Universidad
Carlos III de Madrid

TESIS DOCTORAL
SISTEMA AVANZADO DE PROTOTIPADO RÁPIDO PARA CONTROL EN EXOESQUELETOS Y DISPOSITIVOS MECATRÓNICOS

Autor:
Antonio Flores Caballero
Director:
María Dolores Blanco Rojas
Codirector:
Luis Enrique Moreno Lorente
Tutor:
María Dolores Blanco Rojas

DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Leganés, 15 de Diciembre 2014

TESIS DOCTORAL (THESIS)

SISTEMA AVANZADO DE PROTOTIPADO RÁPIDO PARA CONTROL EN EXOESQUELETOS Y DISPOSITIVOS MECATRÓNICOS.

Autor (Candidate): Antonio Flores Caballero

Director (Adviser): María Dolores Blanco Rojas
Codirector (Co-Adviser): Luis Enrique Moreno Lorente

Tribunal (Review Committee)

Presidente (Chair): Jose Luis Pons Rovira _____

Vocal (Member): Antonio Giménez Fernández _____

Secretario (Secretary): Santiago Garrido Bullón _____

Suplente (Substitute): _____

Título (Grade): Doctorado en Ingeniería Eléctrica, Electrónica y Automática

Calificación: _____

Leganés, 15 de Diciembre de 2014

Esta tesis ha sido financiada por el Ministerio de Ciencia e Innovación del Gobierno de España, y está enmarcada dentro de proyecto HYPER CONSOLIDER-INGENIO 2010 (ref. 2010/00154/001).
Robotics Lab – Universidad Carlos III de Madrid

Abstract

One of the most challenging fields of engineering is system control, which was established with the aim of automating complex systems without human interaction. Thus, by means of the feedback theory, it is intended to modify the most critical system variables to the obtain the desired behavior. Event most of the applications are linear control-based, there exist some occasions where it becomes necessary to apply other approaches for obtaining reasonable results. During the last years, a new type of elements named intelligent materials have been discovered and developed. They yield very interesting properties although they are extremely difficult to control with classical techniques. Even they have a limited number of applications due to their complexity, the actual increase in computer power make them attractive. In such a way, the continuous improvement of their size and performance have converted these intelligent materials into a future reference.

The aim of this dissertation is to suggest solutions to several problems controlling SMA actuators. Due to their high hysteresis, a novel methodology for their identification, adjustment and implementation in the control loop is presented improving the conventional linear methods. Thus, several experimental results and comparisons among several control methods are presented. Furthermore, the initial state search problem in the SMA actuator has been solved.

The presented contributions of this thesis have been tested and fully optimized in order to be installed in a real time actuation platform. In such a way, a new field of applications and future works is established suggesting a wide range of new possibilities.

Keywords: Control engineering, SMA, hysteresis, Prandtl-Ishlinskii, Bouc-Wen, HYPER, Differential Evolution.

Resumen

La necesidad de automatizar sistemas complejos da origen a la ingeniería de control. Utilizando el principio de realimentación, se pretende conseguir que las variables de interés del sistema se acerquen a un comportamiento deseado. En un principio se utilizaron sistemas de control basados en lógica combinacional mediante relés y electrónica analógica para tratar magnitudes más allá de la lógica binaria, estos sistemas adolecían de una gran falta de fiabilidad. Con la aparición del microprocesador, se abrieron las posibilidades del control digital. Con el paso de las décadas, la necesidad de controlar sistemas grandes y complejos en plazos de tiempo cada vez más cortos, provocó la aparición de lenguajes de programación de muy alto nivel, basados en gráficos, que fueron adaptados para su utilización en sistemas de control. Son los denominados hardwares de control de prototipado rápido, más conocidos como Rapid Control Prototyping hardware (RCP).

El objetivo de la presente tesis doctoral es proponer y hacer realidad una solución alternativa al uso de RCP comerciales, sin menoscabo de la capacidad computacional y de entrada/salida. Igualando o superando la facilidad, capacidad de programación e interfaces de intercambio de datos. De forma que se tenga un RCP totalmente compatible con las herramientas de desarrollo de más alto nivel utilizadas en ingeniería por un coste tan extremadamente reducido que permita disponer del RCP en gran número.

Las contribuciones presentadas en esta tesis han sido probadas y optimizadas, por lo que se dispone de la plataforma RCP con actuación en tiempo real. Se está empleando en multitud de Trabajos Fin de Grado, Tesis de Máster y Tesis Doctorales, no ya sólo en esta escuela, también en escuelas extranjeras.

Palabras clave: Control, SMA, histéresis, Prandtl-Ishlinskii, Bouc-Wen, Evolución Diferencial, HYPER.

No hay computador lento,
hay código sin optimizar.

Agradecimientos

Agradecer a mis padres y a mi hermana el estar siempre ahí. Ellos ya sabían que yo sería Doctor en ingeniería, aún cuando ni me lo había planteado, ni aún cuando existía esa posibilidad... ahora que lo pienso, siempre supe que acabaría siendo un experto en las destrezas técnicas que más me gustan, aún cuando no había pensado nunca en ello.

También agradecer al equipo del proyecto HYPER de la Universidad Carlos III de Madrid, María Dolores Blanco Rojas y Luis Enrique Moreno Lorente la confianza depositada en mis destrezas, sabieron ver su utilidad aún cuando éstas se encontraban en estadios incipientes. También debo agradecer lo mismo a Santiago Garrido Bullón, por sus lecciones sobre control siempre que le he consultado y a Ramón Barber Castaño por el interés, en los contenidos de esta presente tesis doctoral, que he logrado despertar en él y el buen uso que está haciendo de la misma.

Agradecer, y no puedo decir en última instancia, pues realmente ésta nunca llega, a los compañeros del laboratorio: Dorin-Sabin Copaci, Álvaro Villoslada, Alejandro Martín Clemente, David Sánchez, Alejandro Lumbier; pues los contenidos de esta tesis les hacen la vida más fácil, y entre todos hemos podido llevar a cabo 'juguetes' increíbles, combinando todo nuestro conocimiento. También agradecer a los betatesters extranjeros y nacionales ajenos a la Universidad Carlos III de Madrid, tales como Marcin Chodnicki, oficial ingeniero del ejército de tierra de Polonia, a José Atienza de la Universidad de Almería y a otros tantos que no tendría espacio suficiente para nombrar, por haber confiado en los contenidos de esta tesis y haberme ayudado a mejorarlos localizando defectos.

La vida se abre camino, el conocimiento se ve que también.

Abreviaturas I

A/D	**A**nalog to **D**igital
API	**A**pplication **P**rogramming **I**nterface
ARCP	**A**dvanced **R**apid **C**ontrol **P**rototyping
BIOS	**B**asic **I**nput **O**utput **S**ystem
CASE	**C**omputer **A**ided **S**oftware **E**ngineering
CAN	**C**ontroller **A**rea **N**etwork
CDC	**C**ommunication **D**evice **C**lass
CCM	**C**ore **C**oupled **M**emory
CCP	**C**an **C**alibration **P**rotocol
CMSIS	ARM **C**ortex-**M** **S**oftware **I**nterface **S**tandard
CPU	**C**entral **P**rocessing **U**nit
DAC	**D**igital to **A**nalog **C**onverter
DCMI	**D**igital **C**amera **M**anagement **I**nterface
DCS	**D**istributed **C**ontrol **S**ystem
DLR	**D**eutsches Zentrum für **L**uft- und **R**aumfahrt (Centro Aeroespacial Alemán)
DMA	**D**irect **M**emory **A**ccess
DOF	**D**egree **O**f **F**reedom or **D**epartment **O**f **D**efense
DSP	**D**igital **S**ignal **P**rocessing/or
EN	**E**uropean **N**orm
EMG	**E**lectro**m**io**g**rafía
EXTI	**Ext**ernal **I**nterrupt
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
FPU	**F**loating **P**rocessing **U**nit
FSMC	**F**lexible **S**tatic **M**emory **C**ontroller
GPIO	**G**eneral **P**urpose **I**nput-**O**utput
GPU	**G**raphics **P**rocessing **U**nit
HID	**H**uman **I**nterface **D**evice
HIL	**H**ardware **I**n the **L**oop
I/O	**I**nput-**O**utput
I2C	**I**nter-**I**ntegrated **C**ircuit

Abreviaturas II

I2S	**I**ntegrated **I**nterchip **S**ound
IEC	**I**nternational **E**lectrotechnical **C**ommission
IRQ	**I**nterrupt **Req**uest
JTAG	**J**oint **T**est **A**ction **G**roup norma IEEE 1149.1
LIN	**L**ocal **I**nterconnect **N**etwork
MBD	**M**odel **B**ased **D**esign
MCU	**M**icrocontroller
MEX	**M**atlab **Ex**ecutable
NP	**N**euro**p**rótesis
NR	**N**euro**r**obot
PID	**P**roporcional-**I**ntegral-**D**erivativo
PIL	**P**rocessor **In** the **L**oop
PWM	**P**ulse **W**idth **M**odulation
RAM	**R**andom **A**ccess **M**emory
RCP	**R**apid **C**ontrol **P**rototyping
ROM	**R**ead **O**nly **M**emory
RTOS	**Real**-**TimeO**perating **S**ystem
SBC	**S**ingle **B**oard **C**omputer
SDK	**S**oftware **D**evelopment **K**it
SIL	**S**oftware **I**n the **L**oop or **S**ecurity **I**ntegration **L**evel
SMA	**S**hape **M**emory **A**lloy
SOC	**S**ystem **O**n **C**hip
SPI	**S**erial **P**eripheral **I**nterface
SWD	**S**erial **W**ire **D**ebug
TFT-LCD	**T**hin **F**ilm **T**ransistor **L**iquid **C**rystal **D**isplay
TLC	**T**arget **L**anguage **C**ompiler
UART	**U**niversal **A**syncronous **R**eceiver/**T**ransmitter
UML	**U**nified **M**odeling **L**anguage
USB	**U**niversal **S**erial **B**us
XCP	E**x**tended **C**alibration **P**rotocol

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

1.1. Origen de esta tesis

Esta tesis tiene su origen en las necesidades del grupo de investigación que colabora en el proyecto HYPER[1] del RoboticsLab, Universidad Carlos III de Madrid, liderados por el Dr. Luis Enrique Moreno Lorente y la Dra. María Dolores Blanco Rojas.

La misión de este grupo de investigación, respecto del proyecto involucrado, es la de analizar, modelar, simular y controlar nuevos sistemas de actuación basados en componentes poco comunes tales como aleaciones con memoria de forma y motores de ultrasonidos, que presentan fuertes no linealidades en su comportamiento.

Dada la compleja naturaleza de estos sistemas de actuación y el estricto plazo temporal del proyecto, se hace imprescindible contar con un entorno software que proporcione las herramientas de análisis científico necesarias, de forma que no sólo facilite las investigaciones sino que permita también ahorrar tiempo. Estos entornos software son aquellos orientados al diseño basado en modelos, en los que los planteamientos y simulaciones pueden realizarse utilizando elementos gráficos, resultando más sencillo transferir las ideas del pensamiento humano al ordenador.

Esta tesis propone la utilización del mismo software de análisis científico para plantear y obtener los controladores hardware para los sistemas de actuación no

[1] Acrónimo del proyecto "Dispositivos Híbridos Neuroprotésicos y Neurorobóticos para Compensación Funcional y Rehabilitación de Trastornos del Movimiento" financiado por el programa Consolider-Ingenio 2010 del Ministerio de Ciencia e innovación del Gobierno de España.

lineales, proponiendo un sistema avanzado de Prototipado Rápido para Control.

1.2. Proyecto HYPER

El objetivo final del proyecto es crear robots de los denominados *wearable-robots*, cuya traducción literal es robots vestibles, y que principalmente están pensados para sustituir y/o ayudar las funciones del esqueleto humano. De aquí que encontremos el otro nombre *wearable-exoskeletons*, o exoesqueletos vestibles. Cuyas características principales son las de comodidad, facilidad de uso y portabilidad.

Figura 1.1: *Logo del proyecto HYPER. (Acrónimo de) Dispositivos Híbridos Neuroprotésicos y Neurorobóticos para Compensación Funcional y Rehabilitación de Trastornos del Movimiento.*

El proyecto HYPER pretende representar un avance significativo en la investigación de dispositivos neurorrobóticos y neuroprotésicos en interacción cercana con el cuerpo humano, tanto en la rehabilitación como en la compensación funcional de trastornos motores en actividades de la vida diaria. El proyecto centra sus actividades en nuevos neurorobots (NR) vestibles y neuroprótesis (NP) que combinan estructuras biológicas y artificiales con el objetivo de superar las principales limitaciones de las soluciones robóticas actuales.

Los principales objetivos del proyecto son la restauración de la función motora en pacientes con lesión medular, a través de la compensación funcional, y promover el re-aprendizaje del control motor en pacientes afectados por accidente cerebro vascular y parálisis cerebral, por medio de un uso integrado de neurorrobots y neuroprótesis.

El proyecto validará funcional y clínicamente el concepto de desarrollo de sistemas Humano-Robot híbridos, para la rehabilitación compensación funcional de trastornos motores bajo el paradigma de asistencia bajo demanda. En HYPER, se asume que el avance en las terapias de rehabilitación física depende de la obtención de una comunicación más transparente, entre los sistemas humanos y las máquinas, y por lo tanto, se explorarán los diferentes niveles de actividad neural humana.

Además se tratarán preguntas fundamentales en la frontera del conocimiento en diferentes disciplinas tecnológicas y científicas. Estas preguntas son investigadas en seis líneas de investigación: biomecánica, control neuromotor, tecnologías de control, tecnologías de sensores actuadores y alimentación, interfaces multimodales cerebro-máquina y la adaptación de sistemas híbridos a escenarios de aplicación.

Dentro de las necesidades del proyecto HYPER, el grupo de la Universidad Carlos III de Madrid tiene asignado el diseño y control de nuevos actuadores para su aplicación sobre el exoesqueleto final. Es por este motivo por el que ha surgido esta tesis y mediante la cual se cumple con los requisitos de control de actuadores dentro del proyecto.

1.3. Prototipado Rápido para Control (RCP)

El concepto de Prototipado Rápido para Control (RCP, del inglés "Rapid Control Prototyping") hace referencia a todas las técnicas software y hardware necesarias para acortar los tiempos de desarrollo y puesta en marcha de sistemas de control, haciendo uso de un alto nivel de abstracción en la programación. El alto nivel de abstracción se obtiene al utilizar lenguajes basados en gráficos, que permiten transferir las ideas del programador al computador de una forma más natural, obviando todos los aspectos de configuración a bajo nivel del sistema hardware, pues estos pasos se realizan automáticamente. La figura 1.2 ilustra el alto nivel de abstracción proporcionado por un lenguaje gráfico frente a los lenguajes textuales tradicionales, como ejemplo se puede considerar que cada cubo del dibujo representa una semana de trabajo, obteniendo los mismos resultados con cada lenguaje.

Utilizar un alto nivel de abstracción en la programación, valiéndose de un lenguaje basado en gráficos, conlleva ventajas incluso para los más expertos en programación textual. Utilizar un lenguaje de este tipo pone al alcance de un público multidisciplinar el hacer uso de un hardware de control real [1, 2], también permite que dicho controlador se comporte exactamente tal y cómo se ha definido su comportamiento en el lenguaje gráfico que le ha servido de programación. Este último punto no sería posible si se tuviera que realizar una transcripción manual desde el lenguaje gráfico a un lenguaje textual, el programador introduciría modificaciones de forma involuntaria [3].

Estos lenguajes de alto nivel de abstracción se encuentran disponibles en entornos software basados en el diseño en base a modelos (del inglés "Model Based

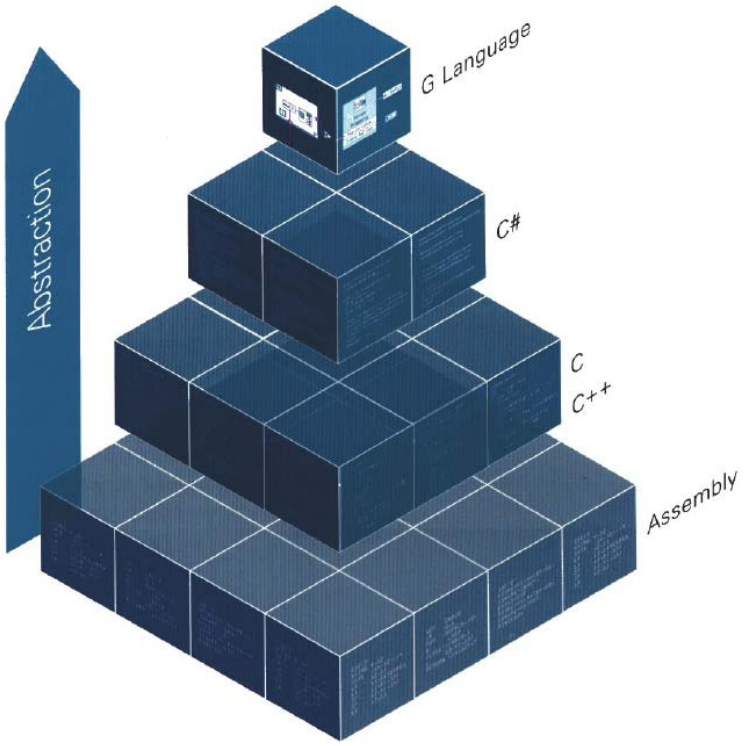


Figura 1.2: *Nivel de abstracción proporcionado por el uso de un lenguaje gráfico frente a otros lenguajes de programación textuales. Fuente: Folleto publicitario LabVIEW®*

Design"), con el mismo lenguaje gráfico se pueden realizar simulaciones de sistemas electromecánicos, identificación de sistemas en base a capturas de datos... además de poder programar los sistemas RCP entre otras muchas tareas.

1.3.1. Concepto actual de Prototipado Rápido para Control

Tradicionalmente el uso de sistemas RCP se ha entendido como un paso intermedio entre el planteamiento teórico del controlador en el laboratorio y su implementación final. La figura 1.3 muestra el concepto actual de RCP tanto en investigación como en empresas.

Los sistemas RCP actualmente están relegados a ser utilizados únicamente en el laboratorio, en la fase de prototipado, como una herramienta que sirve para poner en práctica el controlador teórico y verificar su funcionamiento en un proceso de rápidas iteraciones en espacios de tiempo muy cortos. Debido a este planteamiento, los sistemas RCP generalmente sólo pueden ser programados con un repertorio de funcionalidades básicas, se pueden utilizar los periféricos de entrada/salida

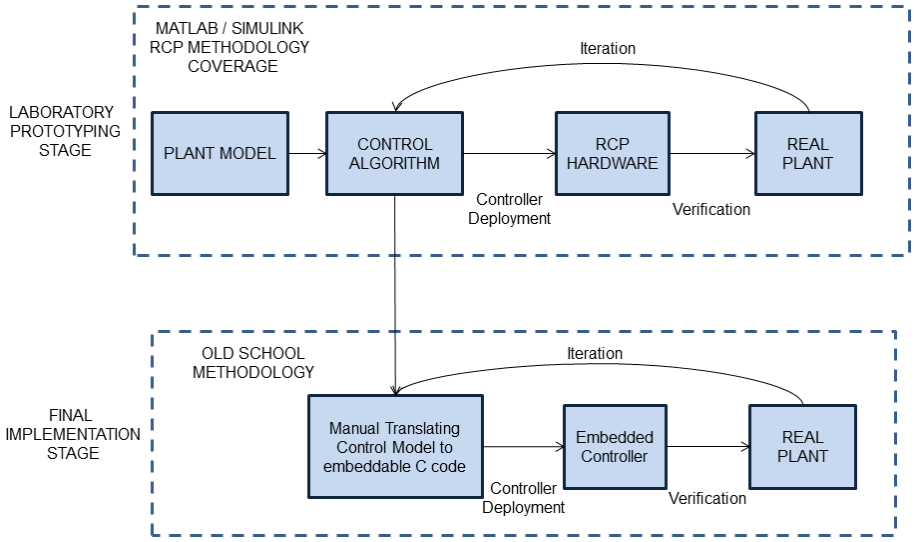


Figura 1.3: *Metología tradicional utilizando un sistema RCP en la etapa de prototipado en laboratorio*

(I/O, del inglés "Input/Output") y se puede establecer una única tarea en Tiempo-Real. El resto de funcionalidades más avanzadas como el tratamiento de errores, la gestión de un entorno multitarea, optimizaciones a nivel de código fuente y arquitectura... quedan postergadas para la implementación final. Dicha implementación final se realiza mediante una transcripción manual a lenguaje textual del modelo gráfico, que tendrá como destino un hardware de control de tamaño reducido y más barato que el sistema RCP utilizado en el laboratorio.

Los mayores incovenientes de realizar la programación manual del sistema de control final, son la pérdida de las ventajas en ahorro de tiempo del sistema RCP y la pérdida de la trazabilidad entre el código escrito de forma manual y el código basados en gráficos del sistema RCP. Esto implica que la etapa de implementación final continúe consumiendo ingentes cantidades de tiempo y personal muy especializado, las pruebas de verificación tendrán que volver a realizarse y con cada fallo encontrado, se tenga que reiniciar el proceso de implementación final. Esta metodología en lo que se refiere a la etapa de implementación final se encuentra explicada de forma gráfica en la figura 1.4.

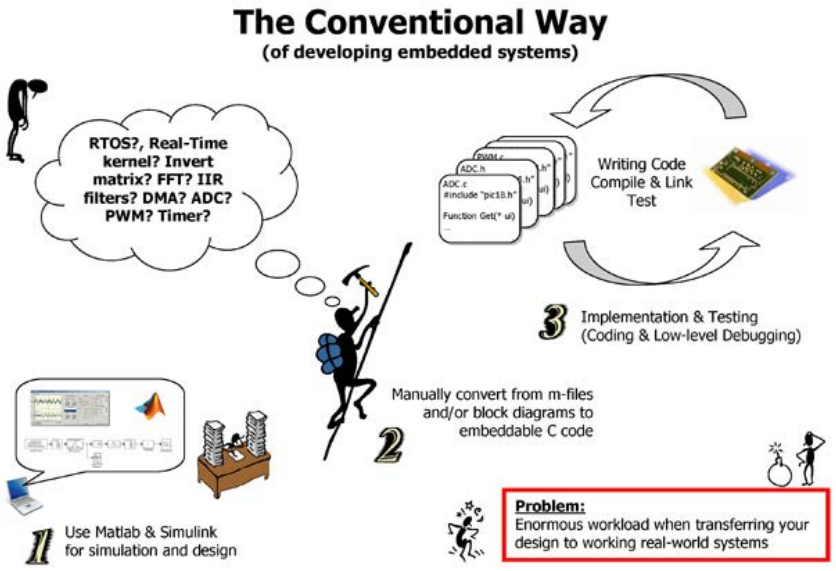


Figura 1.4: *Metología tradicional utilizada para la etapa de implementación final. Imagen cortesía de Aimagin Ltd.*

1.3.2. Nuevo Concepto de Prototipado Rápido para Control

En vista de lo expuesto anteriormente, esta Tesis Doctoral plantea una metodología de trabajo completamente innovadora, utilizar el sistema RCP para cubrir las etapas de desarrollo y experimentación en el laboratorio, así como la etapa de implementación final y verificación al mismo tiempo. Dotando también al sistema RCP de funcionalidades avanzadas sin necesidad de recurrir a integrar éstas de forma manual. Esta propuesta se ilustra en la figura 1.5, eliminando así la necesidad de recurrir a todos los pasos descritos en la figura 1.4.

En el momento en que el sistema funciona perfectamente en el laboratorio, queda listo para su producción masiva, sin necesidad de recurrir a otro hardware de control al que traspasar los resultados reflejados por el sistema RCP. Para cumplir esta novedosa metodología de trabajo hace falta disponer de un controlador digital disponible en el mercado de forma industrializada, producido en serie a bajo coste y orientado a la implementación final, cuyas características computacionales y facilidad de manejo lo hagan apto también para su uso como sistema RCP de laboratorio. De forma que se cubran ambas etapas con un sólo controlador, eliminando de la fase de desarrollo la necesidad de una gran cantidad de personal cualificado, tiempo y recursos económicos, obteniendo unos resultados mucho más satisfactorios pues la implementación final cumple ahora al 100 % el modelo de control. Otra ventaja principal de un sistema RCP como este, es que pone al alcance de

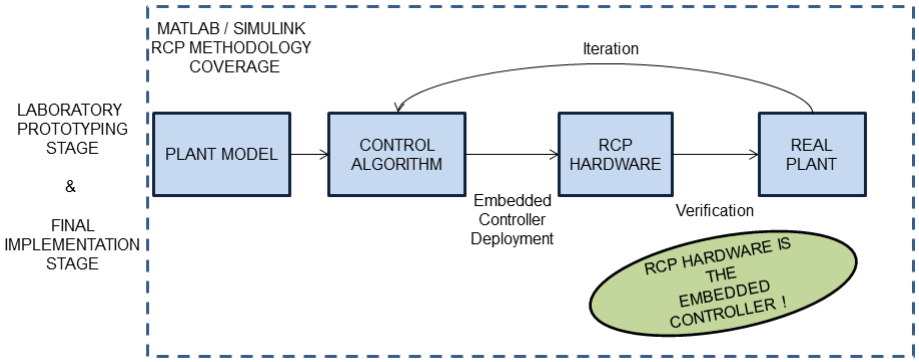


Figura 1.5: *Metología propuesta en esta Tesis para cubrir las etapas de prototipado en laboratorio e implementación final al mismo tiempo*

una audiencia multidisciplinar la capacidad de plantear una teoría de control, hacerla realidad en el laboratorio y difundirla de forma masiva ya integrada en un hardware de control para implementación final, del que dicha audiencia desconoce completamente los detalles de su programación interna a bajo nivel.

