y T_{stab}
1	C_1=270 nF C_2= 1 µF R=330 kΩ	$\frac{-34{,}01}{s^2 + 9{,}091 \cdot s + 34{,}01}$	$\lambda_1 = -4{,}54 + 3{,}65i$ $\lambda_2 = -4{,}54 - 3{,}65i$	$\delta = 0{,}7794$ $T_{stab} = 0{,}88\ s$
2	C_1=100 nF C_2= 1 µF R=120 kΩ	$\frac{-694{,}4}{s^2 + 25 \cdot s + 694{,}4}$	$\lambda_1 = -12{,}5 + 23{,}2i$ $\lambda_2 = -12{,}5 - 23{,}2i$	$\delta = 0{,}4743$ $T_{stab} = 0{,}32\ s$
3	C_1=100 nF C_2= 1 µF R=330 kΩ	$\frac{-91{,}83}{s^2 + 9{,}091 \cdot s + 91{,}83}$	$\lambda_1 = -4{,}54 + 8{,}43i$ $\lambda_2 = -4{,}54 - 8{,}43i$	$\delta = 0{,}4743$ $T_{stab} = 0{,}88\ s$
4	C_1=270 nF C_2= 1 µF R=120 kΩ	$\frac{-257{,}2}{s^2 + 25 \cdot s + 257{,}2}$	$\lambda_1 = -12{,}5 + 10{,}04i$ $\lambda_2 = -12{,}5 - 10{,}04i$	$\delta = 0{,}7794$ $T_{stab} = 0{,}32\ s$

Tabla 8: Caracterización de los sistemas

En cuanto al método de trabajo del conjunto de tarjeta y capa, hay que conectar la tarjeta por medio de un cable USB a un ordenador. Accedemos a la tarjeta por medio de una conexión SSH y ejecutamos sobre la tarjeta el *runtime* FORTE para cargar la aplicación que queremos ejecutar. El montaje completo de la tarjeta y la capa en el entorno habitual de trabajo está representado en la figura 31.

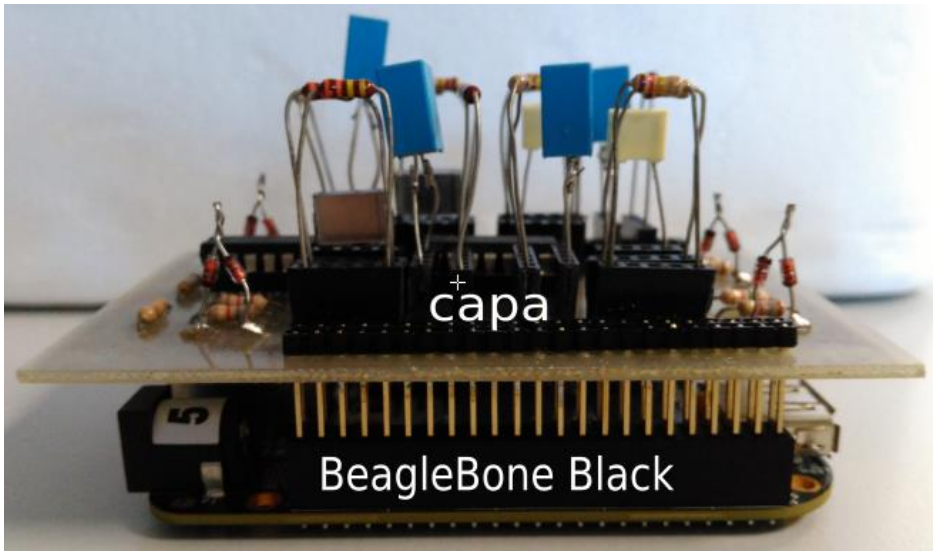


Figura 31: Conjunto de tarjeta BBB y capa

1.8.1.2 Tarjeta para la generación de extrínseca de eventos

Los controladores basados en eventos, aunque algunas veces no se deban ejecutar, consumen de todas formas tiempo de la CPU porque hay que hacer las comprobaciones para determinar si se debe ejecutar o no. Por esta razón se ha desarrollado un circuito que nos permite detectar un cambio significativo en la señal para que el controlador se ejecute. Con este circuito queremos reemplazar el tiempo que nos cuesta realizar una lectura analógica y hacer las comprobaciones, cuando no hace falta que el controlador se ejecute, por la lectura de una señal digital que cuesta mucho menos tiempo que la lectura digital y realizar las comprobaciones.

Como se ha dicho en el apartado del análisis de soluciones se requiere una generación de eventos basada en la metodología *Symmetric-Send-On-Delta* (SSOD), para lo cual se empieza por convertir la señal que hay que medir en un conjunto de señales digitales con un ADC utilizando así el cambio de los bits resultantes de la conversión como la entrada del circuito de detección.

Para el desarrollo de la tarjeta, no se ha seguido la metodología de Huffman, como hemos visto, por la magnitud del circuito resultante, por lo tanto, se ha decidido hacer de otra forma utilizando otro tipo de componentes. El principio de funcionamiento del circuito finalmente adoptado (Planos P.002) consiste en, teniendo como entrada el LSB y el inmediato superior de la señal convertida por el ADC, es decir, el bit 0 y el 1, se guarda el bit 1 cuando la conversión da un número par por medio de un biestable tipo D, y comparar este valor del bit 1 con el anterior valor guardado de este bit, si son diferentes, se envía un evento, es decir, se envía un evento cuando hay un cambio de valor par entre conversiones del ADC. Sin embargo, por las características de conversión del ADC no se sigue este mismo comportamiento cuando hay un cambio de dirección en la señal muestreada, dando como resultado una reducción del valor de la histéresis a la mitad de ese valor. El resultado final es que se envían eventos cada vez que se pasa un umbral de valor δ siempre y cuando no haya cambio de dirección, en cuyo caso, el primer envío se hará cuando se pase un valor δ/2. En la figura 32, podemos ver ilustrada la conversión que se ha implementado.

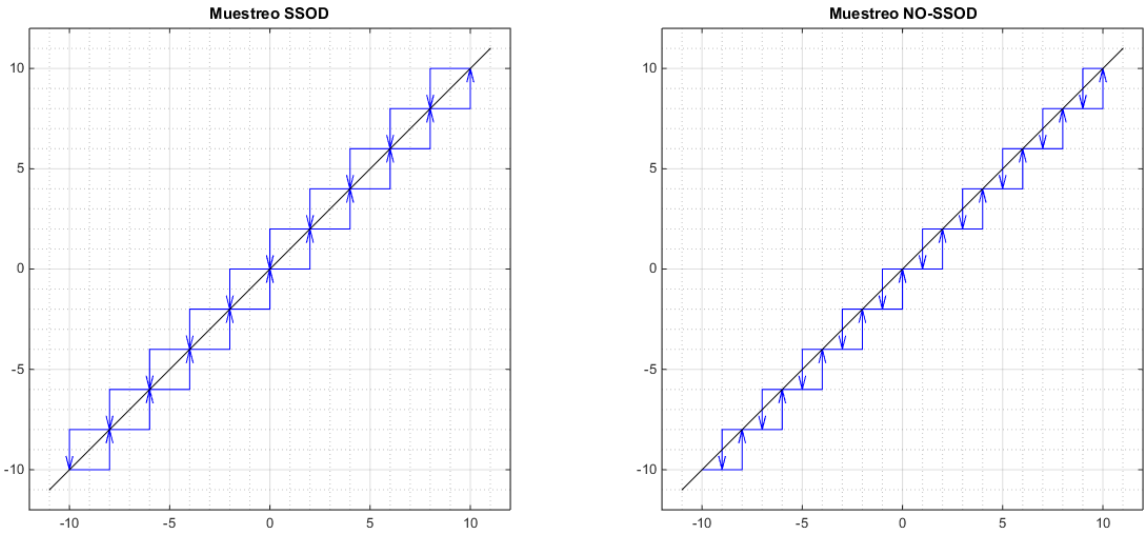


Figura 32: Muestreo SSOD (izquierda) y muestreo alternativo implementado (derecha)

Utilizando este tipo de generación de eventos se sale un poco de la teoría del SSOD en el sentido estricto, pero aun así, se logra el objetivo propuesto. El punto positivo de este tipo de muestreo es que somos más sensibles a los cambios de dirección de la señal y lo somos de la misma forma que lo es un SSOD cuando la señal no cambia de dirección.

El valor de la δ de detección se puede configurar fácilmente en función de los parámetros del ADC. En caso de cambiar los bits 0 y 1 de detección por otros bits, la explicación anteriormente aportada sobre el funcionamiento sigue siendo válida, lo único que cambia es que ahora no es entre conversiones pares consecutivas, es decir, se realiza cada n valores pares, por ejemplo, si se usaran el bit 2 y 1, en vez de cada 2 unidades, sería cada 4 unidades cuando se enviaría un evento. Se puede caracterizar esta δ de envío utilizando la siguiente ecuación:

$$\delta = \frac{FSR}{2^{8-(n_{bit}+1)}}$$

Donde el numerador es el FSR (siglas del inglés *Full Scale Range*) del ADC y el n_{bit} es el valor del menor de los bits usados para la entrada del circuito de detección. En cuanto a valores específicos, el FSR de nuestro ADC está fijado a 5 V, ADC que como se puede apreciar en la ecuación es un ADC de 8 bits.

Podemos comprobar cualitativamente el funcionamiento del generador de eventos conectando una señal sinusoidal a la entrada con el generador de ondas como podemos ver en la figura 33.

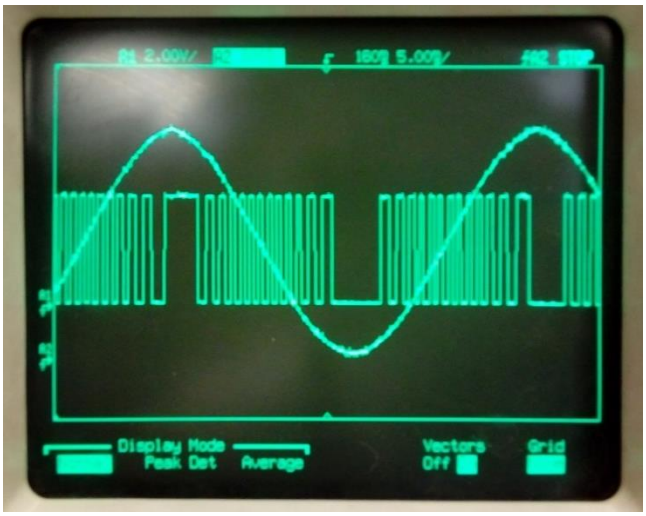


Figura 33: Comprobación del funcionamiento del generador de eventos

Como se puede ver, la señal digital cambia entre el valor alto y bajo con más velocidad en función de la pendiente de la señal de entrada, cambiando más cuando la señal es más grande y no cambiando en absoluto cuando el seno cambia de dirección.

En diferentes ensayos sobre este montaje hemos comprobado que podemos aumentar la frecuencia de una señal triangular de entrada hasta 50 Hz, valor a partir del cual empezamos a perder información en la generación de eventos. Esta velocidad de detección viene dada por la suma acumulativa de los tiempos de conversión del ADC, el tiempo de tratamiento de datos de las puertas lógicas y por el tiempo consumido por los biestables. Podemos concluir pues, que el

circuito es plenamente funcional ya que la dinámica de los sistemas es mucho más lenta que la velocidad de generación de eventos.

En la figura 34 podemos ver el resultado de la tarjeta desarrollada con dos generadores de eventos montados sobre ella. En este caso no se trata de una capa para la BBB si no que se trata de una tarjeta externa debido a picos de consumo de los componentes, por ello podemos ver que la alimentación llega de manera externa por los conectores de los bornes.

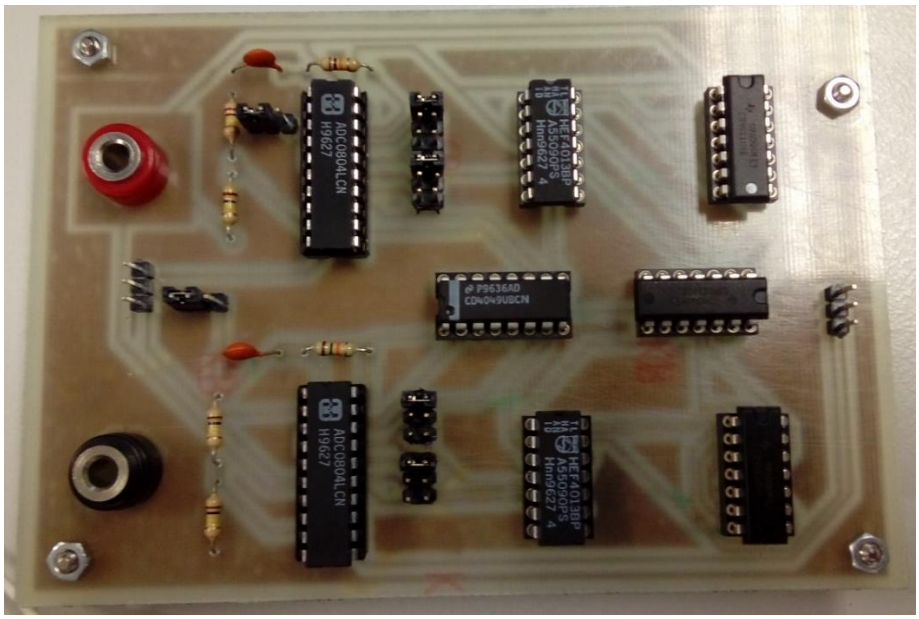


Figura 34: Tarjeta con dos generadores de eventos

En el anexo 2.2.1.2 se demuestra el funcionamiento experimentalmente y se aporta la impresión en PCB del circuito.

1.8.2 Desarrollo de FB

Para poder ejecutar correctamente una aplicación para el control de sistemas son necesarios algunos FB que nos permitan tomar muestras, controlar los sistemas y bloques para actuar sobre el sistema.

Para tomar muestras en este caso vamos a leer entradas analógicas, las cuales toman valores de entre 0 y 1,8 V, pero desarrollaremos también bloques para leer entradas digitales. Además, para controlar el sistema desarrollaremos diferentes FB con controladores PID, particularmente, el controlador PID clásico periódico, un FB que toma como base el controlador PID de Årzén presentado en (Årzén, 1999) y un FB que toma como base el controlador PID de Durand presentado en (Durand & Marchand, 2009b) (para no sobrecargar el proyecto, nombraremos a los controladores como "controlador de Årzén/de Durand" en lugar de "controlador basado en el controlador de Årzén/de Durand", aunque tomamos sólo la base de estos controladores porque los controladores originales presentan algunos defectos). Además, para actuar sobre el sistema utilizaremos un FB que genera una señal PWM que en función del ciclo de trabajo controlará el sistema. Finalmente, y aunque aporten una funcionalidad puramente vinculada al control, hay que desarrollar algunos FB para permitir el correcto funcionamiento de la aplicación.

Pasaremos ahora a describir y explicar los diferentes FB que se han desarrollado a lo largo del proyecto.

1.8.2.1 FB de entrada

Como se ha descrito en la introducción de este apartado se han desarrollado diferentes FB que nos permiten leer tanto señales analógicas como digitales. Estos FB son de tipo SIFB, porque nos ofrecen un servicio de comunicación entre la aplicación y el microprocesador ya que leen directamente los datos de los registros del procesador.

En primer lugar, tenemos el FB que realiza la lectura de la entrada analógica (se ha llamado a este FB "BBBio_AIn") el cual podemos ver en la figura 35.

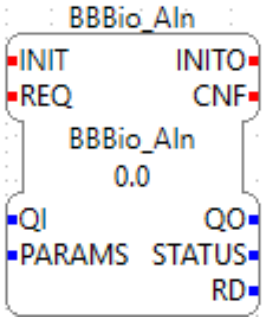


Figura 35: FB para la lectura analógica "BBBio_AIn"

Este FB toma como datos un booleano para habilitar su ejecución y una cadena de caracteres que representa la entrada que debe leer. Como salida, este FB da un booleano para indicar como se ha desarrollado la ejecución, una cadena de caracteres para indicar posibles errores durante la ejecución y un real con el valor leído. En cuanto al control de ejecución, el FB se inicializa cuando recibe un evento sobre la entrada INIT, cuando esta se ha realizado envía un evento por INITO (se haya inicializado correctamente o no) y para leer el valor analógico debe recibir un evento en REQ y envía un evento por CNF cuando esta se ha realizado.

En segundo lugar, tenemos el FB que se encarga de leer entradas digitales (se ha llamado a este FB "BBBio_IX") el cual podemos ver en la figura 36.

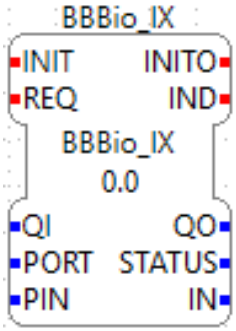


Figura 36: FB para la lectura digital "BBBio_IX"

Este bloque tiene una estructura y comportamiento idénticos al FB anterior, salvo por el hecho de que se necesitan dos cadenas de caracteres para determinar el puerto y el pin a leer y que la salida no es un real, sino un booleano.

Finalmente, se ha desarrollado un FB para indicar cuando la entrada digital ha cambiado de estado (se ha llamado a este FB "BBBio_IX_ChangeDetect"), podemos ver este FB en la figura 37. Este bloque es necesario porque en la documentación técnica del procesador no aparece toda la información para programar interrupciones (si bien es cierto que sí aparece como configurar pines para estas interrupciones no aparece como está distribuida internamente la memoria, en concreto, que registros se deben leer para detectar la interrupción, es decir, no aparece la tabla de vectores de interrupción).

Este FB es igual al FB anterior en cuanto a estructura y funcionamiento excepto que no envía su salida como un booleano si no en forma de eventos. Este FB envía el evento IND si se ha detectado una conmutación o el evento NO_IND en caso contrario.

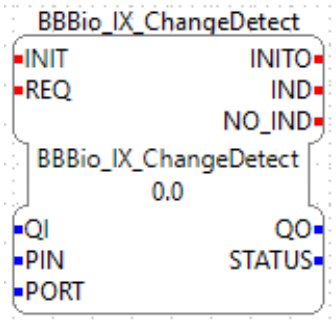


Figura 37: FB para la detección de la conmutación digital "BBBio_IX_ChangeDetect"

1.8.2.2 FB de controladores

Los FB que tiene funciones de control se han modelado bajo la forma de BFB, porque el modelo de ejecución interna, caracterizado por ECC, nos da mucha flexibilidad a la hora de modelar dicho comportamiento. Hay que decir, que aunque los FB que se van a presentar son versiones finales ha habido algunas versiones funcionales anteriores a esta solución y muchas otras versiones que presentaban defectos que se debieron resolver a medida que se detectaron.

Como los FB de esta sección son todos bloques PID hay algunos parámetros y partes del funcionamiento que se repiten. Todos los FB de control tienen como parámetros comunes la medida realizada, la referencia a seguir, los parámetros inherentes al control PID (a saber: ganancia proporcional, tiempo derivativo e integral, ponderación de la referencia y el coeficiente de filtrado del término derivativo), límites de saturación de la salida y una entrada para indicar el tiempo entre ejecuciones (en el caso del PID periódico este dato es constante pero no lo es en los otros controladores). Como salida todos ellos tienen la acción de control.

En cuanto al aspecto del control de ejecución, todos reciben un evento de entrada para inicializar el FB, uno para ejecutarlo y otro para devolver los valores internos a cero (hacer un reset) y como eventos de salida todos tienen un evento para indicar que ha finalizado la etapa de inicialización y, como mínimo, otro evento de salida que indica que se ha ejecutado el FB.

Una vez explicadas las características comunes de todos los FB de control pasamos ahora a describir las particularidades de cada uno de los FB desarrollados.

En primer lugar, tenemos el controlador periódico. Este FB (se ha llamado "Time_driven_PID"), que podemos ver en la figura 38, es el más simple de los tres que vamos a describir en esta

sección y constituye la base para los otros controladores PID. En cuanto a los parámetros, cuenta con los descritos en la parte general y además con un booleano que indica cuando se ha saturado la acción de control.

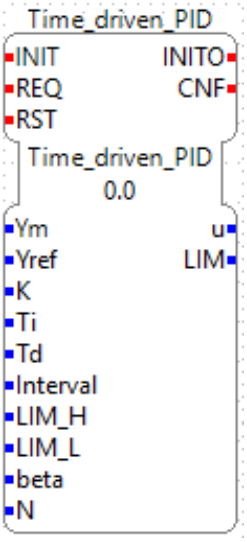


Figura 38: FB del controlador periódico "Time_driven_PID"

Este FB presenta un ECC (representado en la figura 39) constituido de cuatro estados (START, INIT, RST y REQ) en color azul. Las transiciones se franquean con la recepción de eventos o con condiciones lógicas (los eventos son INIT, REQ y RST; las condiciones lógicas es el 1, que es un verdadero lógico) también de color azul. En cada estado se ejecuta un algoritmo (REQ o RESET) representado en color amarillo y se puede enviar un evento de salida (INITO o CNF) representado en verde.

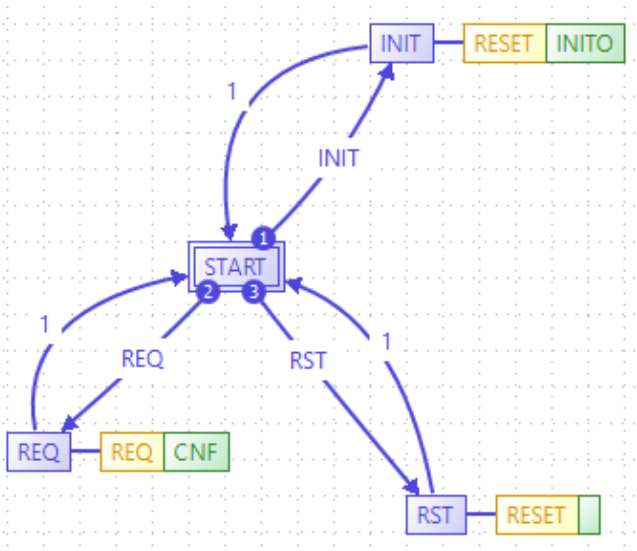


Figura 39: ECC del FB "Time_driven_PID"

En el algoritmo RESET se ponen a cero los parámetros internos del controlador y en el algoritmo REQ se calcula la acción de control del controlador PID con los datos proporcionados.

No nos centraremos en describir los algoritmos de inicialización ni de *reset* porque son simplemente sendas puestas a cero de los parámetros internos del controlador (como por

ejemplo el término integral de la acción de control o el tiempo que ha pasado entre ejecuciones). Pero sí que se tendrá en cuenta el contenido del algoritmo REQ.

En este primer caso, el algoritmo en cuestión no necesita demasiada introducción ya que es el perteneciente al controlador PID periódico, el cual se escribe a continuación:

```
VAR
        up: REAL;
END_VAR;

        up:=K*(beta*Yref-Ym);
        ud:=(Td/(Td+N*Interval))*ud-(K*Td*N/(Td+N*Interval))*(Ym-yold);
        u:=up+ui+ud;

        ui:=ui+(K*Interval/Ti)*(Yref-Ym);
        yold:=Ym;

        IF u<LIM_L THEN
                u:=0.0;
        END_IF;

        IF u>LIM_H  THEN
                u:=100.0;
        END_IF;
```

Como se puede ver en el código, se trata de un controlador PID donde se aplica la acción correspondiente al término integral en la ejecución del siguiente periodo. Además, se observa que la acción de control está saturada entre los valores LIM_H y LIM_L.

En segundo lugar, tenemos el controlador de Årzén. De este controlador se dispone de dos versiones, una que necesita de una llamada periódica al bloque (llamado “Event_driven_PID”) y otro que se puede llamar aperiódicamente (llamado “Event_driven_PID_Ap_Ex”), ambos bloques representados en la figura 40.

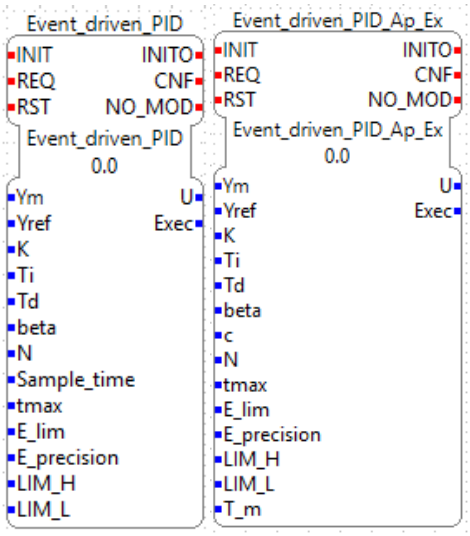


Figura 40: FB del controlador de Årzén con llamada periódica (izquierda) y aperiódica (derecha)

La diferencia entre estos dos bloques reside únicamente en la forma en la que se calcula el tiempo entre ejecuciones. Mientras que en el que necesita una llamada periódica el tiempo es un dato fijo que no cambia entre ejecuciones y que se va incrementando en cada llamada al FB, en el que permite una llamada aperiódica el tiempo se recalcula internamente pasando como dato el tiempo del procesador. La justificación de la existencia de estas dos modelizaciones es que, por un lado, en un entorno de POSIX no se puede acceder al tiempo del procesador, y por otro lado, que leer el tiempo del procesador cuesta un tiempo que si sabemos que es fijo, nos podemos ahorrar.

En la figura 41 se muestra uno de los dos ECC pero, realmente no existe ninguna diferencia en la estructura interna de un FB y del otro. Como rasgo distintivo en la implementación de los CBE en este proyecto podemos ver que las condiciones de ejecución de los controladores se encuentran en las condiciones de flanqueo de las transiciones (hay que darse cuenta que una condición es la otra negada, con lo cual siempre se volverá al punto de partida), es decir, que implementamos las características de ejecución del controlador de Årzén con los elementos que definen la norma IEC-61499.

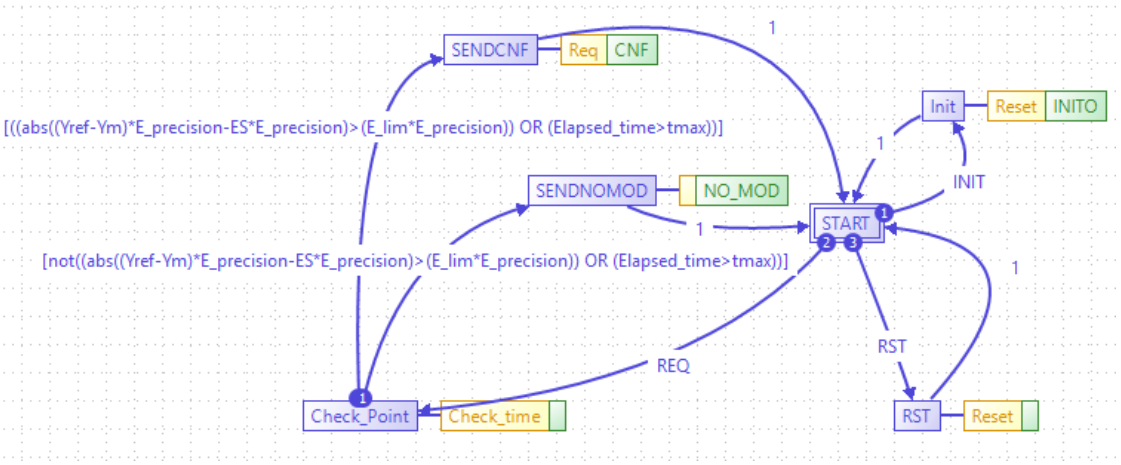


Figura 41: ECC del FB "Event_driven_PID_Ap_Ex"

En cuanto al contenido de los algoritmos, Reset tiene la misma función que en el controlador anterior, Check_time se encarga de calcular el tiempo que ha pasado entre ejecuciones y Req implementa los algoritmos de ejecución del algoritmo de control, los cuales se trataran más adelante.

Con respecto al código original, se ha añadido para las transiciones, la variable E_precision, ya que la función abs devuelve un entero y no un real, por lo tanto hay que dotar de algunas cifras significativas a este testeo de las condiciones de ejecución.

En cuanto al código del controlador de Årzén, podemos encontrarlo detallado en su artículo (Årzén, 1999). Como ya se ha comentado en trabajos posteriores, este controlador ha sido criticado por diferentes autores en especial por la forma en que se calcula el término integral.

En este trabajo se ha tomado como base el controlador de Årzén y se han añadido algunas mejoras en su código, el cual se presenta a continuación.

```
VAR
        up: REAL;
```

```
END_VAR

        ES:=Yref-Ym;
        up:=K*(beta*Yref-Ym);
        UD:=Td/(Td+N*Elapsed_time)*UD+Td/(Td+N*Elapsed_time)*K*N*(c*(Yref-YrefOLD)-(Ym-YOLD));
        UI:=UI+K/Ti*Elapsed_time*(ES);
        U:= up +UD +UI;

        IF U>LIM_H THEN
                U:=100.0;
                IF (ES>0)THEN
                        UI:=UI-K/Ti*Elapsed_time*(ES);
                END_IF;

        END_IF;

        IF U<LIM_L THEN
                U:=0.0;
                IF (ES<0)THEN
                        UI:=UI-K/Ti*Elapsed_time*(ES);
                END_IF;
        END_IF;

        YOLD:=Ym;
        YrefOLD:=Yref;
        Exec:=1;
        LAST_TIME:=T_m;
```

En comparación con el controlador que presenta Årzén en su artículo, este algoritmo presenta varias mejoras. En primer lugar la adición de un término para ponderar la acción del término derivativo (c). En segundo lugar, el cálculo de la acción integral se produce antes de la asignación de esta a la acción de control y se aplica, por tanto, en este cálculo de la acción integral, no en la siguiente ejecución del código como en el caso del controlador original. Finalmente, se han añadido los mecanismos de saturación de la señal de salida (con LIM_H y LIM_L) y *anti-windup* (revirtiendo la modificación del término integral).

Cabe comentar que tanto en el algoritmo de Årzén como en el algoritmo del controlador de Durand, el cual se presentará a continuación, las condiciones de ejecución o no ejecución vienen implementadas en las transiciones como se ha comentado anteriormente.

Finalmente, nos encontramos con el controlador de Durand. De la misma forma que en el caso del controlador de Årzén y por las mismas razones tenemos dos FB con este controlador, uno con llamada periódica y otro con llamada aperiódica, ambos representados en la figura 42.

La estructura interna de estos controladores (llamados “Durand_PID” y “Durand_PID_Ap_Ex”), representada en la figura 43, es igual a la del controlador de Årzén, la única modificación significativa con respecto al anterior se introduce en los algoritmos.

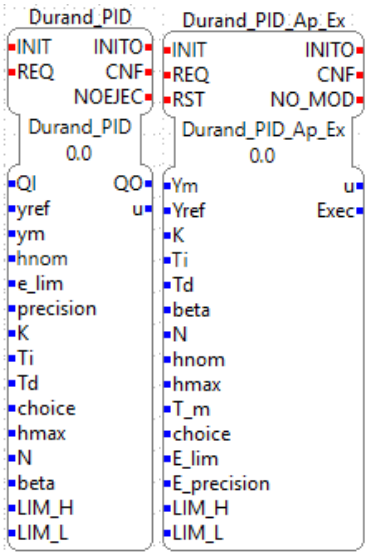


Figura 42: FB del controlador de Durand con llamada periódica (izquierda) y aperiódica (derecha)

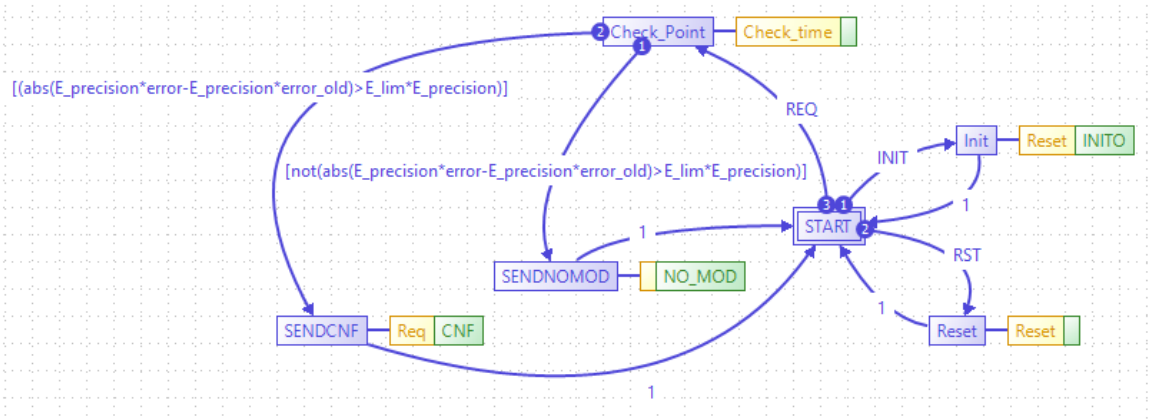


Figura 43: ECC del FB "Durand_PID_Ap_Ex"

Para este ECC se han evaluado diferentes alternativas en cuanto a la modelización, ya que se podría estructurar en varios estados más en función de la elección que se hace para el control según lo especificado en (Durand & Marchand, 2009b), ya que existen cuatro posibilidades. Sin embargo, como no presenta ninguna mejora significativa respecto a la estructura presentada se ha desestimado.

En cuanto al código del controlador de Durand, se ha seguido lo publicado en su artículo (Durand & Marchand, 2009b). Sin embargo, este algoritmo presenta algunos defectos como veremos posteriormente, en gran medida por la falta de conocimiento en cuanto a la caracterización de sus parámetros. El código de este algoritmo se presenta en las siguientes líneas.

```
VAR
        up: REAL;
END_VAR


        up := K*(beta*Yref-Ym);

        IF choice =1 THEN
```

```
            ui := ui+K/Ti*hact*error;
END_IF;
IF choice = 2 THEN
        IF hact>=hmax  THEN
                he :=(hact-hnom)*e_lim+hnom*error;
        ELSE
                he := hact*error;
        END_IF;
        ui := ui +K/Ti*he;
END_IF;
IF choice = 3 THEN
        hact_i :=hact*(exp(hnom-hact));
        ui := ui +K/Ti*hact_i*error;
END_IF;
IF choice =4 THEN
        IF hact>=hmax THEN
                hact_i :=hact*(exp(hnom-hact));
                he :=(hact_i-hnom)*e_lim+hnom*error;
        ELSE
                he :=hact*error;
        END_IF;
        ui := ui +K/Ti*he;
END_IF;

ud := Td/(N*hact+Td)*ud-K*Td*N/(N*hact+Td)*(ym-y_old);

u := up+ui+ud;

IF u>LIM_H THEN
        u :=LIM_H;
END_IF;
IF u<LIM_L THEN
        u :=LIM_L;
END_IF;
Exec:=1;
error_old :=error;
y_old :=ym;
LAST_TM:=T_m;
```

Como se puede ver se ha mantenido el código propuesto por Durand añadiendo únicamente una saturación en la acción de control. Los fallos que se han detectado en este controlador son dos. En primer lugar, el relativo a una de las condiciones de ejecución. Si bien es cierto que la condición de ejecución del controlador de Durand se centra única y exclusivamente en la diferencia entre el error presente y el pasado (condición puramente SSOD), esto no es suficiente, ya que si el control se realiza sobre un sistema que puede estabilizarse en otros puntos este controlador no es capaz de detectar este hecho y se presentaría un error de posición. En segundo lugar, el segundo problema deriva de la forma en la que se realiza el control en función de los parámetros, la cual tiene como resultado la generación de sobreoscilaciones cuando se produce un cambio en la referencia. Sobre este aspecto comentar que no se trata tanto de un fallo del controlador en sí mismo, sino de una falta de conocimiento en cuanto a una caracterización robusta de sus parámetros, ya que haciendo un estudio en profundidad sobre esta caracterización cabría la posibilidad de que se podrían obtener parámetros (en concreto h_{max}) que condujeran a una respuesta del sistema más estable y robusta.

1.8.2.3 FB de salida

Para actuar sobre nuestro sistema se ha utilizado una señal PWM y se ha implementado con un FB de tipo SIFB (figura 44) porque debemos actuar sobre los parámetros del procesador.

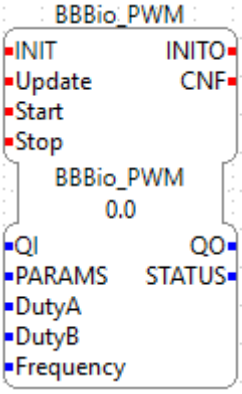


Figura 44: FB para señal PWM "BBBio_PWM"

Este FB (se ha llamado “BBBio_PWM”) tiene como parámetros un booleano para habilitar su ejecución, una cadena de caracteres para indicar que módulo se debe activar (se dispone de varios módulos y cada uno puede proporcionar dos salidas PWM), dos reales para los ciclos de trabajo de las señales de salida y un real adicional para especificar la frecuencia de la señal. Como parámetros de salida tiene un booleano y una cadena de caracteres para indicar como se ha desarrollado tanto la inicialización como la ejecución del bloque.

En cuanto al control de ejecución de este FB, tiene como entradas un evento INIT para inicializar el bloque, un evento Update para actualizar los parámetros (ciclo de trabajo y frecuencia), un evento Start para iniciar la ejecución del bloque y un evento Stop para detenerla. Como eventos de salida este FB cuenta con la confirmación de la inicialización INITO y un evento CNF para indicar el fin de la etapa de ejecución.

1.8.2.4 FB auxiliares

Subsidiariamente, para la correcta ejecución de la aplicación hay que añadir algunos FB que, aunque no formen parte directamente del control, hay que añadirlos para asegurar un correcto funcionamiento. Estos bloques son normalmente del tipo SIFB y en algunos casos son FB que tienen un equivalente en las librerías por defecto de 4DIAC, pero que por desgracia, no soportan ni aseguran una ejecución en tiempo real. En las líneas siguientes describiremos someramente la funcionalidad de los bloques, ya que, desde el punto de vista del control, interés principal del proyecto, no vale la pena entrar demasiado en detalle.

En primer lugar, se han implementado una serie de FB encarados de la generación y gestión de *threads* en Xenomai (representados todos ellos en la figura 45). En concreto, se ha desarrollado un FB que con la recepción de un evento crea un *thread* con una prioridad y una política específicas (llamado “XENO_RT_E_EO”). Tenemos también un FB que envía de forma periódica eventos y crea un *thread* con las propiedades especificadas (llamado “XENO_RT_E_CYCLE”). Se ha desarrollado también un FB que con la recepción de un evento crea dos *threads* y envía un evento a cada uno según las propiedades de prioridad y política especificadas (llamado

"XENO_RT_E_SPLIT"). En la misma línea que el bloque anterior se ha desarrollado un FB que en vez de crear dos *threads* y enviar dos eventos, crea y envía 4 (llamado "XENO_RT_E_SPLIT_4").

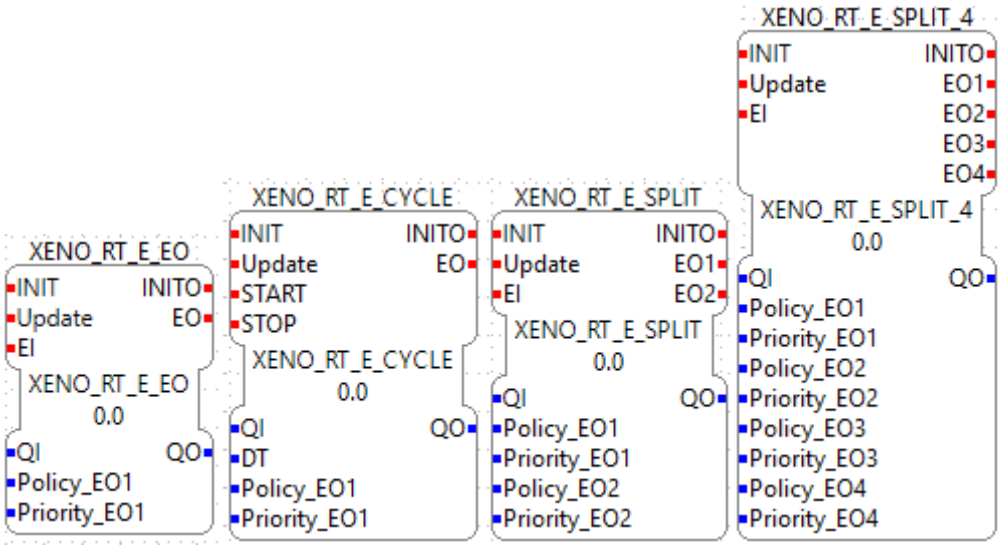


Figura 45: FB encargados de la gestión de threads en Xenomai

Para información detallada sobre cómo se desarrolla la ejecución en tiempo real de los FB en la norma IEC-61499 se recomienda encarecidamente consultar el libro (Zoitl, 2008), cuyo autor es quien desarrolló la propia norma, y donde se explica de forma detallada las diferentes correspondencias que pueden haber entre *threads* y FB. En nuestro caso, un *thread* se corresponde con una cadena de FB, definiendo cadena de FB como el conjunto de todos los bloques comprendidos entre un FB que envía eventos hasta un FB sumidero, que no envía eventos y con el que finaliza la cadena.

Por otro lado, se ha desarrollado un FB que nos permite obtener el tiempo del procesador con precisión de nanosegundos (llamado "F_RT_CLOCK_NS") y un FB que a partir de las medidas y de las referencias de cuatro sistemas da como resultado una prioridad diferente para cada uno de ellos en función del error (llamado "Priority_allocator"), ambos representados en la figura 46. Este último FB está pensado para conectar sus datos de salida a las entradas de prioridad del FB XENO_RT_E_SPLIT_4.

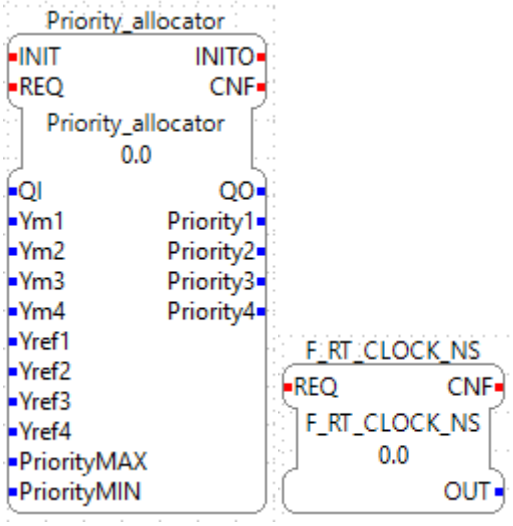


Figura 46: FB auxiliares para el cálculo y obtención de parámetros para el control

1.8.3 Resultados del control

1.8.3.1 Ajuste de controladores

En cuanto al método de ajuste de los algoritmos de CBE utilizados en este proyecto existe un vacío teórico. Hasta ahora, si bien tanto Årzén como Durand han propuesto sus algoritmos PID basados en eventos, ellos mismos no han especificado un método de ajuste genérico para sus controladores.

Dada la falta de métodos de ajuste específicos para este tipo de controladores, una alternativa razonable, aunque no exenta de limitaciones, es el uso de métodos de ajuste tradicionales de PID continuos. Aquí hemos considerado dos métodos relativamente nuevos que intentan minimizar el IA cumpliendo unos requisitos de robustez dados. En concreto se han usado los métodos propuestos en (Sanchis, et al., 2010) y el método AMIGO, detallado en (Åström & Hägglund, 2004).

Para probar los diferentes métodos utilizamos la función de transferencia de uno de los sistemas que vamos a controlar (sistema 1 de la tabla 8) ya que se considera bastante genérico para realizar este estudio. En las siguientes simulaciones se mostrará solo los resultados para este sistema (al cual se le han aplicado las escalas relativas a la acción de control y a la adecuación para la lectura) con la función de transferencia siguiente:

$$G(s) = \frac{0{,}612}{s^2 + 9s + 34}$$

El cual presenta una respuesta ante un escalón como la mostrada en la figura 47.

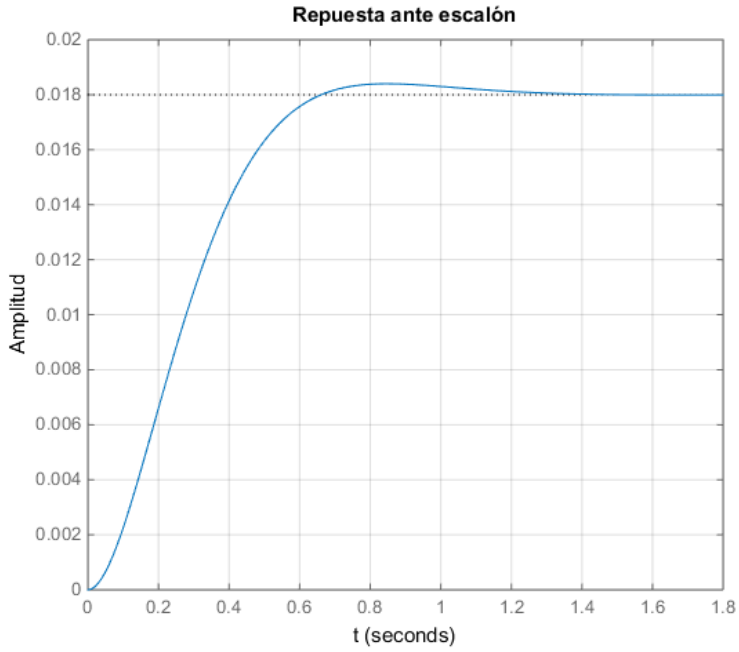


Figura 47: Respuesta ante escalón de un sistema a controlar

Aplicando el método expuesto en (Sanchis, et al., 2010), el cual trata de obtener los parámetros del controlador resolviendo un problema de optimización, se obtienen los siguientes parámetros:

$$K_p = 7130 \quad T_d = 0{,}0248 \quad T_i = 0{,}1039 \quad N = 10$$

Paralelamente, pasamos a obtener los parámetros del controlador con el método AMIGO, publicado en (Åström & Hägglund, 2004).

Este método tiene como principal virtud que los parámetros resultantes confieren al controlador una gran robustez, siendo esta una de las razones por las que se ha tenido en cuenta este método. Los parámetros obtenidos de aplicar la metodología son:

$$K_p = 47{,}772 \quad T_d = 0{,}0445 \quad T_i = 0{,}1922 \quad N = 10$$

Como se puede ver la ganancia proporcional del primer método es muy elevada con respecto a la ganancia obtenida en el segundo controlador. Para probar el impacto de estos parámetros se ha simulado el sistema con los tres controladores estudiados (a saber: periódico, de Årzén y de Durand).

En primer lugar, se simulan los controladores con los parámetros obtenidos por el primer método. En la figura 48 se observa la respuesta de los sistemas con estos parámetros así como una línea vertical cada vez que su respectivo controlador basado en eventos se ejecuta.

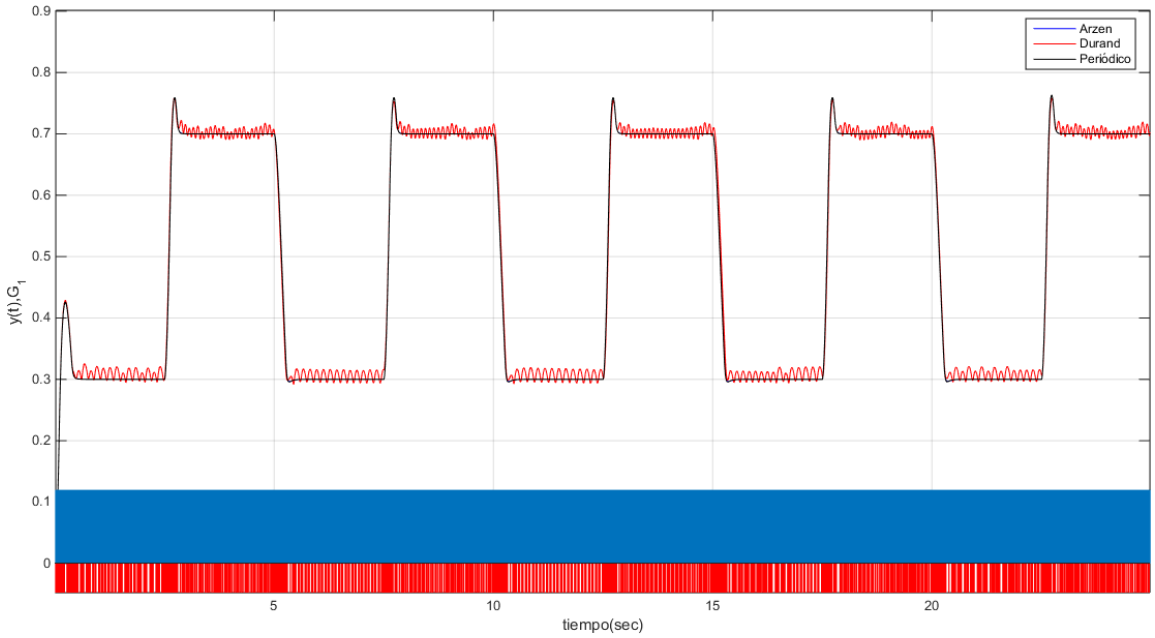


Figura 48: Simulación del sistema con los parametros de minimización del IAE

Aunque en la imagen no se pueda apreciar con claridad, tanto el controlador de Årzén como el de Durand se ejecutan la mayor parte del tiempo con el periodo nominal, especialmente cuando hay un cambio de referencia, aunque después, el controlador de Durand puede pasar algún tiempo sin ejecutarse.

Sin embargo, en el caso del controlador de Durand observamos que esta elección presenta un ciclo límite con lo cual debemos descartar estos parámetros para el controlador de Durand.

Para realizar la comparación, pasamos ahora a comprobar cuál es la respuesta resultante de la simulación de los controladores en el caso de emplear los parámetros obtenidos con el método AMIGO. Las respuestas de estos controladores quedan representadas en la figura 49.

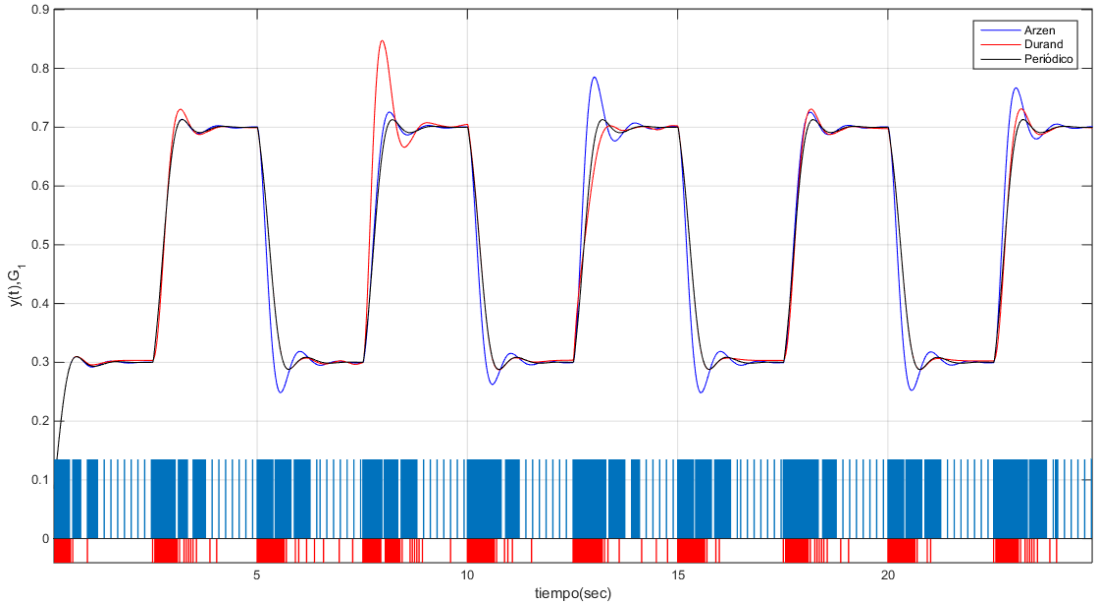


Figura 49: Simulación del sistema con los parámetros del método AMIGO

Como se puede ver en esta figura los controladores no deben hacer tantos cálculos como en los casos anteriores y podemos ver, por ejemplo, que el controlador de Årzén llega a ejecutarse por tiempo máximo en vez de por condiciones basadas en el error o que el controlador de Durand pasa largas etapas sin ejecutarse.

En contraposición a los parámetros utilizados en el caso anterior, para esta simulación el controlador de Durand no oscila, por lo tanto, a priori significa que andamos en el buen camino para la obtención de los parámetros del controlador. Sin embargo, en las simulaciones anteriores no se ha introducido ningún tipo de ruido en la señal. Por lo tanto, en las siguientes simulaciones introduciremos un error aleatorio en la señal medida con valor absoluto máximo δ.

En primer lugar, para el caso del método de minimización del IAE obtenemos la respuesta mostrada en la figura 50.

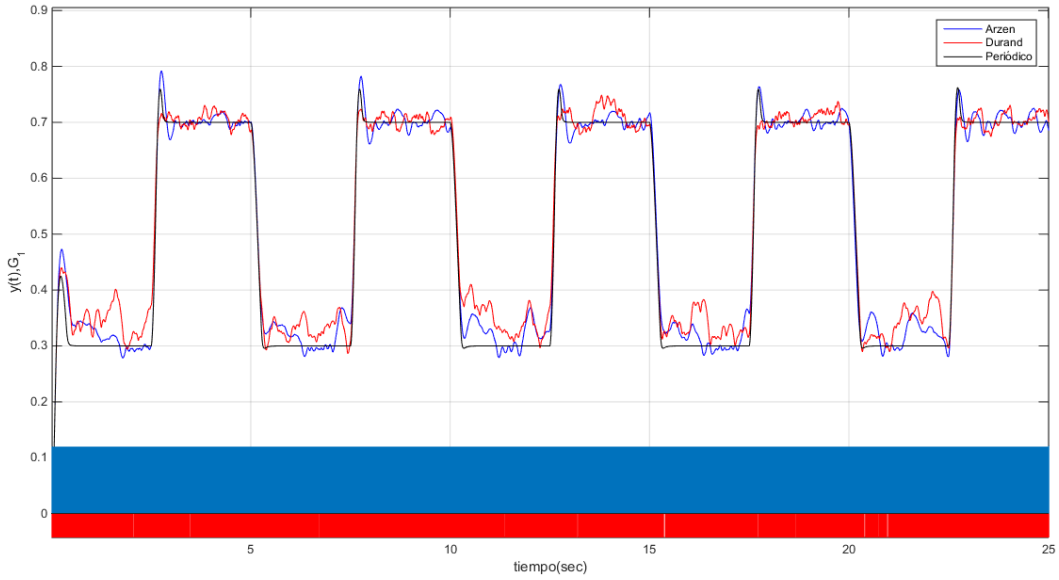


Figura 50: Simulación del sistema con los parametros de minimización del IAE con ruido

Como podemos ver ahora ni el controlador de Durand ni el de Årzén son capaces de controlar el sistema. Esto es debido a la brusquedad del controlador y por tanto a los cambios bruscos en la acción de control, la cual se pude observar en la figura 51.