1.4. Requisitos del proyecto HYPER

El proyecto HYPER plantea dispositivos mecatrónicos para ayudar en la rehabilitación de seres humanos, son dispositivos que interactúan con las extremidades del cuerpo humano imponiendo o ayudando al movimiento. Por tanto es una aplicación en la que la seguridad e integridad del paciente son los elementos prioritarios, no es aceptable que las lesiones del paciente se tornen más graves o surjan nuevas lesiones como fruto del uso de los dispositivos mecatrónicos rehabilitadores.

Por tanto el proyecto HYPER plantea escenarios donde la seguridad es el elemento crítico. Es crítica la seguridad del ser humano. Los dispositivos mecatrónicos que surjan derivados de las investigaciones de los diferentes grupos de investigación involucrados, deben tener en cuenta por encima de todo la seguridad del paciente.

El concepto de seguridad hace alusión a la probabilidad de que un dispositivo funcione correctamente y en caso de fallo o pérdida de la capacidad de funcionamiento, éste permanezca en un estado seguro.

También debido al entorno de aplicación, un entorno clínico y no simplemente de pruebas en un laboratorio, los dispositivos deben diseñarse de forma que posean una alta probabilidad de desempeñar sus funciones correctamente a lo largo de extensos períodos de tiempo.

Para atender los criterios de seguridad existen reglas normalizadas que clasifican la seguridad de un sistema de control en base a su escenario de aplicación, se denomina "Safety Integrity Level" (SIL) y existen cuatro clasificaciones [4, 5, 6]:

Safety Integrity Level 1:

- Sin necesidad de Tiempo-Real
- Programas de tamaño reducido
- Fallos del sistema no implican daños a personas, financieros, a circuitos ni a elementos mecánicos
- No hay elementos mecánicos
- No hay bucles de control
- Ejemplo: Programación de entrenamiento, prueba y error

Safety Integrity Level 2:

- Algunas partes del programa requieren Tiempo-Real
- Fallos del sistema pueden dañar el interés de grupos de personas, provocan pequeños riesgos financieros, provocan pequeños daños a elementos mecánicos
- No hay bucles de control o no son relevantes
- Estándar Recomendado: ISO9001
- Ejemplo: Impresora 3D doméstica

Safety Integrity Level 3:

- Fuerte contenido de Tiempo-Real
- Fallos del sistema pueden dañar físicamente a grupos de personas, provocar riesgos financieros, provocar daños a elementos mecánicos, provocar la destrucción de electrónica de potencia
- Bucles de control sencillos basados en PID
- Estándar Recomendado: TickIT,ISO9001,IEC 61508
- Ejemplo: Dispositivos médicos no críticos en seguridad, dispositivos industriales no críticos en seguridad

Safety Integrity Level 4:

- Completamente basados en Tiempo-Real
- Se requieren tiempos de puesta en marcha reducidos, del orden de milisegundos
- Se requiere tolerancia/recuperación frente a fallos de funcionamiento
- Fallos del sistema pueden dañar de forma grave y físicamente a individuos, provocar grandes riesgos financieros, provocar graves daños a elementos mecánicos, provocar la destrucción de electrónica de potencia, provocar la pérdida de dispositivos que no pueden recuperarse para ser reparados
- Bucles de control sencillos y complejos/adaptativos
- Estándar Recomendado: TickIT,ISO9001,DO178/B/C,IEC 60880,ISO 26262
- Ejemplo: Dispositivos médicos críticos en seguridad, dispositivos industriales críticos en seguridad, control del motor y frenos y transmisión automática de un automóvil, control de sistemas de transporte público, control de sistemas nucleares

Atendiendo a los criterios del proyecto y a la clasificación de los niveles de seguridad para los sistemas de control, el sistema RCP avanzado propuesto en esta tesis debe encajar con el nivel de seguridad SIL 4.

1.5. Objetivos de la tesis

El objetivo de la presente tesis doctoral consiste en plantear y hacer realidad un sistema Avanzado de Prototipado Rápido para Control (ARCP) que supere en posibilidades y facilidad de programación a los sistemas RCP disponibles actualmente.

Este sistema ARCP integrará mediante lenguaje gráfico todas aquellas funcionalidades avanzadas que actualmente se integran de forma manual, de manera que todas las etapas del desarrollo y verificación de un controlador hardware queden integradas en la misma etapa de trabajo, siendo esta etapa la de prototipado en laboratorio e implementación final al mismo tiempo. Obteniendo de esta manera dos grandes ventajas frente al uso tradicional de un sistema RCP, se ahorra mucho más tiempo y recursos humanos especializados, y el controlador que se obtiene cumple al 100 % con el comportamiento descrito por medio del lenguaje gráfico, se elimina el factor humano en la transcripción manual del lenguaje gráfico al lenguaje textual. La figura 1.7 muestra los objetivos de la tesis de forma más visual.

El sistema ARCP tendrá en cuenta todos los requisitos de seguridad del proyecto involucrado así como los requisitos de tamaño y consumo reducidos. Los planteamientos del sistema ARCP se extenderán a diferentes arquitecturas hardware, de forma que queden cubiertos un amplio abanico de escenarios.

El software de análisis científico basado en modelos sobre el que se construirá el soporte del sistema ARCP es MATLAB®/SIMULINK®, debido a su facilidad para trabajar y simular sistemas de múltiples dominios. Los objetivos se enumeran a continuación:

- Precio, coste, tamaño, peso y consumo energético reducido.
- El sistema RCP debe ser completamente determinista y debe cubrir simultáneamente la étapa de prototipado en el laboratorio y la etapa de implementación final, a diferencia de los de sistemas RCP comerciales.
- El sistema debe ser predecible y fiable. Predecible para poder conocer en todo momento el estado y comportamiento del sistema, por lo que se debe evitar el uso de capas de abstracción software; tales como sistemas operativos. La fiabilidad del sistema computacional recae en la sencillez de sus planteamientos, evitar capas de abstracción software redunda igualmente en beneficio de la fiabilidad.

- Gran cantidad de interfaces de entrada/salida analógicas y digitales, embebidas de forma original en el sistema, sin necesidad continúa de añadir placas de expansión electrónicas. La capacidad de comunicación debe hacerse a través del puerto USB® nativo.
- Arquitectura computacional Harvard, con el fin de impedir una posible corrupción del programa de control, con interfaz de programación basada en un estándar reconocido, como JTAG. Con posibilidad de depuración línea a línea. A ser posible el depurador/programador debe estar incluido en cada unidad del RCP.
- El RCP debe poseer una capacidad computacional tal, que se encuentre bastante por encima de la demanda actual del mercado (años 2013/2014) para aplicaciones de control de múltiples actuadores tanto convencionales como no convencionales; asegurándose de esta forma la vigencia temporal del RCP y la posibilidad de probar experimentalmente multitud de posibilidades y teorías de control.
- El sistema RCP debe contar con unidad de procesamiento de coma flotante, para poder trabajar directamente con números decimales de al menos 32 bits (coma flotante precisión simple).
- La programación del RCP debe integrarse completamente y sin fallos/incompatibilidades en el entorno Matlab/Simulink®, de forma que los programas de control desarrollados en Simulink® se integren directa y automáticamente de forma que los pasos intermedios entre Matlab/Simulink® y el compilador para el RCP no requieran intervención humana.
- El requisito anterior exige la creación de una toolbox para Matlab/Simulink® que cubra la totalidad de periféricos y todas sus posibilidades de configuración, la capacidad de funcionar en Tiempo-Real Multi-Tarea, la posibilidad de sincronizar el planificador de Tiempo-Real para varias unidades del RCP, dotar de interfaces de intercambio de datos de alta velocidad mediante interfaz Universal Serial Bus® (USB). Posibilidades muchas de ellas que no se ofrecen en ningún RCP disponible con anterioridad. Deben cubrirse bastantes versiones de MATLAB, tanto de 32 como de 64 bits.
- El sistema RCP se validará mediante la proposición de algoritmos de control posición-velocidad-fuerza para actuadores no convencionales basados en aleaciones con memoria de forma (SMA), también se realizarán otras intervenciones dentro del proyecto HYPER para distintos miembros del consorcio, tales como CIDETEC, Hospital Nacional de Parapléjicos de Toledo utilizando el ARCP, de forma que complementen su validación.

- Se explorarán las capacidades multimedia del controlador digital elegido si las hubiera, y se integrarán en MATLAB®/SIMULINK®.
- El sistema ARCP debe poder ser utilizado por un público multidisciplinar, no se podrá exigir un conocimiento técnico especializado para poder utilizarlo; ha de ser fácil de configurar y autoexplicativo en las ocasiones que así se requiera.

La figura 1.7 muestra el ciclo de desarrollo, denominado ciclo en V [7], para un controlador siguiendo el concepto tradicional de sistema RCP; se obtienen beneficios sólo en la etapa de prototipado en laboratorio (recuadrada en rojo). Debajo se muestra el nuevo concepto de sistema ARCP, en el que todas las etapas están cubiertas por el RCP, también se utiliza para facilitar la etapa de modelado y simulación.

Los citados beneficios se obtienen al entender el concepto de diseño basado en modelos como una 'máquina' con la que se puede diseñar y programar con todo nivel de profundidad y detalle cualquier controlador basado en procesador. La figura 1.6 ilustra este concepto.

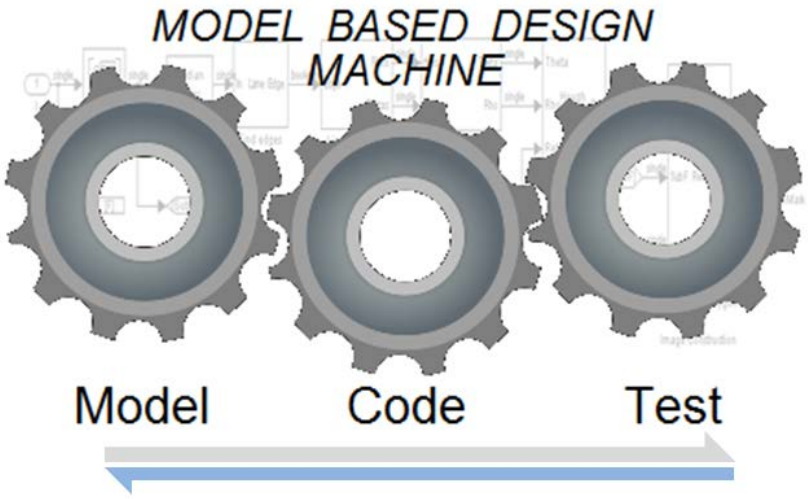


Figura 1.6: *Concepto de diseño basado en modelos en forma de 'máquina' con la que todo tipo de programación, pruebas y desarrollos son posibles.*

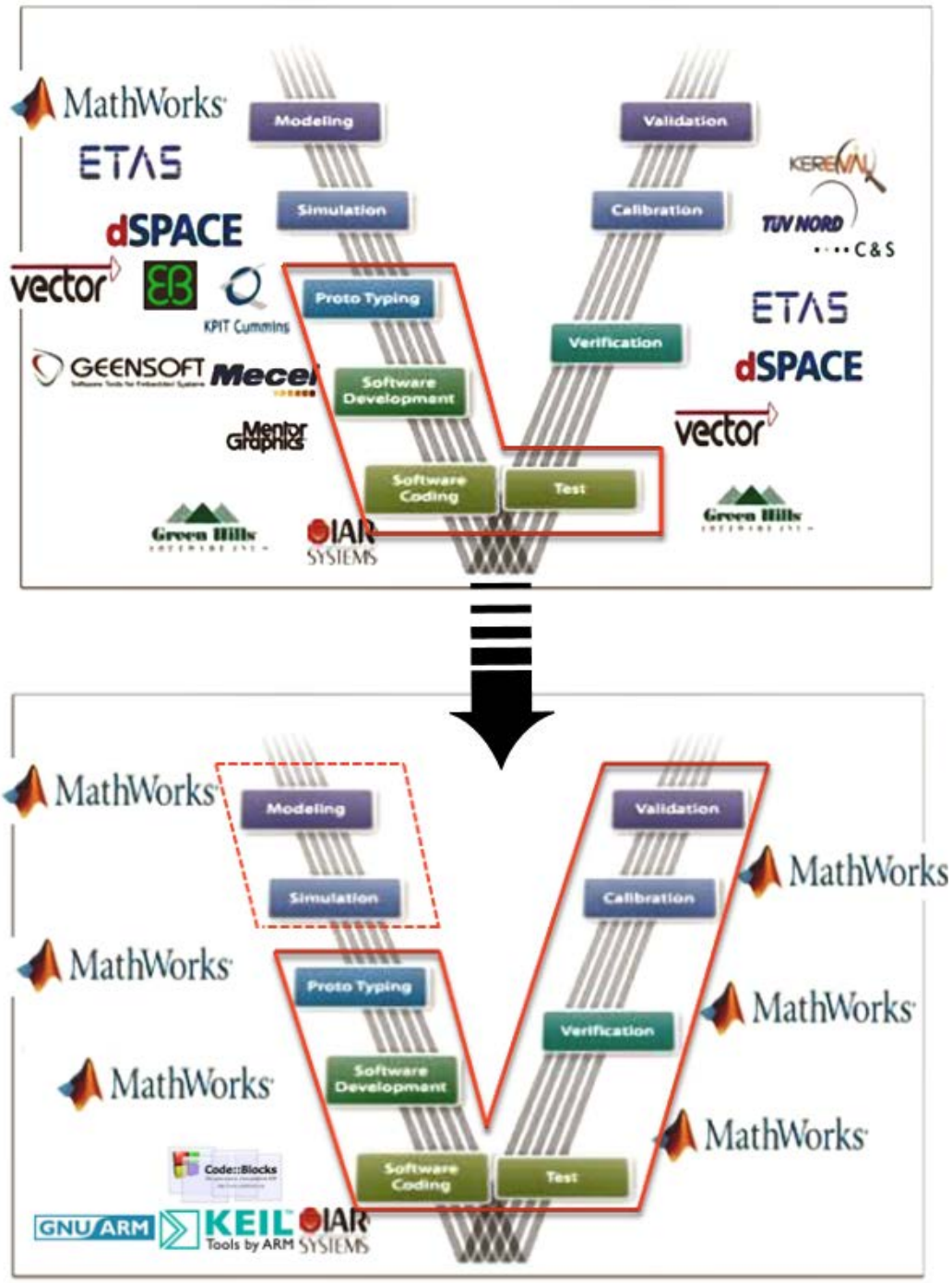


Figura 1.7: *Objetivos de la Tesis reflejados de forma visual.*

1.6. Novedades Esperadas

La creación y utilización de un sistema RCP avanzado (ARCP) representa un paso básico para satisfacer los objetivos de control de actuadores no convencionales comprometidos por la Universidad Carlos III de Madrid (UC3M), en particular el RoboticsLab y su participación enmarcada dentro del proyecto HYPER cuyo objetivo es ayudar en las tareas de rehabilitación y/o recuperación de las funciones motrices en los miembros del cuerpo humano, apoyándose en el uso de la robótica [8]. De este sistema ARCP se esperan determinadas innovaciones en lo que a Prototipado Rápido para Control se refiere.

El sistema Avanzado de Prototipado Rápido para Control integra funcionalidades avanzadas, de forma que el código fuente generado desde el entorno de desarrollo gráfico sea de tal calidad y optimización que no sea necesario recurrir a la programación textual manual en ningún caso.

El ARCP provee del soporte necesario para ejecutar controladores en tiempo-discreto y en tiempo-continuo.

También permite la gestión de múltiples tareas en Tiempo-Real sin la necesidad de utilizar un sistema operativo en Tiempo-Real.

Se pueden gestionar desde el lenguaje gráfico cada uno de los aspectos propios de una arquitectura hardware orientada a control, como la reprogramación de las fuentes de interrupción y la utilización óptima de las capacidades inherentes de cada arquitectura hardware soportada por el ARCP, con especial énfasis en las funciones relacionadas con tratamiento digital de señales (DSP, del inglés "Digital Signal Processing").

La gestión de las capacidades de vídeo y audio por parte del sistema ARCP se realiza de forma poco común, de forma que no perjudique la ejecución de las tareas en Tiempo-Real y aproveche con el máximo beneficio las capacidades de cada arquitectura hardware soportada.

Las comunicaciones basadas en USB® también se encuentran optimizadas de forma que no supongan una sobrecarga para el procesador.

El soporte para MATLAB®/SIMULINK® está aprovechado al máximo e incluso extendido más allá del soporte original, debido a funcionalidades que así lo han requerido.

El éxito en los resultados de esta Tesis Doctoral no sólo ha permitido cumplir los requisitos y satisfacer las necesidades en cuanto al prototipado y experimentación con diversas teorías para control de actuadores [9],tesis doctoral de Dorin-Sabin Copaci [en progreso], tesis doctoral de Álvaro Villoslada Peciña [en progreso], estas tesis enmarcadas dentro del ámbito del proyecto HYPER; sino que dadas sus propiedades y facilidad de uso, el sistema ARCP está siendo rápidamente adoptado a lo largo y ancho del Departamento de Ingeniería de Sistemas y Automática, para la realización de tesis de mástes como las llevadas a cabo por Higor Alves y Cristobal José Gómez [en progreso], así como por el departamento análogo a éste en la Universidad Complutense de Madrid, forjando nuevas colaboraciones con otras universidades, tanto para su uso en proyectos de investigación como en prácticas experimentales para distintos niveles en la formación reglada de ingeniería. Utilizándose también de esta forma como una herramienta educativa.

Capítulo 2

Estado del Arte

A diferencia de muchos otros ámbitos de investigación amparados en el Departamento de Ingeniería de Sistemas y Automática; tales como la robótica humanoide, visión artificial, navegación autónoma de robots terrestres y aéreos... el área de estudio correspondiente al análisis y planteamiento de algoritmos de control, y del hardware que hace de anfitrión para estos algoritmos y su implementación, es un área que cuenta con soporte comercial y figuras en forma de entidades empresariales reconocidas respecto a los sistemas RCP.

Analizando en profundidad, y siempre que se cuente con las destrezas necesarias tanto del bajo como del alto nivel correspondientes a toda implementación propia de la ingeniería de sistemas, resultará sencillo detectar que las soluciones comerciales de RCP carecen en gran medida de muchos aspectos y posibilidades de programación que otorguen toda la libertad necesaria para gobernar al gusto y detalle del ingeniero el sistema controlador digital que está proponiendo. De forma que no vea limitado su correcto funcionamiento en Tiempo-Real para todas las situaciones que puedan llegar a plantearse.

Una vez explicada la actual metodología de trabajo con sistemas RCP y dónde se ubica esta Tesis Doctoral respecto a dichos sistemas, es el momento de analizar las distintas soluciones que se ofrecen, tanto comerciales como surgidas en centros de investigación.

Además de esto, es necesario estudiar bajo qué paradigmas de programación trabajan los sistemas RCP. Dado que han sido diseñados para tareas de control, en

las que el conocimiento del transcurso del tiempo y la capacidad de reaccionar inmediatamente a eventos es vital, es de suponer que no utilicen paradigmas de programación convencionales, sino aquellos orientados a programación en Tiempo-Real. Los sistemas RCP utilizan para su programación lenguajes síncronos, y para dotarlos de mayor nivel de abstracción utilizan lenguajes síncrono-gráficos.

2.1. Lenguajes Síncronos

Un lenguaje de programación síncrono es un lenguaje de programación optimizado para sistemas reactivos. Los sistemas informáticos se pueden clasificar en tres tipos [10], en esta tesis de propone un segundo nombre para cada uno de estos tipos:

- Sistemas de Transformación o de ejecución única: Programas que reciben entradas, las procesan y realizan operaciones matemáticas con ellas. Al concluir las operaciones ofrecen las salidas, que son el resultado de la aplicación de las operaciones matemáticas sobre los datos de entrada y no vuelven a ejecutarse más veces con regularidad a lo largo del tiempo.

- Sistemas Interactivos o de ejecución continua: Programas que funcionan de forma constante, su estado y salidas evolucionan cuando sus entradas cambian. No abandonan la ejecución, sólo terminan cuando se les fuerza a ello.

- Sistemas Reactivos o de ejecución en Tiempo-Real: Son programas que comienzan su ejecución a intervalos regulares de tiempo. Cuando llega el momento temporal de su ejecución, reciben sus entradas, operan con ellas y ofrecen las salidas. Esperando al siguiente evento temporal regular para volver a ejecutarse. La ejecución también puede deberse a eventos indiferentes del tiempo, como señales de interrupción.

Generalmente se han considerado sistemas informáticos en Tiempo-Real a aquellos que pertenecen a la tercera categoría, ya que impone que el programa de control actúe cada determinado tiempo y no pueda rebasar los plazos temporales entre iteraciones. Observando con atención, la tercera categoría se basa en imponer sobre los sistemas informáticos de la primera categoría la condición de volver a ejecutarse cada determinado intervalo de tiempo.

La segunda categoría también resulta relevante desde el punto de vista de un sistema de control, siempre que puedan darse determinadas condiciones. Un ejemplo de programa de control perteneciente a la segunda categoría son los controladores basados en tiempo-continuo, estos controladores están leyendo constantemente las entradas, operan con ellas, dan su salida e inmediatamente vuelven al principio, sin esperar a ningún evento temporal para reiniciar su ejecución. Estos procesos en tiempo-continuo son especialmente útiles si el sistema informático sobre el que funcionan puede interrumpir su ejecución, dejarla inconclusa, y ejecutar un proceso reactivo en su lugar, una vez terminada la ejecución del proceso reactivo el sistema informático retomará la ejecución del proceso en tiempo-continuo

donde lo abandonó. En un sistema RCP es realmente útil poder contar con un proceso en tiempo-continuo cuando se deben analizar o adquirir grandes cantidades de datos y debido a ello resulta difícil determinar el tiempo necesario entre iteraciones, o también cuando se dispone de un sistema de control muy distribuido y no se puede predecir en qué momentos se van a recibir datos, en estos casos conviene que un proceso en tiempo-continuo realice estas labores, ya que un proceso en tiempo-real podría rebasar su plazo de ejecución o podría perder datos vitales enviados desde otros controladores.

El sistema RCP propuesto en esta tesis doctoral tiene en cuenta todo estos aspectos, y permite utilizar tanto procesos en tiempo-continuo como en tiempo-real de la manera que se acaba de describir.

Volviendo al asunto de los lenguajes de programación síncronos, estos se idearon para poder programar sistemas informáticos reactivos, para programar procesos de control en tiempo-real.

Los lenguajes de programación síncronos abstraen al programador del concepto de tiempo o de cómo se obtiene en determinada arquitectura computacional la noción del transcurso del tiempo. En su lugar, se emplea el concepto de ”Tick”, por lo tanto un programa de control o una parte de él se ejecuta cada determinados ticks, y el valor de los ticks se incrementa cada determinado plazo de tiempo. También son lenguajes que consideran que una vez acontece el evento temporal, el programa se ejecuta instantáneamente y no es necesario tener en cuenta un orden en la ejecución de las partes del programa, éste se estructura en recibir las entradas, procesarlas y dar las salidas; por lo que nunca pueden darse las salidas antes de haber recibido las entradas, y las salidas siempre tendrán su origen en las operaciones con las entradas.

Con estas consideraciones, estos lenguajes poseen una semántica determinista (no da lugar a ambigüedades), el comportamiento de los programas es determinista (se puede conocer cuanto tardan en ejecutarse) y por tanto pueden analizarse de forma formal, verificarse en base a criterios establecidos y posteriormente dar lugar a código ejecutable por un procesador que cumpla lo dictado por la semántica determinista.

Estos lenguajes síncronos tienen su origen en Francia, en la década de 1980. Se idearon para dar lugar a programas de control en ámbitos nucleares, militares y aeronaúticos.

2.1.1. Esterel

Esterel es un lenguaje de programación orientado a los paradigmas de paralelismo y concurrencia. La metodología de programación se basa en la creación de instancias. Las instancias son rutinas de programa que se ejecutan en paralelo a todas las demás rutinas, por lo que una instancia puede detenerse en un bucle de espera sin afectar a la ejecución de las demás instancias. Lo más parecido a una ejecución bajo el paradigma de Tiempo-Real es hacer que una instancia espere un determinado tiempo antes de volver a ejecutarse, pudiéndose lograr un comportamiento similar al ofrecido por el Tiempo-Real.

Un ejemplo de programa con dos instancias en Esterel es el siguiente:

```
module CD:
input D;
output C;
 present D else
   emit C
end module

module ABRO:
input A, B, R;
output O;
loop
  [ await A || await B ];
  emit O
each R
end module
```

En el programa anterior, escrito en lenguaje Esterel, la primera instancia llamada ”CD” comprueba si D es mayor que 0 y en caso afirmativo su salida es la señal C. La segunda instancia denominada ”ABRO” comprueba si R es mayor que 0, en caso afirmativo se queda esperando a que A y B sean ambos mayores de 0, cuando esa condición ocurra la salida de la instancia será la señal O.