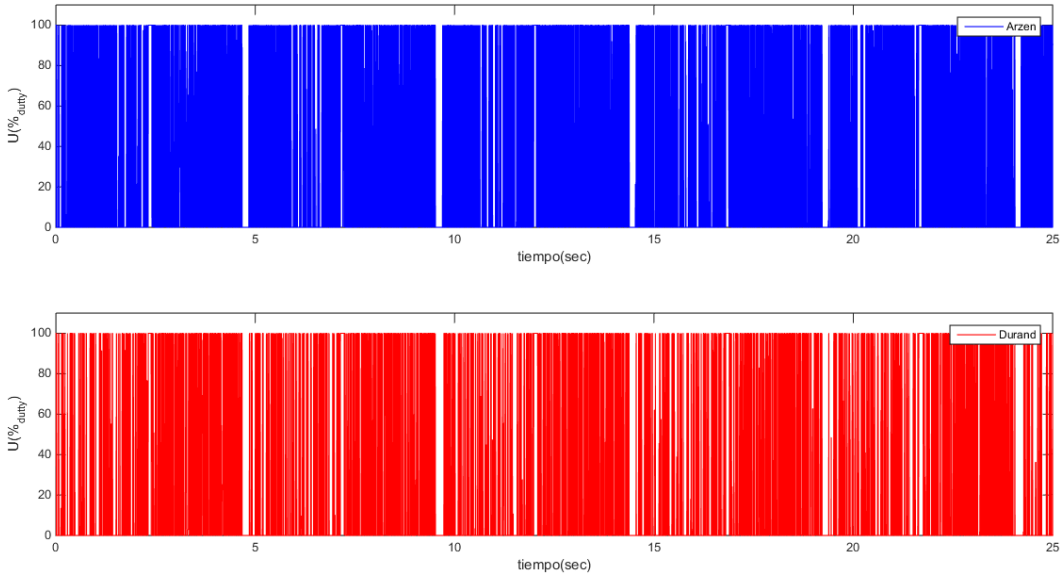


Figura 51: Acción de control de los controladores de Årzén y Durand

Para finalizar las comparaciones, se ha simulado también la respuesta del sistema con una lectura ruidosa para los parámetros del controlador obtenidos con el método AMIGO, obteniendo la respuesta mostrada en la figura 52.

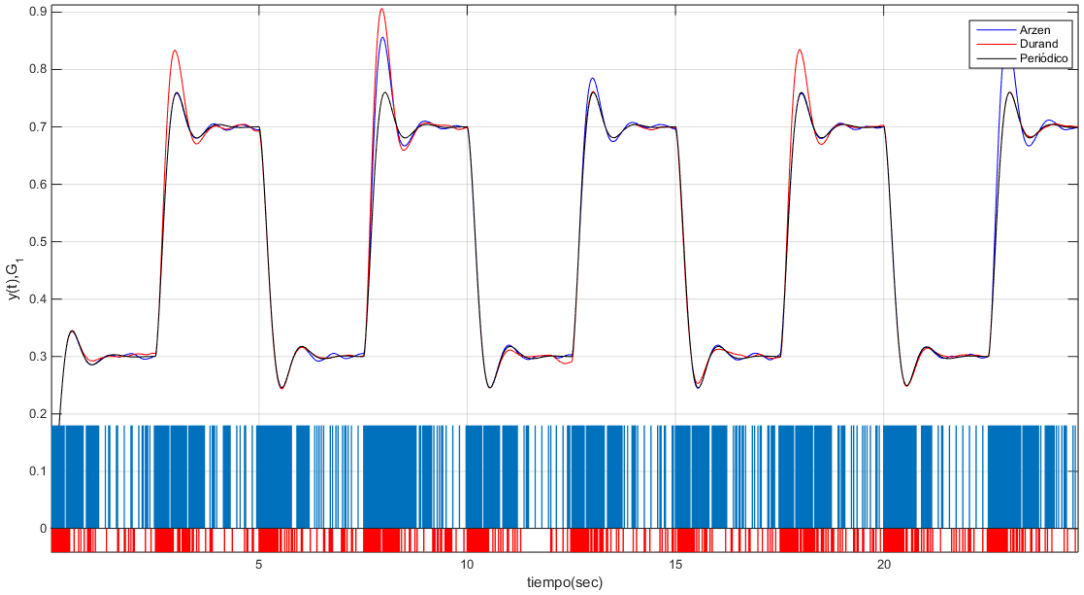


Figura 52: Simulación del sistema con los parámetros del método AMIGO con ruido

Como se puede ver en la figura 52, con los parámetros proporcionados por este controlador sí que se puede controlar el sistema cuando se introduce error en la medida, teniendo como única repercusión que los controladores deben ejecutarse más veces que en el caso donde no tenemos ruido a la entrada.

Como punto final a esta comparación, se pude ver muy claramente que el controlador sintonizado con el método de minimización del IAE es mucho más agresivo que el controlador

con los parámetros del método AMIGO, ya que teniendo ambos el mismo error aleatorio en la señal medida, al tener el primero una ganancia proporcional 150 veces superior a la del segundo, esta perturbación se ha amplificado de igual forma causando que los sistemas no puedan controlarse.

Esta sección, en la que se han probado distintos métodos de ajuste para controladores continuos para caracterizar los controladores basados en eventos de ejecución periódica, no hace más que evidenciar la falta de métodos sistemáticos para la caracterización de los parámetros de los controladores periódicos basados en eventos, como los utilizados en este trabajo.

1.8.3.2 Análisis de la respuesta de los controladores desarrollados

Para esta parte de los resultados se va a utilizar como sistema a controlar el sistema 1 presentado en la tabla 8. Se ha escogido este sistema ya que se considera que es lo suficientemente representativo como para que los resultados obtenidos con el mismo sirvan como resultados generales para los demás.

1.8.3.2.1 PID periódico

Como se ha comentado anteriormente se ha utilizado el método AMIGO, detallado en (Åström & Hägglund, 2004), para la caracterización de los parámetros del controlador debido a la gran robustez que ofrecen los controladores. Como recordatorio, los parámetros que se han obtenido son:

$$K = 47.772 \quad T_i = 0.1922 \quad T_d = 0.0445 \quad N = 10$$

Con respecto a la implementación de los algoritmos, esta se realiza única y exclusivamente mediante una red de FB. En la figura 53, se representa la modelización completa para el caso de la aplicación del controlador PID periódico. En ella se han marcado varias zonas con colores indicando la funcionalidad de cada una. En azul se ha marcado la red de FB encargada de realizar el control, en verde se ha marcado la parte encargada de realizar el cambio en la referencia para el controlador y en amarillo se ha marcado la parte de guardado de los datos, la cual se realiza con un FB incluido en las librerías por defecto de 4DIAC.

Los FB que no se han enmarcado en ningún grupo son un bloque de espera, introducido por si es necesario que mientras se acaben de realizar el resto de inicializaciones de los bloques; y un bloque que recibe eventos directamente del recurso y que los transmite al resto de la aplicación.

En las páginas siguientes se detallarán las secciones relevantes de esta modelización así como el funcionamiento completo de la misma.

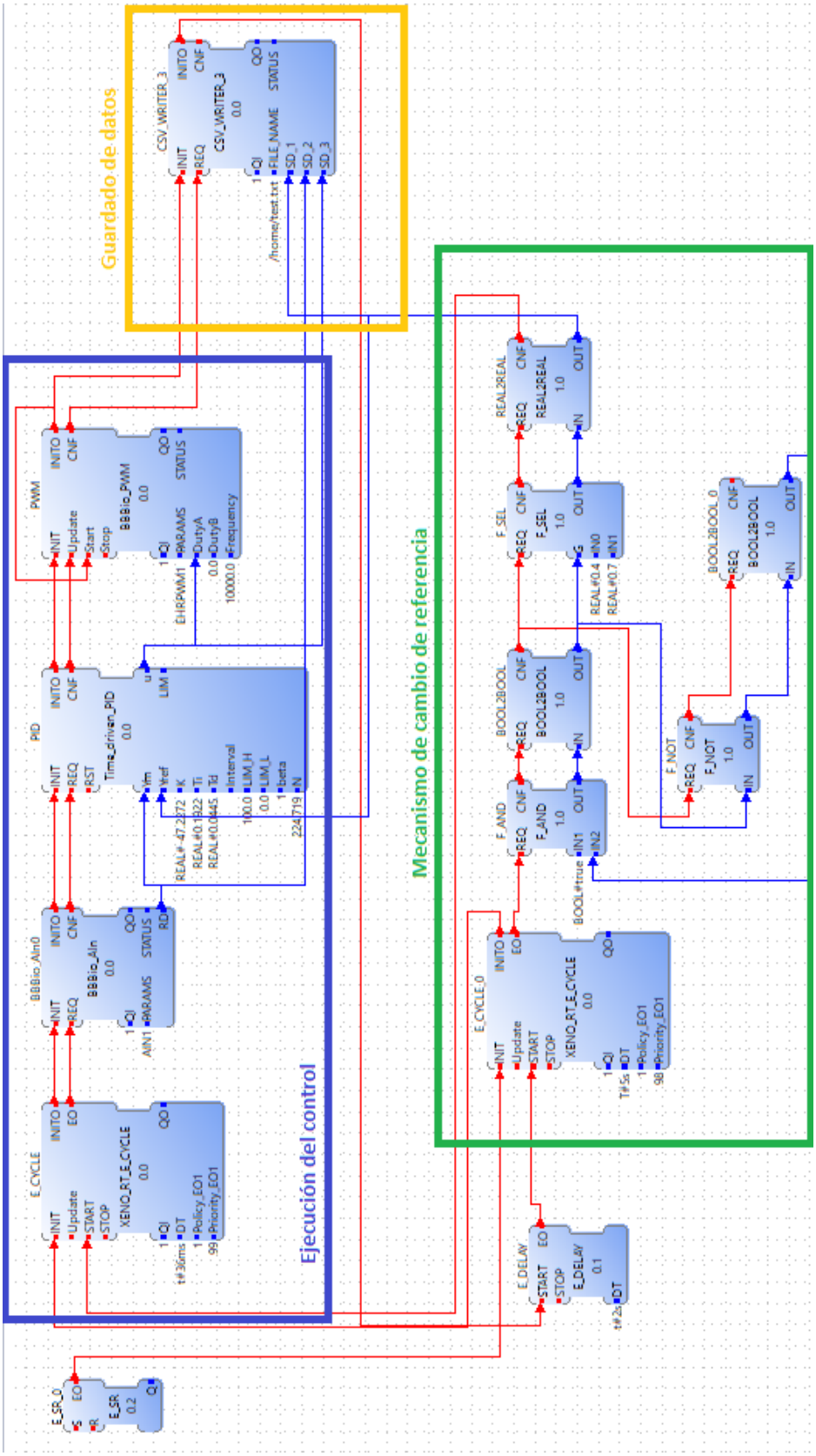


Figura 53: Modelización completa de la aplicación del control periódico

En primer lugar, hay una parte, que es común a todas las modelizaciones, que es el mecanismo de cambio en la referencia. Como es común a todos los algoritmos, la mostraremos una sola vez

para no repetir información y dar más importancia al resto de elementos. La modelización del cambio de referencia (mostrada en la figura 54), consiste en un algoritmo formado por un FB que envía de forma periódica eventos (bloque "E_CYCLE_0") que activan un FB que hace una operación AND (bloque "F_AND") entre un valor verdadero y el valor enviado en la ejecución anterior del algoritmo negado (con un bloque "F_NOT"), de esta forma se obtiene un valor que a cada iteración cambia entre verdadero y falso. Este valor, entra como un dato en un FB de selección (bloque "F_SEL") que en función de este valor da una salida u otra de entre las que se han especificado (y que son los valores de la referencia que queremos). Además se han añadido algunos bloques que se encargan de la conversión de datos a un tipo en concreto (bloques de tipo "BOOL2BOOL" y "REAL2REAL"), ya que la salida de los bloques anteriores es de tipo ANY.

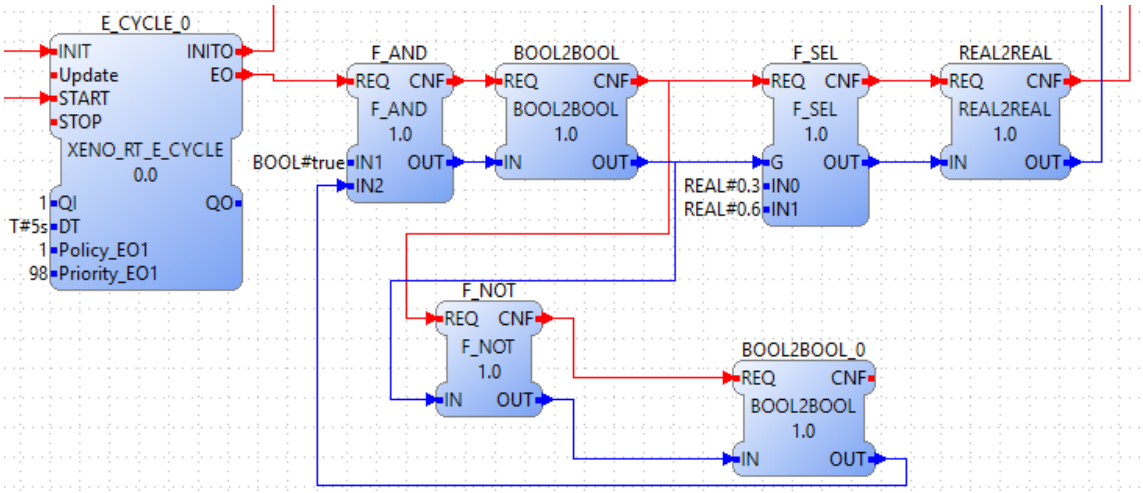


Figura 54: Mecanismo de cambio de referencia

Para la implementación del algoritmo de control (representado en la figura 55), se ha utilizado un FB que envía de forma periódica eventos (con el periodo de muestreo del sistema), un FB que lee el pin analógico donde va conectada la salida del sistema, un FB con el controlador PID periódico, el cual da una salida saturada entre 0 y 100 para regular el ciclo de trabajo de la señal PWM, y finalmente se guardan los datos para poder visualizarlos (este guardado de datos se omitirá en las siguientes figuras ya que no aporta nada al control).

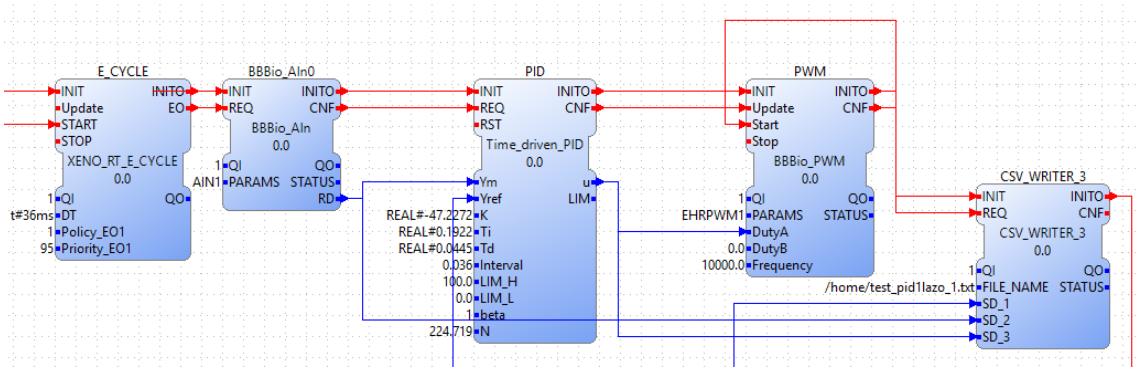


Figura 55: Implementación del PID periódico

En cuanto a los resultados obtenidos de ejecutar este controlador en este sistema, podemos verlos la figura 56, donde se puede ver la respuesta del sistema así como la acción de control que se ha utilizado. Mirando esta figura, se puede observar la suavidad con la que el controlador

corrige la acción de control para seguir la referencia sin reaccionar a posibles perturbaciones (como la perturbación aleatoria que se produce en torno al segundo 27,5).

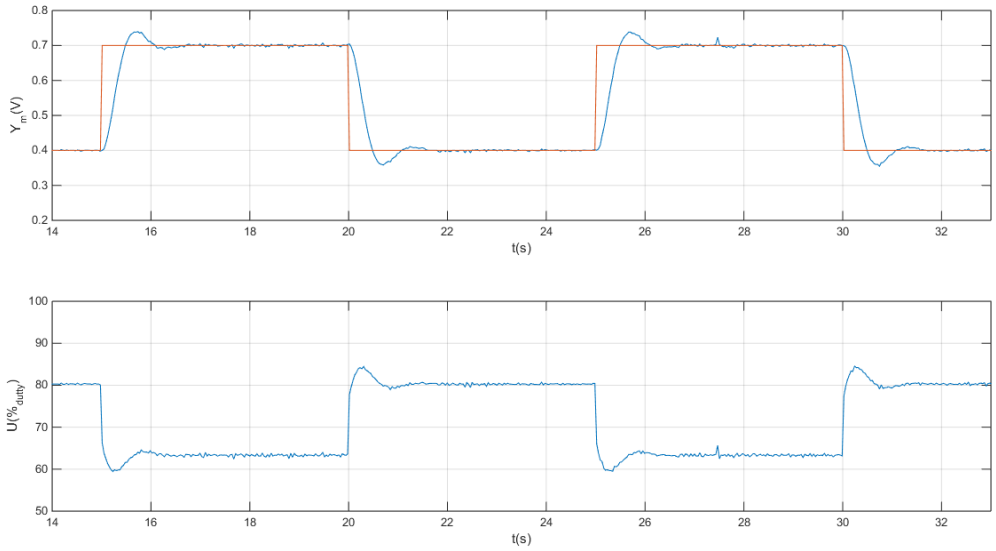


Figura 56: Respuesta del controlador PID periódico

En cuanto al consumo de los recursos (tiempo de cálculo del procesador), vemos en la figura 57 todos los procesos que se ejecutan en la BeagleBone Black durante la ejecución de la aplicación. En esta figura se puede observar que los cuatro *threads* inherentes a la ejecución de FORTE con Xenomai que son los procesos con PID (en este contexto PID se refiere al identificador del proceso) del 1609 al 1619, después vemos el *thread* encarado de la inicialización de todos los FB que necesitan una inicialización, con PID 1701 (este *thread* es hijo de uno de los *threads* inherentes a FORTE, el *thread* "ecet", por tanto hereda sus características) y finalmente vemos el cambio de la referencia (PID 1702) que tiene una prioridad de 98, y el consumo de la ejecución del control (PID 1706) que tiene una prioridad de 95 y que no supera el 0.8 % del tiempo del tiempo de cálculo del procesador.

```
Every 1.0s: cat stat

CPU  PID    MSW        CSW        PF    STAT       %CPU  NAME
  0  0      0          134745     0     00500080   98.3  ROOT
  0  1609   3          4          0     00300182    0.0  forte
  0  1611   1          134374     0     00300184    0.6  forte
  0  1618   83         176        0     00300182    0.0  forte
  0  1619   295        301        0     00300380    0.0  forte
  0  1701   6          9          1     00300182    0.0  forte
  0  1702   1          24         0     00300182    0.0  forte
  0  1706   1          11001      0     00300182    0.8  forte
  0  0      0          174998     0     00000000    0.2  IRQ67: [tim



Every 1.0s: cat sched

CPU  PID    CLASS  PRI      TIMEOUT   TIMEBASE    STAT       NAME
  0  0      idle    -1      -         master      R          ROOT
  0  1609   rt      99      -         master      W          forte
  0  1611   rt      99      781us     master      D          forte
  0  1618   rt      90      -         master      W          forte
  0  1619   rt      90      -         master      X          forte
  0  1701   rt      90      -         master      W          forte
  0  1702   rt      98      -         master      W          forte
  0  1706   rt      95      -         master      W          forte
```

Figura 57: Estado del procesador durante la ejecución del PID periódico

A pesar de la utilidad de saber cuánto tiempo del procesador consume cada *thread*, no se mostrará en los siguientes controladores PID, en primer lugar, porque el porcentaje de consumo del procesador en un CBE no es constante en toda la ejecución porque la ejecución de los inspectores no es regular, y en segundo lugar, porque los porcentajes son muy bajos.

1.8.3.2.2 PID de Årzén

Como se ha comentado en el análisis de soluciones, la teoría para sintonizar los controladores PID basados en eventos no está muy desarrollada. De hecho, no existen unos estudios que indiquen cómo se comporta un controlador en cuanto a la robustez y a la aparición de ciclos límites.

Por esa razón, su gran robustez, se ha decidido usar el método AMIGO. Para este controlador solo queda ahora ajustar los parámetros δ y t_{max} propios a este controlador para caracterizarlo completamente. Se ha escogido un valor de δ aceptable, que se ajusta a los valores de operación del sistema, y se ha escogido un valor para el tiempo máximo que es cinco veces el tiempo nominal de ejecución:

$$\delta = 0.01 \quad t_{max} = 200\,ms$$

En cuanto a la implementación del algoritmo (mostrada en la figura 58) es muy similar al caso anterior, la única diferencia consiste en que se ha introducido el bloque F_RT_CLOCK_NS para pasarle el tiempo del procesador al bloque PID, el cual calcula internamente el tiempo entre ejecuciones. Para este apartado se ha utilizado el bloque Event_driven_PID_Ap_Ex, pero en caso de utilizar el otro FB desarrollado la modelización hubiera sido igual a la del caso anterior.

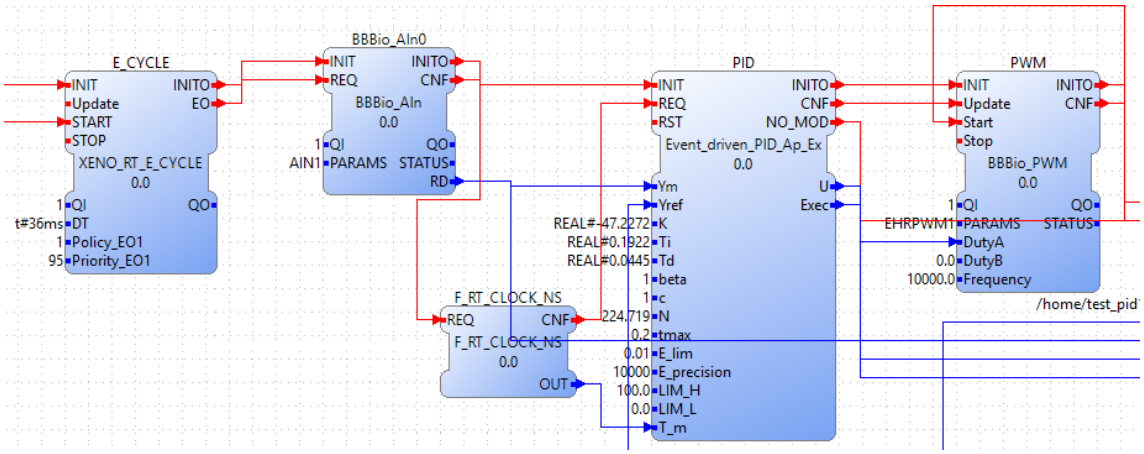


Figura 58: Implementación del PID de Årzén

En cuanto a los resultados, se ha utilizado el FB de control presentado en la modelización, pero los resultados obtenidos con el otro FB desarrollado son muy similares. En la figura 59 podemos ver en primer lugar la respuesta ante escalón del sistema junto con las ejecuciones del controlador (marcadas con un trazo vertical en amarillo en cada ejecución) y en segundo lugar la acción de control que da el sistema. Se puede apreciar que el controlador puede actuar correctamente sobre el sistema con menos ejecuciones que en el caso del PID periódico, de hecho, puede observarse que en los instantes posteriores a un cambio en la referencia el controlador se ejecuta con un periodo igual al periodo de muestreo nominal, pero que, a medida

que nos alejamos de este cambio se ejecuta con otro periodo, el fijado por el tiempo máximo que puede pasar sin ejecutarse este controlador. Con respecto a la acción de control, podemos observar que no presenta saturaciones (como en el caso del PID periódico), aunque en este caso es un poco más brusca que en el anterior control.

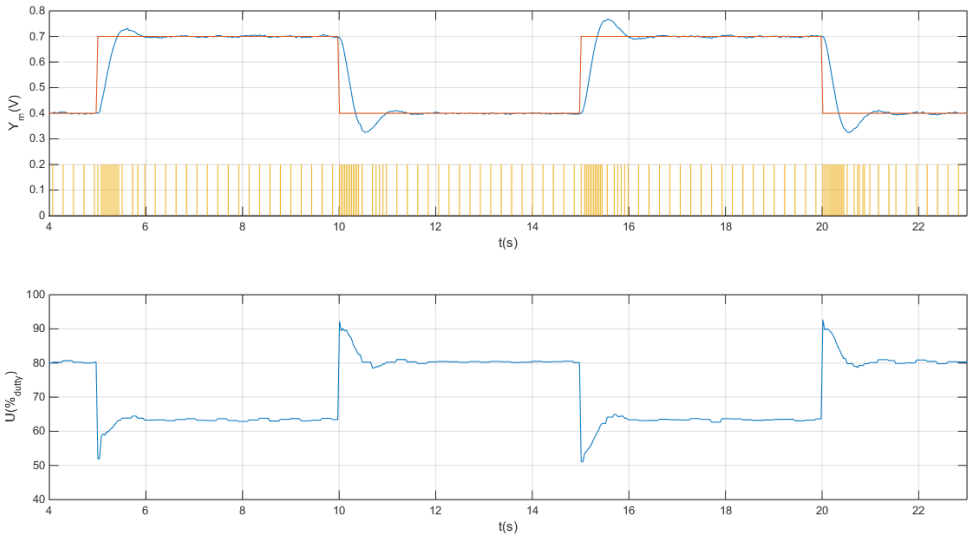


Figura 59: Respuesta del controlador PID de Årzén

1.8.3.2.3 PID de Durand

Para el PID de Durand y como en el caso del controlador de Årzén se utilizará los mismos parámetros que se obtuvieron con el método AMIGO. Esta vez, el tiempo máximo no es un parámetro tan restrictivo como en el controlador de Årzén porque la ejecución de este controlador no depende del tiempo que haya pasado entre ejecuciones, la ejecución depende exclusivamente de la diferencia entre el error medido y el error precedente. Es por esta razón que se puede escoger un tiempo máximo más grande que en el controlador de Årzén (ya que este tiempo máximo solo repercute en el cálculo del término integral de la acción de control). Para realizar los experimentos utilizaremos la elección (*choice*) 4 entre las opciones de control que nos ofrece el controlador de Durand.

Como consecuencia de lo expuesto en el párrafo anterior, la elección de un δ apropiado se convierte en algo más crítico. Se utilizará el mismo valor que en el caso anterior porque es un valor lo suficientemente pequeño para los valores que tratamos en este sistema. Sintetizando:

$$\delta = 0.01 \quad t_{max} = 1\ s$$

En lo concerniente a la implementación de los FB para el control no hay ninguna diferencia con respecto al caso anterior, lo único que se ha cambiado en el modelo ha sido el FB del controlador.

En la figura 60 podemos ver la respuesta del controlador de Durand. Con respecto al controlador de Årzén podemos observar que el número de ejecuciones ha disminuido significativamente de media, se puede ver especialmente en el intervalo que va desde el segundo 15 hasta el 20. Observando la respuesta podemos ver que el sistema puede seguir la referencia pero vemos que en algunos cambios de referencia el controlador puede presentar algún *overshoot* (los defectos de este controlador se tratarán en el apartado siguiente) e incluso la acción de control

puede llegar a saturar. Si omitimos estos problemas, la respuesta del sistema es muy buena y el número de ejecuciones es menor que en el caso anterior.

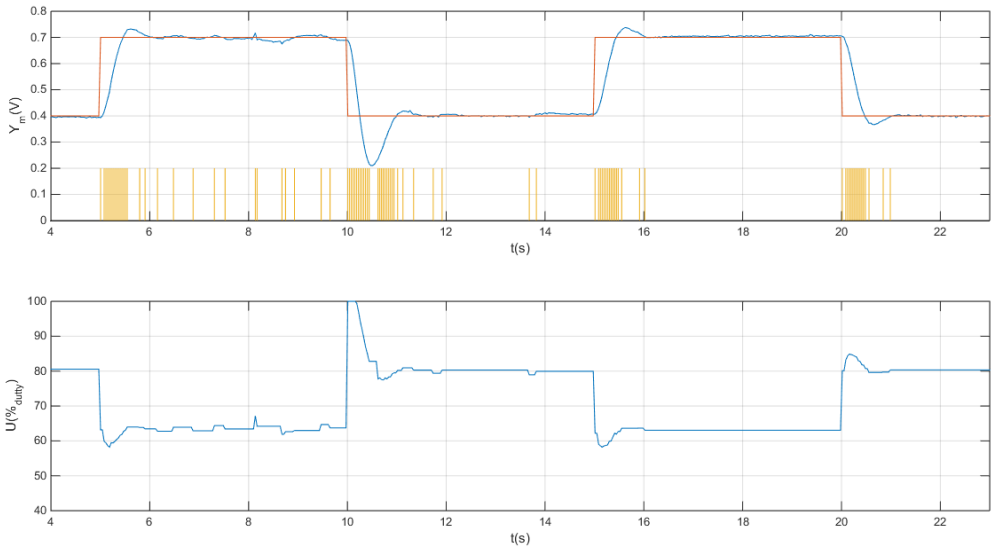


Figura 60: Respuesta del controlador PID de Durand

1.8.3.2.4 Comparación y análisis de resultados

A la vista de los resultados anteriores, podemos ver que los tres controladores PID pueden controlar el sistema de forma satisfactoria. Pero presentan diferencias en cuanto al número de ejecuciones y respuesta las cuales se pueden observar superponiendo los resultados de los experimentos anteriores en la figura 61.

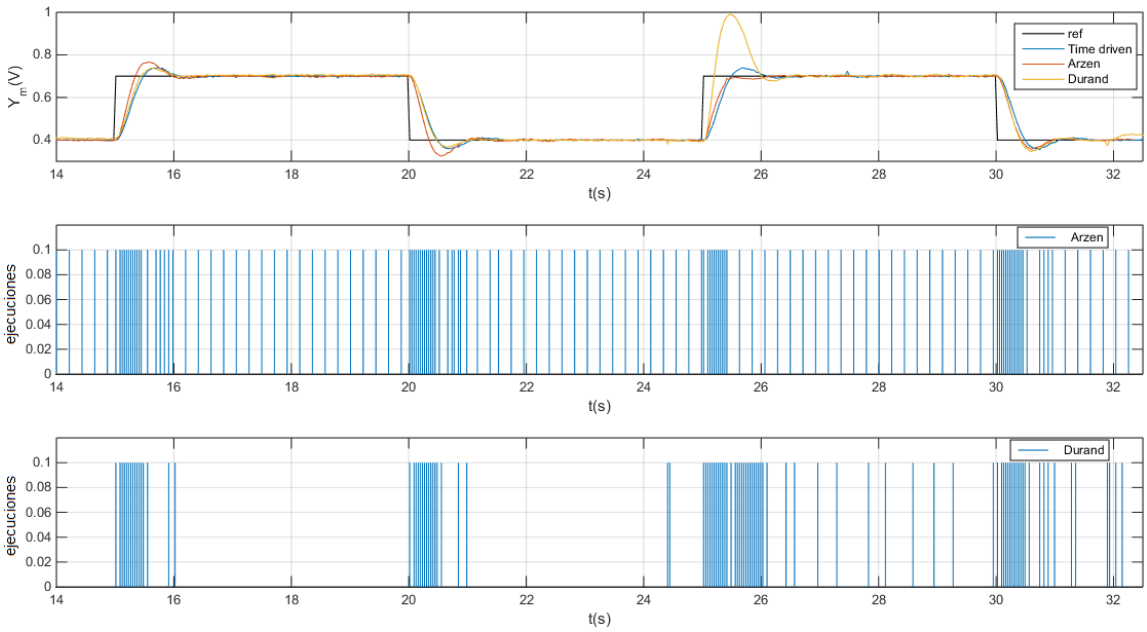


Figura 61: Comparación de los tres controladores PID

En la figura 61 se observa la respuesta de los sistemas y las ejecuciones de los controladores basadas en eventos. Se puede observar con claridad, que el inspector de Årzén se ejecuta de

forma periódica con el periodo fijado por el tiempo máximo cuando el sistema se estabiliza y con el periodo nominal en los alrededores del cambio de referencia. Sin embargo, con respecto al controlador de Durand, se ha querido poner en evidencia los problemas y también las ventajas que se desprenden analizando esta imagen.

Por un lado vemos que necesita muchas menos ejecuciones de media que el controlador de Årzén, y por supuesto que el controlador periódico. Pero por el otro lado, observamos que dependiendo de la casuística de las ejecuciones del controlador (es decir de cuando les llegue un evento) podemos tener una dinámica no deseada, en concreto, en el cambio del valor en la referencia del segundo 25 podemos ver que el sistema presenta una oscilación importante y es debida al evento que llega al controlador alrededor del segundo 24.5, porque, como entre esa ejecución y la siguiente no pasó el tiempo máximo especificado, el controlador realiza un cálculo que hace aumentar demasiado el término integral de la acción de control. Además, como podemos ver hacia el final de la gráfica, si la señal leída se estabiliza dentro del umbral δ que le especificamos al controlador, este no se ejecuta y nos podemos encontrar con un error estático. Esto no sucede con el controlador de Årzén porque introduce una ejecución "periódica". Por estas razones utilizaremos solo el controlador de Årzén para los otros casos de control de este proyecto.

De todas formas, con los datos de que se dispone podemos realizar un análisis temporal en cuanto a los tiempos de ejecución de los controladores basados en eventos con respecto a los controladores periódicos. En primer lugar caracterizamos los tiempos de ejecución de ambos controladores como:

$$t_{perio} = N(t_{AIN} + t_{PID} + t_{PWM})$$

$$t_{ev} = N(\%_{ejec}(t_{AIN} + t_{ejec} + t_{PWM} + t_{CLK}) + (1 - \%_{ejec})(t_{AIN} + t_{noejec} + t_{CLK}))$$

Donde t_{perio} y t_{ev} son los tiempos que consumen las aplicaciones de los controladores periódico y basado en eventos respectivamente y todos los tiempos que aparecen son los tiempos de ejecución de cada uno de los FB que aparecen (se hace distinción entre ejecución y no ejecución del FB del controlador basado en eventos). También se debe tener en cuenta que los CBE no se ejecutan todo el tiempo por eso también debemos introducir el porcentaje de veces que se ejecuta. Igualando las expresiones para saber qué porcentaje de ejecuciones se necesita para que, de media, se consuma el mismo tiempo del procesador que con el controlador periódico obtenemos:

$$t_{perio} = t_{ev}; \quad \%_{ejec} = \frac{t_{PID} + t_{PWM} - t_{CLK} - t_{noejec}}{t_{ejec} + t_{PWM} - t_{noejec}}$$

Utilizando los valores experimentales de los tiempos (obtenidos mediante una caracterización temporal de los FB, detallada en el anexo 2.2.2), se ha obtenido que para tener el mismo tiempo de uso del procesador hace falta que el controlador basado en eventos se ejecute un 77,15% de las veces.

Hace falta puntualizar que este resultado es una media, ya que los controladores basados en eventos cuando deben ejecutarse toman más tiempo de ejecución que los controladores periódicos, ya que deben ajustar los coeficientes internos del controlador.

Para ver realmente cual es el impacto de los controladores basados en eventos con respecto a los controladores periódicos en función del porcentaje de ejecución realizamos el cociente entre los tiempos de ejecución, este cociente $\%_t$ es el porcentaje de utilización temporal de los controladores basados en eventos con respecto a los controladores periódicos.

$$\%_t = \frac{t_{ev}}{t_{perio}}$$

$$\%_t = \frac{N\left(\%_{ejec}\left(t_{AIN} + t_{ejec} + t_{PWM} + t_{CLK}\right) + \left(1 - \%_{ejec}\right)\left(t_{AIN} + t_{noejec} + t_{CLK}\right)\right)}{N\left(t_{AIN} + t_{PID} + t_{PWM}\right)}$$

$$\%_t = \frac{t_{ejec} - t_{noejec} + t_{PWM}}{t_{AIN} + t_{PID} + t_{PWM}} \cdot \%_{ejec} + \frac{t_{AIN} + t_{noejec} + t_{CLK}}{t_{AIN} + t_{PID} + t_{PWM}}$$

Podemos ver la representación gráfica de esta ecuación en la figura 62.

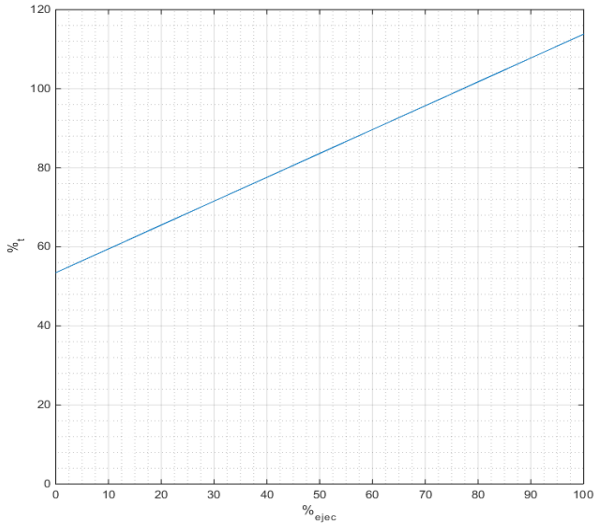


Figura 62: Relación entre el porcentaje de ejecución y el coste temporal de los CBE

Como podemos observar en la figura 62, encontramos el resultado particular anterior, para un 77,15% del porcentaje de ejecución obtenemos el mismo tiempo que usaría un controlador periódico. Además se puede ver que para un 100% del porcentaje de ejecución del controlador basado en eventos se incurre en un sobrecoste temporal del 13,8%. En el caso concreto del controlador de Årzén, la ejecución temporal que se deriva del tiempo máximo, se corresponde con un porcentaje de ejecución del 20%, lo cual significa, que cuando la señal se estabiliza nos permite ahorrar un 35% del tiempo del procesador con respecto al controlador periódico. Sin embargo, para el controlador de Durand, idealmente en estado estacionario no se debería ejecutar nunca, por lo tanto, el consumo descendería en casi un 50%. Esto significa que, aunque en algunos tramos se consuma más tiempo del procesador, el ahorro es mucho más importante que el sobrecoste en el consumo temporal.

Llegados a este punto, y para acabar las comparaciones entre los controladores, vamos a ver cuál es el comportamiento del controlador periódico si lo ejecutamos, en vez de con el periodo nominal, con el tiempo máximo del controlador de Årzén. Podemos observar el resultado en la figura 63.

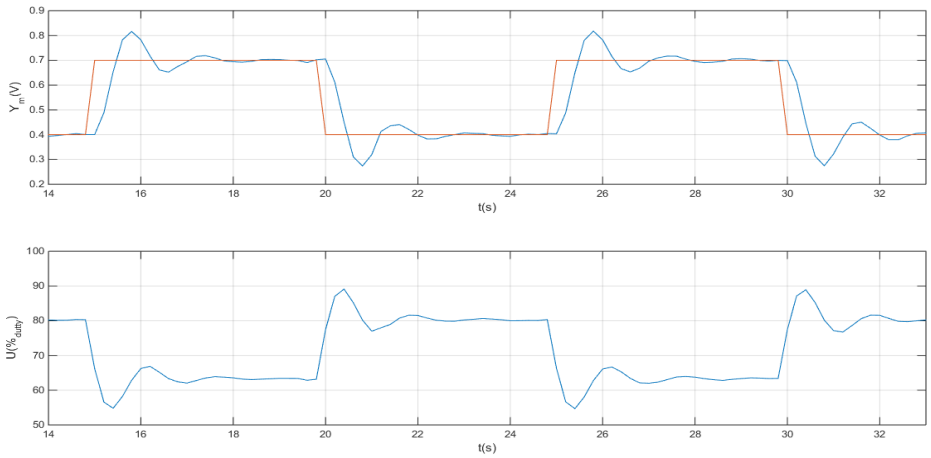


Figura 63: Controlador PID periódico con periodo de 200 ms

Como se observa en la figura 63, el controlador aún es capaz de controlar satisfactoriamente el sistema, sin embargo, el sistema tarda más tiempo en estabilizarse (3 segundos frente a 1 segundo), y tanto la respuesta como la acción de control varían más, pero sin llegar a presentar ciclos límite.

1.8.3.3 Análisis de la respuesta de un bucle de control con generación extrínseca de eventos

En esta parte trataremos de ahorrarnos la parte de no ejecución del controlador basado en eventos, es decir, la del tiempo correspondiente a la lectura de una entrada analógica y una no ejecución del bucle de control. El objetivo de este ensayo es de reducir el tiempo de cálculo del procesador porque como hemos visto en el apartado anterior tenemos un término en la ordenada en el origen cuyo origen se debe a esta no-ejecución.

Para ahorrarnos este tiempo, se utilizará tanto el circuito generador de eventos presentado anteriormente y como las herramientas disponibles en la modelización para cubrir todas las condiciones de ejecución del controlador para que este no tenga que realizar dichas comprobaciones (esquema explicativo en la figura 64).

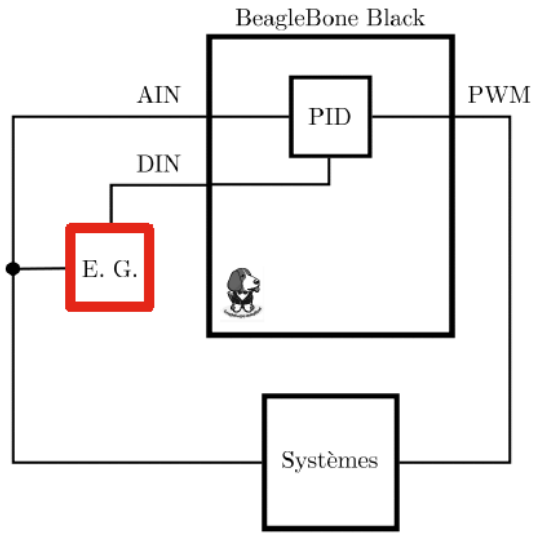


Figura 64: Esquema de la generación externa de eventos

Para realizar los experimentos se ha utilizado el mismo sistema que en la parte anterior (sistema 1 de la tabla 7).

Como el tiempo entre ejecuciones no tiene por qué ser regular se va a utilizar el controlador de Årzén, ya que es el que mejor respuesta presenta de entre los controladores basados en eventos estudiados.

Para que el controlador de Årzén se ejecute se debe cumplir:

$$abs(e - es) > e_{lim}\ OR\ h_{act} \geq h_{max}$$

La segunda condición hace que el controlador no pueda pasar más de un tiempo h_{max} sin ejecutarse. En cuanto a la primera, si la desarrollamos tenemos que:

$$e = y_{ref}(k) - y(k) \quad es = y_{ref}(k-n) - y(k-n)$$

Por lo tanto:

$$abs(e - es) = abs\left(y_{ref}(k) - y(k) - \left(y_{ref}(k-n) - y(k-n)\right)\right)$$

$$abs\left(\underbrace{\left(y_{ref}(k) - y_{ref}(k-n)\right)}_{\boxed{1}} + \underbrace{\left(y(k-n) - y(k)\right)}_{\boxed{2}}\right) > e_{lim}$$

De esta última ecuación se pueden extraer dos casos, el primero, el algoritmo se debe ejecutar si se produce un cambio en la referencia (los cambios en la referencia se consideran siempre más grandes que el umbral δ, en la ecuación e_{lim}) [1]; el segundo caso es directamente que se produzca un cambio de δ en la señal medida entre la última ejecución y la comprobación actual [2], esta condición tiene una gran similitud con un muestreo SSOD siempre y cuando se realice la comprobación lo suficientemente rápido.

Para que esta generación externa de eventos tuviera el máximo de sentido posible, esta detección se debería hacer según dos posibles metodologías. La primera de ellas sería implementar la detección por medio de interrupciones en el procesador de la BeagleBone Black, y la segunda opción sería externalizar la detección y mantenerla hasta que se leyera, de forma que con la lectura se reseteara la lectura. Esta última forma sería la más coherente con el espíritu de la norma de hacer las aplicaciones lo más independientes posible de los sistemas que los ejecutan. Sin embargo, el circuito desarrollado no contempla esta opción y a la hora de programar las interrupciones nos hemos encontrado con dificultades al no tener una documentación completa ni del vector de interrupciones del procesador ni de la gestión específica de la memoria interna. Por estas razones se ha utilizado un método de detección alternativo, hacer un *polling* (operación de consulta constante de un recurso, en este caso, de la entrada digital) sobre la entrada digital.

Por esta razón, se ha desarrollado el FB BBBio_IX_ChangeDetect. Este FB compara la entrada digital con la anterior entrada guardada y si son diferentes envía un evento de salida. La consulta constante se hará por tanto a este FB.

Teniendo en cuenta las diferentes condiciones de ejecución del controlador de Årzén se pasa ahora a implementarlas. Para ello tenemos dos opciones, implementarlas externamente o en el modelado. En el modelado se ha implementado que se envíe un evento al controlador cuando se produzca un cambio en la referencia, y además, se ha añadido un FB que envía eventos periódicamente al controlador con periodo igual al tiempo máximo de no ejecución del controlador. Finalmente, la condición que queda se ha implementado con el circuito de detección de eventos y con el FB BBBio_IX_ChangeDetect.

En la figura 65 se presenta el modelado completo de la aplicación para el control con generación externa de eventos. Como la red de FB es extensa no se aprecia con nitidez, pero el FB encargado de la detección se ha puesto a ejecutar con un periodo de 1 ms para hacer el *polling*. A esta detección se le da una prioridad más baja (95) que a la ejecución del controlador (99) para que esta comprobación no interrumpa la ejecución del controlador. Cuando se detecta un evento con este bloque se cambia la prioridad del *thread* para que las siguientes comprobaciones no interrumpan la ejecución del controlador.

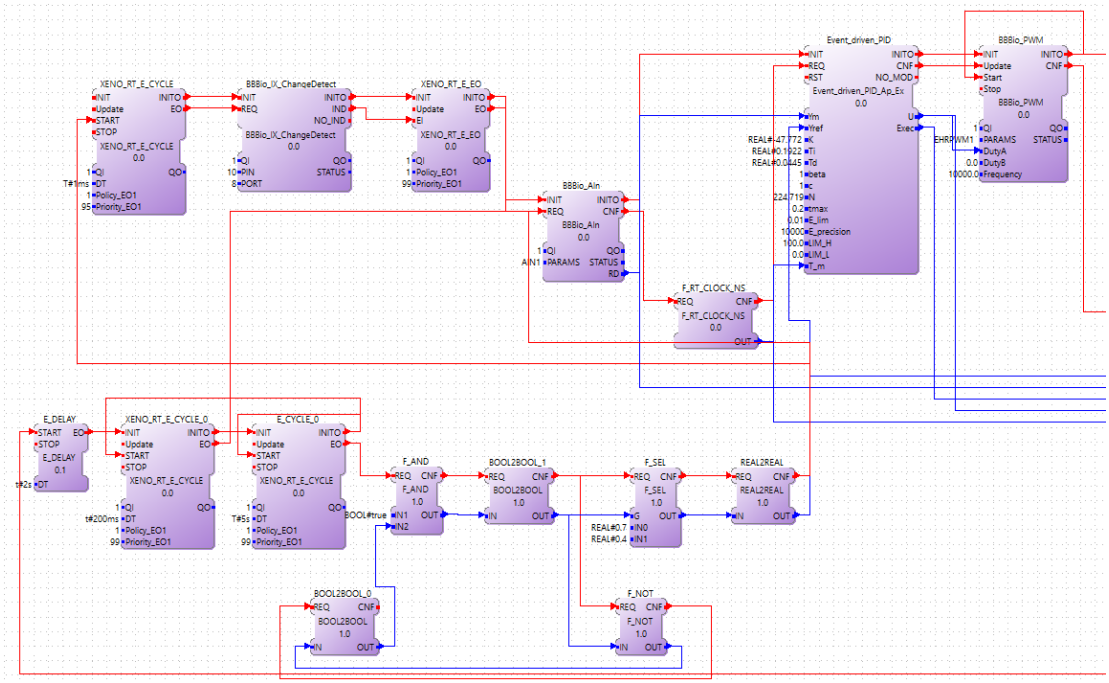


Figura 65: Modelado de la aplicación para la generación extrínseca de eventos

Se quiere hacer hincapié en que esta modelización es funcional pero no es óptima, y se dejará para futuros trabajos un estudio en más profundidad de la modelización y la implementación de la generación externa de eventos de forma óptima.

En cuanto a la respuesta del sistema, la cual podemos ver en la figura 66, se observa que presenta una pequeña sobreoscilación con respecto al controlador periódico. Esta sobreoscilación es debida a las características del controlador de Årzén, ya que cuando pasa mucho tiempo sin ejecutarse el término integral de la acción de control aumenta demasiado y esto repercute en gran mesura sobre la respuesta del sistema. Además se puede ver que la acción de control contiene algunos picos bruscos, este comportamiento de la acción de control

podría deberse a la sucesión consecutiva de eventos en muy poco tiempo, aunque para probar esto deberían realizarse algunos estudios en más profundidad que exceden del marco del proyecto.

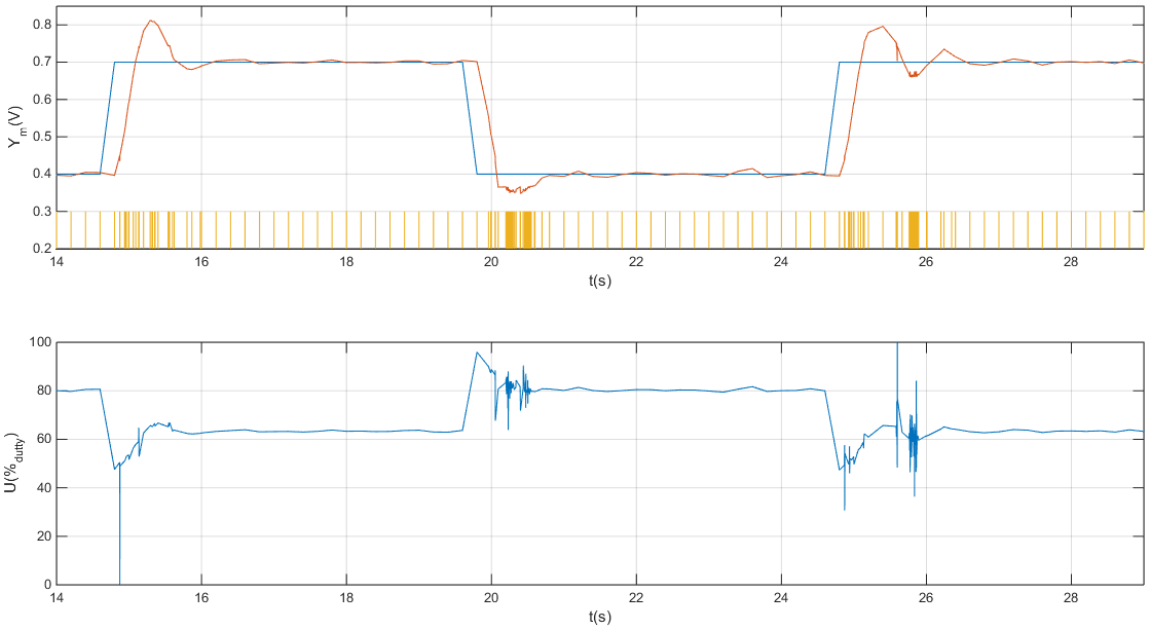


Figura 66: Respuesta del controlador de Årzén con generación externa de eventos

Sin embargo, podemos observar que el control se desarrolla correctamente, y podemos ver los rasgos distintivos del controlador de Årzén: las ejecuciones con un período nominal cuando hay un cambio de referencia y hasta que el sistema se estabiliza y el período máximo entre ejecuciones cuando el sistema ya está estabilizado.

1.8.3.4 Análisis del control con cuatro bucles de control simultáneos

1.8.3.4.1 Análisis de la respuesta de cuatro sistemas iguales

Para esta parte se van a utilizar cuatro sistemas iguales al sistema estudiado hasta ahora.

Llegados a este punto se pueden contemplar numerosas modelizaciones para la aplicación, pero en este caso en particular vamos a controlar los cuatro sistemas con el controlador de Årzén utilizando una gestión de ejecución basada en las prioridades que nos ofrece Xenomai. Estas prioridades serán determinadas en función del error de los sistemas. Pensamos que otorgándole una prioridad más elevada a los *threads* de los sistemas que tienen más error nos permitirá corregirlos más rápidamente y mejorar por tanto su respuesta. Se realizará la comparación con una gestión donde todos los *threads* tienen la misma prioridad, esta modelización hace que se desarrolle el control de un sistema después de otro independientemente del error que estos sistemas tengan.

Se ha modelado en primer lugar la aplicación con las prioridades variable en función del error con la ayuda de los FB Priority_Allocator y con el FB XENO_RT_E_SPLIT_4 como se muestra esquemáticamente en la figura 67.