Esterel genera código en lenguaje C o en VHDL/Verilog si el destino final es un circuito electrónico en lugar de un procesador. En el caso de un procesador, el paralelismo no es infinito o puede no existir, así que podría implementarse en lenguaje C de la siguiente manera:

```
static unsigned char CD_end;
static unsigned char ABRO_end;
```

```
static unsigned int exe;

void main()
{
CD_end = 0x00;
ABRO_end = 0x00;
exe = 0x00000000;
 while(1)
  {
  Paralelizador();
  }
}

void Paralelizador()
{
switch(exe)
{
  case 0:
     if(CD_end == 0x00)
       {
       CD();
       }
       exe++;
  case 1:
     if(ABRO_end == 0x00)
       {
       ABRO();
       }
}
 //Enables re-execution of all Esterel instances
 exe = 0x00;
}

uint32 CD(uint32(D),uint32(C))
{
if(CD_end == 0x00)
{
 CD_end = 0x01;
  //Executes Esterel code
  if(D>0)
```

```
   {
   CD_end = 0x00;
   return(C);
   }
 CD_end = 0x00;
 return NULL;
 //End of Esterel code
}
}

uint32 ABRO(uint32(A),uint32(B),uint32(R),uint32(O))
{
if(ABRO_end = 0x00)
{
 ABRO_end = 0x01;
  //Executes Esterel code
  if(R>0)
   {
    while( (A<0) & (B<0) )
     {
     Paralelizador();
     }
     ABRO_end = 0x00;
     return(O);
   }
  ABRO_end = 0x00;
  return NULL;
  //End of Esterel code
}
}
```

El código anterior presenta una posible implementación en lenguage C del código Esterel explicado anteriormente, el paralelismo en la ejecución se consigue con llamadas recursivas a la función ”Paralelizador()” en las que se inhibe la ejecución de las funciones que han quedado bloqueadas esperando a algunas señales. También se va guardando el orden de la ejecución, para que si se llama a la función paralelizador de nuevo, no se vuelvan a ejecutar las funciones que ya se han ejecutado (para esto sirve la variable ”exe”). Una vez que se ha terminado de iterar, ”exe” vuelve a valer 0 permitiendo que se ejecuten de nuevo todas las funciones.

En actualidad, Esterel es el lenguaje que está detrás del sistema de programación gráfica basada en modelos SCADE Systems®. La figura 2.1 ilustra cómo se programa de forma gráfica con el entorno de SCADE Systems®, la apariencia y metodología es muy similar a SIMULINK®, aunque la generación de código se realiza en dos pasos, primero se transcribe a código Esterel y a continuación se traduce a lenguaje C o VHDL/Verilog.

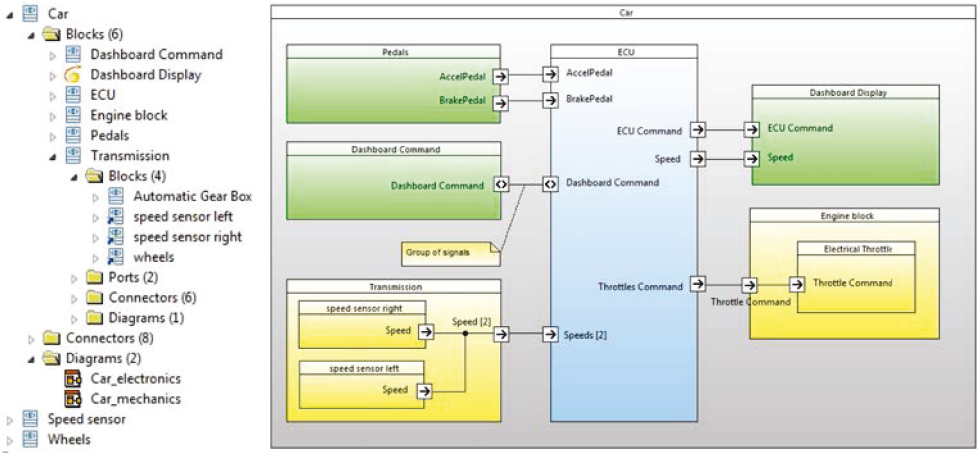


Figura 2.1: *Programación basada en modelos en el entorno SCADE Systems®.*

El sistema SCADE Systems® soporta algunos objetivos hardware tanto basados en procesador como en FPGA, pero no permite añadir soporte de hardware personalizado. Se utiliza en entornos nucleares y militares debido a las pruebas de certificación que ha superado con éxito:

- DO-178B qualified up to Level A and DO-178C Ready for Civilian and Military Aeronautics
- IEC 61508 certified at SIL 3 by TÜV for Industry
- EN 50128 certified at SIL 3/4 by TÜV for Rail Transportation
- IEC 60880 compliant for Nuclear Energy
- ISO 26262 for Automotive

2.1.2. Lustre

Estrechamente relacionado con el lenguaje Esterel, aunque se considera formalmente un lenguaje distinto de Esterel en realidad es una forma de ampliación del primero, en el que por primera vez aparece el concepto de relojes, señales de reloj y sincronía en base a reloj. Extiende la sintaxis de Esterel con nuevas funcionalidades manteniendo los principios formales. También se encuentra integrado en los entornos de desarrollo SCADE Systems®. Estos sistemas permiten utilizar, por debajo del sistema de programación gráfico, Esterel o Lustre, por defecto utilizan Lustre al ser este una extensión mejorada de Esterel [11]. Los compiladores de código Lustre generan código en lenguaje C.

2.1.3. Signal

Signal es un lenguaje de programación síncrono textual que soporta diferentes tiempos de muestreo para cada parte del programa que se implemente[2], como novedad respecto al lenguaje Esterel. Desarrollado durante la década de 1980 [12] en paralelo a Esterel y Lustre, comenzó a utilizarse de forma gráfica, como un lenguaje de programación basada en modelos bastante pronto, pareciéndose en cuanto a concepto y metodología de trabajo a MATLAB/SIMULINK®. Los compiladores de código signal generan código en lenguaje C.

2.1.4. LabVIEW®

Es un lenguaje de programación síncrono basado en gráficos, de carácter comercial. Su origen tiene lugar en 1976, pero es en 1986 cuando comienza a comercializarse. En un principio fue concebido para conectarse remotamente a instrumentos de medida y control de procesos, más adelante en el tiempo le fue otorgada la capacidad de poder programar dichos instrumentos. Sus áreas de aplicación están centradas en la automatización de la adquisición de medidas y tratamiento de las mismas, automatizar sistemas de verificación y calibración, y diseñar sistemas de control embebidos. Originalmente no está contemplada la posibilidad de incluir bloques gráficos que contengan programación textual, pero complementos software adicionales permiten ampliar sus opciones. Estas ampliaciones basadas en lenguaje textual vienen a suplir deficiencias en cuanto a cálculo matemático de carácter científico que no habían sido tenidas en cuenta originalmente.

[2]http://www.irisa.fr/espresso/Polychrony/introToSignal.php , Julio 2014

2.1.5. MATLAB®/SIMULINK®

MATLAB® es un lenguaje textual de mayor nivel de abstracción que el lenguaje C, en el que trabajar y operar matemáticamente con vectores y matrices resulta sencillo en comparación con los lenguajes textuales tradicionales. También mostrar los resultados de las operaciones en pantalla se encuentra muy simplificado, esto unido al alto nivel de abstracción en lo que a cálculo matemático se refiere, cuenta con un gran repertorio de funciones que realizan operaciones muy extensas y complejas, ha ayudado a su asentamiento como estándar en ámbitos de trabajo científico.

Aún con el potencial descrito este lenguaje textual no es un lenguaje síncrono, esto lo suple una extensión de software denominada SIMULINK®, que proporciona un lenguaje síncrono basado en gráficos apoyado internamente por la funcionalidad del lenguaje textual MATLAB®. SIMULINK® está indicado para el diseño y programación de sistemas de control en Tiempo-Real y Tiempo-Continuo. También está preparado para diseñar modelos matemáticos, mediante el lenguaje gráfico y el textual, de sistemas reales. Cubriendo gran cantidad de dominios tanto eléctricos, mecánicos e hidráulicos/neumáticos. Sobre estos modelos se pueden realizar simulaciones, para comprobar el comportamiento de un sistema sin necesidad de construirlo en la realidad. Estas simulaciones también pueden ser programadas en un sistema de control hardware de forma que se disponga de un sistema virtual perteneciente a cualquiera de estos dominios físicos.

MATLAB fue ideado por Cleve Moler en 1984, aunque el lenguaje en que se basa, denominado M es muy anterior datando de 1970. El lenguaje M se ideó para proporcionar un gran nivel de abstracción en el uso de las bibliotecas de funciones matemáticas LINPACK y EISPACK [13, 14]. SIMULINK® aparece en 1990, bajo el nombre de SIMULAB (Simulation Laboratory), cambiando a su nombre actual en 1992 en la cuarta versión de MATLAB®.

Una particularidad, que ha permitido una mayor aceptación de este software en numerosos ámbitos científicos, es su retrocompatibilidad. Programas escritos en lenguaje M en versiones muy antiguas de MATLAB® pueden funcionar en versiones actuales, al igual que sucede con los modelos programados en versiones antiguas de SIMULINK®, aunque ya no se pueden importar modelos anteriores a la versión 14 (año 2004) de MATLAB®/SIMULINK® debido a profundos cambios en la estructura interna de los modelos de SIMULINK® en dicha versión.

2.1.6. ADA

ADA es un lenguaje de programación textual desarrollado al amparo del Departamento de Defensa (DOF) de los Estados Unidos a mediados de la década de 1970. La intención fue crear un lenguaje de programación orientado a la programación de sistemas embebidos en Tiempo-Real, con un fuerte formalismo y adecuación a normativas de seguridad que permitiera desplazar al resto de lenguajes de programación del ámbito de la ingeniería de control, de forma que se trabajara siempre con este lenguaje que garantiza mayor integridad. El lenguaje está diseñado para evitar errores de programación triviales, éstos no tienen cabida en la sintaxis de ADA. Dicha sintaxis está orientada para facilitar su lectura y entendimiento por parte de un ser humano, en lugar de simplificar el proceso de programación como sucede con otros lenguajes textuales, esto se decidió hacerlo así para que múltiples programadores revisaran el mismo código con la mayor facilidad de lectura posible. El lenguaje no tiene un carácter propiamente síncrono, pues no funciona bajo el paradigma de ejecución en Tiempo-Real, pero manejando los tiempos de espera en las tareas puede lograrse un comportamiento similar.

Para aquellos usuarios con experiencia previa en el uso de lenguajes textuales tales como C, Pascal... la sintaxis de ADA se revela como sencilla de entender a simple vista, cambian algunos aspectos como la asignación de valores. Respecto a la multitarea y al soporte de Tiempo-Real, estas características son inherentes al lenguaje; esta preparado para ejecutar múltiples tareas en aparente paralelismo, con la posibilidad de crear tareas que no tienen porque ejecutarse indefinidamente, pueden terminar su ejecución y volver a ser llamadas después. Las tareas también pueden detenerse y controlarse su ejecución a voluntad, así como destruirlas. Las tareas se declaran con el identificador ”task” y comienzan su ejecución sin necesidad de ser invocadas, al comienzo del programa principal. Las sentencias de control tales como ”switch”, ”while”, ”loop”, ”if-else” terminan siempre con ”end”. A continuación se muestra un ejemplo de programa en lenguaje ADA con dos tareas, en realidad se ejecutarán tres tareas, el programa principal y las tareas ”Control temperatura” y ”Control presión”. Merece especial atención la sentencia ”delay” que permite que la tarea espere el tiempo indicado hasta volver a ejecutarse, siendo esta la manera de gestionar el tiempo en ADA. En la declaración de cada tarea (”task body”), previamente al código de la lógica (”begin”), se declaran las variables que utilizará la tarea.

```
procedure Control_temperatura_y_presión

  task Control_temperatura;
  task Control_presión;
```

```
  task body Control_temperatura is
    Temperatura: TTemperatura;
    Temperatura_referencia: TTemperatura := 55;
    AcciónCalefactor: TAcciónCalefactor;
  begin
    loop
      Temperatura := Leer;
      AccciónCalefactor := Control (Temperatura, Temp_referencia);
      Escribir (AcciónCalefactor);
      delay (2.0);
    end loop;
  end Control_temperatura;

  task body Control_presión is
    Presión: TPresión;
    Presión_referencia: TPresión := 315;
    PosiciónValvula: TPosiciónVálvula;
  begin
    loop
      Presión := Leer;
      PosiciónVálvula := Control (Presión, Presión_referencia);
      Escribir (PosiciónVálvula);
      delay (3.0);
    end loop;
  end Control_presión;

begin
  -- Comienzan a ejecutarse Control_temperatura y Control_presión;
  null; -- Un cuerpo de procedimiento no puede estar vacío.
end Control_temperatura_y_presión;
```

El concepto del lenguaje ADA es muy similar a los actuales lenguajes multitarea ampliamente utilizados para videojuegos, debido a la gran facilidad que aportan en el manejo y gestión de programas multitarea. Uno de estos lenguajes de carácter gratuito es el lenguaje Fénix/BennuGD. El anterior programa escrito en ADA puede transcribirse casi directamente a lenguaje Fénix de la siguiente manera, con la salvedad de que las tareas se denominan ”process” y el programa principal ”program”, por lo demás la sintaxis es muy similar a la sintaxis de ADA:

```
GLOBAL //Declaración de variables globales
INT Temperatura_In;
INT Presión_In;
INT AcciónCalefactor;
INT Posición_Valvula;
END

program Control_temperatura_y_presión //Programa principal
begin
   set_fps(0);//Ejecución en modo continuo
   set_video(320,240,8);//Vídeo a 320 x 240 x 8 bits
   timer[0].mode=MSECS;//Timer 0 en milisegundos
   timer[0]=0;//Arrancar el timer
   Control_temperatura();//Se invocan las tareas
   Control_presión();
   loop
      frame;
      if(key(_esc))//'escape' termina el programa
         let_me_alone();//Destruye las tareas en ejecución
         break;//Permite salir del bucle loop
      end
   end
   exit();//Termina el programa
end

process Control_temperatura
INT Temperatura;//INT es un entero de 32 bits
INT Temperatura_Referencia = 55;
begin
   loop
      Temperatura = Temperatura_In;
      AcciónCalefactor = Control(Temperatura,Temperatura_Referencia);
      Escribir(AcciónCalefactor);
   end
   wait(2000);//Esperar dos mil milisegundos,
end //timer configurado para  milisegundos

process Control_presión
INT Presión;//INT es un entero de 32 bits
INT Presión_Referencia = 315;
```

```
begin
   loop
      Presión = Presión_In;
      Posición_Valvula = Control(Presión,Presión_Referencia);
      Escribir(Posición_Valvula);
   end
   wait(3000);//Esperar tres mil milisegundos,
end//timer configurado para milisegundos

Function int wait(int t)//Función esperar tiempo
Begin
    t += timer[0];
    While(timer[0]<t) frame; End
    return t-timer[0];
End
```

El programa en lenguaje Fénix requiere en cada tarea de la llamada a la sentencia ”frame” que devuelve la ejecución al nivel jerárquico inmediatamente superior, permitiendo que se ejecuten el resto de tareas, cuando no queden tareas por ejecutar se ejecutará el programa principal, y al ejecutarse en éste la sentencia ”frame” se actualizará la pantalla, si hay algo que dibujar, y se volverá a iterar la ejecución de todas las tareas. ADA posee un mecanismo de ejecución similar para llevar a cabo la ejecución multitarea, aunque no se encuentre de manera explícita en el lenguaje.

2.1.7. Lenguaje Unificado de Modelado. UML

El lenguaje unificado de modelado (UML, del inglés "Unified Modeling Language") es un lenguaje gráfico que permite visualizar y construir un sistema de naturaleza diversa, desde el esquema de un modelo de negocio hasta la descripción de forma visual de un sistema de control.

El lenguaje UML en principio sólo provee de la capacidad para describir el sistema y mostrarlo de forma gráfica, pero sin posibilidad de poder llegar a ser ejecutable. Posteriormente la compañía Rational Software® integró la capacidad de generar código fuente basado en texto a partir de un modelo programado en UML y la compañía de origen israelí I-Logix® logró que se pudieran programar dispositivos controladores hardware diversos y personalizables mediante el lenguaje de alto nivel UML; de forma que desde el propio lenguaje UML se puede disponer de acceso a los recursos hardware de I/O, configurarlos y utilizarlos. Además esta compañía proveyó al lenguaje UML de la capacidad de poder ser ejecutable, con lo que se alcanzó la capacidad de poder realizar simulaciones de diseños basados en modelos gráficos con el lenguaje UML. Ambas compañías fueron adquiridas por IBM® y los resultados de ambas adquisiciones dieron lugar al software de desarrollo basado en modelo IBM Rational Rhapsody®. La figura 2.2 muestra un modelo de control programado con lenguaje gráfico UML en el citado software de IBM®. La gran similitud entre este software y SIMULINK® hace que se pueda diseñar el modelo con ambos sofware sin grandes problemas. Existen más entornos MBD que hacen uso del lenguaje UML, pero el de IBM® es el más representativo.

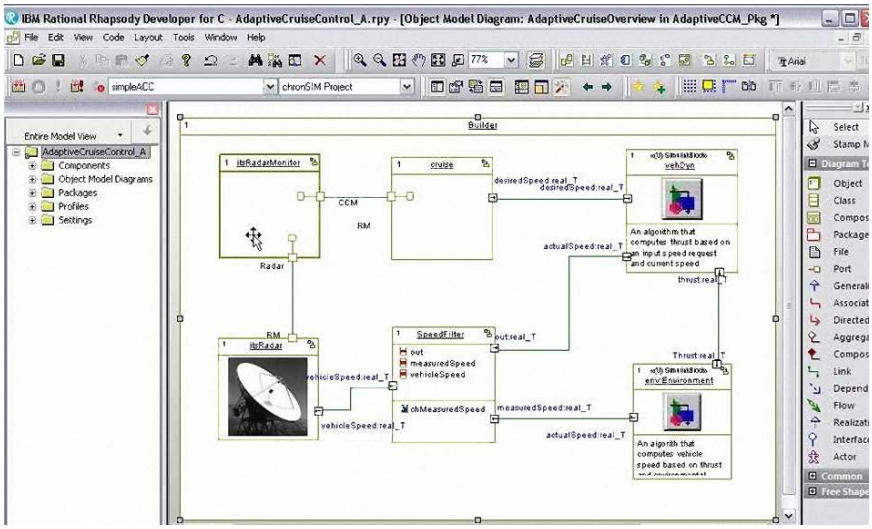


Figura 2.2: *Programación basada en modelos utilizando lenguaje UML*

2.1.8. Lenguaje de modelado de sistemas. SysML

El lenguaje de modelado de sistemas (SysML, del inglés "Systems Modeling Language") es una extensión del lenguaje UML que además de permitir la representación de un software mediante gráficos, funcionalidad original del lenguaje UML, permite además integrar elementos físicos interconectados con los elementos puramente de lógica computacional, dando lugar a un lenguaje gráfico que permite simular sistemas o conectarse o generar código fuente específico para una arquitectura computacional además de código fuente genérico, siendo éste último el necesario para implementar la lógica. SysML también incorpora la descripción visual de máquinas de estado. Hoy día es común considerar que el lenguaje UML integre también las características propias de la extensión del lenguaje proporcionada por SysML.

2.2. Herramientas CASE

La herramientas CASE (del inglés " Computer Aided Software Engineering") representan un conjunto de programas cuya finalidad es aumentar la productividad, eficiencia y fiabilidad en el desarrollo de software; esto se traduce en una reducción considerable del tiempo y recursos materiales y humanos necesarios. Para lograr estos beneficios, una herramienta CASE hace uso de lenguajes basados en gráficos que proveen de un alto nivel de abstracción al programador además de automatizar los procesos de escritura del código en formato textual y comprobación de errores y adecuación del mismo a normativas de seguridad.

Una herramienta CASE puede cubrir uno o varios ámbitos de trabajo, existen herramientas orientadas únicamente a la creación de interfaces de usuario para sistemas Windows®/Linux, otras se encuentran más especializadas en la realización de simulaciones o programación de dispositivos hardware embebidos...

Ejemplos de herramientas CASE son MATLAB/SIMULINK®, LabVIEW® y la familia de programas IBM Rational®, entre otros. El primer caso es ampliamente utilizado en ingeniería de control debido a la gran cantidad de herramientas software adicionales (denominadas toolboxes) que facilitan el trabajo de los ingenieros, la segunda herramienta posee un amplio soporte en lo referente a fabricantes de dispositivos hardware industriales y el tercer caso cubre aspectos muy amplios, desde el desarrollo de sistemas de control embebidos hasta interfaces de usuario en PC, pero no dispone de toolboxes adicionales que apoyen el trabajo propio de la ingeniería de control.

2.3. RCP Comerciales

En las siguientes secciones se hará un análisis de la diferentes opciones que ofrece el mercado para trabajar con sistemas RCP, son las soluciones más utilizadas en el ámbito industrial, automovilístico y aeronáutico. Para cada uno de ellos se presentará la arquitectura hardware en la que se basan, el entorno de desarrollo para el que están orientados tales como Matlab®, LabView®, RT-Lab®... así cómo las posibilidades de programación en lenguaje gráfico que ofrecen, sus puntos fuertes y sus puntos débiles. También se señalará si se trata de sistemas RCP abiertos, o lo suficientemente abiertos como para poder expandir sus posibilidades e incrementar su repertorio de funciones del lenguaje gráfico.

2.3.1. xPC Target®

xPC Target® [15] es la nomenclatura que recibe un complemento software producido por la compañía MathWorks®, para su herramienta MATLAB® / SIMULINK®. De forma que permite la ejecución bajo el paradigma de Tiempo-Real de programas desarrollados mediante el lenguaje gráfico de SIMULINK® en los objetivos hardware soportados por xPC Target®.

La arquitectura computacional para la que está orientada la toolbox xPC Target® son aquellas basadas en procesadores Intel x86 e Intel x64, y compatibles. Cualquier ordenador PC es factible de ser utilizado con la toolbox xPC Target®, aunque para poder ser utilizado para algoritmos de control se hace necesario disponer de interfaces de entrada/salida analógicas y digitales, estas interfaces son añadidas mediante placas de expansión al ordenador PC donde se va a ejecutar el modelo de control.

La toolbox xPC Target® provee de controladores para manejar las diferentes placas de entrada/salida desde el lenguaje gráfico de Simulink®. Esta placas de expansión están fabricadas por terceras compañías. El soporte para SIMULINK® de estas placas de expansión de entrada/salida, pueden verse en la figura 2.3, se facilita por medio de acuerdos comerciales entre los fabricantes y MathWorks® [3], acuerdos que cubrirán un determinado número de ediciones de MATLAB®/SIMULINK®. No obstante, algunos autores [16], [17] han presentado desarrollos para otorgar soporte bajo xPC Target® para hardware personalizado.

La metodología de trabajo con este sistema RCP consiste en utilizar Simulink®

[3] The MathWorks® extensión y cobertura a hardware y fabricantes, Julio 2014

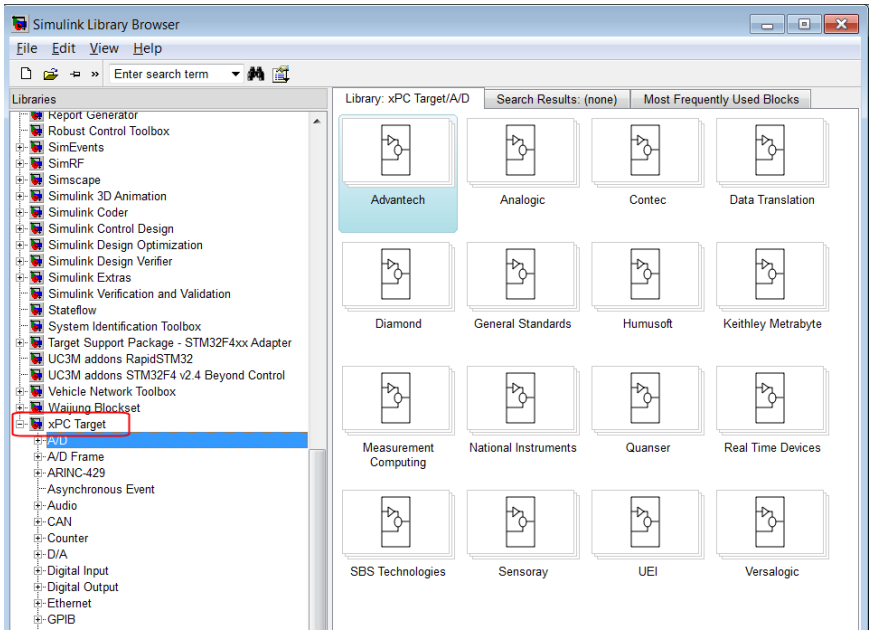


Figura 2.3: *Bloques de programación gráfica de la toolbox xPC Target®,una misma funcionalidad como conversores analógico-digitales presentan bloques distintos para cada fabricante*

para programar de forma visual el modelo de control, así como para modificar parámetros del controlador y depurar el rendimiento del sistema de control. Simulink® compilará el modelo y lo descargará hacia la estación de trabajo donde está instalado el entorno software en Tiempo-Real de xPC Target®, es en la máquina que cuenta con el entorno xPC Target® donde se ejecutará el modelo de control. La comunicación entre ambos ordenadores es a través de conexión ethernet. Este modelo de funcionamiento se muestra en la figura 2.4 .