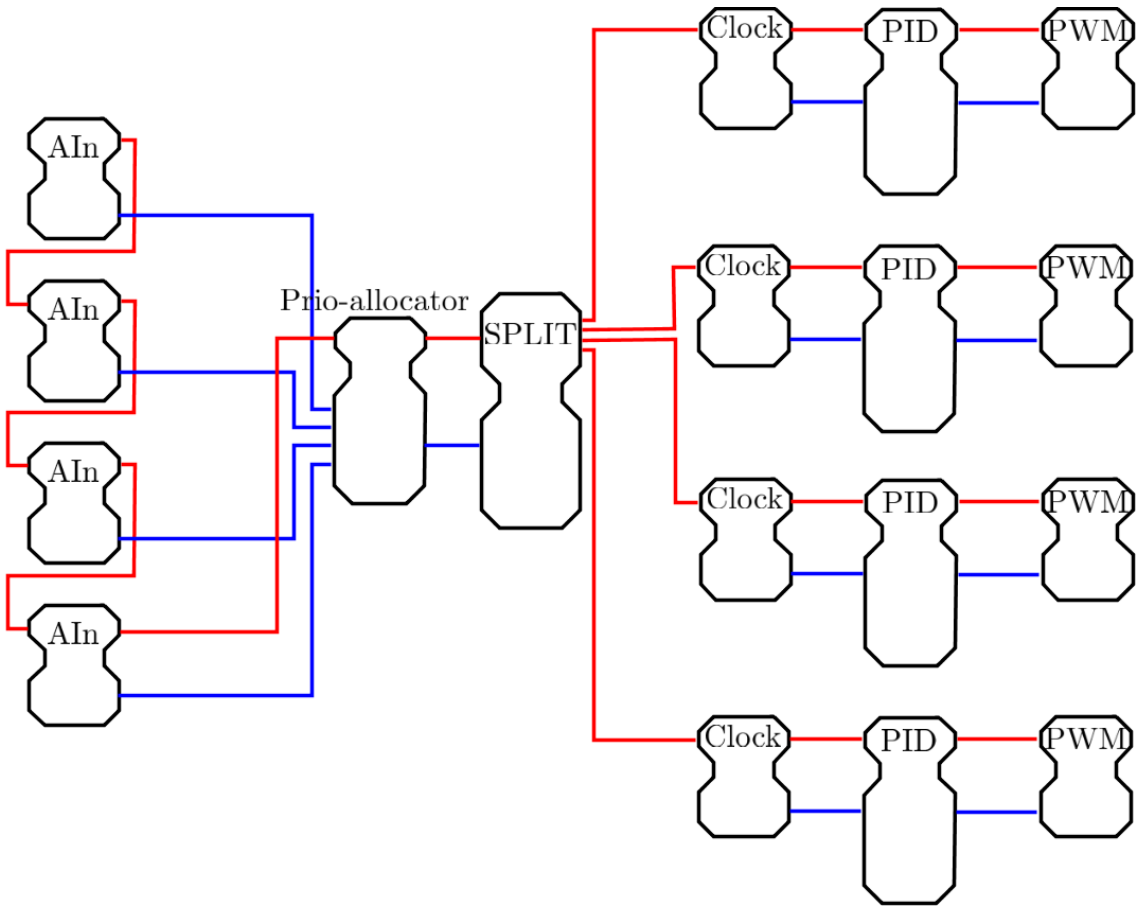


Figura 67: Esquema de la aplicación con prioridades basadas en el error

Solamente se han incluido los FB más significativos de este modelado (se han omitido los FB encargados de enviar eventos periódicamente, los FB que guardan los datos y el sistema de cambio de referencia para dar más claridad al esquema).

Como configuraciones específicas, se ha considerado diferentes alternativas en cuanto a las prioridades, escogiendo finalmente las prioridades más altas (de 96 a 99) para la ejecución de los sistemas, seguidamente se realiza la lectura (prioridad 95) y por último el cambio en la referencia (prioridad 94). Realizando el modelado de esta forma nos aseguramos que la ejecución de los algoritmos de control no será interrumpida por las otras acciones, y dándole a la lectura una prioridad más alta que al cambio de referencia disminuimos el tiempo entre lectura y ejecución cuando estas dos tareas confluyan.

Por otro lado, para hacer la comprobación cuando los cuatro sistemas se ejecutan con la misma prioridad, se utiliza una modelización, mostrada esquemáticamente en la figura 68, donde cada cadena que compone un *thread*, formada por un FB que envía eventos periódicamente, FB de lectura analógica y del tiempo de procesador, FB de control y FB de modificación de la salida; se ejecuta con una prioridad y periodicidad que nosotros le fijamos.

Para este caso todas las prioridades están fijadas al mismo valor, igual que las prioridades, pero la virtud de esta modelización reside en que podemos cambiar el periodo de ejecución de cada bucle y la prioridad con la que se ejecuta.

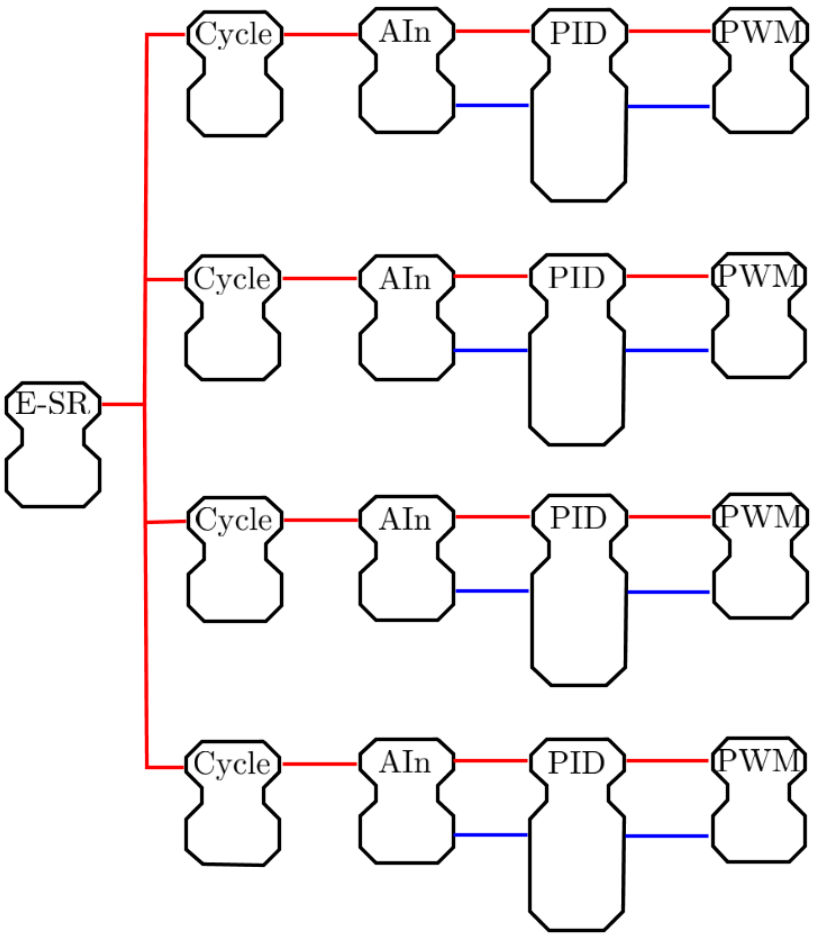


Figura 68: Esquema de la aplicación con prioridades individuales

En cuanto a los parámetros de los controladores, se han utilizado los mismos que en los experimentos anteriores, es decir los obtenidos con el método AMIGO.

En la figura 69, podemos ver la respuesta de los cuatro sistemas con la modelización presentada en la figura 67, y en la figura 70 la acción de control resultante.

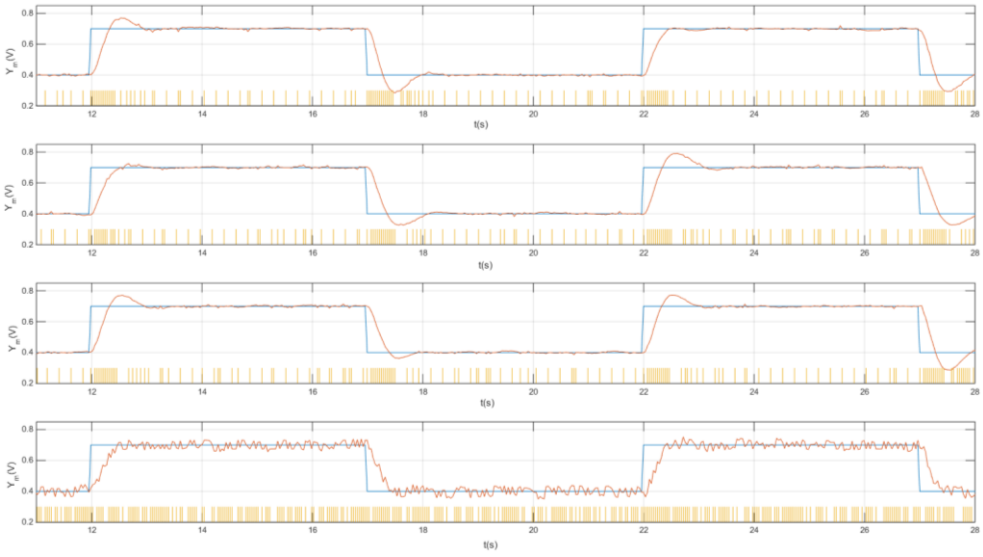


Figura 69: Respuesta de cuatro sistemas iguales con prioridades recalculadas

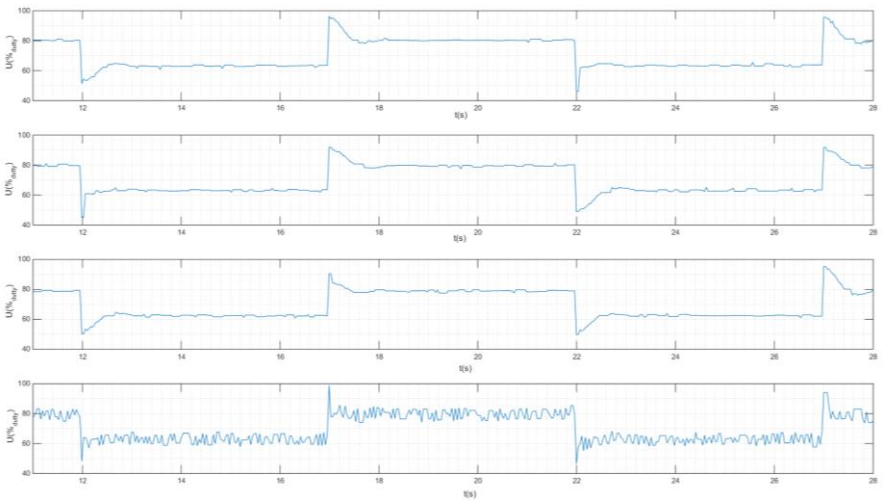


Figura 70: Acción de control de cuatro sistemas iguales con prioridades recalculadas

Por otro lado, en la figura 71 podemos ver la respuesta de los cuatro sistemas con la modelización presentada en la figura 68, y en la figura 72 la acción de control resultante

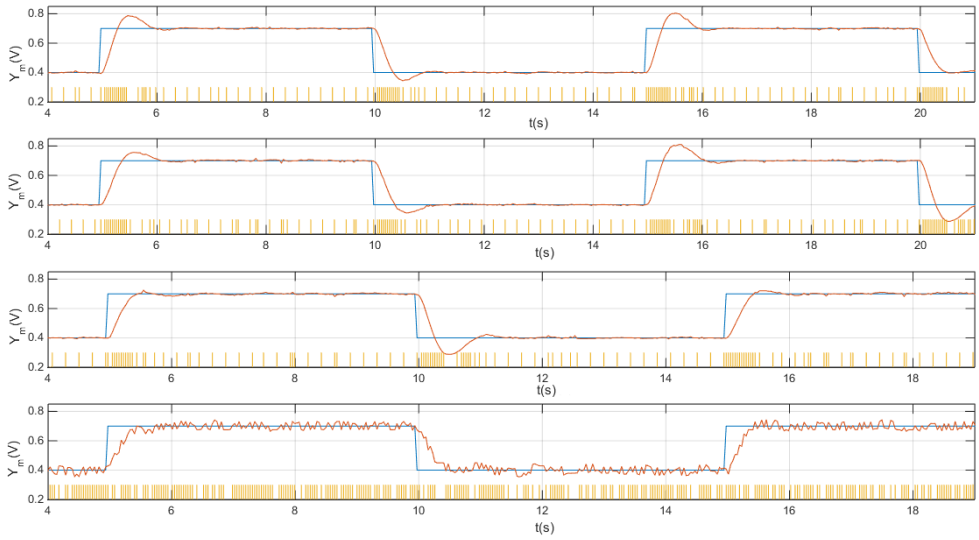


Figura 71: Respuesta de cuatro sistemas iguales con prioridades iguales

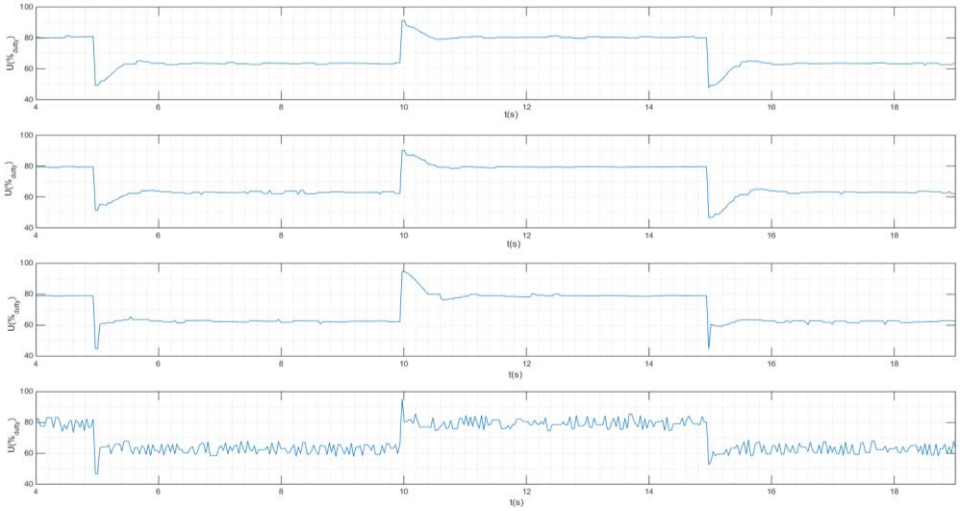


Figura 72: Acción de control de cuatro sistemas iguales con prioridades iguales

Analizando los resultados obtenidos de los experimentos anteriores nos damos cuenta que el cuarto sistema presenta una lectura de la señal analógica ruidosa. Esta lectura conduce al sistema a presentar una respuesta ruidosa así como una acción de control igualmente ruidosa.

En cuanto a las respuestas de los sistemas, no se observan diferencias significativas entre las dos experimentaciones ni tampoco en la acción de control. Este hecho nos lleva a concluir que para los sistemas controlados, tanto una modelización como la otra son perfectamente capaces de realizar el control de forma satisfactoria.

1.8.3.4.2 Análisis de la respuesta de cuatro sistemas diferentes

Para los experimentos que se van a detallar a continuación usaremos los cuatro sistemas presentados en la tabla 8.

Para este análisis, en cuanto a las características del control, se utilizará la modelización donde se asigna a cada sistema un *thread* con una prioridad fija pero diferente a la de los demás, la cual se le asignará en función de la dinámica del sistema, porque, para este experimento, cada sistema a controlar tiene un período de muestreo y una dinámica diferente. La asignación de estas prioridades se hace en función de la dinámica, ya que los sistemas que varían más rápidamente pensamos que deben tener más prioridad en la ejecución para mejorar su respuesta.

Igual que en el caso anterior, se hará una comparación con un experimento donde todos los sistemas tienen la misma prioridad. Para los experimentos utilizaremos la modelización presentada en la figura 68 para ambos experimentos porque esta modelización nos permite asignar diferentes prioridades a cada *thread* cambiando solo un parámetro.

Pasamos ahora a caracterizar los controladores con el método AMIGO, como en los casos anteriores, y se han obtenido los parámetros para todos los controladores utilizados. Además, asignamos a cada sistema una prioridad basada en su dinámica. Podemos ver todos los datos en la tabla 9.

Sistema	K	T_i	T_d	N	$T_{muestreo}$(s)	T_{max}(s)	Prioridad
1	47,772	0,1922	0,0445	10	0,0361	0,1806	96
2	29,3628	0,0297	0,0091	10	0,0071	0,0355	99
3	28,9707	0,0812	0,0251	10	0,0195	0,0977	97
4	48,3042	0,0705	0,0161	10	0,013	0,0651	98

Tabla 9: Parámetros de configuración para los controladores de los cuatro sistemas

Una vez caracterizados los sistemas con los parámetros anteriormente mostrados podemos proceder a visualizar los resultados de los experimentos.

En primer lugar, tenemos la respuesta (figura 73) y la acción de control resultante (figura 74) cuando controlamos los diferentes sistemas con una prioridad diferente en función de la dinámica.

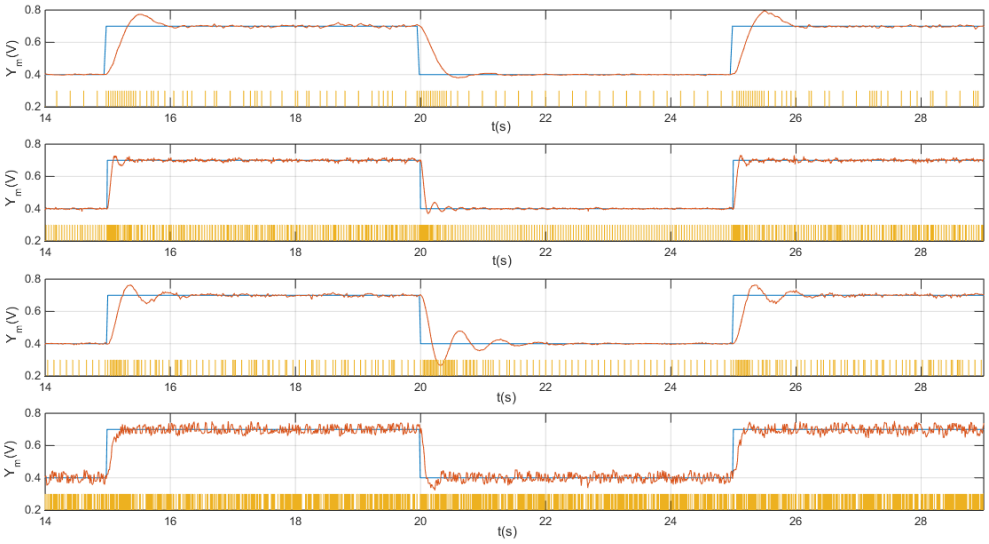


Figura 73: Respuesta de los cuatro sistemas con diferentes prioridades

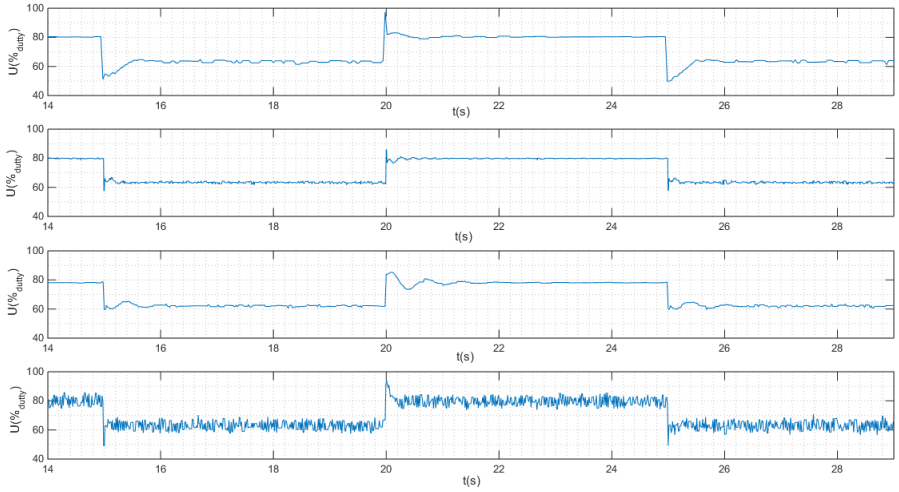


Figura 74: Acción de control de los cuatro sistemas con diferentes prioridades

En segundo lugar, podemos ver la respuesta (figura 75) y la acción de control (figura 76) cuando controlamos todos los sistemas con la misma prioridad.

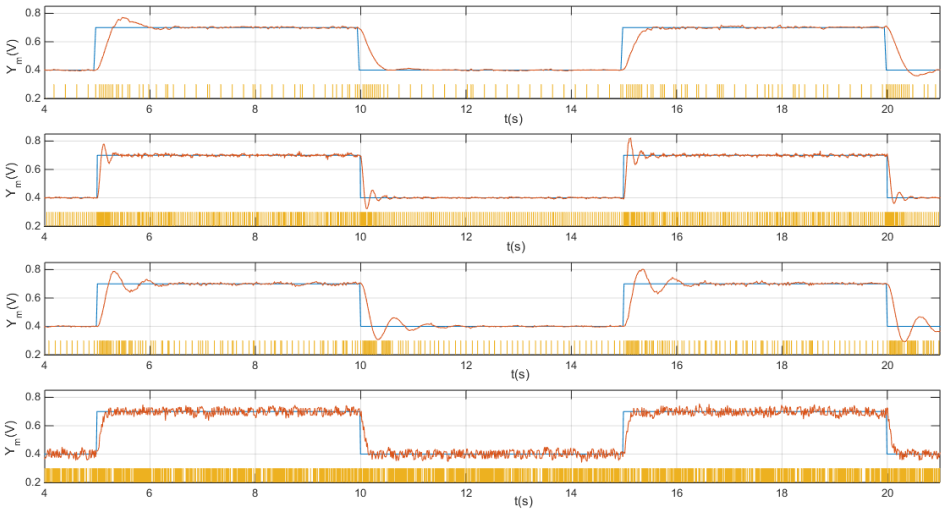


Figura 75: Respuesta de los cuatro sistemas con prioridades iguales

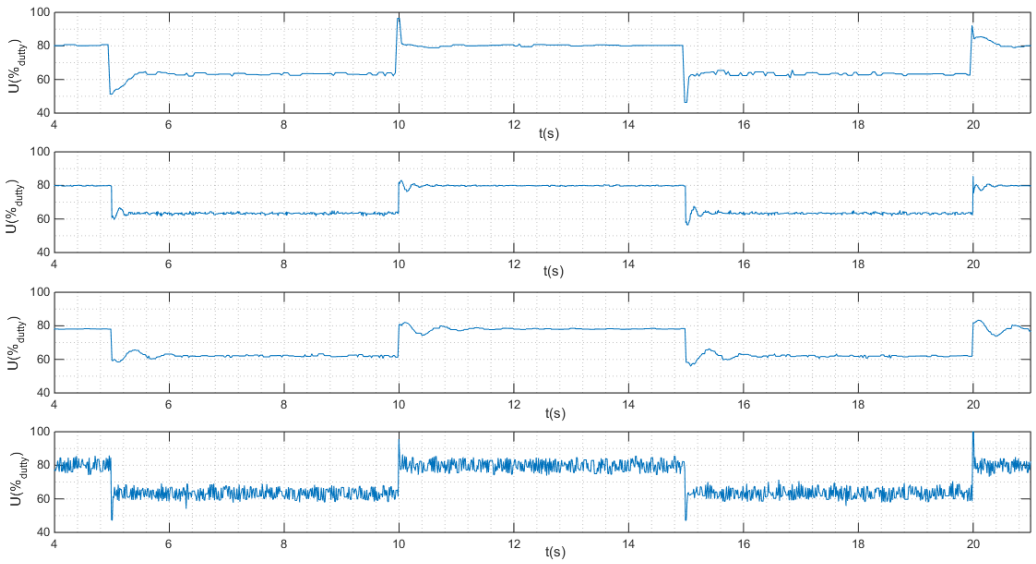


Figura 76: Acción de control de los cuatro sistemas con prioridades iguales

A la vista de los resultados no podemos observar ninguna diferencia significativa entre una asignación de prioridades y la otra. Afinando, se puede ver que el sistema 2 oscila un poco más cuando hay un cambio de referencia y que el sistema 1 tiene una mejor respuesta en el caso de las prioridades iguales, pero esto no constituye una diferencia bastante significativa para que se pueda considerar como una consecuencia de la modelización, de hecho, en los sistemas 3 y 4 no se aprecian diferencias (lo mismo pasa con las acciones de control).

Podemos, por tanto, concluir que como en el caso estudiado en la parte precedente, no se observa ninguna mejora, y que las diferencias entre una respuesta y otra no es consecuencia de la modelización sino de la casuística en la llegada de eventos.

1.8.3.5 Estudio temporal de los CBE

Después de todas las modelizaciones y las pruebas realizadas, tanto en el caso de control de un solo sistema como en el caso de cuatro lazos a controlar, se ha visto cómo la tarjeta es perfectamente capaz de controlar los sistemas sin ninguna dificultad.

En este punto cabe preguntarse por qué los resultados que se han obtenido han sido siempre favorables, porque, aunque se ha demostrado que el estándar IEC-61499 y el control basado en eventos son capaces de desenvolverse bastante bien en todos los casos estudiados, parece que independientemente de la modelización que se escoja para el caso multi-sistema se puede realizar un control satisfactorio, y que esta modelización no tiene ningún impacto sobre la respuesta. Para esto razón debemos hacer un estudio temporal de cómo se comporta el conjunto de sistema y controlador para ver qué se produce cuando se ejecuta un algoritmo en términos de coste computacional.

Para comprender mejor como funciona realmente la aplicación para cualquiera de los casos estudiados podemos resumir el modo de ejecución de la siguiente forma: queremos controlar N sistemas dónde cada uno envía eventos con un periodo determinado y cada sistema tiene una duración de ejecución de sus algoritmos diferente. Además, el *runtime* que ejecuta la aplicación actualiza la pila de tareas a ejecutar con un periodo determinado y lo ejecuta en la CPU

gestionada por el sistema operativo (Xenomai), donde una vez ingresadas las tareas se ejecutan en orden una tras otra hasta que se vacía la pila.

Para realizar esta caracterización temporal nos centraremos en el estudio concreto del caso expuesto en el apartado 1.8.3.4.1 de este trabajo, el cual tiene cuatro sistemas iguales con dos modelizaciones distintas.

En primer lugar, para el caso donde se tiene todos los sistemas con prioridades fijas tenemos que el tiempo de ejecución de cada sistema es de:

$$t_{fijo} = \begin{cases} t_{AIN} + t_{CLK} + t_{ejec} + t_{PWM} + t_{Write} \\ t_{AIN} + t_{CLK} + t_{nejec} + t_{Write} \end{cases}$$

Utilizando los resultados obtenidos en la caracterización temporal de los FB detallada en el anexo 2.2.2:

$$t_{fijo} = \begin{cases} t_{fijo-MAX} = 10{,}3 + 3{,}3 + 17{,}8 + 15{,}3 + 22{,}8 = 69{,}5\ \mu s \\ t_{fijo-MIN} = 10{,}3 + 3{,}3 + 8{,}3 + 22{,}8 = 44{,}7\ \mu s \end{cases}$$

Por lo tanto, en el peor de los casos en cuanto a tiempos de ejecución, es decir, cuando se tienen que ejecutar los cuatro controladores tenemos que el tiempo máximo es de:

$$t_{fijo-TOT-MAX} = 69{,}5 \cdot 4 = 278\ \mu s$$

Por otro lado, cuando tenemos la modelización donde las prioridades de los *threads* se recalculan en función del error en cada ejecución, la expresión del tiempo consumido es la siguiente:

$$t_{recalculo-TOT} = t_{calculo} + t_{ctrls}$$

Donde:

$$t_{calculo} = 4 \cdot t_{AIN} + t_{prioallo} + t_{split4} = 4 \cdot 10{,}3 + 17{,}8 + 9.3 = 68{,}3\ \mu s$$

$$t_{ctrls} = \begin{cases} t_{ctrls-MAX} = 4 \cdot \left(t_{CLK} + t_{ejec} + t_{PWM} + t_{Write}\right) = 4 \cdot (3{,}3 + 17{,}8 + 15{,}3 + 22{,}8) = 236{,}8\ \mu s \\ t_{ctrls-MIN} = 4 \cdot \left(t_{CLK} + t_{nejec} + t_{Write}\right) = 4 \cdot (3{,}3 + 8{,}3 + 22{,}8) = 137{,}6\ \mu s \end{cases}$$

Y en total para el peor caso:

$$t_{recalculo-TOT} = 236{,}8 + 68{,}3 = 305{,}1\ \mu s$$

Finalmente, en cuanto a los periodos, el periodo de envío de eventos es de 36 ms, el cual está fijado al tiempo de muestreo del sistema y por tanto de ejecución del PID; y el periodo de comprobación por parte del *runtime* de la pila de ejecución es de 1 ms, especificado en la compilación de FORTE.

Para que resulte más clara la situación podemos hacer una representación gráfica (figura 77) del caso que consume más tiempo para este estudio concreto, es decir, en el caso del re-cálculo de las prioridades cuando todos los sistemas deben ejecutarse.

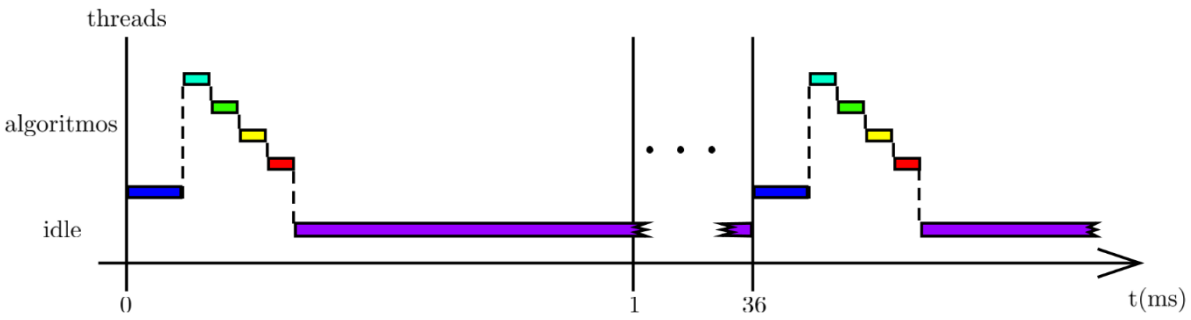


Figura 77: Gráfico del tiempo consumido en el caso estudiado

Podemos concluir pues, que con cualquier modelización de las que podamos emplear, como los tiempos de ejecución del algoritmo son mucho más rápidos que la dinámica del sistema, seremos capaces de ejecutar los cuatro sistemas sin problemas.

Dejamos como trabajos futuros comprobar cuál es el impacto de las diferentes modelizaciones que se han propuesto y si son o no son mejores que la modelización con la misma prioridad para los cuatro sistemas. Para esto, habría que cambiar los componentes pasivos de la tarjeta de manera que la dinámica del sistema requiera un periodo de muestreo similar al tiempo de actualización de la pila de FORTE y al mismo tiempo para que este período sea más próximo a la orden de magnitud de los tiempos de ejecución del algoritmo, es decir, un periodo de muestreo y de actualización de la pila del orden de los 300 µs. Sin embargo, habría que comprobar también que la tarjeta puede soportar sistemas tan rápidos porque si la dinámica de los sistemas es demasiado rápida el amplificador operacional no podrá actuar correctamente por posibles límites en relación al *Slew-Rate*.

Para comparar los tiempos con los que se puede llegar a trabajar, podemos compararlos con un PLC industrial, por ejemplo con el PLC de Schneider de que disponemos, el Modicon M340 que tiene un periodo mínimo de ejecución de tareas cíclicas de 1 ms. Podemos ver, que con un sistema operativo con unas atribuciones de tiempo real como Xenomai podemos llegar hasta un periodo cíclico de aproximadamente 300 µs en el caso más crítico de cálculo para la modelización que más tiempo consume. No nos queda más que preguntar hasta que orden de tiempos de ejecución se podría llegar si se utiliza un microprocesador que soportas la carga y la ejecución de las aplicaciones y del *runtime* de la norma IEC-61499.

1.8.4 Conclusiones y perspectivas de trabajo

En este trabajo se ha visto cómo se implementan los controladores basados en eventos utilizando la norma IEC-61499 para la programación de los sistemas de control. Además, se ha realizado un estudio donde se ha tomado en consideración los aspectos relativos al coste computacional de los algoritmos y a las respuestas del sistema que hay que controlar. Asimismo, para ser capaces de comparar los resultados teóricos con resultados experimentales se han desarrollado diferentes tarjetas electrónicas que permiten realizar el control sobre sistemas reales.

En cuanto al coste computacional se ha obtenido un modelo, en el caso del control en un solo sistema, que nos permite comparar de forma sistemática los controladores basados en eventos con los controladores periódicos. Este modelo, relaciona el porcentaje de ejecuciones de un controlador basado en eventos con el porcentaje del tiempo de ejecución de un CBE respecto al controlador periódico. Este modelo permite, por tanto, conocer el ahorro en términos de tiempo de ejecución, que se obtiene para un porcentaje de ejecución bajo, o el sobrecoste computacional en el que se incurre para un porcentaje de ejecución alto, tomando siempre como referencia al controlador periódico.

En cuanto a la generación de eventos se ha visto que se puede pasar de una generación intrínseca a una generación extrínseca de los eventos. Este hecho, aunque la realización final no haya sido la mejor posible (queremos recalcar que el uso de interrupciones sería mejor, sin ningún tipo de dudas, que hacer un *polling*), nos permite prescindir, en el mejor de los casos, de una parte del tiempo consumido, en concreto, del tiempo consumido por el conjunto de acciones que se realizan cuando no es necesaria la ejecución del controlador basado en eventos. Todavía no podemos expresar cuánto tiempo exactamente somos capaces de ahorrar con esta generación extrínseca de los eventos debido a las dificultades que se encontraron, las cuales nos impidieron realizar un estudio en profundidad pero las ideas propuestas en este trabajo pueden establecer una base para futuros estudios en relación a este tema.

Además, en cuanto al control con varios sistemas se ha visto que existe más de una modelización correcta para los sistemas que se han controlado en este trabajo, porque el tiempo de ejecución de los algoritmos es demasiado pequeño con respecto al período de muestreo y también con relación al período de actualización de la pila de ejecución del *runtime*. En este sentido, y tal y como se ha planteado en apartados anteriores, resulta necesario realizar futuras investigaciones donde se pruebe el efecto de las asignaciones de prioridades al control, para ello es necesario realizar experiencias donde el tiempo de ejecución sí que sea crítico.

De todas formas, se ha mostrado de forma experimental la viabilidad del estándar IEC-61499 para la implementación de algoritmos basados en eventos. Cabe recalcar la potencia de este estándar, porque dependiendo de la aplicación que se quiera desarrollar se pueden lograr tiempos de ejecución muy pequeños.

Además de los trabajos que se han propuesto, sería interesante estudiar en profundidad la aplicación del estándar a diferentes casos, como el control de muchos sistemas donde la red de comunicación sea un recurso compartido (y por tanto crítico), o la aplicación de este estándar a otro tipo de controladores que necesiten mucho más tiempo de cálculo que un controlador PID, como lo es un control óptimo que requiera la solución de la optimización en-línea, pero basado también en eventos.

Por otra parte, sería también conveniente realizar un estudio sobre cómo se comportan los sistemas y los controladores si en vez de utilizar un muestreo tipo SSOD se utiliza otro tipo de muestreo, como por ejemplo, un muestreo basado en el error acumulado. Es decir, cual es el impacto que tienen los diferentes sistemas de generación de eventos sobre el control y en el caso de que se considere necesario, desarrollar un controlador. Además, y quizás lo más urgente, sería desarrollar un método de ajuste para los controladores basados en eventos donde se pudieran escoger los parámetros de los mismos relacionándolos con parámetros que tengan en cuenta la robustez de los controladores.

1.9 Planificación

Este proyecto se ha desarrollado en torno a tres puntos fundamentales: concepción y realización de tarjetas electrónicas, programación de FB para el correcto funcionamiento de las aplicaciones y desarrollo de las mismas y realización de experiencias de control.

Al tratarse de un proyecto de investigación, no es posible seguir una planificación lineal bien definida, ya que se tienen que ir resolviendo los problemas a medida que van apareciendo. Sin embargo, sí se ha seguido, a grandes rasgos, una planificación general que viene detallada en la figura 78.

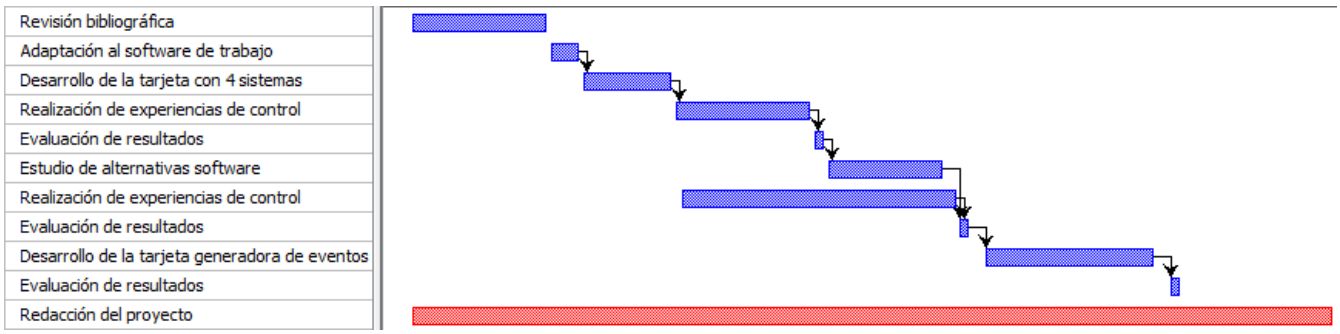


Figura 78: Diagrama de Gantt del proyecto

1.10 Orden de prioridad entre los documentos

El orden de prioridad entre los documentos es el que por defecto se especifica en la norma UNE-157001, a saber: Planos, Pliego de condiciones, Presupuesto y Memoria.

2. Anexos

2.1 Documentación de partida

La documentación de partida para este proyecto se encuentra listada y referenciada en la bibliografía, punto 1.4.4 de la Memoria.

2.2 Cálculos

2.2.1 Detalles sobre los circuitos montados

En esta sección de los anexos se explican en profundidad algunos aspectos adicionales en relación a los circuitos electrónicos desarrollados. En concreto se va a tratar la implementación en PCB de los mismos y el funcionamiento en concreto del generador de eventos externos.

2.2.1.1 Implementación en PCB

En primer lugar, vamos a tratar la tarjeta que contiene los cuatro sistemas a controlar. En esta tarjeta se han modelado los conectadores de la BeagleBone Black como bandas de pines normales porque la separación entre un pin y otro es una longitud estandarizada, y el esquemático del que se disponía de la tarjeta BeagleBone Black presentaba algunos problemas a la hora de realizar las conexiones. Podemos consultar el esquemático de este circuito en la sección de planos, en el plano P.001.

Cómo podemos ver en dicho plano, aunque se haya modelado el circuito con las representaciones de resistencias y condensadores, después dejaremos su emplazamiento para instalar pequeños zócalos para poner los componentes con el valor que queramos. Se puede ver tanto en este esquemático como en el montaje final que se incluyen diodos para proteger la tarjeta BeagleBone Black frente a picos de tensión.

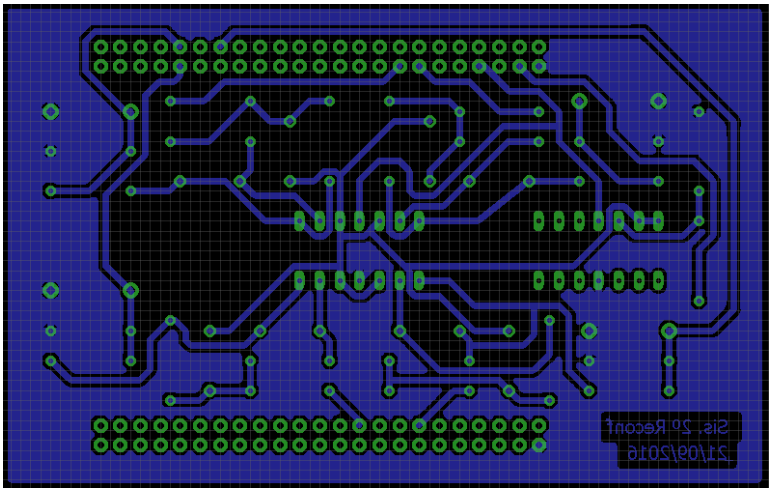


Figura 79: PCB de la tarjeta que contiene los circuitos a controlar

Cómo podemos ver en la figura 79 se ha conectado todo en una única capa, la inferior. Se ha realizado de este modo para facilitar de modelización y para evitar problemas en la fabricación de la tarjeta. El hecho de modelar de ese modo nos impidió conectar dos puntos que quedaban alejados en el diseño, problema que se ha resuelto con la soldadura de un cable entre estos.

Por otra parte, tenemos la tarjeta encargada de la detección y el envío de los eventos. En la fase de diseño de esta tarjeta decidimos fuera una tarjeta externa, no como la tarjeta anterior que se ha modelado pensando en que fuera una capa que se pone sobre la BeagleBone Black. Además, se decidió que para este circuito no fuera la BeagleBone Black quien proporcionara la alimentación a la tarjeta, porque se comprobó experimentalmente que pueden sucederse picos de consumo cuando se inicia el funcionamiento de la tarjeta.

Además, como se ha comentado en la memoria del proyecto, se ha añadido la posibilidad de ajustar los niveles de detección en función de los bits que se escojan a la entrada del circuito, esto se hace por medio de unos *jumpers* que conectaremos en una posición u otra según nuestras necesitadas.

Esta tarjeta, es más compleja que la tarjeta anterior (cuyo circuito esquemático se puede ver en el plano P.002), y aunque el espacio de que disponemos es más grande, no podemos crear un circuito desmesuradamente grande, por ello, se ha decidido poner solo dos generadores de eventos para poder realizar experimentos en dos sistemas. Además, se ha realizado el montaje con puertas NAND (con las que se monta una puerta XOR) porque era mejor en cuanto al rendimiento porque una puerta XOR (en vez de las cuatro NAND) era demasiado rápida y conducía a errores en la dinámica de generación.

Para el diseño del PCB, como en la otra tarjeta, se ha modelado totalmente en una capa, pero esta vez como el circuito es más complejo que en el caso anterior, se han tenido que dejar más conexiones pendientes por medio de cables. La figura 80 muestra el PCB del circuito generador de eventos.

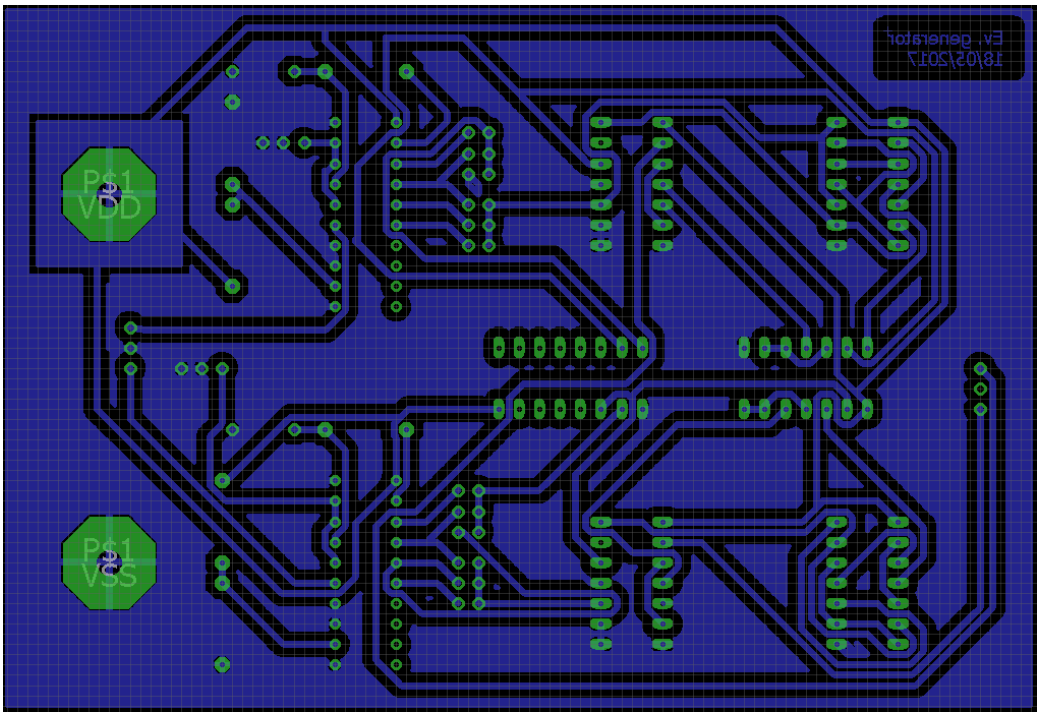


Figura 80: PCB de la tarjeta generadora de eventos externos

2.2.1.2 Detalle del funcionamiento del circuito generador de eventos

Para mostrar el correcto funcionamiento del circuito generador de eventos se han realizado varias pruebas donde ponemos a la entrada del circuito una señal triangular, cuya pendiente es constante, y por lo tanto, también debería serlo la duración de los pulsos generados. Para esta prueba se han escogido los bits 1 y 2, sabiendo además que el FSR de nuestro ADC es de 5V podemos calcular δ:

$$\delta = \frac{FSR}{2^{8-(n_{bit}+1)}} = \frac{5}{2^{8-(1+1)}} = 0{,}07812\ V$$

El funcionamiento (generación de eventos) se muestra en las figuras 81 y 82.

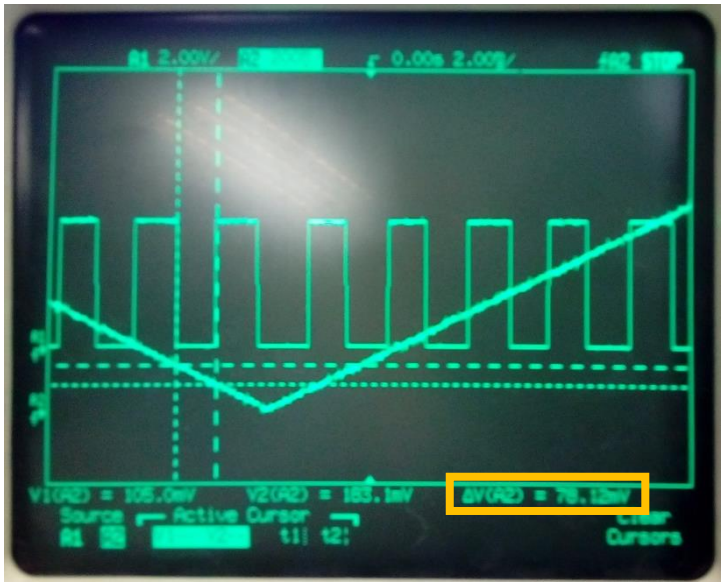


Figura 81: Circuito generador de eventos frente a una señal con pendiente constante

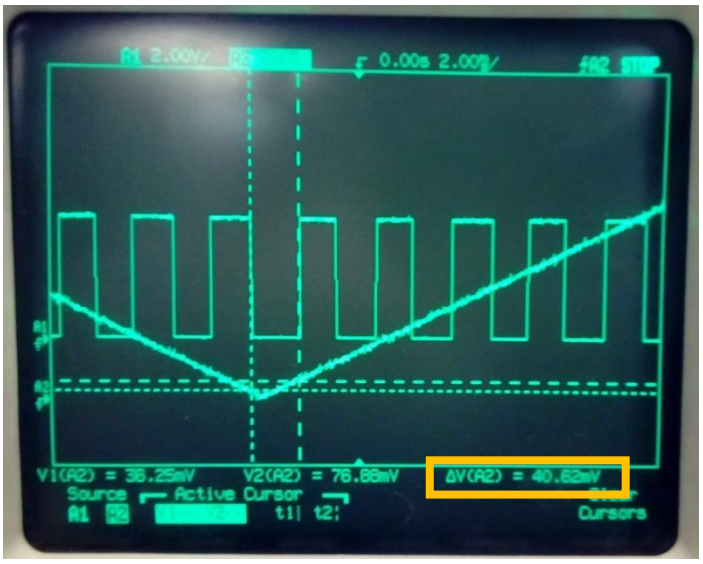


Figura 82: Circuito generador de eventos frente a un cambio de dirección de la señal de entrada

En estas gráficas se puede observar claramente el método de generación de eventos que se obtiene, como se había explicado en la figura 40, donde se puede ver que para un cambio de la señal muestreada en una misma dirección (sea incrementándose o disminuyéndose) se obtiene el mismo comportamiento que para un muestreo tipo SSOD, como muestra la figura 81; pero que, para un cambio de dirección de la señal muestreada, se reduce el umbral de detección a la mitad, en la figura 82 se puede apreciar claramente esta reducción.

2.2.2 Caracterización temporal de los FB

En esta parte vamos a caracterizar al principal FB que han sido utilizados para realizar el control de los sistemas. Con este fin modelamos una aplicación para guardar el tiempo al principio y al final de cada ejecución, de este modo, luego se hace la diferencia y se representan en forma de histograma. La aplicación que se utilizó es la mostrada en la figura 83.

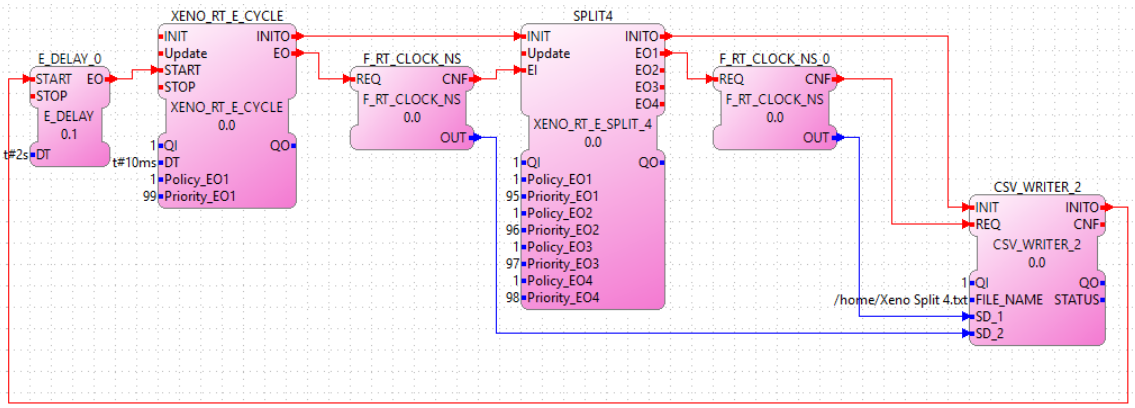


Figura 83: Aplicación para la caracterización temporal de los FB

En esta figura se ilustra el caso concreto de la caracterización del FB XENO_RT_E_SPLIT_4, pero la estructura se mantiene para los otros FB caracterizados. Basta con cambiar el FB SPLIT4 por el FB que se quiere caracterizar. Como característica particular de esta modelización, para la obtención del tiempo que realmente consumen los FB, hay que restar a los tiempos que se obtendrán el tiempo que consumen los dos FB F_RT_CLOCK_NS, tiempo que se ha caracterizado y que obtuvimos fue del orden de 2.2 µs. En las gráficas que se presentarán posteriormente se presentarán los tiempos obtenidos directamente de esta modelización.

Veremos además, que en todos los gráficos hay algunas ejecuciones que toman demasiado tiempo con relación a la media. Asumimos que este hecho es debido a la forma de ejecutarse de Xenomai, el cual se ejecuta como una interrupción del sistema operativo. Sin embargo, como este hecho no aparece muy a menudo, consideramos que para simplificar los cálculos podemos despreciar estas ejecuciones para tomar el tiempo medio de ejecución del FB.

En las figuras siguientes (figuras de la 84 a la 97)se puede ver resultados interesantes como por qué pasar de una lectura analógica a una digital, o la caracterización de los controladores cuando se ejecutan y cuando no. Además en los casos donde se pueden caracterizar varios tiempos siempre se ha escogido hacerlo de la manera en que más tiempo consumen (XENO_RT_E_SPLIT_4 o PWM cuando se actualiza el valor de la salida).