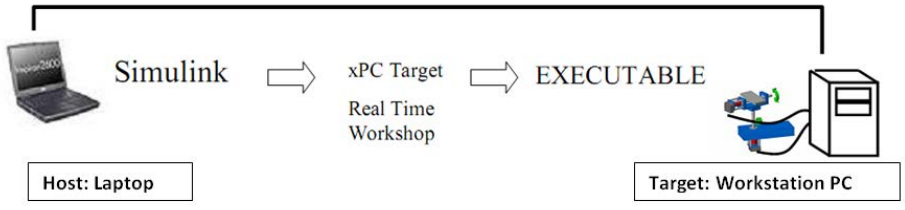


Figura 2.4: *Metodología de trabajo con xPC Target®, se desarrolla el modelo de control en Simulink® y se le hace funcionar en Tiempo-Real en la estación de trabajo*

Generalmente, el ordenador donde se ejecuta el modelo de control en Tiempo-Real (ordenador Target), necesita de la supervisión de un ordenador anfitrión (ordenador Host), en el que se desarrolla el modelo de control en Simulink® y se capturan y envían datos desde y hacia el ordenador Target. Para obtener un funcionamiento completamente autónomo, en el que el ordenador Target no requiera del ordenador Host, se hace necesario adquirir el paquete de software xPC Target Embedded Option®.

Para logar esta funcionalidad autónoma, en la que la estación de trabajo no necesita de la supervisión de ningún otro computador para funcionar, se tiene que configurar desde el ordenador supervisor que dicha estación de trabajo con xPC Target® pueda arrancar directamente en el entorno de Tiempo-Real de xPC Target® mediante la creación de un disco de arranque (disco compacto, unidad de almacenamiento USB o tarjeta de memoria SD), el medio de arranque se crea desde el ordenador que hace de supervisor, el cual conoce la configuración hardware de la estación de trabajo, por lo que la unidad de arranque contendrá los controladores de las placas de entrada/salida necesarios así como el modelo de control ya compilado.

El procedimiento de creación de un controlador digital autónomo basado en un ordenador PC mediante un disco de arranque se utiliza para ordenadores de escritorio, pero existe otra opción, mucha más difundida, en la que también se utiliza una arquitectura basada en Intel® x86-x64, pero de reducido tamaño que carece de unidad óptica para la lectura de discos, en este caso el entorno software en Tiempo-Real se encuentra en una tarjeta de memoria Secure Digital (tarjeta SD). Estas arquitecturas PC de pequeño tamaño, denominados computadores de placa única (Single Board Computer, SBC) están basadas en el estándar PC 104 con un tamaño de 96 x 90 mm, organizado y controlado por el consorcio PC 104 [4]. Cabe señalar que existen computadores basados en el estándar PC 104 que no están basados en la arquitectura Intel® x86 o x64, estos PC 104 están basados en arquitecturas PowerPC de IBM. Estos computadores PC 104 no están soportados por la toolbox xPC Target®.

Pese al pequeño tamaño inicial de este formato de ordenadores PC, como puede verse en la figura 2.5, las distintas placas de expansión de entrada/salida necesarios se van apilando a lo alto, conectándose a la interfaz Peripheral Component Interconnect (PCI) o Peripheral Component Interconnect Express (PCI-Express).

La conexión del bus PCI/PCI-Express presente en el estándar PC 104 está diseñada para que se pueda conectar una placa de expansión encima de otra, en la

[4]http://www.pc104.org/ , PC-104 estandar e interfaces, Última consulta Enero 2014

Figura 2.5: *Computador en placa única basado en estándar PC 104*

figura 2.6 se muestra un computador de estándar PC 104 funcionando bajo xPC Target®, en este caso se encuentran varias placas de entrada/salida analógicas y digitales, entre ellas una para bus CAN. Se aprecia como el sistema RCP alcanza un tamaño en forma de torre considerable.

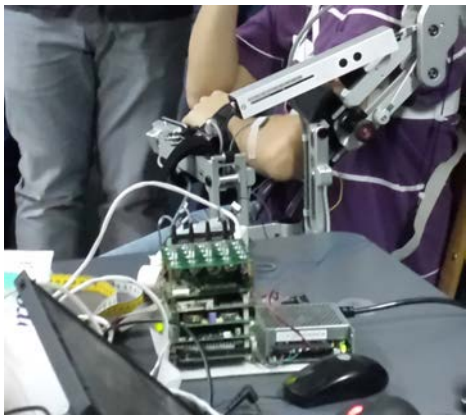


Figura 2.6: *Sistema RCP con xPC Target® sobre arquitectura PC 104 con procesador Intel Atom, con múltiples tarjetas de expansión de entrada/salida*

Pese a tratarse de una arquitectura muy bien concebida, no puede cumplir muchos de los objetivos propuestos en esta Tesis (véase el apartado 1.5). El tamaño y el consumo energético no son adecuados para las aplicaciones de control embebido, en las que el hardware de control se ubica en el propio exoesqueleto o sistema para rehabilitación. Por otra parte el coste de un sistema RCP basado en xPC Target® es bastante alto, la escasez de fabricantes para el estándar PC 104 hace que sea un sistema caro de adquirir; la placa base con procesador Intel® x86 (Intel® Atom®) para este estándar no se localiza por menos de 200€ y algunas placas de

expansión, como la de bus CAN, que cuentan con un único fabricante, incurre en un coste de 600€. Contando tan sólo con el factor precio, sería imposible disponer de un número suficiente de sistemas RCP en cualquier laboratorio docente o de investigación.

Por otro lado, la capacidad computacional se encuentra sobredimensionada para control de sistemas de actuación electromecánicos, y dentro de las posibilidades de programación de la toolbox xPC Target® no se ha encontrado la posibilidad de gobernar distintas tareas en Tiempo-Real o aprovechar el tiempo libre del procesador entre iteraciones de la tarea en Tiempo-Real. Toda funcionalidad de entrada/-salida debe ser añadida de forma externa mediante placas de expansión, desde entrada/salida digital y medios de comunicación básicos para sensores tales como SSI, SPI, I2C...

Este sistema RCP encuentra una buena acogida en centros de investigación, ejemplos de su utilización para controlar exoesqueletos similares al propuesto en el proyecto HYPER los encontramos en las siguientes referencias [18], [19], [20], [21], [22] y en el propio exoesqueleto del proyecto HYPER para el tren inferior humano, exoesqueleto H2-HAL. También es muy utilizado para sistemas mecatrónicos como robots [23], [24] o para máquinas-herramientas basadas en control numérico [25]; en definitiva, para tareas de control de todo tipo de sistemas electromecánicos y motores [26], [27] así como gestión de sistemas de almacenamiento y generación de energía eléctrica [28].

2.3.2. CompactRIO® y MyRIO®

CompactRIO® y MyRIO® son sistemas embebidos y reconfigurables orientados para control, adquisición y procesamiento de datos. La arquitectura del hardware del sistema CompactRIO®/MyRIO® está basada en una FPGA, que ejerce de medio de entrada/salida, y en un procesador sobre el que se ejecuta un sistema operativo en Tiempo-Real, en este caso el sistema operativo es VxWorks®, del fabricante Wind River Systems® [5]. Al procesador no se le ha otorgado capacidad innata de entrada/salida mediante periféricos, sino que debe comunicarse con la FPGA de forma que ésta le transmita los datos provenientes de las interfaces de entrada/salida que se hayan sintetizado en la FPGA, o se hayan conectado a la FPGA mediante módulos de expansión externos. El concepto de esta arquitectura se muestra en la figura 2.7 de forma simplificada.

Figura 2.7: *Arquitectura de un sistema RCP CompactRIO®*

Estos sistemas se programan con herramientas de programación gráfica de National Instruments® LabVIEW®, la figura 2.8 muestra un pequeño ejemplo de programación visual en LabView®. Se trata de un lenguaje gráfico, que se utiliza de manera ligeramente diferente al proporcionado por Matlab/Simulink®, que hace más hincapié en aspectos de más bajo nivel referentes al uso específico de una FPGA.

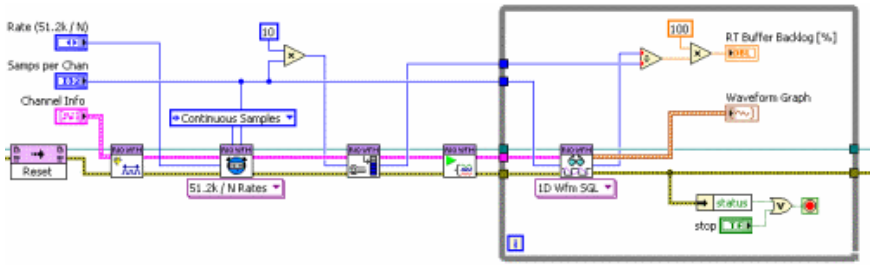


Figura 2.8: *Programación en lenguaje gráfico de LabView®*

Los sistemas CompactRIO® y MyRIO® han sido evaluados gracias a la oferta de

[5] http://www.windriver.com/products/vxworks/ , VxWorks RTOS, Enero 2014

talleres prácticos gratuitos por parte de National Instruments®, talleres impartidos en las dependencias de la Universidad Carlos III de Madrid.

La metodología de trabajo se basa en realizar programas en lenguaje gráfico, para el procesador y para la FPGA por separado. La comunicación entre ambos componentes no es directa o sencilla, se deben utilizar unos mecanismos específicos que dependen del modelo de procesador y de FPGA presentes, no es fácil generalizar este mecanismo. Mecanismo que requiere establecer una sincronización en el envío/recepción de datos entre procesador y FPGA, aspecto que hace perder nivel de abstracción al lenguaje gráfico de LabView.

Por otro lado, disponer de una FPGA como dispositivo intermedio en la adquisición de datos hace posible establecer un tratamiento de los datos antes de ser recuperados por el procesador, este procesamiento de datos puede realizarse sólo con números enteros, números decimales en punto fijo o únicamente bits. La longitud en bits de los datos hace decrecer rápidamente la cantidad de celdas lógicas disponibles en la FPGA y los números decimales en punto flotante no están soportados, a menos que la FPGA cuente con una o varias unidades de coprocesamiento, unidades de cálculo no disponibles en los modelos CompactRIO®/MyRIO® de menor coste. La capacidad de procesamiento de la FPGA se encuentra circunscrita a un repertorio limitado de operaciones, tales como suma, multiplicación, resta... particularidades que acotan su campo de aplicación. Generalmente son sistemas muy utilizados por ingenieros de telecomunicaciones en modulación/demodulación y encriptación/desencriptación de señales digitales, ya que las operaciones a realizar están cubiertas por este limitado repertorio y una FPGA las realiza en cuestión de nanosegundos, en cambio para operaciones más complejas en las que interviene ejecución condicional, un procesador es mucho más apropiado, estos aspectos se encuentran más profundamente detallados en el apartado 2.3.2 .

Se deben resaltar varios aspectos muy importantes en el uso de LabView®, este sistema RCP pese a contar con un sistema operativo en Tiempo-Real, no permite gobernar ni conocer el número de tareas en Tiempo-Real lanzadas por el sistema operativo, utilizando el lenguaje gráfico. En cambio programando el sistema en lenguaje C sería posible, pero de esta forma se perdería la abstracción y versatilidad del lenguaje gráfico. Tampoco es posible crear soporte de hardware personalizado en LabView®, esta posibilidad no existe, sólo se puede utilizar el hardware ofertado por National Instruments®.

Al igual que los sistemas basados en xPC Target®, CompactRIO® se emplea en un amplio repertorio de aplicaciones de control embebido. Se encuentran exoesqueletos [29], sistemas de control más o menos clásicos [30], [31], tratamiento de

señales de alta frecuencia [32], sistemas de encriptación/desencriptación de datos en formato digital [33] y sistemas de visión artificial [34].

Los sistemas RCP CompactRIO® y MyRIO® generalmente se presentan con el formato mostrado en la figura 2.9.

Figura 2.9: *A la derecha sistema RCP CompactRIO® y a la izquierda sistema RCP MyRIO®*

El formato básico de CompactRIO®, denominado NI cRIO-9076®, integra un procesador en tiempo real de 400 MHz con un LX45 FPGA y posee cuatro ranuras para módulos de I/O de la Serie C. Estos módulos de entrada/salida proveen de capacidades de envío y adquisición de datos analógicos, digitales y trenes de pulsos modulados en ancho de pulso (PWM), bus CAN, bus LIN, Profibus, puerto serie RS-232 y RS-485, Temporizadores... existe un amplio repertorio, y dentro de cada modalidad se pueden elegir rangos de tensión, salida/entrada diferencial o unipolar, optoacoplados...

La versiones extendidas de la familia de sistemas RCP CompactRIO® presentan volúmenes superiores y capacidad de conectar aún más módulos de I/O.

Respecto al funcionamiento y comodidad de manejo de este sistema RCP; una vez configurado el target (procesador y modelo de FPGA), se realiza un programa en lenguaje gráfico de LabView®, tanto para el procesador como para la FPGA, un programa sencillo puede ser un circuito lógico combinacional de tres puertas lógicas a nivel de bits para la FPGA, sin programa para el procesador de Tiempo-Real. La imágenes muestran el tiempo de espera hasta que el modelo se convierte en código, se compila/sintetiza y se transfiere al hardware, siempre que se utilice la FPGA el tiempo total de compilación y síntesis estará cercano a los 10 minutos, como puede verse en la figura 2.10. Tiempo que se debe esperar siempre entre iteraciones del modelo de control. Si se utiliza también un programa para el procesador de Tiempo-Real, y otro para la FPGA, el tiempo total a esperar serán los

aproximadamente 10 minutos de síntesis para la FPGA y un minuto para la generación y compilación del código para el procesador. En el sistema MyRIO® es posible utilizar la interfaz SPI del procesador para acceder a los periféricos de entrada/salida sin necesidad de utilizar la FPGA, por lo que se pueden evitar los largos tiempos de síntesis de la FPGA. Para el sistema CompactRIO® la interfaz de comunicación con el PC es ethernet, el sistema MyRIO® dispone de interfaz USB. Lo errores cometidos en la programación de la FPGA se muestran en la etapa de generación de código VHDL y síntesis, LabVIEW no detecta estos errores antes de entrar en esta fase.

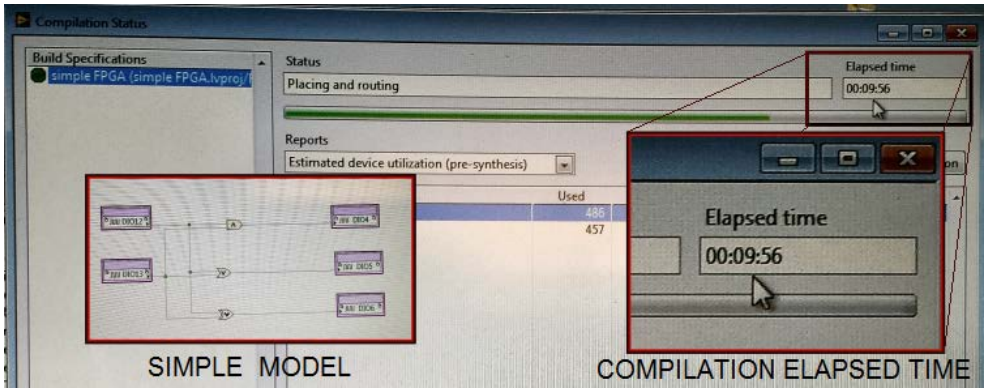


Figura 2.10: *Compilación de un modelo sencillo en LabView® para el RCP CompactRIO®. Se muestra a la izquierda el modelo y a la derecha el tiempo transcurrido desde que comenzó el proceso de sistensís y compilación*

2.3.3. dSPACE®

dSPACE® es una compañía fundada en 1988 en Alemania, centrada en ofrecer productos software y hardware para prototipado rápido para control (RCP) que cumplan con todos los estándares y certificaciones exigidos por la industria automovilística, aeronáutica, dispositivos médicos y cadenas de producción [6].

Los productos software y hardware de dSPACE® se integran en Matlab/Simulink®, algunos de ellos son herramientas adicionales a modo de toolboxes muy específicas, orientadas en gran medida al mundo del control para automóvil. Los modelos de Simulink®, diseñados de forma gráfica, se ejecutan en su hardware de Tiempo-Real, pero este hardware no está orientado en ningún caso para utilizarse en la implementación final. Son soluciones para la etapa de prototipado en laboratorio y verificación, no cubren la última etapa.

En los sistemas RCP de dSPACE® se distinguen dos tendencias, una de ellas orientada a probar el modelo de control en un hardware de Tiempo-Real, modelo de simulación con procesador real, del inglés PIL "Processor-in-the-loop". La otra tendencia está orientada a probar el modelo de control en Tiempo-Real junto con la simulación de la planta física a controlar, ejecutándose todo ello en Tiempo-Real, modelo de simulación HIL "Hardware-in-the-loop", en el que se tiene una planta virtual y un controlador real, ambos funcionando en un hardware de Tiempo-Real, con tiempos de muestreo del orden de microsegundos. El hardware de dSPACE® destinado a simulaciones HIL tiene forma de "mainframe" de gran potencia computacional. La gran mayoría de sistemas RCP de dSPACE® están basados en procesador, utilizando a día de hoy procesadores PowerPC® de IBM®; el uso de FPGA por parte de este fabricante queda relegado a módulos de expansión de I/O para tratar señales digitales de alta frecuencia.

Dentro del repertorio de sistemas RCP, de aquellos orientados a ejecutar el modelo de control en Tiempo-Real, encontramos tarjetas de expansión para bus PCI orientadas a su uso en ordenadores PC; estas tarjetas proporcionan capacidades de I/O al PC. El procesador que ejecuta el modelo diseñado en Simulink® se encuentra en la propia tarjeta, funcionando bajo el sistema operativo en Tiempo-Real QNX®, de la compañía QNX Software Systems Limited® que ahora mismo pertenece al grupo Blackberry® [7].

[6] https://www.dspace.com , Información de la compañía, Enero 2014

[7] http://www.qnx.com/company/ , QNX Software Systems Limited, Blackberry, Enero 2014

Las tarjetas de expansión para ordenadores PC, como el modelo DS1104 mostrado en la figura 2.11 ,proporcionan un sistema RCP con procesador propio modelo PowerPC 603e a 250 Mhz con 32KB de memoria caché y unidad de coma flotante de 64 bits, con 32 MB de memoria SDRAM y 8 MB de memoria de programa ROM. Respecto a la I/O que proporciona la tarjeta provee de 5 timers de 32 bits, 1 timer de 64 bits, 1 A/D de 16 bits, 4 A/D de 12 bits, 8 D/A 16 bits, 20 I/O digital, 2 entradas de encoder con contador de 24 bits, 1 UART RS-232/485.

Figura 2.11: *Sistema RCP DS1104 de dSPACE® para ordenadores PC*

La tarjeta DS1104 también cuenta con un microcontrolador Texas Instruments® TMS320F240 [8], cuyo procesador es de 16 bits sin unidad de cálculo en coma flotante, funcionando a una frecuencia de reloj de 20 Mhz, contando con 32KB de memoria de programa ROM y 8 KB de memoria RAM, proporciona 7 salidas PWM, 4 entradas PWM, 1 puerto SPI y 14 I/O digitales adicionales. El consumo de la tarjeta DS1104 es de 18.5 Wattios. Su coste aproximado es de 950€ o 4800€[9] junto con sus toolboxes para Simulink®.

Además de esta tarjeta, existen otras de mayores capacidades, en las que cada parte está separada, una tarjeta con un procesador en Tiempo-Real a 1Ghz, a la que se conectan otras tarjetas de expansión que ejercen de módulos de I/O. El repertorio es muy amplio y completo, adecuándose a las complejas o específicas necesidades que se requiera cubrir.

Respecto a los sistemas RCP cuyo funcionamiento es totalmente independiente de un ordenador PC, dSPACE® cuenta con un hardware denominado MicroAutoBox, en formato de tamaño más reducido con procesador IBM PowerPC® a 900 Mhz o con un tamaño algo mayor con procesador Intel ATOM, la versión con procesador Intel® no puede funcionar en Tiempo-Real con el sistema operativo QNX®. Se debe resaltar que se trata de un sistema RCP muy orientado al mundo del automóvil pues sus interfaces CAN y LIN cuentan con soporte para protocolo

[8] http://www.ti.com/product/tms320f240 , Información sobre el MCU
[9] Fuente: Consulta al departamento de ventas de dSPACE® , año 2013

de calibración de control del automóvil estandarizado por bus CAN (Can Calibration Protocol, CCP) o a través de algunas otras interfaces de comunicaciones denominándose en ese caso Protocolo de Calibración Extendido (XCP).

La figura 2.12 muestra el cuidado aspecto de un sistema RCP MicroAutoBox.

Figura 2.12: *Sistema RCP MicroAutoBox*

Para dotar a este RCP de capacidades de entrada/salida se le debe dotar de las apropiadas tarjetas de I/O, la figura 2.13 muestra dónde se conectan dichas tarjetas de expansión del sistema MicroAutoBox.

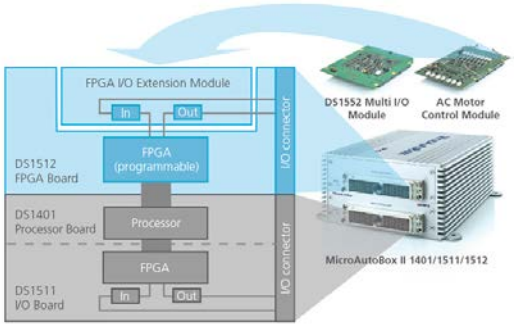


Figura 2.13: *Posibilidades de I/O del sistema RCP MicroAutoBox*

En los sistemas MicroAutoBox en Tiempo-Real sólo puede colocarse una tarjeta de expansión I/O. El precio para universidades, junto con sus toolboxes de Simulink® y una tarjeta de I/O a elegir asciende a 12500€[10] por unidad.

Todos estos sistemas que se han descrito, se convierten en sistemas de prototipado rápido si se instalan y utilizan sus respectivas toolboxes para Simulink®, esta familia de toolboxes se denominan Real-Time Interfaces Blocksets (RTI). Cada funcionalidad de I/O tiene su propia toolbox que se vende por separado, así encontramos una toolbox genérica para manejar el procesador de Tiempo-Real, gobernar

[10]Fuente: Consulta al departamento de ventas de dSPACE® , año 2013

las tareas del sistema operativo en Tiempo-Real QNX y se necesitarán las toolboxes correspondientes a las tarjetas de expansión de las que dispongas. La figura 2.14 muestra los bloques de programación gráfica disponibles para el bus CAN de cualquier sistema RCP de dSPACE®, conjunto de bloques denominado RTI CAN.

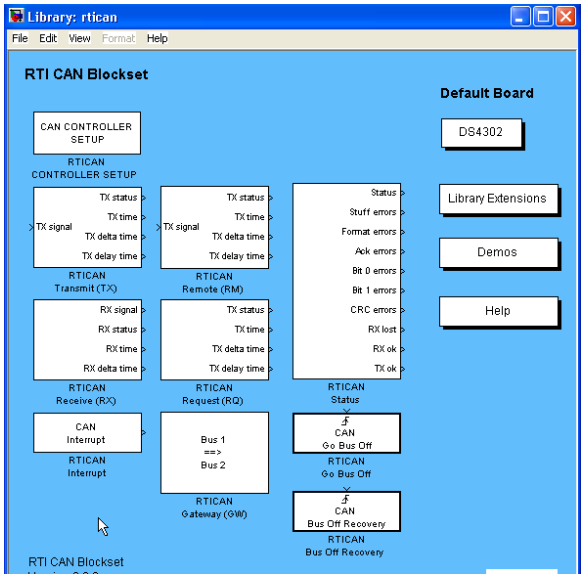


Figura 2.14: *Blockset para Simulink® de la toolbox RTI CAN*

Proyectos en el seno de la investigación utilizando sistemas RCP de dSPACE® DS1103/DS1104 (tarjetas de expansión para PC) los encontramos en áreas como el almacenamiento y producción de energía eléctrica [35], [36], [37], [38] muy frecuentemente en control de motores eléctricos [39], [40], [41], [42], [43]. Mientras que el uso en robótica es mucho menos frecuente que los anteriores [44], [45]. Para visión artificial no hay resultados destacables, mientras que el RCP MicroAutoBox se utiliza generalmente para control embarcado en automóvil [46], [47], [48].

2.3.4. IBM Rational®

La familia de programas IBM Rational® cubre un amplio espectro en lo que a ámbitos de trabajo se refiere, haciendo un uso intensivo de los lenguajes UML y SysML. Existen programas dentro de esta familia que permiten crear interfaces de usuario y aplicaciones de gestión y consulta de bases de datos para ordenadores PC, todo mediante dicho lenguaje basado en gráficos generando el código fuente textual en C, C++, ADA o Java.

Otros programas de la familia Rational sirven para llevar a cabo simulaciones de sistemas multidominio; sistemas electromecánicos, hidraúlicos... y otros, como Rational Rhapsody® sirven para diseñar mediante gráficos, controladores hardware embebidos para ciertas arquitecturas soportadas o para la arquitectura que el usuario decida, permitiendo dotar al programa Rhapsody de soporte hardware adicional.

Todos los programas de la amplia familia Rational se pueden integrar entre sí, como si de un único entorno basado en modelos se tratara, complementando sus funcionalidades. Las características de Rhapsody son tan similares a SIMULINK® que ambos software se pueden relacionar entre sí, transfiriendo los modelos gráficos desde Rhapsody a SIMULINK® y viceversa, pese a que SIMULINK® no está basado en UML/SysML. La figura 2.15 ilustra un modelo de Rhapsody importado en SIMULINK®. Uno de los motivos relevantes al utilizar varios software MBD para realizar la misma simulación es la comparación de resultados.

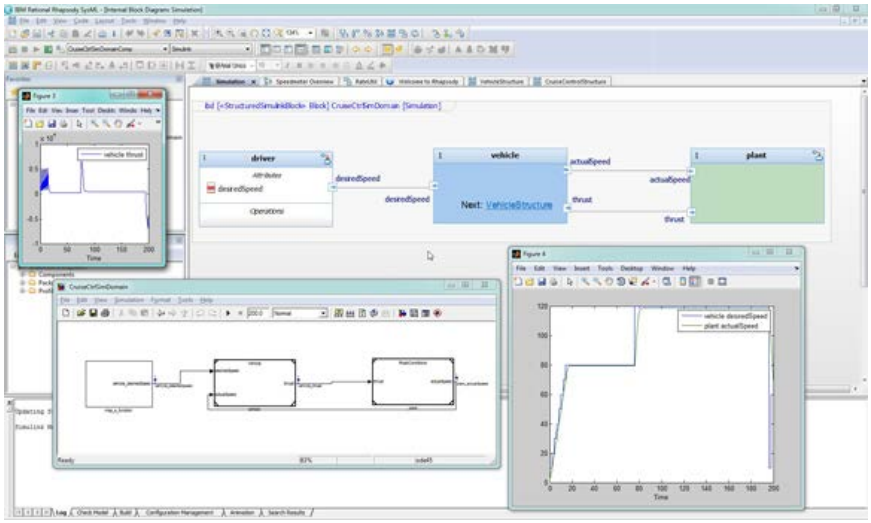


Figura 2.15: *Un modelo diseñado en IBM Rational Rhapsody® importado en SIMULINK®*

2.3.5. Conclusiones RCP comerciales

Una vez analizados los sistemas RCP comerciales más representativos, siendo también los más utilizados en el ámbito referente a la investigación, se pueden extraer una serie de conclusiones. Nótese que en el anexo 1 se localiza el análisis de más sistemas RCP de este tipo.
En primer lugar, se puede apreciar la profundidad del soporte en los lenguajes gráficos de cada sistema RCP comercial, resultando fácil diferenciar entre las toolboxes que proporcionan una funcionalidad básica y aquellas que proporcionan una funcionalidad avanzada.

Por el término ***funcionalidad básica*** se entiende aquellas toolboxes de programación gráfica para sistemas RCP que permiten configurar el controlador digital; configurar el procesador, y proporcionan acceso para manejar de forma gráfica los periféricos de I/O.

Mientras que el término ***funcionalidad avanzada*** hace referencia a aquellas toolboxes de programación gráfica que además de incluir las funcionalidades básicas de I/O, habilitan el funcionamiento bajo el paradigma de multitarea en Tiempo-Real, haciendo uso del tiempo libre entre iteraciones del modelo en Tiempo-Real y/o habilitando el uso de un Sistema Operativo en Tiempo-Real (RTOS, del inglés *Real-Time Operating System*), permitiendo también gobernar las distintas tareas. También entra dentro de las características avanzadas de un lenguaje de programación gráfico para sistemas de control, el hecho de que el mismo RCP pueda medir tiempos de ejecución con gran precisión (al menos en términos de microsegundos) del algoritmo completo y de las partes de éste. Así mismo, toda funcionalidad que caiga más allá del dominio propio de las funcionalidades de I/O, funcionalidades tales como la opción de sincronizar el temporizador de Tiempo-Real entre varios sistemas RCP, capacidades de audio y vídeo, son consideradas funcionalidades avanzadas. También se consideran funcionalidades avanzadas la capacidad dentro de un lenguaje de programación gráfico de poder modificar las rutinas de atención a interrupción, no son funcionalidades cuya presencia sea frecuente en las diferentes toolboxes.

De los sistemas RCP comerciales más utilizados en investigación, aquellos que presentan una funcionalidad avanzada son los proporcionados por dSPACE® y RT-Lab®, pues permiten gobernar el RTOS sobre el que hacen funcionar el modelo de control de SIMULINK® y debido a la gran cantidad de expansiones software en forma de más bloques que profundizan en las capacidades I/O de forma muy específica.

Los casos referentes a xPC Target® y CompactRIO®/MyRIO® son casos enigmáticos. Pese a ser sistemas RCP que cuentan con un RTOS, éste no es gobernable desde

el entorno de programación gráfico, la generación y gestión de tareas no se encuentra al acceso del programador. Se ha de considerar que son sistemas RCP que cuentan con una toolbox, en el primer caso para SIMULINK®, en el segundo caso para LabView®; que proporcionan una funcionalidad básica, con algunas características avanzadas como la capacidad de programar de forma personalizada una rutina de atención a interrupción, como es el caso de algunas tarjetas de expansión de I/O digital de unos pocos fabricantes para la toolbox xPC Target®. Respecto las funcionalidades de audio y vídeo, tan sólo xPC Target® cuenta con capacidad para mostrar información en forma de texto a través de un puerto para monitor VGA. El resto de sistemas no poseen ninguna capacidad multimedia accesible desde el lenguaje de alto nivel de abstracción. También xPC Target® cuenta con capacidades USB limitadas, al permitir conectar determinados modelos de WebCAM. En la tabla 2.1, mostrada en la siguiente página se ofrece un resumen con las capacidades de cada sistema RCP comercial, accesibles desde el lenguaje basado en gráficos.

El resto de sistemas RCP comerciales además de tener una presencia casi nula, o muy poca representación en el sector de la investigación, también suelen presentar de forma generalizada unas toolboxes de programación gráfica que proporcionan únicamente funcionalidad básica.

Retomando los objetivos propuestos en esta tesis encontramos que la totalidad de estos sistemas RCP comerciales están diseñados teniendo en mente la manera tradicional de trabajar con un RCP, sírvase la figura 1.3 como recordatorio, de forma que son útiles para prototipar los sistemas de control en el laboratorio, pero no resulta posible englobar con ellos la etapa de implementación final. Además tanto la potencia computacional, como las dimensiones físicas y el consumo energético de estos sistemas comerciales quedan muy alejados de los objetivos y necesidades de esta tesis y el proyecto que le sirve de marco. También desde el punto de vista del coste a nivel económico. Por lo que la utilización de cualquiera de estos sistemas queda descartada. Respecto a los costes de adquisición de los diferentes sistemas RPC comerciales, la tabla 2.2 muestra la inversión necesaria tanto para adquirir una unidad hardware como el mantenimiento de las licencias software necesarias para un puesto de trabajo.

Tabla 2.1: *Funcionalidades básicas y avanzadas disponibles desde el lenguaje gráfico para cada sistema RCP comercial*

	FUNCIONALIDADES BÁSICAS					FUNCIONALIDADES AVANZADAS									
RCP System	Real-Time Scheduler	I/O	IRQ I/O	DMA I/O	Time Profiling	RTOS Multitask	Non RTOS Multitask	Task Manager	Timer IRQs	Continous Time	Virtual Plant	MCU Sync	Exception Manager	USB I/O	Multimedia
XPC Target	Yes	Yes	Yes	Yes	Yes	Partially Supported	No	No	No	No	Yes	No	No	Partially Supported	Partially Supported
CompactRIO MyRIO	Yes	Yes	Yes	Yes	Yes	Partially Supported	No	No	No	No	No	No	No	No	No
dSPACE	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	Yes	No	No	No	No
RT LAB	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	Yes	No	No	No	No
MicroGEN 555	Yes	Yes	Yes	Partially Supported	Partially Supported	No	No	No	No	No	No	No	No	No	No
HILINK RAPCON	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No	No	Partially Supported	No

Fabricante	Sistema RCP	Coste € (Hardware + Software)
dSPACE	DS1104	5800
dSPACE	Mainframes	30000 - 90000
RT-Lab	PC 104	10000
RT-Lab	Mainframes	42000 - 120000
National Instruments	CompactRIO	3000
National Instruments	MyRIO	1500
MathWorks	xPC Target (PC 104)	5600 + (200 - 600) per I/O module
Add2	MicroGen555	3800

Tabla 2.2: *Comparación costes de adquisición para los sistemas RCP comerciales. Se incluyen los costes de una unidad hardware y de las licencias software necesarias para un puesto de trabajo.*

En los costes económicos reflejados en la tabla anterior no se han tenido en cuenta los costes de adquisición de los requisitos software previos, tales como los supuestos por la compra de MATLAB® o LabVIEW®.

Asimismo, algunos de estos fabricantes cobran licencias por el uso de su software, toolboxes para SIMULINK® o LabVIEW®, por cada puesto de trabajo (en inglés se denomina "engineering seat based license"). National Instruments® y MathWorks® llevan a cabo estas prácticas. En cambio, dSPACE®, RT-Lab®, Add2® permiten la instalación de sus toolboxes en múltiples puestos de trabajo al mismo precio. Las diferentes toolboxes sólo pueden funcionar si el sistema RCP del fabricante se encuentra conectado al ordenador, independientemente de si las toolboxes están instaladas en un ordenador o en decenas.

El caso de los computadores de gran potencia computacional, los denominados "mainframes", es diferente pues sus costes de adquisición incluyen toolboxes muy específicas y complejas orientadas a sectores de los considerados críticos, como el sector aeronáutico o el control de sistemas basados en energía nuclear.

2.4. Toolboxes Programación Gráfica para MCU

Una vez estudiados los sistemas RCP comerciales disponibles y comparadas las prestaciones e inconvenientes de éstos con los objetivos fijados en esta tesis, se llega al punto en el que se puede estudiar si las posibilidades que ofrecen los microcontroladores actuales (años 2013-2014) para control de sistemas electromecánicos, y sus opciones para programar estos últimos mediante lenguajes de programación basados en gráficos, permiten que su manejo sea igual o similar a un sistema RCP de los estudiados en el apartado anterior.

Para alcanzar el éxito en estos planteamientos, el microcontrolador debe proveer de una buena capacidad computacional y de una gran cantidad de periféricos de I/O.

En los apartados sucesivos se estudiarán diferentes familias de microcontroladores, explicando ligeramente su arquitectura y el soporte para programación gráfica en Matlab/Simulink® existente para ellos.

2.4.1. dsPIC Blockset®

Los microcontroladores (MCU) PIC®, dsPIC24-30-33F® [11] son MCU cuyo procesador central maneja instrucciones de 16 bits (de ahora en adelante se dirá que es un MCU de 16 bits) del fabricante MicroChip® [12].

Disponen de suficientes periféricos de I/O para resultar útiles para utilizarlos en ingeniería de control, así mismo existe una toolbox de programación gráfica para Matlab/Simulink® denominada "MPLAB 16-Bit Device Blocks for Simulink" que provee de funcionalidad básica para programar el MCU mediante el lenguaje gráfico de Simulink®, estas funcionalidades no cubren la totalidad de los periféricos presentes en esta familia de MCU. Sólo hay bloques para manejar las I/O analógicas y digitales, las salidas PWM y PWM para motores brushless, I2C, SPI, CAN y UART.

No hay bloques de funcionalidad avanzada, no se puede manejar el puerto

[11] http://www.microchip.com/pagehandler/en-us/family/16bit/architecture/dspic33f.html?f=4 , Información sobre los MCU, Enero 2014

[12] http://www.microchip.com/pagehandler/en-us/aboutus/home.html , web del fabricante, Enero 2014

USB, las entradas PWM, tampoco se pueden manejar los timers al antojo del usuario ni reprogramar rutinas de IRQ; no está presente el bloque de interrupción externa (EXTI) entre otros. No hay soporte para RTOS ni para utilizar el tiempo libre entre iteraciones del modelo en Tiempo-Real.

La figura 2.16 proporciona una idea de la extensión en número de los bloques de programación gráfica disponibles en la toolbox para Simulink® de los MCU de 16 Bits PIC.

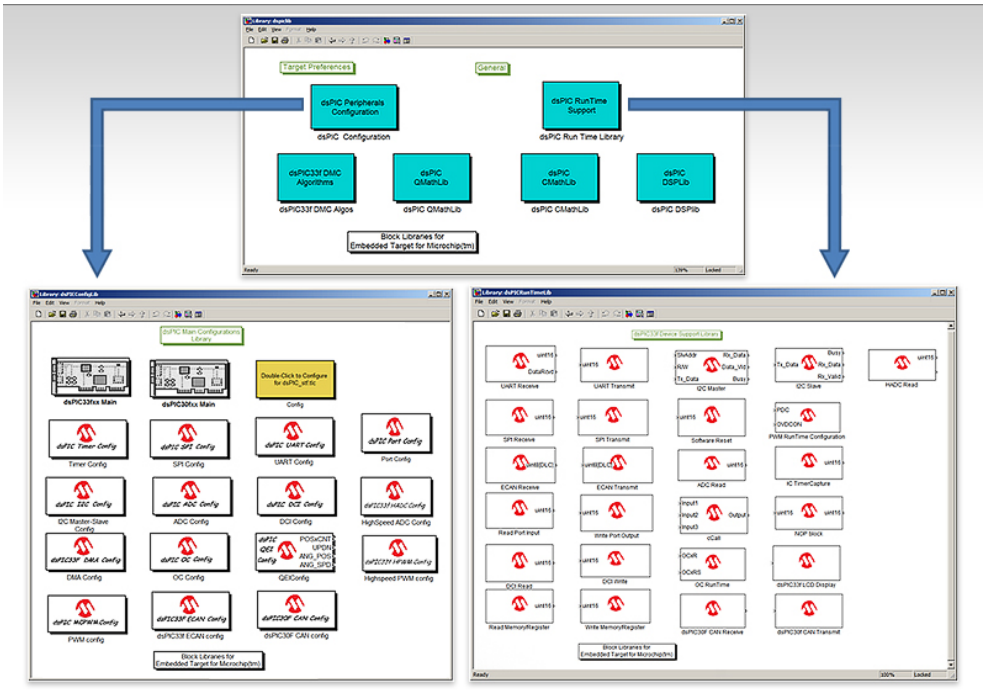


Figura 2.16: *Toolbox MPLAB 16-Bit Device Blocks para programación gráfica en Simulink® de MCU de 16 bits de MicroChip*

Esta toolbox sólo funciona bajo entorno Windows® y versiones de Matlab® de 32 bits. Requiere también del entorno de desarrollo integrado (IDE) MPLAB® de MicroChip®. El coste de adquisición de la toolbox de programación gráfica y del entorno IDE junto con un kit de desarrollo más un programador/depurador asciende a un total de 1495€[13].

La toolbox soporta únicamente las MCU de 16 bits, y no las familias de 32 bits

[13]Consulta realizada en Noviembre de 2013

aparecidas a finales del año 2013 [14], esto junto con el alto coste de los kits de desarrollo para los MCU PIC de 32 bits, en torno a 120€ sin programador[15] han hecho que se haya descartado su utilización en esta tesis.

2.4.2. FreeScale HCS12®

FreeScale HCS12® es un MCU de 16 bits del Fabricante FreeScale® orientado al sector del automóvil, cuyas características en lo que respecta a capacidades I/O y computacionales son similares a la familia de MCU de 16 bits de MicroChip®. Es un MCU muy veterano, aunque sigue produciendo en masa.

Pese a continuar en la línea de producción, el soporte para Matlab/Simulink® que otorgaba funcionalidades básicas ha sido descontinuado a partir del año 2014, en la última consulta de Noviembre de 2013, la toolbox costaba 1000€. Como se ha indicado, ahora ya no se encuentra disponible en la web del fabricante FreeScale®, y éste redirige a la toolbox desarrollada por SimuQuest® para los MCU de la familia HCS12.

Cabe decir que existe otra toolbox para Simulink® de uso gratuito para fines educativos y de investigación, que proporciona una funcionalidad muy básica para MCU HCS12, como curiosidad soporta el protocolo de calibración por bus CAN (CCP) propio del sector del automóvil. Esta toolbox se denomina "ertha blockset" [16] de la compañía Feaser [17].

En el apartado dedicado al soporte para Matlab/Simulink® por parte de la compañía SimuQuest® se profundizará más en dichas toolboxes.

2.4.3. Motorola/FreeScale MPC555

El MCU de Motorola® MPC555, ahora fabricado por FreeScale®, es un MCU de 32 bits diseñado desde su concepción para perdurar en el tiempo, para mantener una vigencia temporal. Esta característica se materializa en una capacidad para contener hasta 4 MB de ROM y 4 MB de RAM, unidad de cálculo de coma flotante de 32 bits y hasta una velocidad de CPU de 132 Mhz desde sus inicios en

[14] http://www.microchip.com/pagehandler/en-us/family/32bit/ , Información sobre los MCU de 32 bits, Enero 2014

[15] Consulta realizada en Febrero de 2014

[16] http://www.feaser.com/en/ertha.php , Información sobre la toolbox, Enero 2014

[17] http://www.feaser.com/en/about.php , Información sobre la compañía, Enero 2014

las versiones más caras. La CPU de este MCU es un PowerPC de IBM®, éstas son CPU originalmente orientadas a arquitecturas computacionales Von Neumann, en este caso al dotar a la CPU de una gran cantidad de periféricos de I/O orientados a ingeniería de control y de una memoria de programa y de datos separadas, se ha convertido automáticamente en una arquitectura Harvard, como todo MCU.

Las primeras versiones de este MCU fueron fabricadas bajo la marca de Motorola®, en el año 2004 los MCU de Motorola® pasaron a ser fabricados por FreeScale®, por lo que este potente MCU fué diseñado con anterioridad al año 2004. Pese a su antigüedad sigue siendo una opción muy competente e interesante.

Resulta obvio, que dado el carácter de dispositivo de alta cualificación dentro del mercado de los MCU, dispone de multitud de periféricos de I/O de forma que satisfaga todas las necesidades.

Respecto al soporte para programación gráfica de este MCU, se basa en una toolbox para Simulink® ofrecida por FreeScale®, la toolbox se denomina "RAppID" y provee de funcionalidad avanzada, se pueden gestionar las tareas del RTOS OSEK, también de FreeScale® y medir tiempos de ejecución. El coste de adquisición de la toolbox es de aproximadamente 7000€ (impuestos incluídos)[18]. Además se necesitará alguno de estos tres IDE: DIAB [19], GHS[20] o el compilador de FreeScale®.

El principal problema para adoptar este MCU como sistema RCP es su alto precio y el hecho de que a día de hoy, año 2014, los MCU de 32 bits de otros fabricantes están alcanzado capacidades computacionales similares a este MPC555 a precios mucho menores. Respecto al coste tan sólo el integrado del MCU en su versión más barata corriendo a 132 Mhz, con 2 MB de ROM y 64 KB de RAM cuesta 44.5€, mientras que un kit de desarrollo alcanza los 632€ para una versión del MCU con 512 KB de RAM [21].

[18]Consulta realizada en Febrero de 2014

[19]http://www.windriver.com/products/development_suite/wind_river_compiler/ , Enero 2014

[20]http://www.ghs.com/products/compiler.html , enero 2014

[21]Consultas realizadas en Farnell España en Enero 2014

2.4.4. SimuQuest®

SimuQuest® es una compañía fundada en el año 2001 [22], se centra en el diseño de sistemas de control a partir de modelo gráficos, utilizando Matlab/Simulink® para modelar matemáticamente la planta, diseñar el controlador y probarlo en la simulación de la misma y en la planta real mediante sistemas RCP. Ofrecen sus servicios para crear toolboxes de Simulink® para la arquitectura hardware que el cliente proponga, de forma que se generen nuevos sistemas RCP al gusto del usuario. La herramienta software destacada de esta compañía se denomina "Quantiphi", es una toolbox para Simulink® que habilita la programación gráfica con funcionalidades básicas para los MCU mostrados en la tabla 2.3 :

Renesas	32 bits	SH725xx
		R250 ECU
	16 bits	R8C/3X - 36x, 38x
		RL78 - G14
		RL78 - F13, F14
FreeScale	32 bits	MPC56x (actual MPC555 MCU family)
	8 - 16 bits	S12X - XEP, XEG, XET, XS
		S12G
		S12 - DG, DP, DJ, C, GC
		Control Base ECU
Microchip	8 bits	PIC18F2480 - PIC18F2580
Any MCU	x bits	Contact for custom blockset

Tabla 2.3: *MCU soportados en distintas versiones de la toolbox "Quantiphi"*

De los MCU mostrados en la tabla 2.3, los que pueden resultar útiles para formar un sistema RCP son los de 32 bits, pero ambos, tanto los SH72xx de Renesas® como los basados en MPC555 de FreeScale® son demasiado caros, contando tan sólo la adquisición del circuito integrado (CI). Un SH7211 cuesta 33€ por unidad contando tan sólo con 32 KB de RAM y cualquier MCU basado en MPC555 es más caro. El precio de los kits de desarrollo de cualquiera de estas familias de MCU es bastante caro, desde los 500€ aproximados[23] para un MCU SH7211 a los 632€ de un kit para MPC555.

Por otro lado, el coste de adquisición de la toolbox "Quantiphi" para MPC555 o SH72xx se encuentra en torno a 6000 - 7000€.

[22] http://www.simuquestinc.com/about , Información sobre la compañía, Enero 2014
[23] Consultado en la web de Renesas, Febrero de 2014

2.4.5. FlowCode®

FlowCode es un software independiente de Matlab® y de LabView® que proporciona un lenguaje de programación gráfico y un entorno de simulación para dispositivos electromecánicos. Pertenece a la compañía Matrix [24], fundada en 1993.

El paquete de software FlowCode® ha alcanzado su sexta versión, evoluciona rápido pues en 2012 estaba disponible la cuarta versión. Inicialmente pensado como herramienta de programación gráfica para MCU, se incluyeron opciones de simulación haciéndose más completo.

El lenguaje de programación gráfico basa su funcionamiento en diagramas de flujo, al estilo de Simulink® StateFlow®; por lo que los resultados y programas que se obtienen son muy semejantes a los orientados para autómatas. Son programas que evolucionan de un estado a otro a partir del cumplimiento de una condición de transición, puede definirse también como sistemas autómatas de Moore y/o Mealy. Pese a la sencillez en la programación, no puede alcanzar la flexibilidad que si otorgan Simulink® o LabView®, tampoco dispone de la multitud de toolboxes o herramientas software adicionales de las que se compone Matlab®, aunque si dispone de funcionalidades equivalentes a algunas partes de las toolboxes "DSP System Toolbox", parte de "SimMechanics" y "SimRF", por una fracción muy pequeña del coste de Matlab®.

El entorno de trabajo es el mostrado en la figura 2.17 , en la que se puede apreciar el lenguaje de programación basado en flujogramas, el modelo electromecánico y la visualización de datos adquiridos desde el MCU.

Comparando el lenguaje de programación con el proporcionado por Simulink StateFlow®, mostrado en la figura 2.18, existe gran similitud entre ambos.

Este software no es integrable con Matlab® o LabView®, su soporte para MCU es muy extenso cubriendo la totalidad de MCU PIC, dsPIC24, dsPIC32, Atmega, Arduino, y algunos MCU con CPU ARM del fabricante Atmel®. Su coste es de 300€ para la base de software y de 120€ para el soporte de cada familia de MCU. También se permite la creación de soporte personalizado por parte del usuario para integrar en la herramienta cualquier MCU a su elección.