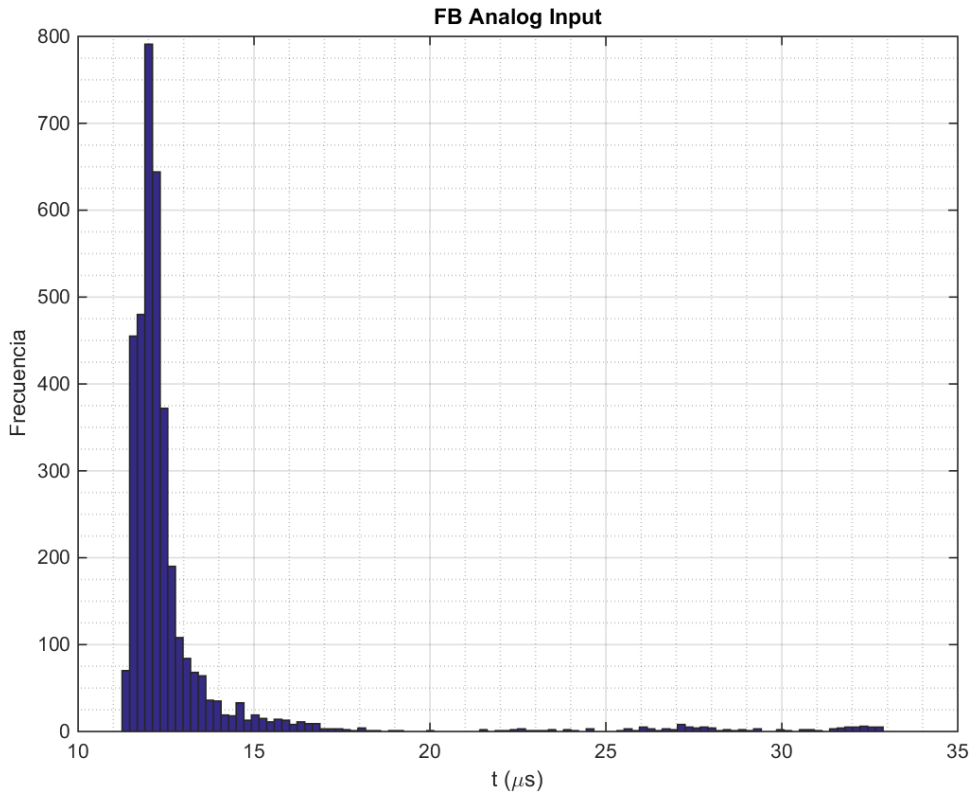


Figura 84: Caracterización temporal del FB "BBBio_Ain"

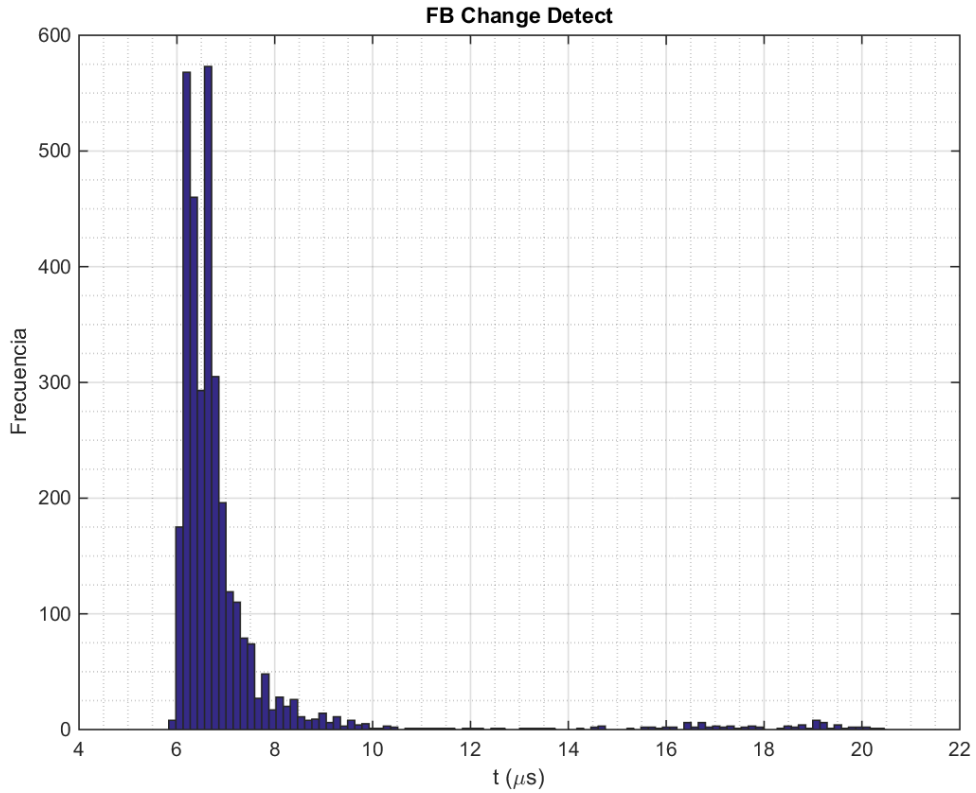


Figura 85: Caracterización temporal del FB "BBBio_IX_ChangeDetect"

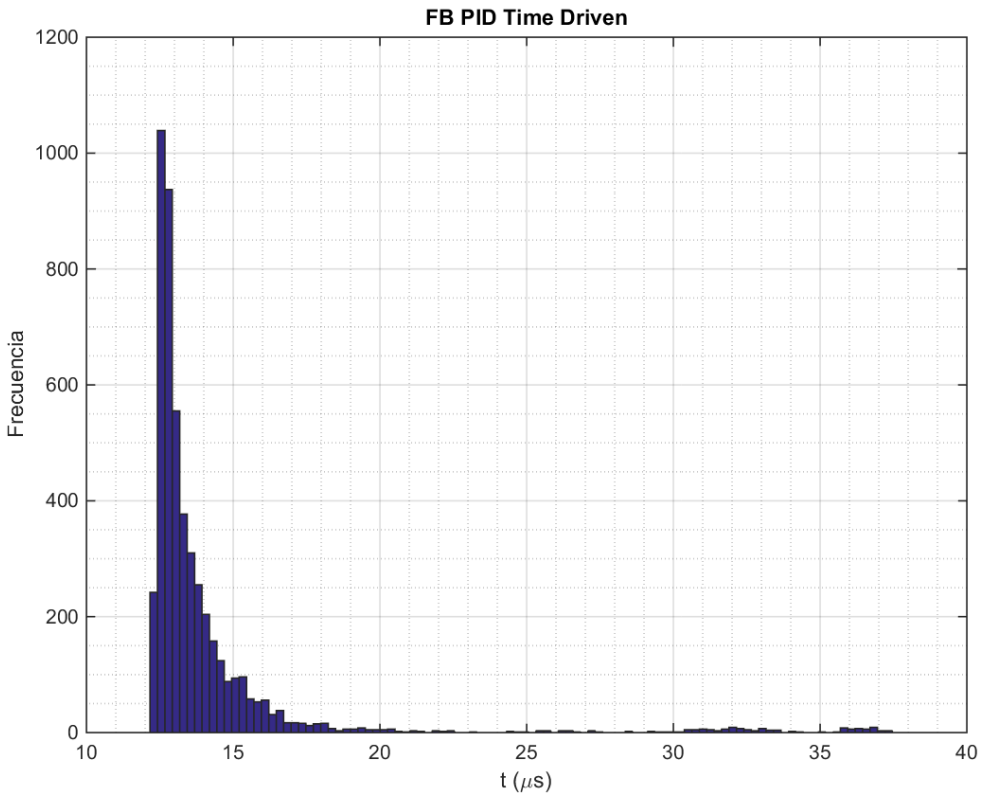


Figura 86: Caracterización temporal del FB "Time_driven_PID"

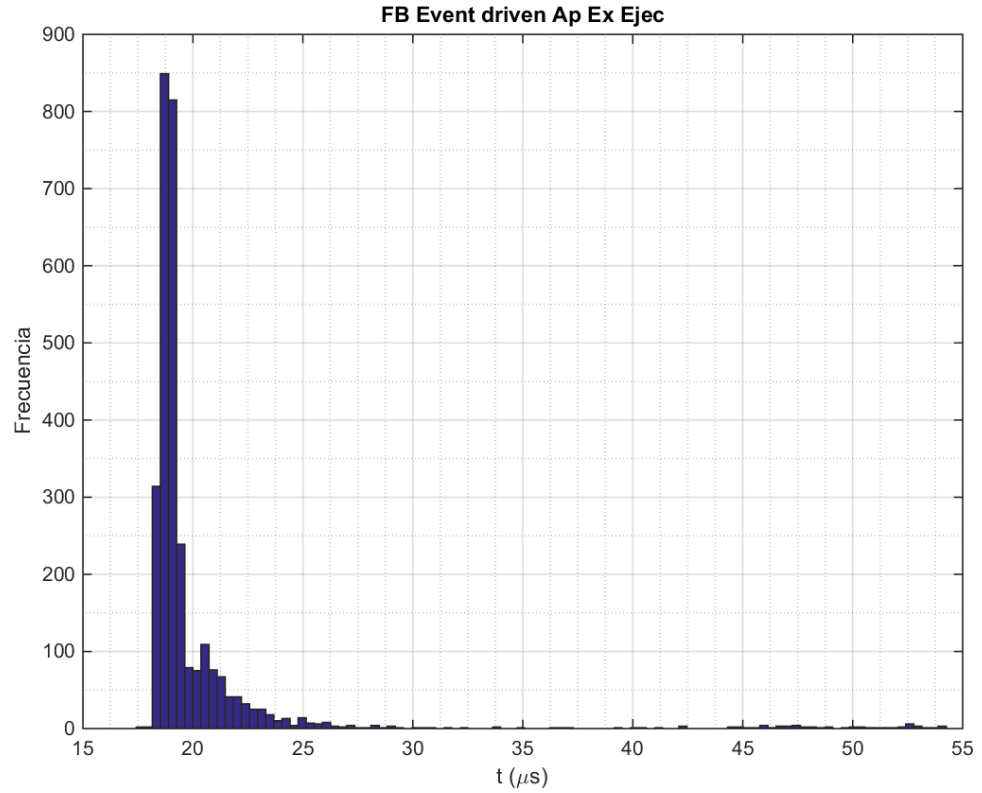


Figura 87: Caracterización temporal del FB "Event_driven_PID_Ap_Ex" cuando se ejecuta

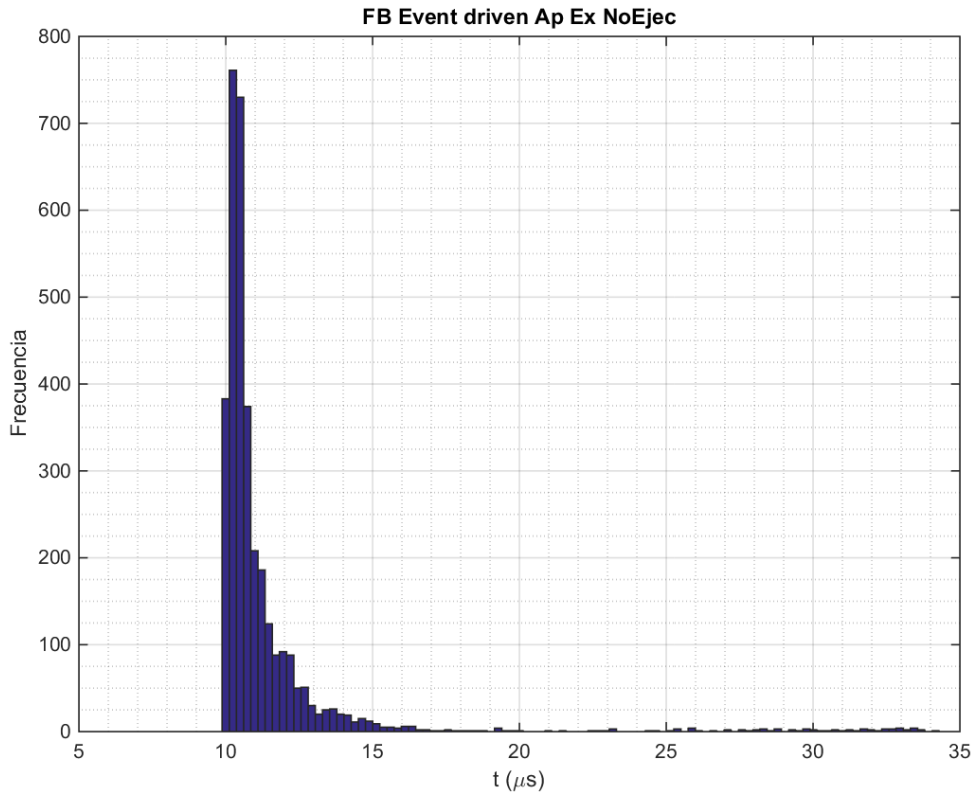


Figura 88: Caracterización temporal del FB "Event_driven_PID_Ap_Ex" cuando no se ejecuta

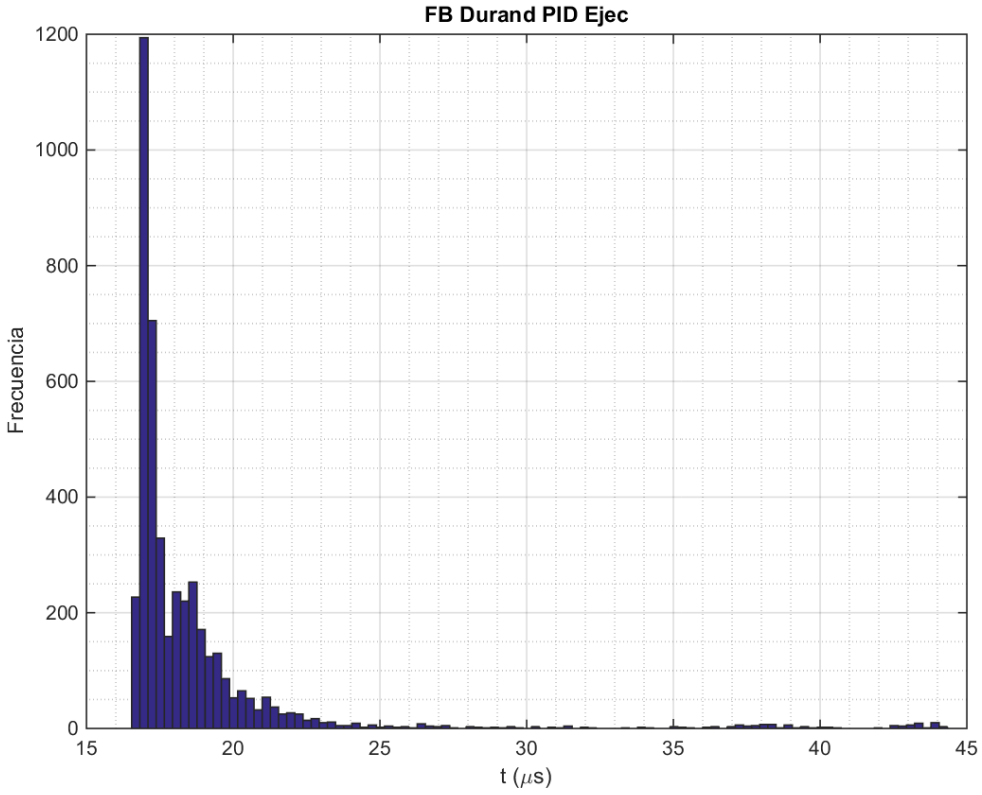


Figura 89: Caracterización temporal del FB "Durand_PID_Ap_Ex" cuando se ejecuta

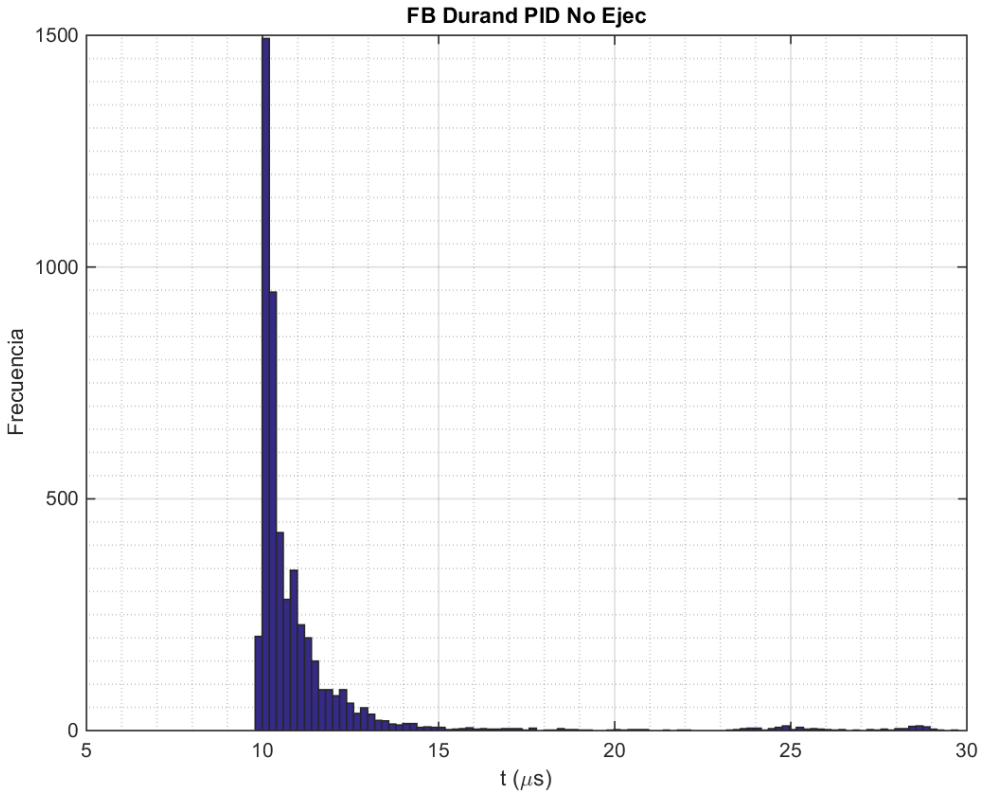


Figura 90: Caracterización temporal del FB "Durand_PID_Ap_Ex" cuando no se ejecuta

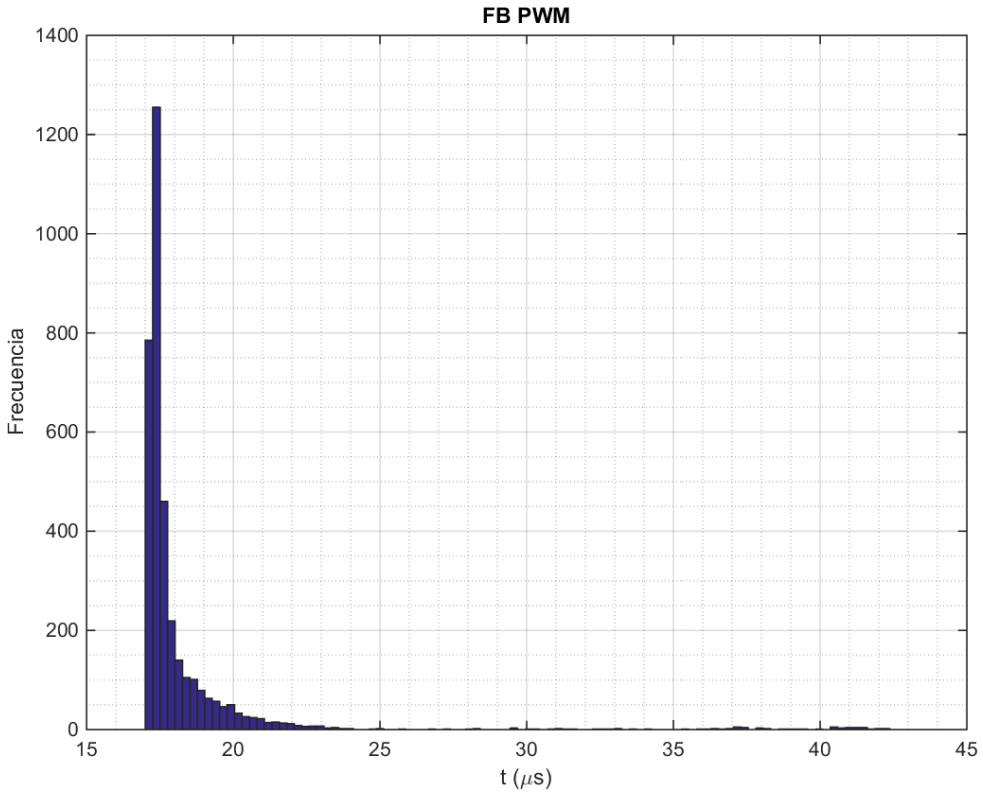


Figura 91: Caracterización temporal del FB "BBBio_PWM"

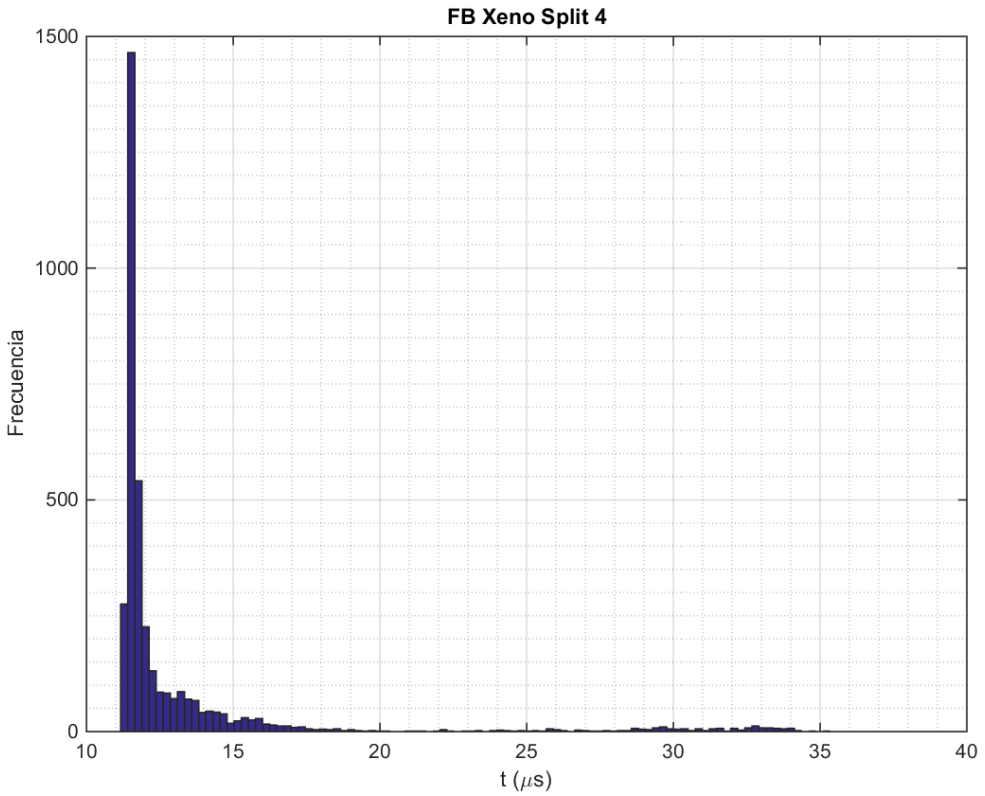


Figura 92: Caracterización temporal del FB "XENO_RT_E_SPLIT_4"

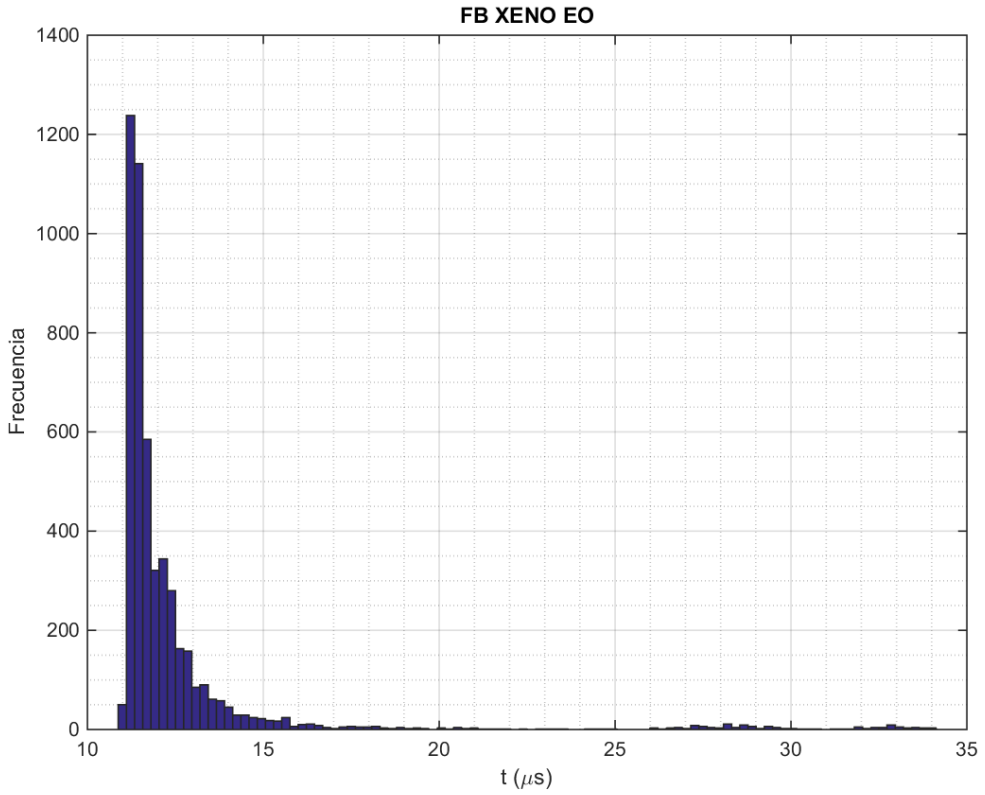


Figura 93: Caracterización temporal del FB "XENO_RT_E_EO"

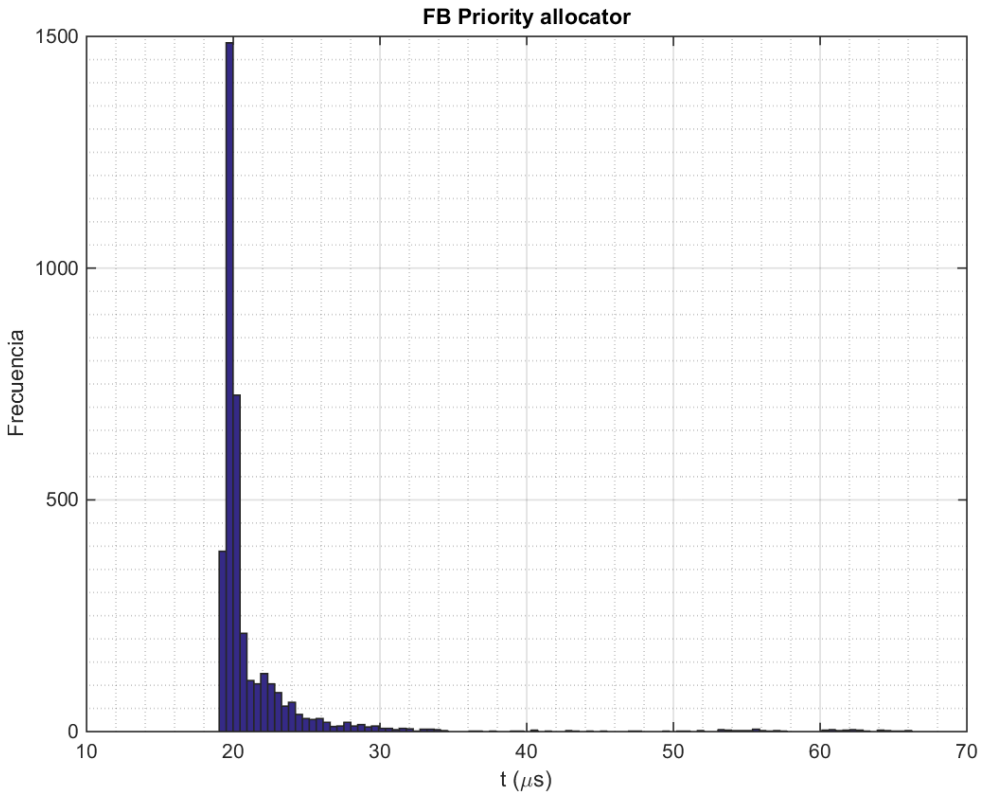


Figura 94: Caracterización temporal del FB "Priority_allocator"

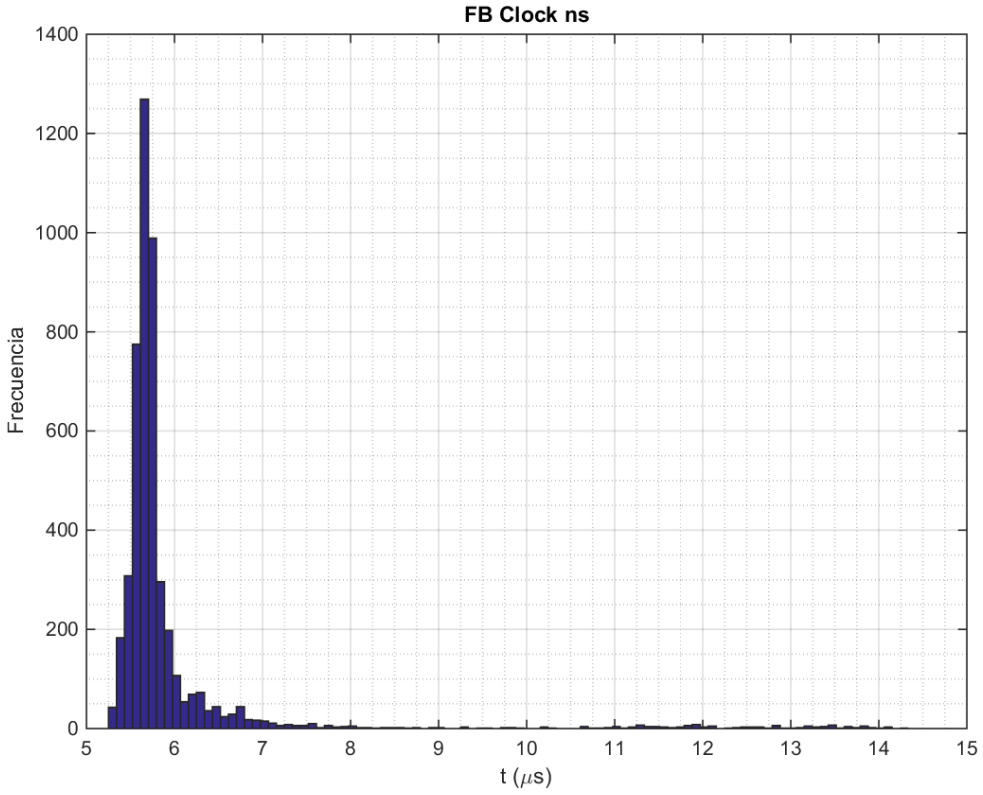


Figura 95: Caracterización temporal del FB "F_RT_CLOCK_NS"