[24] http://www.matrixmultimedia.com/about.php , Información sobre la compañía, Enero 2014

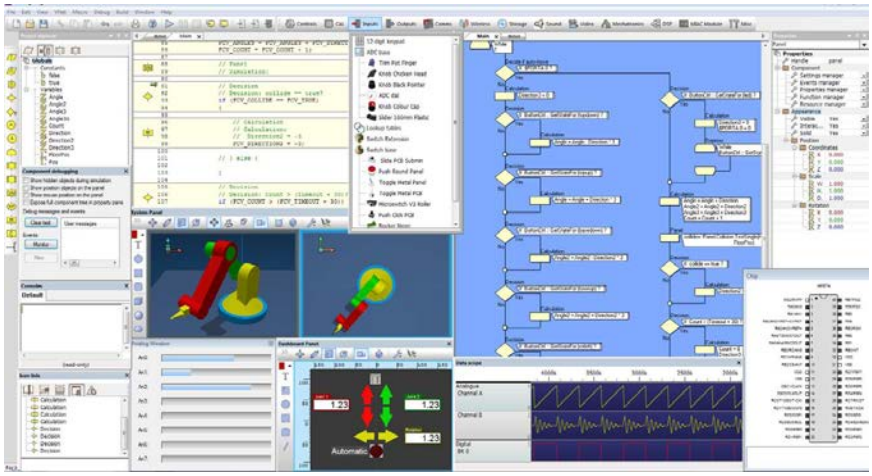


Figura 2.17: *Entorno de trabajo del software FlowCode®*

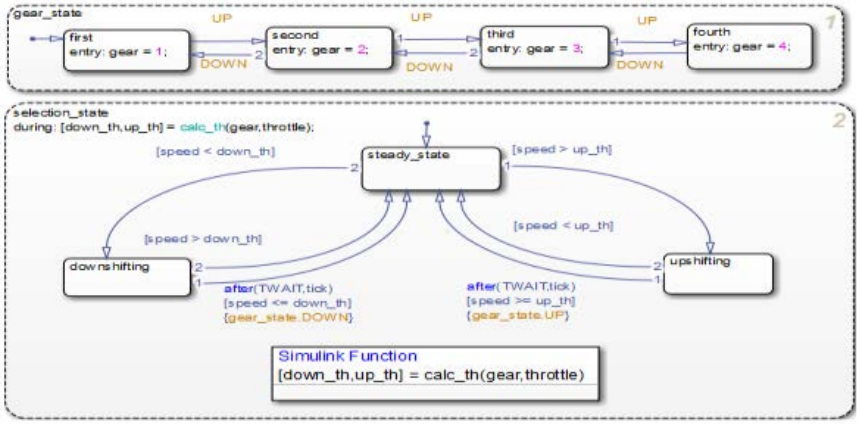


Figura 2.18: *Programa realizado de forma gráfica en Simulink Stateflow®*

Respecto a su utilización en el campo de la investigación se encuentran ejemplos de usos en aplicaciones de control avanzadas, como el presentado en [49], en el que se utiliza para programar el firmware de un MCU que gobierna un sistema de terapia médica por láser para tratamiento de heridas.

2.4.6. Aimagin®

Aimagin® es una compañía fundada en Febrero de 2010 en el sudeste asiático, con sede en Bangkok, Tailandia [25]. Su dedicación principal consiste en la difusión de los sistemas RCP y diseño basado en modelos en entornos educacionales. Impartiendo cursos sobre el uso de Matlab/Simulink® y sus distintas toolboxes más directamente relacionadas con la ingeniería de control, como "System Identification Toolbox", "DSP System Toolbox", "Fuzzy Logic Toolbox", "Curve Fitting Toolbox" ... tanto en universidades, instituciones dedicadas a la investigación y empresas tecnológicas. Así mismo los cursos se imparten mediante talleres prácticos utilizando sistemas RCP basados en MCU y plantas a controlar en forma de pequeñas maquetas electromecánicas.

Este concepto de enseñanza surge de la frustración que sintieron los socios fundadores durante su aprendizaje sobre ingeniería de control, aprendizaje basado en clases magistrales de contenido eminentemente teórico, con poca carga práctica, con hardware que aportaba escasa flexibilidad.

En aras de mejorar la flexibilidad del hardware de control, crearon soporte para Matlab/Simulink® para la familia de MCU STM32F1xx de ST Microelectronics® [26]. Los diferentes modelos de MCU de la familia STM32F1xx son compatibles a nivel de código fuente, configuración y uso de periféricos entre todos ellos, por lo que el código en lenguaje C escrito para uno de ellos, sirve para un modelo superior o inferior de MCU en la escala.

El soporte para SIMULINK® para la familia de MCU STM32F10x fue financiado como proyecto de investigación a nivel nacional por la Armada Real Tailandesa, entre los años 2008 y 2009, para proveer un medio RCP para el aprendizaje de cursos de teoría de control. El resultado es la toolbox para Simulink® "RapidSTM32", aunque este soporte de programación gráfica o diseño basado en modelos sólo cubre parte de la funcionalidad básica. Dicha toolbox presenta graves defectos de funcionamiento, estos defectos se ponen de manifiesto con los bloques para puertos de comunicación, éstos no pueden encontrarse por duplicado, haciendo imposible la utilización de varios sensores o sistemas conectados al mismo puerto, el código fuente generado tendrá que ser modificado a mano, por lo que se hace necesario un conocimiento profundo de la arquitectura y se pierden las ventajas del alto nivel de abstracción proporcionado por SIMULINK®.

[25] https://www.aimagin.com/about-us/ , Información sobre la compañía, Enero 2014

[26] http://www.st.com/web/en/catalog/mmc/FM141/SC1169/SS1031 , Información sobre el MCU STM32F1xx, Enero 2014

Estos defectos se deben en primera instancia a la gran dificultad que entraña adaptar el código fuente en lenguaje C, a las características funcionales de Matlab/Simulink Coder®, cómo se relaciona éste con las variables y configuraciones aportadas por los bloques de Simulink®, y las limitaciones en el lenguaje de generación de código fuente, lenguaje "Target Language Compiler" (TLC).

La figura 2.19 muestra la toolbox RapidSTM32 que sólo está soportada en sistemas Windows de 32 bits y cubren versiones de Matlab® desde la 2009a hasta 2011b, en posteriores versiones no funciona bien por incompatibilidades de las funciones de más alto nivel de la API de lenguaje C de Matlab®, que no fueron tenidas en cuenta, aspecto éste muy importante para lograr compatibilidad entre distintas versiones de Matlab®.

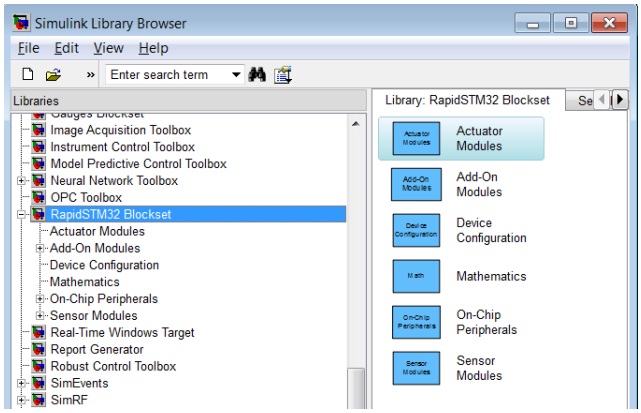


Figura 2.19: *Toolbox RapidSTM32*

Desde finales del año 2011, Aimagin® comenzó a trabajar en dotar al MCU STM32F4xx (aparecido a mediados del año 2011) de una toolbox para SIMULINK®, evitando los errores cometidos con la anterior toolbox. Esta toolbox al igual que la anterior, sólo dota al sistema de funcionalidad básica.

2.4.7. Matlab® 2013b low cost targets

Hasta ahora, en la extensa descripción realizada sobre los distintos sistemas RCP y sus toolboxes de programación gráfica o diseño basado en modelos, se ha constatado que Matlab® es una herramienta software omnipresente en la gran mayoría de los RCP.

Pese a que las características de generación de código de forma automática llevan presentes largo tiempo en Matlab/Simulink®, es a partir de la versión 2013 cuando estas funciones reciben por parte de MathWorks® mayor interés por su difusión y publicidad, enfocándolo como una parte más visible de su herramienta, denotando de esta forma el gran interés tanto pasado como presente por todas las características de prototipado rápido para control por parte de investigadores y desarrolladores de sistemas de control. No sólo se ha publicitado más, también se han añadido importantes propiedades que hasta ahora eran consideradas carencias, como el poder indicar a Simulink Coder® cual es el más complejo tipo de dato que puede estar presente en el código generado.

Otra propiedad añadida en la versión 2013 es la de poder reemplazar el código fuente generado por lo bloques oficiales de la toolbox "DSP System Toolbox" por código fuente específico para la arquitectura hardware que estés utilizando, en el caso de una arquitectura basada en procesadores ARM, se puede utilizar el código de la librería "ARM CMSIS DSP functions". Obviamente esta misma propiedad es aplicable a otras arquitecturas.

En esta versión 2013, se ha añadido la posibilidad de instalar en la librería de Simulink® algunos objetivos hardware tales como MCU o computadores de tarjeta única (Single Board Computer, SBC) de bajo coste como Raspberry Pi o BeagleBoard, más adelante se tratará más en detalle las características de estos sistemas. El soporte ofrecido una vez instalado, corresponde con parte de las funcionalidades básicas, conversores A/D, I/O digital, UART... y en los sistemas más potentes capacidades para conectar una webcam y adquirir imágenes para su tratamiento posterior con la "Computer Vision System" toolbox.

De forma general, Matlab 2013 ofrece soporte básico o muy básico de Simulink® para algunos MCU de Analog Devices®, Texas Instruments® C2000, STM32F4xx, Arduino y para SBC Raspberry Pi, BeagleBoard y PandaBoard. Como ya se ha comentado, la funcionalidad proporcionada no cubre completamente el nivel básico, y dependiendo del sistema, habrá más o menos bloques disponibles, por ejemplo los de Texas Instruments son los más completos, disponiendo de bloques para bus

CAN, SPI, I2C y otros desarrollados por la misma Texas Instruments® para control de motores brushless, como transformación Clarke. Pero en ningún caso cubren la totalidad de los periféricos y sus opciones, y tampoco suelen alcanzar peldaños de las funcionalidades avanzadas, a excepción de los proporcionados gratuitamente por Texas Instruments®.

Siendo estas toolboxes una nueva característica de la versión 2013, sólo son compatibles con ésta. Son toolboxes que pueden modificarse y ampliarse, al menos en las que se ha probado a instalar.

2.4.8. STM32 Matlab Target

En Abril de 2013, la compañía fabricante del MCU STM32F4xx, ST Microelectronics, publicó un blockset para programación MBD o programación gráfica sobre Matlab/Simulink® versión 2013b. La última versión de la toolbox corresponde con la fecha del 4 de Octubre de 2013, denominada "STM32 MAT Target 3.0" [27].

Este añadido software para Simulink® provee de algunas funcionalidades básicas, tales como poder utilizar los conversores A/D, salidas PWM (sólo para 6 timers de los 12 disponibles, los que admiten fuente de interrupción IRQ), I/O digital, entrada para encoder relativo y comunicaciones por UART. También permite utilizar algunas fuentes de IRQ, como la IRQ Externa o la IRQ provocada por lecturas del periférico A/D. Cabe resaltar que las posibilidades del A/D están ampliamente cubiertas, en profundidad.

La figura 2.20 muestra la toolbox instalada y los periféricos soportados:

Respecto a lo que podríamos llamar "manejabilidad", facilidad de manejo, lo que los ingleses llamarían capacidades "user friendly" de esta toolbox, se ha de destacar que la configuración de los bloques presupone que el usuario de la toolbox posee amplios conocimientos sobre la arquitectura y capacidades de los periféricos del STM32F4, por lo que resultaría muy difícil de utilizar por un público multidisciplinar.

Como ejemplo de la dificultad en su utilización, se propone un ejemplo sencillo, en el que se desea realizar una lectura de dos datos analógicos a través de dos puertos de entrada, mediante el conversor A/D. Lo primero que se ha de hacer es configurar el conversor A/D, la información proporcionada por el panel de configuración del propio bloque no es autoexplicativa, dificultando el proceso de

[27] http://www.st.com/web/en/catalog/tools/PF258513 , Febrero 2014

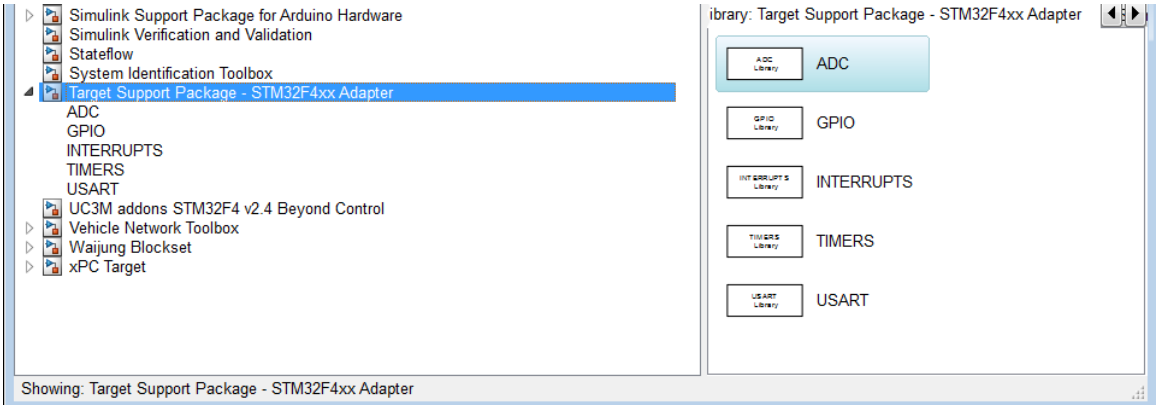


Figura 2.20: *Toolbox para MCU STM32F4xx de ST Microelectronics®*

configuración. También contrariamente a lo que sería lógico suponer, se ha de configurar con el bloque de inicialización de puertos qué pines del MCU van a ser entrada analógica y por último para poder leer el dato analógico, se ha de utilizar el bloque de lectura A/D, se exige conocimiento previo respecto a qué canal del conversor A/D se corresponden los pines que se han configurado anteriormente, el bloque de lectura A/D no indica en su panel de configuración qué pines corresponden a determinados canales. Lo conveniente es que la asignación de los canales A/D se realice automáticamente dependiendo de qué puertos de entrada ha seleccionado el usuario, evitándole la necesidad de un conocimiento profundo de la arquitectura.

Esta toolbox requiere por parte del usuario una gran cantidad de conocimientos sobre el funcionamiento interno de los MCU soportados, haciendo que no sea indicada para ser utilizada por un grupo multidisciplinar; se pierde gran parte de las ventajas de utilizar un lenguaje de alto nivel de abstracción. La figura 2.21 muestra de forma soslayada la cantidad de parámetros y máscaras de configuración a utilizar con esta toolbox para habilitar la función de leer un par de datos analógicos:

Como nota adicional respecto a la toolbox de ST Microelectronics®, el bloque gráfico que permite configurar IRQ contiene el error de permitir configurar más de dos salidas como fuente de IRQ de forma simultánea, si se configura de esta manera, Matlab® dejará de funcionar pues éste no soporta más de una salida como fuente de IRQ por bloque (es una limitación de las toolboxes Simulink Coder® y Embedded Coder® que se encuentra debidamente documentada), aunque el código interno de la función compilada (de la API en C de Matlab®) esté bien programada. Como posible solución, desde esta tesis se propone adaptar el código de la función para que la segunda salida y sucesivas recurrieran al uso de llamadas a

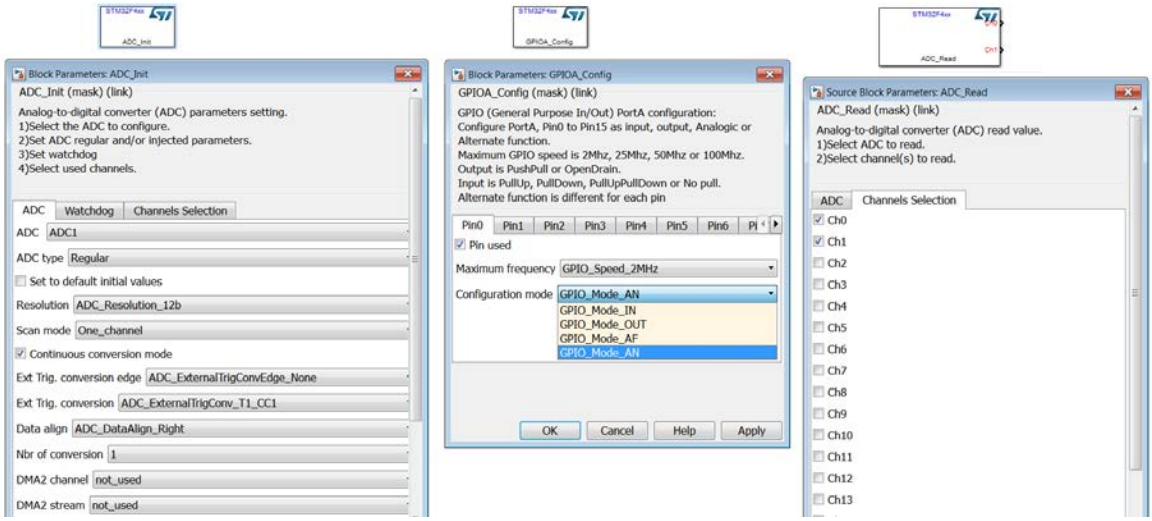


Figura 2.21: *Pasos para configurar dos entradas del conversor A/D en la toolbox para MCU STM32F4xx de ST Microelectronics®*

funciones asíncronas en lugar de producir llamadas a IRQ; debido a que a nivel de código fuente ambos conceptos presentan comportamientos idénticos.

2.4.9. Conclusiones Toolboxes para MCU

Todas las toolboxes que proporcionan soporte de programación gráfica para unos u otros MCU, ya sean gratuitas o de pago, tienen algo en común, todas ellas proporcionan acceso sólo a funcionalidades básicas. Las funcionalidades avanzadas quedan relegadas a ser integradas de forma manual, modificando el código fuente generado con estas toolboxes, perdiéndose las ventajas de emplear un alto nivel de abstracción en su programación.

Pese a esta aparente desventaja, todas estas toolboxes para MCU no dejan de ser buenos ejemplos de las posibilidades de integrar la programación de un MCU en un entorno de desarrollo basado en modelos como es SIMULINK®. También dadas las características actuales de los MCU (años 2013 - 2014), la flexibilidad y gran variedad de I/O que integran y su buena capacidad computacional, hacen que el utilizar alguna de estas toolboxes como base de un sistema RCP más avanzado se presente como una alternativa viable.

2.5. Sistemas RCP con origen no comercial

En los apartados anteriores se han estudiado herramientas software y elementos hardware que componen sistemas RCP o pueden ser utilizados con ese fin. En su práctica totalidad se han tratado de sistemas comerciales, presentando mayor o menor difusión en unos u otros campos de aplicación, generalmente dependiendo de los sectores a los que se enfoca cada sistema.

Ahora, en este presente apartado, se estudiarán las herramientas software utilizadas como base de desarrollo y programación de alto nivel de abstracción para sistemas RCP, cuyo origen se encuentre en centros de investigación. Estas herramientas software originalmente no fueron concebidas para dar cabida a sistemas RCP, son propiedades que fueron añadidas como fruto de la evolución lógica. Esto se explica debido a que todo programa informático destinado a la realización de simulaciones y modelados de sistemas físicos electromecánicos, acaba incorporando la posibilidad de simular también los algoritmos de control que los gobiernan. El siguiente paso es poder generar el código fuente del algoritmo planteado y probarlo en un controlador digital para controlar el sistema electromecánico real.

Las herramientas de desarrollo estudiadas en este apartado son ScicosLab, Scilab y Modelica Standar Library Blockset.

2.5.1. ScicosLab®

ScicosLab® es un software para modelado y simulación multidominio, y cálculo de carácter científico gratuito, desarrollado bajo el ámparo del equipo de investigación Meta2/Metalau en INRIA y ENPC.

INRIA "Institut National de Recherche en Informatique et en Automatique" [28] , cuya traducción literal del francés es Instituto Nacional de Investigación en Informática y Automática, es un centro de investigación francés especializado en Ciencias de la Computación, teoría de control y matemáticas aplicadas, creado en 1967. El ENPC "École nationale des ponts et chaussées" [29] cuya traducción es Escuela Nacional de Puentes y Calzadas, fundada en 1747.

[28] www.inria.fr , Febrero 2014
[29] www.enpc.fr , Febrero 2014

Los equipos de investigación responsables, denominados Metalau y Meta2 [30] han sido los encargados de ir desarrollando las distintas capacidades del software Scilab®, a partir de los contenidos generados por numerosas aportaciones en forma de tesis doctorales, puede resumirse de esta forma: cada funcionalidad de Scilab® es fruto de una tesis doctoral. Los objetivos de este equipo de investigación son el desarrollo de metodologías computacionales para llevar a cabo simulaciones, modelados matemáticos, algoritmos de control, integración en hardware, detección de fallos... todo aquello relacionado con métodos, algoritmos y software para ingeniería de control automática.

Pese a que se ha comenzado nombrando un software denominado ScicosLab® y se ha continuado nombrándolo Scilab®; no es una incongruencia, ni representa una confusión en la redacción, se debe conocer un poco la historia de Scilab® para entender este enunciado.

El comienzo del desarrollo de Scilab® tiene lugar en 1982, tras haber mantenido contacto con las versiones de dominio público de Matlab®. En el INRIA a este nuevo software similar en concepto a Matlab® se le denominó Basile y se comenzó a comercializar en 1984, de la mano de Simulog®, vendiéndose hasta finales de la década de los 80 . A partir de 1990 el equipo de investigación Meta2 en INRIA y el ENPC decidieron crear una versión gratuita de Basile, pasando a llamarla Scilab®. Fueron los distintos estudiantes de doctorado del equipo Meta2 y Metalau junto con sus correspondientes directores de proyecto/tesis quienes crearon el software Scilab y todas sus toolboxes. Los directores y tutores principales han sido y continúan siendo Jean-Pierre Quadrat, S. Steer, F. Delebecque y J. Ph. Chancelier. Aunque debe mencionarse que Francois Delebecque lleva trabajando en la creación de Scilab® desde 1982 y es el responsable de multitud de propiedades y capacidades del actual ScicosLab®, como la compatibilidad con las funciones .mex de Matlab®, la toolbox "systems and control toolbox"...

Desde el año 2003, el relevo en el desarrollo de Scilab fué tomado por el consorcio Scilab, "Scilab Consortium". El INRIA y el ENPC decidieron mantener el desarrollo de su propia herramienta software (programada en lenguaje C) al que tuvieron que rebautizar con el nombre de ScicosLab para evitar toda confusión con el actual Scilab® (programado en Java) , originado en la escisión del año 2003 .

Scicos® representa el entorno de modelado y programación en base a gráficos de ScicosLab®. Es su lenguaje de más alto nivel de abstracción, desarrollado en su totalidad mediante las aportaciones realizadas por numerosas tesis doctorales:

[30] http://www.inria.fr/en/teams/metalau , Febrero 2014

una tesis doctoral para gestionar la generación del código fuente, otra tesis doctoral para integrar las funciones de unas u otras toolboxes de ScicosLab® para poder utilizarlas bajo Scicos®... y un largo etcétera de tesis doctorales y sus aportaciones. Desde el año 2008 el desarrollo de la herramienta Scicos® cuenta con su propio equipo de investigación. La figura 2.22 muestra el aspecto visual que presenta el entorno Scicos®, como puede apreciarse resulta muy similar al resto de lenguajes de programación basados en gráficos vistos en otros programas de modelado, simulación y diseño de algoritmos de control.

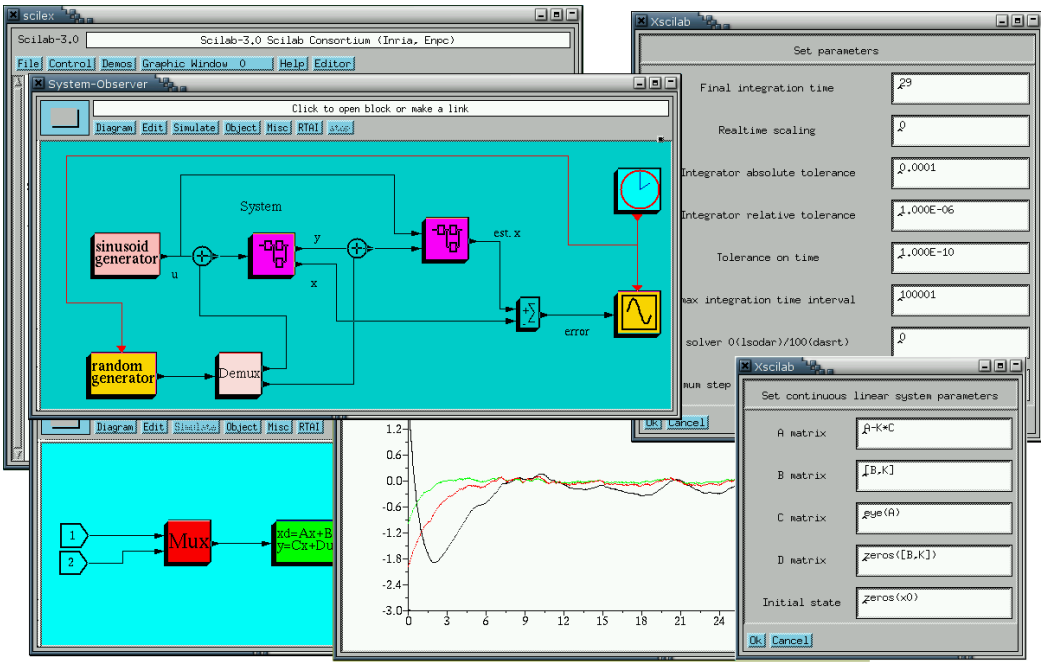


Figura 2.22: *Aspecto visual del entorno Scicos®*

Respecto al uso de ScicosLab® como base de programación para sistemas RCP; originalmente las capacidades de generación de código fuente en lenguaje C están limitadas a generar código en forma de funciones, en lugar de crear el código de un programa independiente. Se obtiene el código fuente en lenguaje C de una función que recibe como argumentos las entradas a los bloques gráficos de Scicos® y como salida un puntero a una estructura donde se guardan los resultados de cada una de la salidas del diagrama de bloques diseñado en Scicos®. Para poder generar el código fuente de programas completos, con su inicialización de periféricos y del reloj para el planificador temporal, gestión de interrupciones y demás, se hace

necesaria la instalación de una ampliación en forma de toolbox gratuita proporcionada por la compañía Evidence® [31] bajo su marca de software Erika Enterprise® [32]. Esta toolbox para ScicosLab® denominada Scicos-Flex [33] modifica el procedimiento general de generación de código, de forma que se genere el código en lenguaje C de un programa completo independiente en lugar de una única función. Esta toolbox provee de soporte para los objetivos hardware proporcionados por Evidence® y para los MCU dsPIC16® de Microchip®.

La instalación de la toolbox Scicos-Flex conlleva multitud de pasos, enunciados a continuación:

1. Instalar Cygwin, un conjunto de herramientas desarrollado por Cygnus® (adquirida por Red Hat®), que sirven para otorgar un comportamiento similar a sistemas Unix a los sistemas operativos Windows®.

2. Instalar Microsoft Visual C++ 2008 - 2010 , versiones más actuales no son válidas pues requieren profundas modificaciones manuales a la toolbox ScicosFlex de forma previa a su instalación.

3. Instalar ScicosLab 4.4.1

4. Instalar el entorno de desarrollo integrado (IDE) de Microchip®, denominado MPLAB.

5. Instalar el compilador de lenguaje C de Microchip®, de los denominados C30. Actualmente se trata de compiladores descontinuados y por tanto son muy difíciles de encontrar. Ya no están disponibles en la web oficial y los actuales compiladores XC de Michochip® no son compatibles con ScicosLab® y Scicos-Flex. Afortunadamente, en las pruebas y evaluaciones de Scicos-Flex realizadas en esta tesis doctoral se pudo localizar una versión obsoleta del compilador Microchip® C30.

6. Instalar la última versión disponible, en Febrero de 2014, de ScicosLab Pack.

7. Ejecutar el script de intalación "install.sce" desde la consola de comandos de ScicosLab. Este script se encuentra dentro del directorio donde se localicen los ficheros de la toolbox ScicosFlex.

[31] http://www.evidence.eu.com/en/company/about-us.html , Febrero 2014
[32] http://erika.tuxfamily.org/drupal/ , Febrero 2014
[33] http://erika.tuxfamily.org/drupal/scilabscicos.html , Febrero 2014

Durante la instalación de la toolbox ScicosFlex se van comprobando los requisitos previos, en el caso de que alguno de los programas requeridos no estuviera instalado o no fuese compatible, el script de instalación lo indicaría. Los mensajes de error que se pudieron observar correspondieron con las versiones de Microsoft Visual C++ y con el no reconocimiento de los compiladores XC de Microchip®. La figura 2.23 muestra los mensajes en pantalla en ScicosLab una vez instalada correctamente la toolbox ScicosFlex.

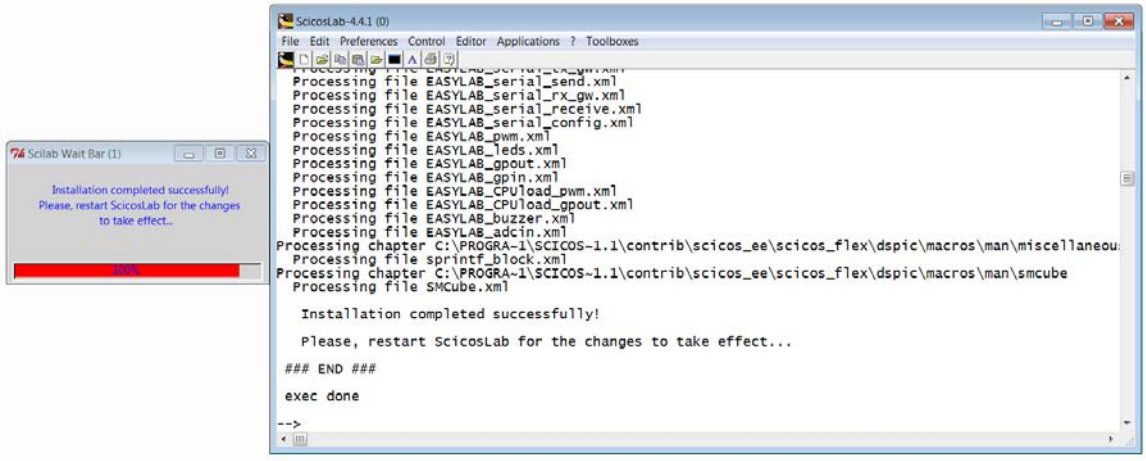


Figura 2.23: *Mensajes mostrados en la consola de comandos de ScicosLab® tras intalar ScicosFlex satisfactoriamente*

Una vez instalada la toolbox ScicosFlex, se procedió a realizar algunas pruebas a fin de comprender mejor la herramienta y poder evaluar tanto su funcionamiento como sus capacidades. Tras la instalación de la toolbox, aparece la opción de elegir dos modelos para la generación de código fuente, el predeterminado de ScicosLab que genera todo el código como una única función en lenguaje C, o el proporcionado por ScicosFlex orientado a MCU dsPIC16®, para los modelos de MCU utilizados en los sistemas RCP vendidos por la compañía Evidence® o para cualquier dsPIC16®. Existen algunos modelos de ejemplo ya incluidos con ScicosLab®, preparados para un MCU dsPIC16® como objetivo hardware.

Tras realizar diferentes pruebas y ensayos, y tras haber estudiado y entendido los mecanismos de generación de código fuente de ScicosLab® y ScicosFlex se encontraron algunos inconvenientes que no están explicados de forma explícita en su documentación. Cada toolbox utiliza sus propios bloques de programación gráfica que a su vez están adaptados para generar código fuente siguiendo uno u otro esquema. Los bloques de ScicosLab® no son compatibles a nivel de generación de

código con los bloques de ScicosFlex y viceversa, lo que plantea que para otorgar soporte a cualquier otro objetivo hardware no sólo se han de crear los bloques que permitan configurar y utilizar los periféricos de I/O, así como la generación de la función principal y configuración de los relojes e interrupciones del sistema; sino que además habría que adaptar individualmente cada uno de los bloques de ScicosLab, lo que representaría una gran carga de trabajo. La figura 2.24 ilustra la manera en que se han implementado los bloques gráficos, se puede observar la duplicidad en los bloques de programación gráfica para cada apartado dependiendo del objetivo hardware al que están destinados.

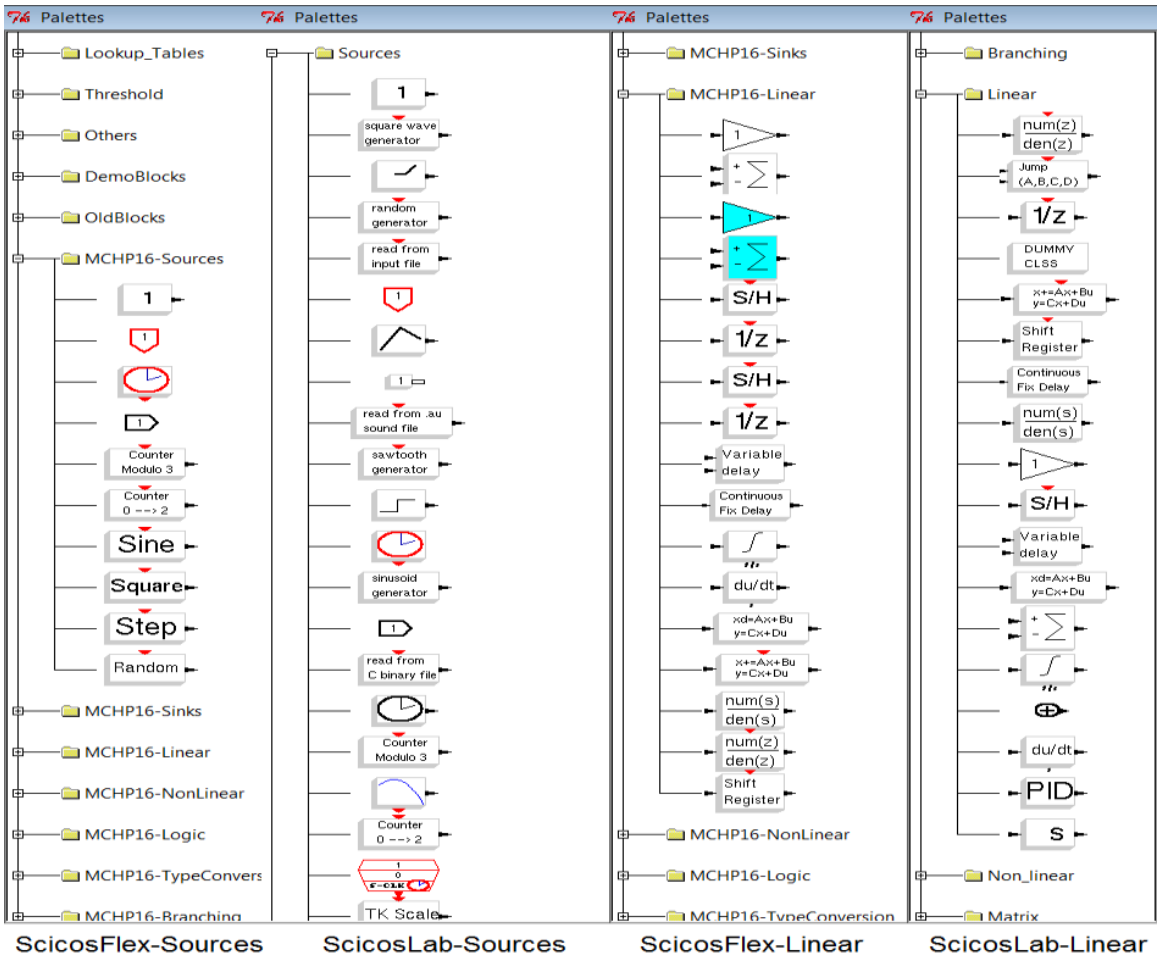


Figura 2.24: *Duplicidad en los bloques de programación gráfica, las mismas funcionalidades tanto para ScicosFlex como para ScicosLab, incompatibles entre sí*

Otro de los inconvenientes observados, tanto en el código fuente en lenguaje C generado como en el uso de ScicosLab®, y que se encuentra documentado oficialmente, es la ausencia de soporte para tipo de datos en coma flotante de precisión simple, en lenguaje C sería tipo "float". Es un gran inconveniente teniendo en cuenta que los MCU con soporte hardware para tipo de datos flotante sólo soportan precisión simple, mientras que la mayoría de los MCU siguen trabajando sólo con números enteros. Los tipos de datos para enteros soportados en ScicosLab® son enteros con o sin signo de 32 bits, tipos de datos de 16 y 8 bits no están soportados. Tampoco están soportados tipos de datos flotantes en punto fijo.

Más inconvenientes detectados, y oficialmente documentados, de la toolbox ScicosFlex® es la limitación en la gestión de los tiempos de muestreo. En el código generado por defecto bajo ScicosLab, la evolución temporal del programa es responsabilidad del usuario, que tendría que ejecutar la función en lenguaje C generada bajo un entorno gobernado por un sistema operativo en tiempo real (RTOS). Respecto a ScicosFlex, el tiempo base que se configura para los MCU dsPIC16 es siempre de 1 ms. , esta es la resolución temporal, no se adapta dinámicamente. Ni ScicosFlex ni ScicosLab proveen de los mecanismos para lograr la adaptación dinámica en la configuración inicial del temporizador del sistema (comúnmente conocido por SysTick Timer). Por tanto todos los tiempos de muestreo de los algoritmos de control diseñados en Scicos se redondean a múltiplos del tiempo base, siempre serán múltiplos de 1 ms. También el período de las señales PWM para MCU dsPIC está fijado siempre en 1 ms. Para cambiar esta resolución temporal tanto del SysTick Timer como del generador PWM han de realizarse modificaciones manualmente.

Una ventaja que si se ha encontrado en ScicosFlex, es la integración del RTOS gratuito y con certificación para utilizarlo tanto en industria como en automoción de Erika Enterprise®, de hecho desde Scicos y mediante la toolbox ScicosFlex puede organizarse la ejecución de diferentes tareas, todo programando desde el más alto nivel de abstracción gracias al lenguaje gráfico, esta es una funcionalidad avanzada que se encuentra presente en muy pocas toolboxes comerciales para MCU, como ya se ha analizado anteriormente.

Como ventaja adicional, el proceso de creación de nuevos bloques gráficos que generen código fuente y la creación de soporte para objetivos hardware adicionales está documentada y existen ejemplos de cómo hacerlo. El proceso de creación de nuevos bloques y soporte hardware adicional está basado en lenguaje .xml, aunque las etapas y nombres de las funciones a utilizar son muy similares a los pasos necesarios en Matlab/Simulink®, con la salvedad de que en ScicosLab es muy difícil

crear bloques gráficos cuya generación de código fuente sea compatible con multitud de objetivos hardware, debido a las particularidades del lenguaje de script que organiza la generación del código.

Por otro lado, más allá de las herramientas y toolboxes de carácter gratuito para ScicosLab®, y tras conocer las limitaciones de la toolbox gratuita ScicosFlex® cabe comentar otra toolbox orientada a la generación de código fuente de origen comercial, destinada a su uso bajo ScicosLab®. Esta toolbox se denomina E4Coder® de la compañía Evidence® distribuida y comercializada bajo su firma de software Erika Enterprise®.

E4Coder® [34], producto comercial, palia las limitaciones de ScicosLab y ScicosFlex en lo que a generación de código fuente en lenguaje C y soporte para tipos de datos se refiere, también soluciona el anterior problema de no poder adaptar la resolución temporal del Timer del Sistema o SysTick Timer. La generación de código puede utilizar el RTOS de Erika Enterprise® o bien no utilizarlo en absoluto, a elección del usuario. También se incluye la documentación necesaria para adaptar las funcionalidades de E4Coder® a cualquier MCU.

Los precios de E4Coder, cuya licencia es por cada ordenador en que se encuentre instalado (en inglés "per seat license" o "engineering seat based license") sin límite de tiempo con un año de actualizaciones, son los siguientes [35] :

- Licencia por computador para empresas, 3000€
- 1 año de actualizaciones para empresas, 900€ por computador
- Licencia por computador para entornos académicos, 500€
- 1 año de actualizaciones para entornos académicos, 150€ por computador

[34] http://www.e4coder.com/ , Febrero 2014
[35] http://www.e4coder.com/content/e4coder-prices , Febrero 2014

2.5.2. Scilab®

Scilab® es el software basado en MBD originado en la escisión del año 2003 acontecida en el seno del desarrollo de ScicosLab®, se cambió el nombre para poder diferenciar ambos proyectos. Al igual que su antecesor es un software destinado al modelado y simulación de sistemas dinámicos, análisis de datos, diseño de controladores...

La versión actual (a fecha de Julio de 2014) es la 5.5.0 . El aspecto gráfico de Scilab® es bastante menos arcaico que el de ScicosLab®® y la cantidad de toolboxes disponibles mucho mayor, debido en gran medida a que Scilab® está desarrollado por un consorcio cuyos integrantes son más numerosos que los que han quedado desarrollando ScicosLab®. La ventana principal de la aplicación, mostrada en la figura 2.25, es bastante similar a la presentada por MATLAB®, consola de comandos en el centro, espacio de trabajo con variables a la derecha y contenido del directorio a la izquierda.

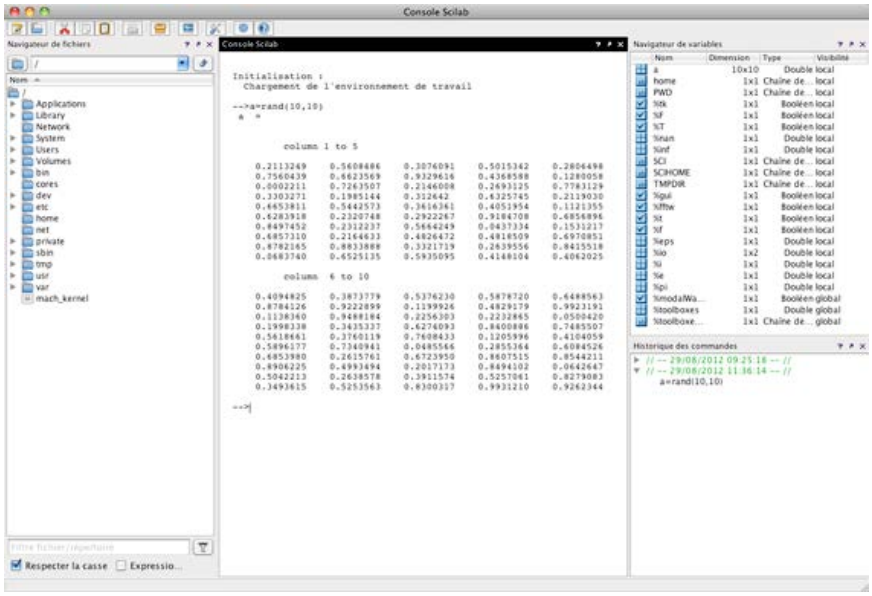


Figura 2.25: *Ventana principal de la aplicación Scilab®*

El entorno de desarrollo basado en lenguaje gráfico se denomina Xcos y desde él se puede acceder a la lista de toolboxes disponibles y descargarlas e instalarlas fácilmente, mediante el denominado "AuTomatic mOdules Management" (ATOMS) de Scilab®. El aspecto gráfico de Xcos se muestra en la figura 2.26.

Scilab® adolece de los mismos problemas que ScicosLab® respecto a su utilización como medio de programación y diseño de controladores para un sistema RCP.

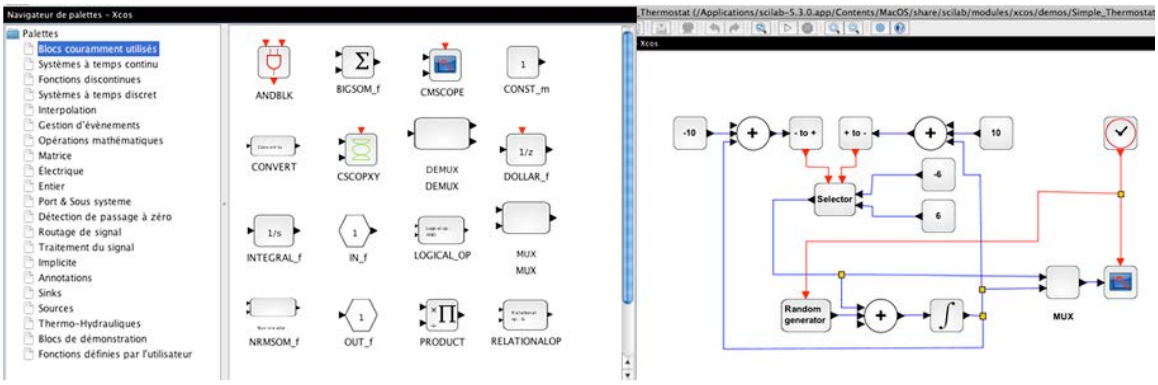


Figura 2.26: *Entorno de programación basado en gráficos Xcos. A la izquierda lista de bloques gráficos. A la derecha un ejemplo de modelo en tiempo-discreto en Xcos.*

El código fuente en lenguaje C que se genera a partir de un modelo desarrollado en Xcos es una única función sin gestión del tiempo de muestreo ni de I/O. Instalando la toolbox OpenRTDynamics mediante ATOMS se genera además una función que implementa un planificador temporal, pero sigue sin generarse el programa completo con su configuración de interrupción temporal y de I/O, necesario para un sistema RCP.
La solución a estos problemas pasa por las entidades de carácter comercial Erika Enterprise® o Scilab Enterprise®[36], en el caso de los productos gratuitos ofrecidos por Erika Enterprise® no han podido instalarse sobre la versión actual de Scilab®.

Aunque Scilab soporta una gran variedad de tipos de datos enteros, tanto con signo como sin él, sigue sin soportar tipos de datos flotantes de precisión simple o de punto fijo; tipos de datos imprescindibles para la implementación de los modelos gráficos en un sistema RCP.

[36]http://www.scilab-enterprises.com/ , Julio 2014

2.5.3. Kernel Mode based systems. MATLAB/SIMULINK® external mode

Existe un procedimiento por el que pueden ejecutarse programas bajo el paradigma de Tiempo-Real en el sistema operativo Windows® o Linux. Técnicamente estos programas no se ejecutarían sobre Windows®, sino debajo de él. Para entender este modo de funcionamiento es necesario antes explicar cómo funciona el núcleo de ejecución de los sistemas operativos Windows®, en Linux es similar.

Windows® opera bajo 4 modos de privilegio, las restricciones respecto a los privilegios del sistema obtienen su definición respecto de a qué recursos del hardware se permite el acceso para un programa determinado. La figura 2.27 ilustra de forma gráfica los 4 modos de acceso a los recursos hardware que proporciona un sistema operativo basado en Windows®.

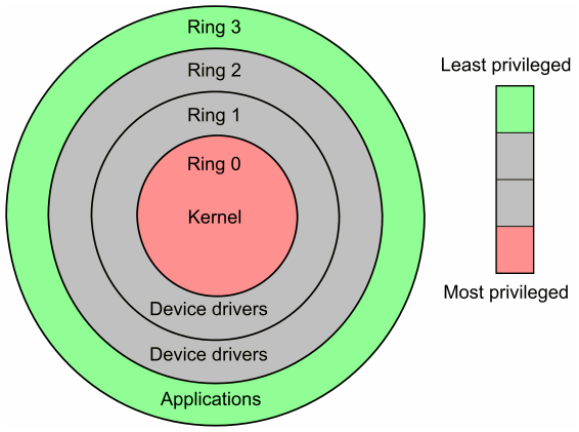


Figura 2.27: *Modos de acceso a los recursos hardware implementados en un sistema operativo Windows®*

En el modo de funcionamiento normal, denominado "modo de usuario" (*user mode*), los programas funcionan bajo la supervisión del núcleo de ejecución de Windows®, en el anillo de privilegios número 3. Toda solicitud de acceso a los dispositivos hardware, incluyendo periféricos y memoria RAM de un programa ejecutándose en modo usuario, es revisada por Windows® para comprobar si es correcta y posible. En caso contrario, el núcleo de ejecución de Windows® excluye el programa causante del error de ser ejecutado en la próxima iteración y se mostrará por pantalla el acostumbrado mensaje respecto a que dicho programa ha provocado un fallo... Windows® liberará la memoria y recursos hardware utilizados por el programa, ya que conoce cuáles son porque el programa se los ha estado solicitando constantemente. Bajo este paradigma no es posible la ejecución

en Tiempo-Real de un programa, no se puede garantizar que cada iteración de un programa se ejecute cada determinado intervalo de tiempo.

Los drivers, o manejadores de los dispositivos hardware operan entre los anillos de privilegios 2 a 0. Por motivos de rendimiento los drivers del adaptador de vídeo y del adaptador de audio funcionan en el anillo de privilegios 0, a excepción del sistema operativo Windows Vista®, donde se les obliga a funcionar en modos de privilegios mayores, de ahí el bajo rendimiento que siempre se ha acusado a Windows Vista® . Este es el motivo principal por el que un ordenador pueda colgarse sin mensaje de error y quede con la pantalla congelada, mostrando la última imagen contenida en el "framebuffer", o memoria de vídeo dedicada a la representación en pantalla. En esos casos, el driver de vídeo o el de audio han fallado en algo; situación que puede verse agravada en aplicaciones multimedia intensivas como programas de edición CAD o videojuegos.

Respecto a los drivers de los dispositivos hardware y su ubicación en los anillos de privilegio, cabe destacar que los drivers son accesibles por programas que se ejecuten en el mismo anillo de privilegio o en uno superior; pero un programa que se ejecute en un anillo de privilegio inferior no podrá acceder a los recursos habilitados por un driver ubicado en un nivel superior.

Esta característica de Windows® hace que sea posible ejecutar un programa en el anillo de privilegio 0, denominado Kernel, pudiéndose acceder al Timer del sistema y reprogramarse la frecuencia de las interrupciones temporales, así como ejecutar un programa cada vez que acontezca una interrupción temporal. Esta interrupción temporal permite ejecutar un programa bajo el paradigma de Tiempo-Real siempre que el código esté basado solamente en tiempo-discreto. Esta propiedad es explotada por la toolbox de MATLAB/SIMULINK® Real-Time Windows Target®, permitiendo ejecución monotarea soportando distintos tiempos de muestreo con planificador temporal incluído. Resulta necesario que el programa a ejecutar en modo Kernel termine su ejecución antes de la llegada de la próxima interrupción temporal, para dejar tiempo a Windows® de administrarse, en caso contrario el sistema se colgará, con el único síntoma de que la pantalla se queda congelada. Dependiendo de la complejidad del programa a ejecutar en Tiempo-Real, será más o menos fácil bloquear completamente el sistema, programas que tengan que realizar grandes análisis de datos no son aconsejables, pues es difícil que sean puramente deterministas.

En el modo Kernel se pierden las capacidades de I/O otorgadas por los drivers de los dispositivos. Tan sólo drivers para hardware de I/O muy especializados

otorgan soporte para el modo Kernel. De forma general se puede acceder al puerto serie RS-232 si la placa base del ordenador posee alguno, pues su uso está normalizado, y se accede a través del BIOS ("Basic Input Output"). También es posible utilizar el puerto paralelo.