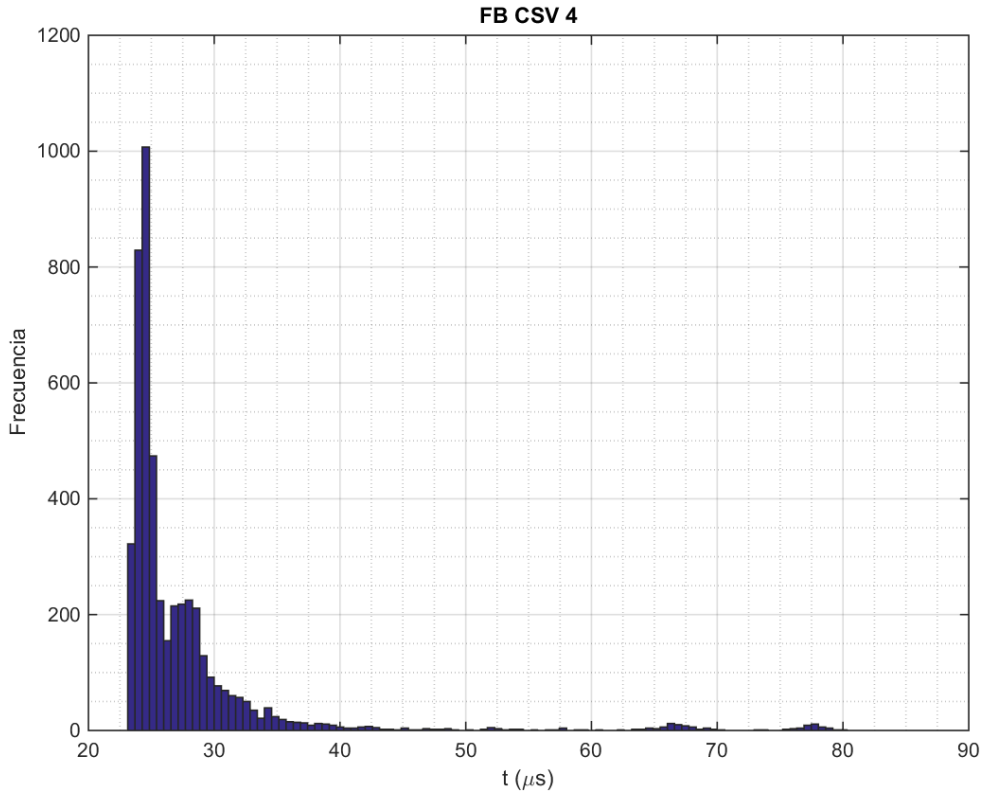


Figura 96: Caracterización temporal del FB de escritura en archivo "CSV_4"

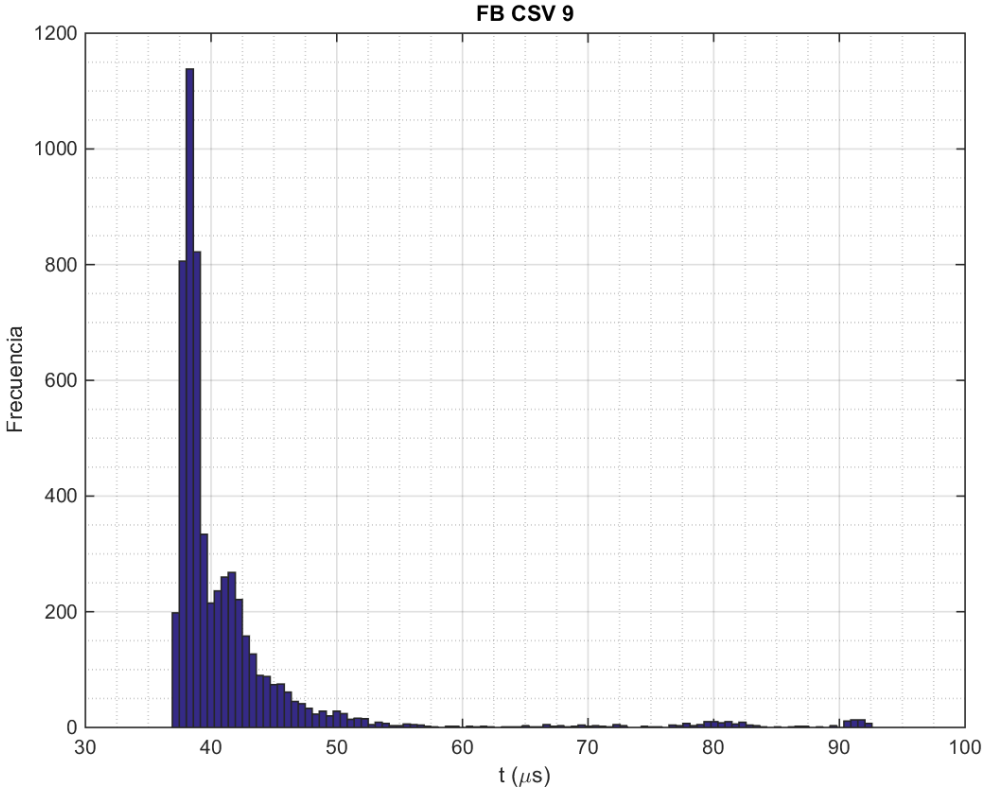
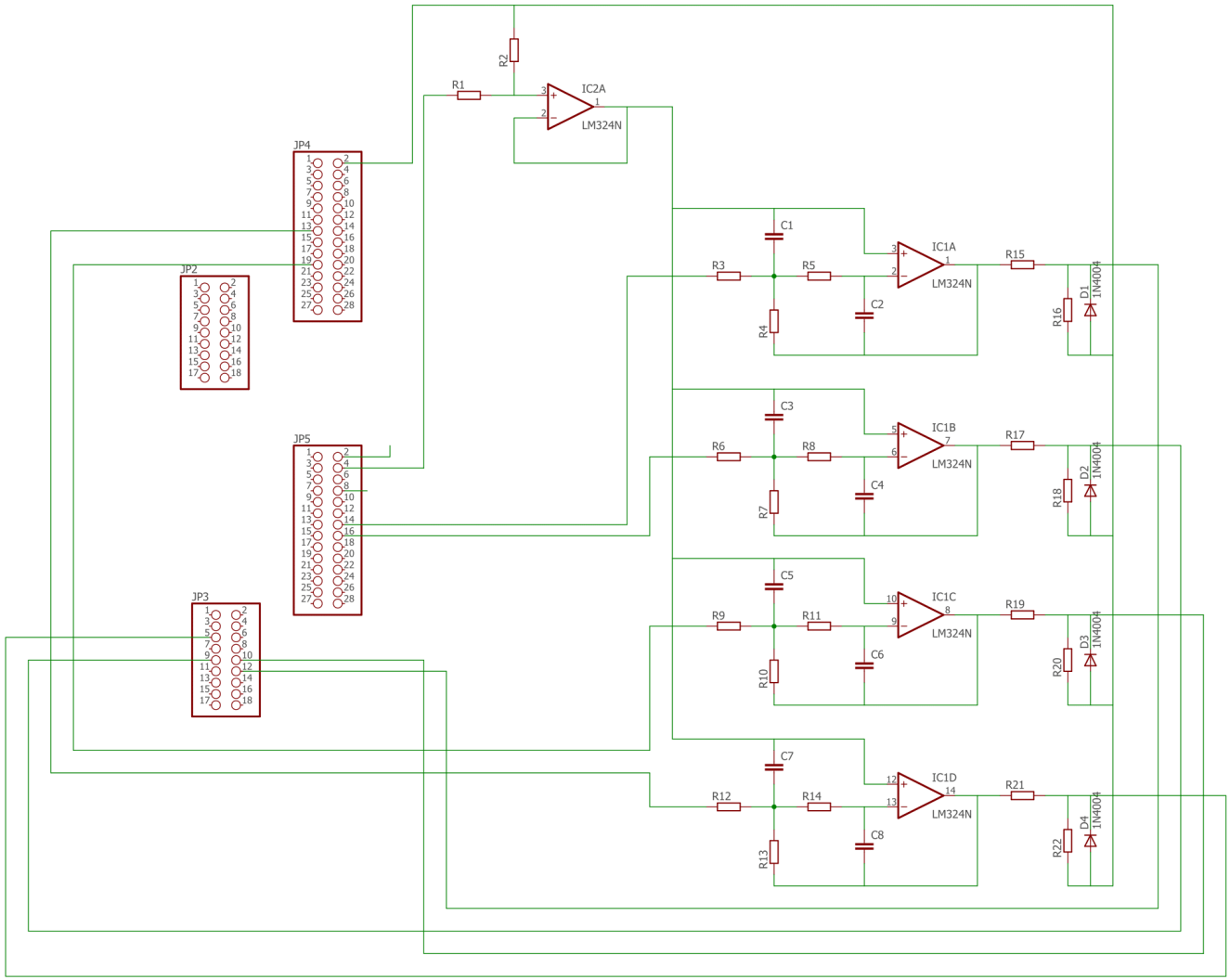


Figura 97: Caracterización temporal del FB de escritura en archivo "CSV_9"

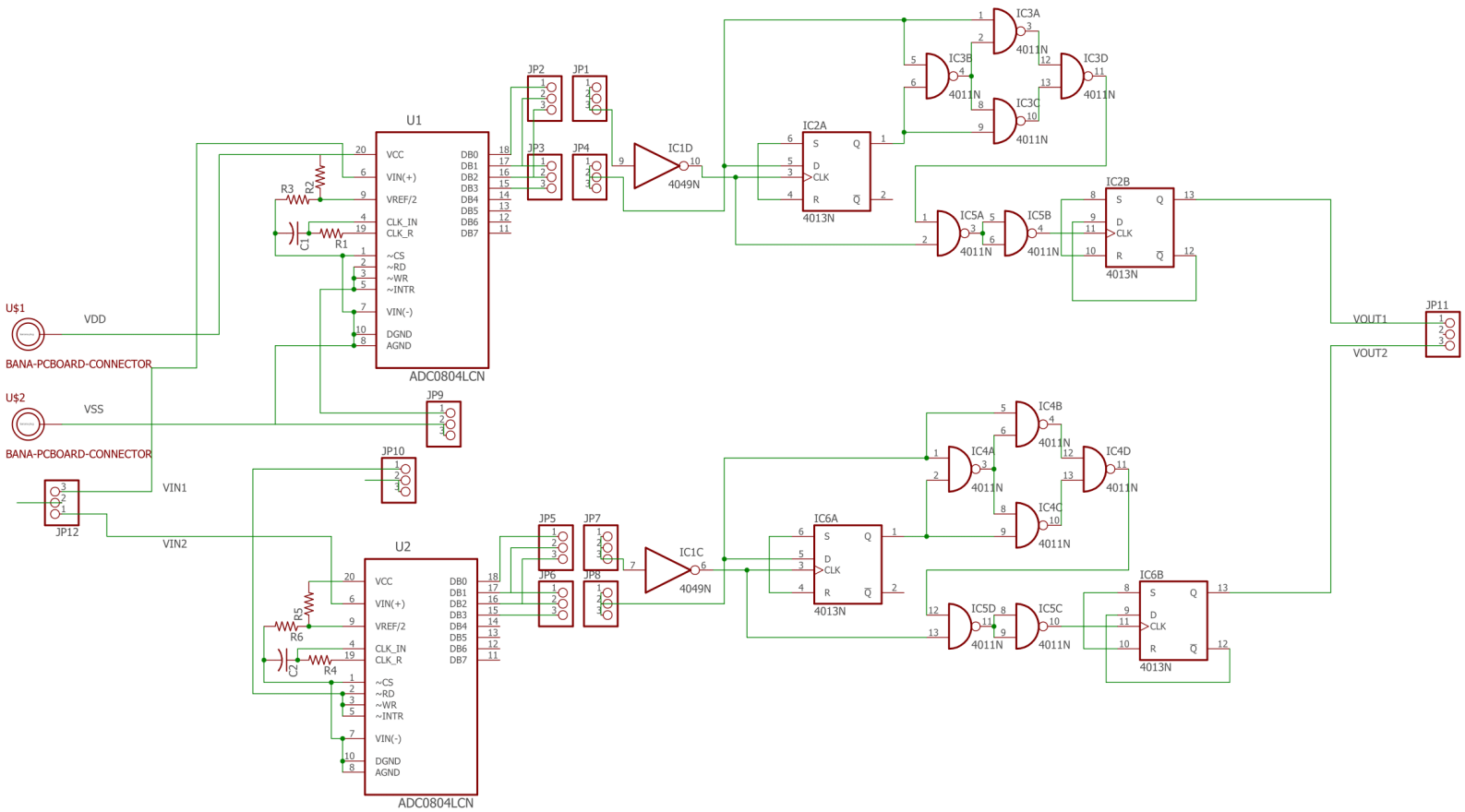
3. Planos

P.001 Esquemático de la tarjeta con 4 sistemas

P.002 Esquemático de la tarjeta generadora de eventos



Titulo: **Esquemático de la tarjeta con 4 sistemas**		UNIVERSITAT JAUME I
Nº de Plano: P.001	Fecha: 21-09-2016	Autor: Oscar Miguel Escrig
Formato: A4	Escala: 1:1	Archivo: Capa_BBB.sch



Titulo: ***Esquemático de la tarjeta generadora de eventos***		UNIVERSITAT JAUME I
Nº de Plano: P.002	Fecha: 18-05-2017	Autor: Oscar Miguel Escrig
Formato: A4	Escala: 1:1	Archivo: Eventos_hardware.sch

4. Pliego de condiciones

4.1 Definición y alcance

El objeto de este documento es establecer las condiciones técnicas, económicas, administrativas y legales para que el presente proyecto pueda materializarse en las condiciones especificadas, evitando posibles interpretaciones diferentes de las deseadas.

Como ya se estableció en el capítulo de objetos de la memoria, el objeto último del presente proyecto es realizar un estudio sobre la implementación de algoritmos de control basado en eventos usando el estándar IEC-61499, por lo que en este documento se establecen las condiciones de obligado cumplimiento que hay que respetar para reproducir los experimentos realizados así como las pautas para el correcto desarrollo de las tarjetas creadas.

En el caso de no cumplimiento de las condiciones que aquí se estipulan podría resultar en unos resultados en cuanto al control que no se ajustarían al presentado. Caso en el que el autor queda desvinculado completamente de las responsabilidades derivadas de su implementación y operación.

Así mismo, también se desvinculan las responsabilidades derivadas de una errónea utilización del sistema de control.

Por último, las condiciones aquí expuestas tienen prioridad sobre cualquier otro tipo de condición que se haya dado en el resto de documentos del proyecto. En caso de que algún aspecto de la implementación no quede definido, o su interpretación pueda ser ambigua, se priorizara lo descrito en la memoria en primer lugar y en los presupuestos en segundo.

4.2 Pliego de especificaciones técnicas particulares

4.2.1 Especificaciones de los equipos

Para la reproducción de los experimentos se deberán utilizar unos equipos informáticos con unas prestaciones iguales, o superiores, a la de los equipos utilizados para validar el desarrollo, de forma que estos puedan asegurar la correcta ejecución de todas las tareas del control. Por tanto, cada uno de los equipos sobre los que se ejecute un *runtime* del sistema de control deberá cumplir con las siguientes condiciones (en nuestro caso tenemos solo una tarjeta pero nada impide realizar el control de forma distribuida):

- Procesador: Procesador multi-hilo, con al menos 2 hilos físicos y 4 lógicos, con una frecuencia base por hilo de procesamiento de 2 GHz. Los procesadores tendrán en preferencia una arquitectura de 32 bits, si bien también se podrán usar procesadores de 64 bits, siempre que los mismos sean completamente compatibles con las aplicaciones de 32 bits.
- Memoria RAM: Memoria física instalada del tipo DD3 y superior a 4 GB, con una frecuencia de muestreo mayor de 1333MHz

- Memoria dura: Espacio disponible de al menos 500 MB para albergar los *runtime* y 4DIAC (este en el caso de que se utilice para el desarrollo).
- Tarjeta gráfica: no sujeta a condiciones
- Interfaces de red: Capaz de soportar una comunicación con protocolo TCP/IP para comunicar en red local (carga de *runtime* y descarga de archivos).
- Sistema operativo: Microsoft Windows XP o superior (revisar en función de las versiones de las diferentes herramientas software utilizadas).

4.2.2 Especificaciones del *runtime* FORTE

El *runtime* utilizado en este proyecto, FORTE, debe cumplir algunas condiciones. Para reproducir los resultados, se puede partir de la versión 1.8.4 (o anteriores), proporcionada por 4DIAC, y se le han añadido para su compilación todos los FB desarrollados en este proyecto (el repositorio completo está disponible en https://github.com/EstebanQuerol/Black_FORTE). Aunque en el proyecto no se especifique la metodología de programación se deben incluir los nuevos ficheros incluidos en el proyecto al camino de compilación, mediante la modificación de algunos archivos, para posteriormente compilarlos con la herramienta Eclipse.

4.2.3 Especificaciones de ejecución

Para la ejecución de los algoritmos de control utilizando el sistema desarrollado se deberán seguir los siguientes pasos:

1. Instalar la capa con los sistemas a controlar, así como el circuito generador externo de eventos, en función del tipo de control que se quiera llevar a cabo, conectándolas a los pines para los que el usuario haya programado las entradas de las aplicaciones.
2. Conectar la tarjeta BeagleBone Black a un ordenador con un USB, y proporcionar alimentación al circuito generador de eventos, iniciando el ADC de este circuito.
3. Acceder a la tarjeta BeagleBone Black desde el ordenador con una conexión SSH en calidad de superusuario (para casos donde se requiera habilitar algún permiso).
4. Ejecutar el *runtime* FORTE sobre la BeagleBone Black encargada del control.
5. Descargar el software de control sobre la BeagleBone Black utilizando el programa 4DIAC.

4.2.4 Especificaciones sobre el funcionamiento

El prototipo de control deberá ser operado por personal que haya sido formado para poder utilizar tanto el software de control, como las tarjetas presentadas, conociendo sus características técnicas y límites de funcionamiento.

5. Mediciones

Las diferentes partes en las que se divide el proyecto, también denominadas bajo el nombre de unidades de obra, se definirán en los siguientes apartados.

5.1 Concepción, diseño, construcción y pruebas de las tarjetas desarrolladas

Esta parte del trabajo consiste en la evaluación de los requisitos que se necesitan cumplir para la funcionalidad que se quiera aportar, evaluar la conveniencia de implementarla con un circuito electrónico más o menos complejo frente a una implementación por ordenador de esta funcionalidad, y en el caso de ser necesario el circuito, concebir un circuito, lo más sencillo posible para poder imprimirlo en una capa, y que cumpla con los requisitos y las metas propuestas según una serie de condiciones de utilización, habiendo comprobado su correcto funcionamiento.

5.2 Desarrollo y pruebas de las librerías de FB

Para que se pueda llevar a cabo el control de los sistemas de forma satisfactoria no solo se deben implementar los FB que se encargan directamente del control de los sistemas, sino que además se deben implementar todos los FB que permitan gestionar de forma correcta las redes de FB. Por otra parte, se deben desarrollar todas las aplicaciones para el control, ya que, no es suficiente con los bloques si no que es necesario realizar un modelado correcto.

En el caso de detectar fallo en alguno de los elementos se debe buscar la causa, bien sea por un fallo en el *runtime*, en el modelado de la aplicación o en la implementación del código y subsanarlo. Este proceso es iterativo ya que se pueden dar varios motivos para un fallo y en algunas ocasiones resulta costosa la detección de la fuente del error.

5.3 Configuración y ajuste de controladores y comprobación del funcionamiento

Una vez presentados todos los elementos para realizar las pruebas, solo queda realizar los experimentos. Para ello se deben ajustar los controladores, comprobar que ofrecen una respuesta adecuada a las simulaciones realizadas y por tanto validar el modelo y la implementación. Una vez realizados estos experimentos, hay que analizar los resultados para sacar conclusiones sobre posibles mejoras, en cuanto al CBE o a la implementación de la aplicación de control se refiere.

6. Presupuesto

Para esta sección se empleará la estructura empleada en la parte de Mediciones y se finalizará con un balance global

6.1 Costes derivados de los trabajos electrónicos

Para esta parte se han desglosado los costes en función de la finalidad que tengan. En primer lugar encontramos los costes para la realización de la placa que contiene los 4 sistemas a controlar:

Nombre	Componente	Encapsulado	Descripción	Precio (en €)
C1	C-EU075-032X103	C075-032X103	CAPACITOR	0,1670
C2	C-EU075-032X103	C075-032X103	CAPACITOR	0,1822
C3	C-EU075-032X103	C075-032X103	CAPACITOR	0,1584
C4	C-EU075-032X103	C075-032X103	CAPACITOR	0,1360
C5	C-EU075-032X103	C075-032X103	CAPACITOR	0,1688
C6	C-EU075-032X103	C075-032X103	CAPACITOR	0,1672
C7	C-EU075-032X103	C075-032X103	CAPACITOR	0,1910
C8	C-EU075-032X103	C075-032X103	CAPACITOR	0,1438
D1	1N4004	DO41-10	DIODE	0,0790
D2	1N4004	DO41-10	DIODE	0,0790
D3	1N4004	DO41-10	DIODE	0,0790
D4	1N4004	DO41-10	DIODE	0,0790
IC1	LM324N	DIL14	OP AMP	0,7000
IC2	LM324N	DIL14	OP AMP	0,7000
R1	R-EU_0207/10	0207/10	RESISTOR	0,0210
R2	R-EU_0207/10	0207/10	RESISTOR	0,0192
R3	R-EU_0207/10	0207/10	RESISTOR	0,0174
R4	R-EU_0207/10	0207/10	RESISTOR	0,0182
R5	R-EU_0207/10	0207/10	RESISTOR	0,0230
R6	R-EU_0207/10	0207/10	RESISTOR	0,0190
R7	R-EU_0207/10	0207/10	RESISTOR	0,0238
R8	R-EU_0207/10	0207/10	RESISTOR	0,0200
R9	R-EU_0207/10	0207/10	RESISTOR	0,0242
R10	R-EU_0207/10	0207/10	RESISTOR	0,0232
R11	R-EU_0207/10	0207/10	RESISTOR	0,0250
R12	R-EU_0207/10	0207/10	RESISTOR	0,0188
R13	R-EU_0207/10	0207/10	RESISTOR	0,0188
R14	R-EU_0207/10	0207/10	RESISTOR	0,0246
R15	R-EU_0207/10	0207/10	RESISTOR	0,0246
R16	R-EU_0207/10	0207/10	RESISTOR	0,0204

Tabla 10: Desglose de los componentes para la realización de la tarjeta con los sistemas (1 de 2)

Nombre	Componente	Encapsulado	Descripción	Precio (en €)
R17	R-EU_0207/10	0207/10	RESISTOR	0,0188
R18	R-EU_0207/10	0207/10	RESISTOR	0,0196
R19	R-EU_0207/10	0207/10	RESISTOR	0,0252
R20	R-EU_0207/10	0207/10	RESISTOR	0,0212
R21	R-EU_0207/10	0207/10	RESISTOR	0,0178
R22	R-EU_0207/10	0207/10	RESISTOR	0,0198
Placa	Placa que contiene el circuito			4,7200
TOTAL (€)				**8,2140**

Tabla 11: Desglose de los componentes para la realización de la tarjeta con los sistemas (2 de 2)

En segundo lugar, continuamos con los costes para la tarjeta generadora de eventos externos:

Nombre	Componente	Encapsulado	Descripción	Precio (en €)
C1	C-US075-032X103	C075-032X103	CAPACITOR	0,1670
C2	C-US075-032X103	C075-032X103	CAPACITOR	0,1670
IC1	4049N	DIL16	Hex inverting BUFFER	0,327
IC2	4013N	DIL14	Dual D FLIP FLOP	0,353
IC3	4011N	DIL14	Quad 2-input NAND	0,204
IC4	4011N	DIL14	Quad 2-input NAND	0,204
IC5	4011N	DIL14	Quad 2-input NAND	0,204
IC6	4013N	DIL14	Dual D FLIP FLOP	0,353
JP1 – JP12	PINHD-1X3	Se compran en tiras de 20	PIN HEADER	1,48
R1	R-US_0207/10	0207/10	RESISTOR	0,0180
R2	R-US_0207/10	0207/10	RESISTOR	0,0210
R3	R-US_0207/10	0207/10	RESISTOR	0,0210
R4	R-US_0207/10	0207/10	RESISTOR	0,0180
R5	R-US_0207/10	0207/10	RESISTOR	0,0210
R6	R-US_0207/10	0207/10	RESISTOR	0,0210
U$1	BANA-CONNECTOR	BANA_PLUG	Banana Plug to PC Board	1,99
U$2	BANA-CONNECTOR	BANA_PLUG	Banana Plug to PC Board	1,99
U1	ADC0804LCN	DIP254P762X508-20	A/D Converter	2,498
U2	ADC0804LCN	DIP254P762X508-20	A/D Converter	2,498
Placa	Placa que contiene el circuito			4,7200
TOTAL (€)				**17,2750**

Tabla 12: Desglose de los componentes para la realización de la tarjeta generadora de eventos

Finalmente, encontramos los costes de la mano de obra empleada:

Concepto	Coste	Unidades	Coste (en €)
BeagleBone Black	40	1	40
Trabajos de electrónica	50 (por hora)	100 (horas)	5000
TOTAL (€)			**5040**

Tabla 13: Costes de ingeniería, electrónica

6.2 Costes derivados de los trabajos de programación

En esta parte no se han empleado materiales físicos, y en cuanto los virtuales (licencias de programas, etc.), se trata de programas *opensource*, con lo cual solo queda incluir el trabajo de ingeniería.

Trabajo de Ingeniería	Coste/hora	Horas	Coste (en €)
Programación, testeo y validación de los FB	50	75	3750

Tabla 14: Costes de ingeniería, programación

6.3 Costes derivados de los trabajos de automatización

En esta parte, como en la parte anterior, no se han empleado materiales físicos, pero en este caso sí que se debe incluir las licencias al trabajo de ingeniería.

Concepto	Coste	Unidades	Coste (en €)
Licencia Matlab	2000	1	2000
Trabajos relativos a la automatización	50 (por hora)	100 (horas)	5000
TOTAL (€)			**7000**

Tabla 15: Costes relativos a los trabajos de automatización

6.4 Presupuesto final

Finalmente juntamos las tres categorías anteriormente desglosadas para obtener el coste total del proyecto.

Concepto	Coste (en €)
Costes derivados de los trabajos electrónicos	5065,489
Costes derivados de los trabajos programación	3750
Costes derivados de los trabajos automatización	7000
TOTAL (€)	**15815,489**

Tabla 16: Presupuesto global del proyecto

UNIVERSITAT JAUME·I