Como ventajas de este modo de funcionamiento se encuentra el poder ejecutar un programa bajo el paradigma de Tiempo-Real en Windows®. Como desventajas se presenta la imposibilidad de realizar multitarea en Tiempo-Real, tampoco se puede implementar código orientado a su ejecución en tiempo-continuo y se pierde la práctica totalidad de las capacidades de I/O salvo el puerto RS-232 real y el puerto paralelo. También son desventajas la facilidad de bloquear el sistema al operar únicamente en este modo de funcionamiento y el tener que compartir el modo Kernel con los drivers de vídeo y audio, lo que impedirá establecer un determinismo absoluto en la ejecución del programa de control.

Existen sistemas RCP utilizados en centros de investigación y educativos que hacen uso del modo Kernel y de la toolbox Real-Time Windows Target® para la ejecución en Tiempo-Real monotarea de programas de control diseñados en SIMULINK® [50, 51, 52, 53], que utilizan el ordenador para ejecutar el programa de control y se comunican mediante RS-232 con un microcontrolador que hace las funciones de tarjeta de adquisición de datos. Se trata de una metodología de trabajo bastante difundida en centros de investigación, puesto que disponer de un sistema RCP comercial resulta por lo general demasiado caro de adquirir y mantener; pero el disponer de un ordenador con puerto RS-232 y programar un microcontrolador para adquirir datos con sus periféricos y enviar estos por RS-232 es mucho más barato. Se pueden crear drivers personalizados para dispositivos de I/O, estos drivers no deben hacerse siguiendo las normas de Windows®, sino que son más parecidos a cómo se hacían los drivers para el sistema operativo MS-DOS®.

En esta sección se ha descrito una metodología de trabajo en Tiempo-Real basada en el modo Kernel de Windows® y como es explotada por la toolbox de MATLAB/SIMULINK® Real-Time Windows Target®. Este principio de funcionamiento es el empleado por el denominado modo externo de SIMULINK®. Además de los inconvenientes ya descritos, se localizan otros derivados de las limitaciones de algunas toolboxes de SIMULINK® de poder operar en modo externo, los visualizadores gráficos de SIMMECHANICS®, y las toolboxes SIMRF® para modelado de sistemas eléctricos y SIMHIDRAULICS® entre otras, no funcionan cuando SIMULINK® se comunica con un programa que se ejecuta en modo externo, por lo que se limitan las capacidades de modelado y simulación al utilizar esta metodología.

2.5.4. Modelica®

Modelica® es un lenguaje formal basado en ecuaciones para describir sistemas mecánicos, eléctricos, neumáticos... mecatrónicos en general. Aparte de ser un lenguaje textual, la versión actual de Modelica®, versión 3.3, incluye un entorno de simulación y modelado basado en la utilización de lenguajes gráficos, al estilo de SIMULINK®.

Modelica® está desarrollado al amparo de una organización sin ánimo de lucro denominada Modelica Association. Fundada en 1996 cuenta con miembros y entidades procedentes de Europa, Estados Unidos, Canadá y Asia. Además del uso gratuito, existen versiones comerciales de Modelica® contenidas en MapleSIM®, Wolfram Modeler®, Dymola® que incluyen soporte adicional al gratuito.

Respecto al uso de Modelica® y su entorno de modelado basado en gráficos como base para un sistema RCP es posible, pues Modelica® incluye la funcionalidad de convertir a código en lenguaje C los modelos basados en gráficos. La metodología de trabajo con Modelica se muestra en la figura 2.28.

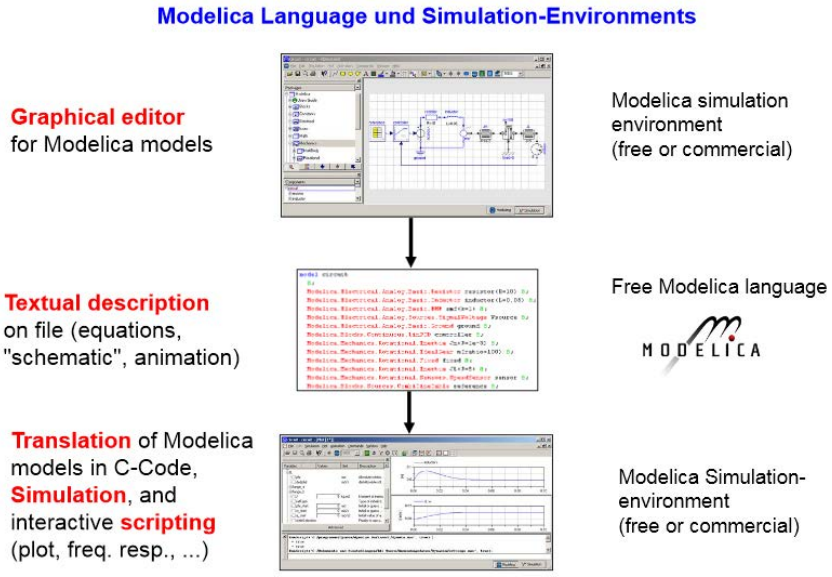


Figura 2.28: *Convirtiendo un modelo gráfico en Modelica® a código fuente en lenguaje C. Fuente: Presentación "Modelica Overview", www.modelica.org.*

La generación automática del código en lenguaje C da lugar a una función que recrea el comportamiento del modelo descrito, pero sin ofrecer respaldo a tiempos de muestreo ni planificadores temporales. Esta circunstancia cambia al utilizar

la librería gratuita "Embedded Systems", con origen en el Centro de Robótica y Mecatrónica del DLR[37]. Utilizando el entorno gráfico de Modelica® y la librería Embedded Systems el DLR diseña los controladores para sus robots. La figura 2.29 ilustra el concepto de la librería Embedded Systems para Modelica®; los bloques gráficos que permiten utilizar los recursos I/O del controlador necesitan ser reconfigurados para adecuarse al hardware que quieras utilizar, así como reescribir todo el código C que genera cada bloque.

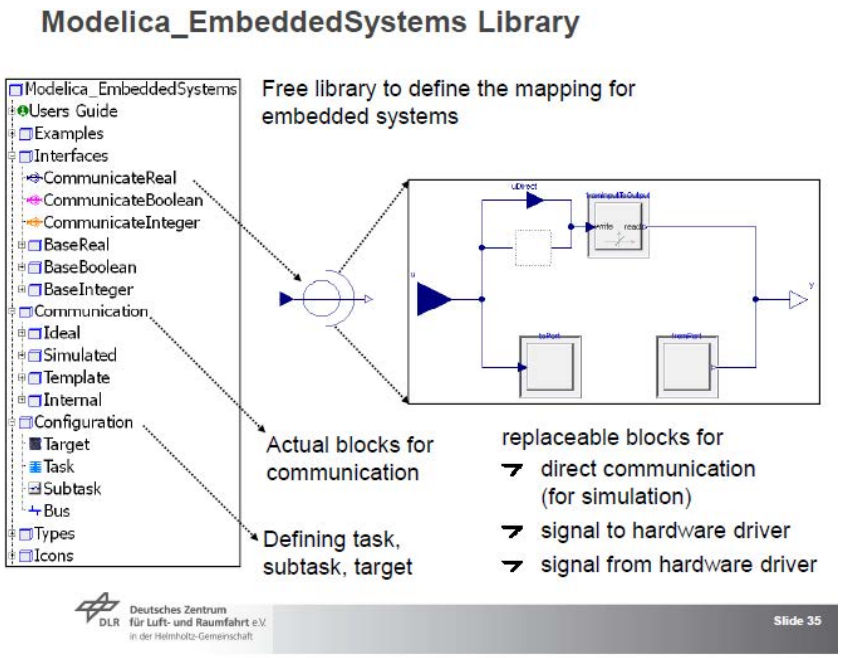


Figura 2.29: *Concepto de la librería para sistema embebidos. Fuente: Presentación "Modelica Embedded Library", German Aerospace Center (DLR)*

Una grata particularidad de este sistema de Prototipado Rápido para Control es que permite generar el código fuente en lenguaje C no sólo para un controlador, sino para múltiples controladores al mismo tiempo, e incluso estos controladores pueden ser diferentes entre sí, siempre que se cuente con una versión personalizada y reconfigurada de la librería Embedded Systems para cada hardware objetivo. Modelica® y Embedded Systems Library conforman la base software de un sistema RCP que permite administrar distintas tareas bajo un sistema operativo en Tiempo-Real, crearlas y detenerlas; son características propias de sistemas RCP avanzados. La configuración de los periféricos de I/O se realiza mediante paneles de configuración, la figura 2.31 muestra un ejemplo de estos paneles.

[37]http://www.dlr.de/rmc/rm/en/desktopdefault.aspx/tabid-8016/ , Julio 2014

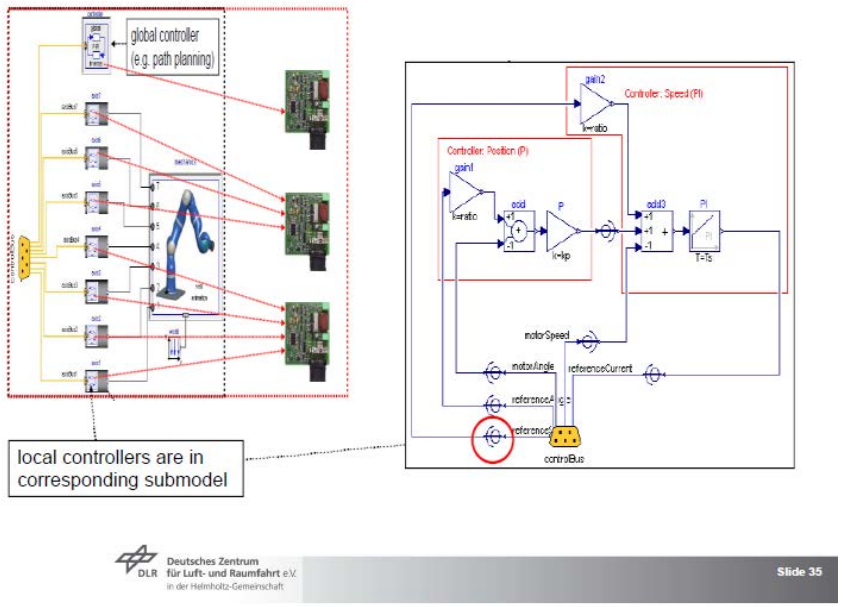


Figura 2.30: *Esquema de control para el brazo robótico LWR. Fuente: Presentación "Modelica Embedded Library", DLR*

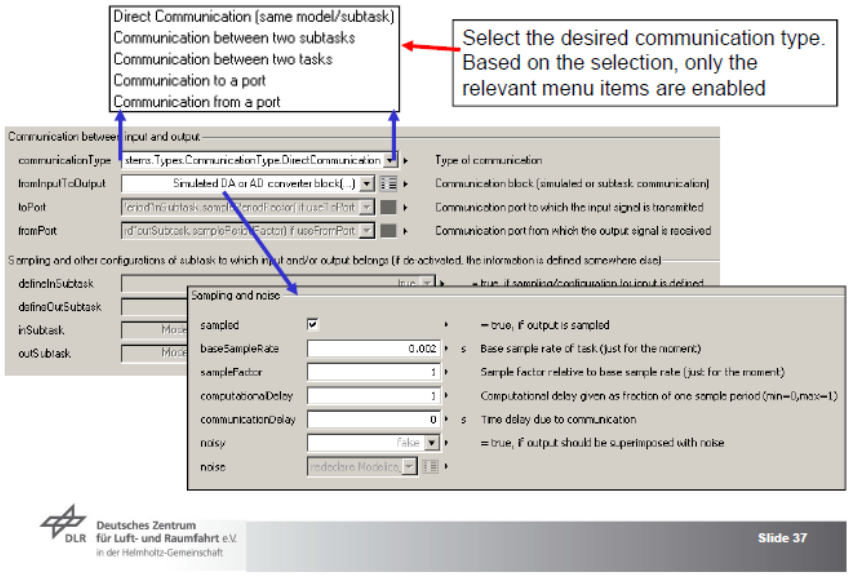


Figura 2.31: *Configuración de las opciones de I/O hardware en el entorno gráfico de Modelica®. Fuente: Presentación "Modelica Embedded Library", DLR*

2.5.5. Signal

Signal es un lenguaje de programación basado en gráficos que admite múltiples tiempos de muestreo para cada una de las partes del algoritmo que se diseñe. Aunque también es un lenguaje textual, inicialmente se concibió para utilizarse de forma gráfica, por los beneficios que el diseño basado en modelos implica. En este sentido es idéntico a MATLAB/SIMULINK®. Se comenzó a diseñar en la década de 1980 en paralelo a los lenguajes Esterel y Lustre. Se diseñó intencionadamente para programar controladores embebidos en Tiempo-Real, por lo que el lenguaje admite modificaciones para soportar hardware personalizado. El aspecto gráfico del entorno de edición es el mostrado en la figura 2.32. Tanto el entorno de desarrollo basado en modelos gráficos, como las herramientas de generación de código fuente en lenguaje C son gratuitas[38]. Debido a estas características cumple con los requisitos para considerar la base software de un sistema RCP, y dados sus orígenes en el centro de investigación INRIA en Francia, se encuentra en esta sección, a pesar de haber iniciado su andadura simplemente como un lenguaje síncrono textual.

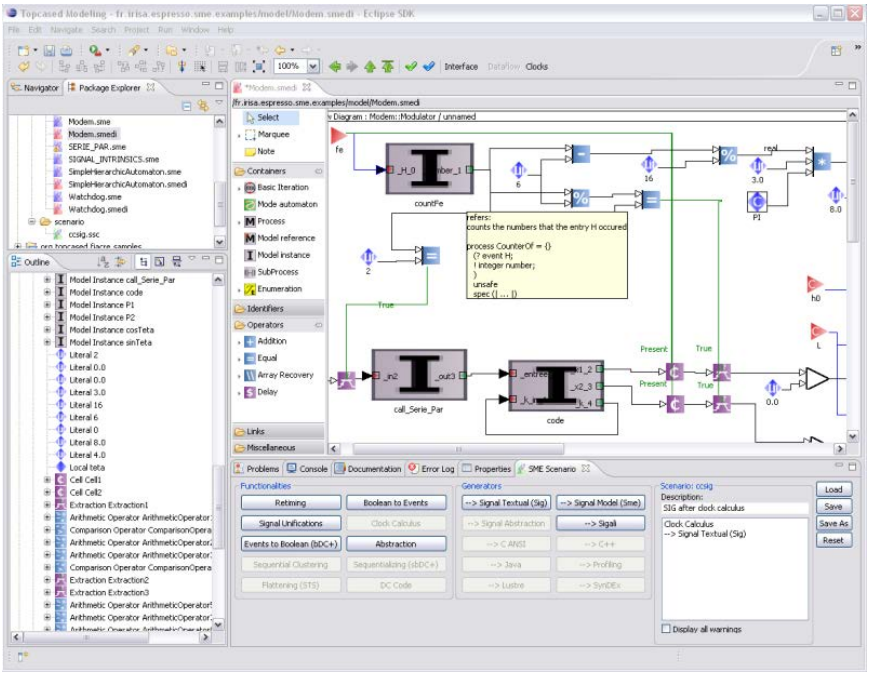


Figura 2.32: *Programación basada en modelos en el entorno gráfico de la implementación actual del lenguaje Signal.*

Aunque la metodología de programación sea similar a la de SIMULINK®, los

[38]http://www.irisa.fr/espresso/Polychrony/ , Julio 2014

procedimientos de trabajo exigen una mayor profundidad y detalle en cuanto a implementación y diseño de programas. Se trabajo a un nivel de abstracción menor, por lo que el usuario deber ser el encargado de propagar los tiempos de muestreo por los diferentes bloques que utilice. Tampoco está preparado para soportar tipos de datos vectoriales/matriciales, es responsabilidad del usuario lidiar con estas cuestiones, el programa no facilita estas tareas al usuario.

Como tipos de datos soportados están los enteros con o sin signo y el tipo de datos flotante double, se carece de soporte para flotantes de precisión simple y tipos de datos en punto fijo.

Signal se utiliza en multitud de proyectos de investigación [54, 55, 56], es la base software de los sistemas RCP personalizados utilizados generalmente por los grupos de investigación del INRIA relacionados.

2.5.6. Conclusiones RCP no comerciales

Los sistemas RCP de origen no comercial utilizan una base software, un entorno de desarrollo basado en modelos que proporciona un lenguaje de programación basado en gráficos, de mayor o menor nivel de abstracción. Generalmente estos software basados en MBD no estaban orientados a poder programar controladores hardware independientes de un ordenador PC, a excepción de Signal. Éstas son características que han sido añadidas con posterioridad y en casos como los de ScicosLab® o Scilab® ha implicado severas limitaciones que han impedido su difusión y aceptación generalizada como base de un sistema RCP. También se encuentran casos de sistemas RCP encerrados en un profundo hermetismo como son las extensiones de Modelica® del centro de investigación DLR, o aquellas que conllevan limitaciones en cuanto a capacidades multitarea y de tamaño del sistema hardware como es utilizar el modo kernel de Windows® o Linux.

Estas limitaciones han impedido la utilización de cualquiera de estos software MBD como base del sistema RCP avanzado presentado en esta tesis.

Capítulo 3

Elección del hardware

Para construir el sistema ARCP deseado, acorde a los objetivos propuestos, es necesario disponer del hardware de control idóneo de entre todos los candidatos disponibles en el mercado. Antes de comenzar a analizar posibles candidatos, se recordarán brevemente las cualidades deseadas:

- Precio, coste, tamaño, peso y consumo energético reducido
- Gran cantidad de interfaces de entrada/salida analógicas y digitales, embebidas de forma original en el sistema, sin necesidad continúa de añadir placas de expansión electrónicas o componentes adicionales.
- Es preferible que la arquitectura computacional sea Harvard en lugar de Von Neumman, de forma que el programa de control no pueda corromperse en ningún caso.
- Número de componentes muy reducido con el fin de maximizar la fiabilidad del sistema, un sistema en chip único (SOC, del inglés "System On a Chip") sería deseable.
- Tiempo de puesta en marcha del orden de milisegundos

A continuación se describirán algunos posibles candidatos y se recordarán en qué consisten las arquitecturas computacionales Von Neumann y Harvard.

3.1. Arquitectura Von Neumann

La arquitectura computacional denominada Von Neumann, basada en procesador, tiene como característica principal la utilización de una memoria única, tanto para albergar instrucciones como datos. La figura 3.1 ilustra el esquema básico de esta arquitectura computacional. Esta arquitectura computacional comparte el mismo bus de acceso a la memoria tanto para recuperar instrucciones del programa como para acceder o escribir datos.

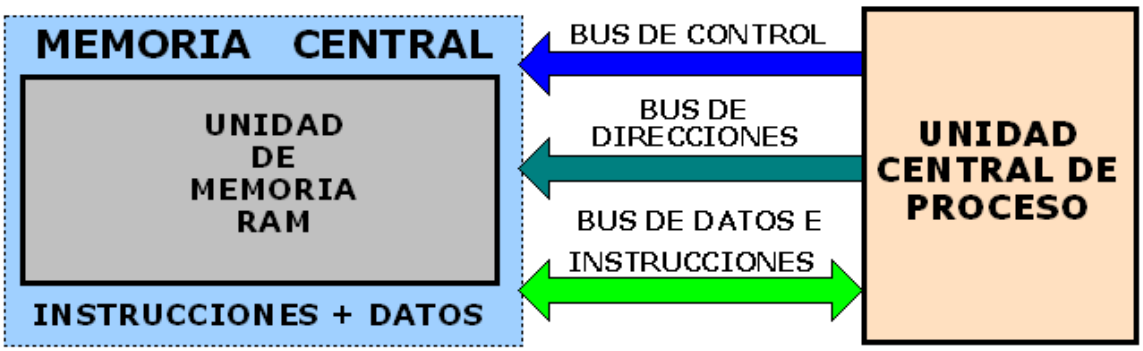


Figura 3.1: *Esquema básico arquitectura computacional Von Neumann*

Como ventajas de esta arquitectura se localizan las siguientes:

- Es una arquitectura fácilmente escalable, resulta sencillo ampliar la memoria disponible.
- Al compartir los buses tanto para acceder al programa como a los datos, los costes de fabricación son contenidos.
- El tamaño del programa albergado en la memoria puede ser variable; esto y las ventajas anteriores han hecho que esta sea la arquitectura computacional preferida para computadores de propósito general.

Como desventajas:

- Compartir un mismo bus para acceder al programa y a los datos provoca un cuello de botella, puesto que no se puede acceder a las instrucciones y a los datos a la vez. Esto provoca un detrimento en el rendimiento de la CPU del sistema.
- Compartir la misma memoria para el programa y para los datos puede ocasionar que tanto el programa o los datos se corrompan, con la gravedad que puede suponer esta eventualidad para un sistema de control mecatrónico de carácter crítico en seguridad.

3.2. Arquitectura Harvard

La arquitectura computacional denominada Harvard, tiene como característica principal la utilización de memorias separadas, tanto para albergar instrucciones como datos. La figura 3.2 ilustra el esquema básico de esta arquitectura computacional, al que se ha añadido un elemento que no estaba comtemplado en la forma básica de la misma. Este elemento es el bus de acceso directo a memoria DMA (del inglés *Direct Memory Access*), elemento presente en la mayoría de sistemas computacionales basados en arquitectura Harvard. Las arquitecturas Von Neumann también poseen buses DMA, que presentan características muy avanzadas y se conocen con el nombre de *bus mastering*, aunque emplean el bus del sistema, véase la figura 3.1, en lugar de emplear un bus separado como en muchas arquitecturas Harvard. Un bus DMA permite transferir datos entre los periféricos y la memoria RAM sin necesitar el concurso de la CPU, por lo que mientras acontecen las transacciones DMA, la CPU puede seguir ejecutando instrucciones.

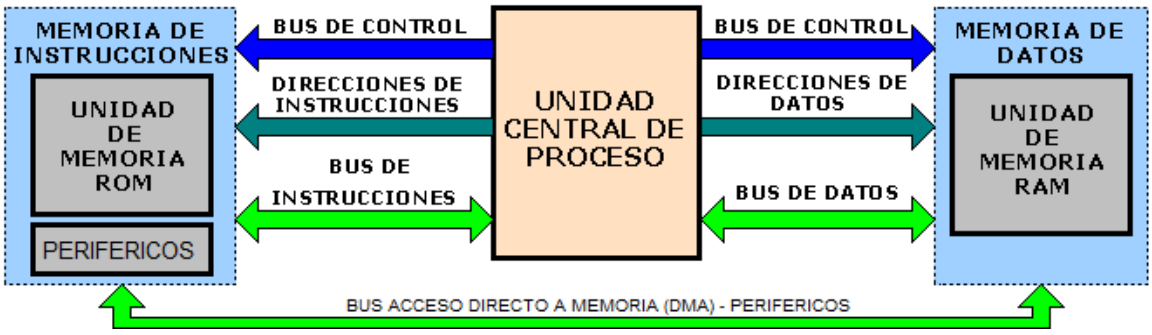


Figura 3.2: *Esquema de la arquitectura computacional Harvard*

Como ventajas de esta arquitectura:

- Emplear buses y memorias separadas para instrucciones y para datos imposibilita el hecho de que la memoria de programa pueda corromperse, lo que conlleva una mayor fiabilidad.
- El sistema puede adquirir instrucciones y datos al mismo tiempo, maximizando el rendimiento de la CPU; en esta arquitectura cualquier CPU rinde mejor que utilizándola bajo una arquitectura Von Neumann.
- La memoria de programa suele ser no volátil; esto junto con la mayor fiabilidad del sistema lo han hecho idóneo para aplicaciones de control embebido y/o crítico en seguridad.

- El mayor rendimiento que se obtiene para cualquier CPU, posibilita emplear una arquitectura Harvard con anchos de banda, en los buses de instrucciones y datos, más lentos que los que harían falta en una arquitectura Von Neumann. Característica que repercute en un sistema computacional más barato y sencillo de fabricar al emplear tecnologías relativamente obsoletas.
- Los periféricos de I/O suelen ubicarse en una de las dos secciones de memoria, la de programa o la de datos, y suelen disponer de un bus DMA para llevar a cabo transferencias con la memoria RAM.
- El haber enfocado esta arquitectura hacia sistemas embebidos ha permitido que los requerimientos en la cantidad de memoria ROM y RAM sean bastante reducidos en comparación con las arquitecturas Von Neumann. Esto permite encapsular un sistema Harvard con múltiples interfaces de I/O en un único circuito integrado. Con la reducción de espacio y consumo energético que conlleva.

Como desventajas:

- El emplear buses separados para datos e instrucciones hace que sea difícil y costoso plantear un sistema computacional de propósito general con esta arquitectura. Debido a que la duplicidad de buses y módulos de memoria encarecería el coste en su fabricación.
- La reducida cantidad de memoria ROM y RAM que puede albergar un sistema contenido en un único circuito integrado es a día de hoy, Septiembre de 2014, bastante limitada para aplicaciones de procesamiento de vídeo. Por lo que este tipo de aplicaciones se encuentran bastante limitadas, aunque ya es posible hacer tratamientos básicos de imagen, detección de formas a pocos muestreos por segundo, con sistemas Harvard de chip único.

3.3. RoBoard

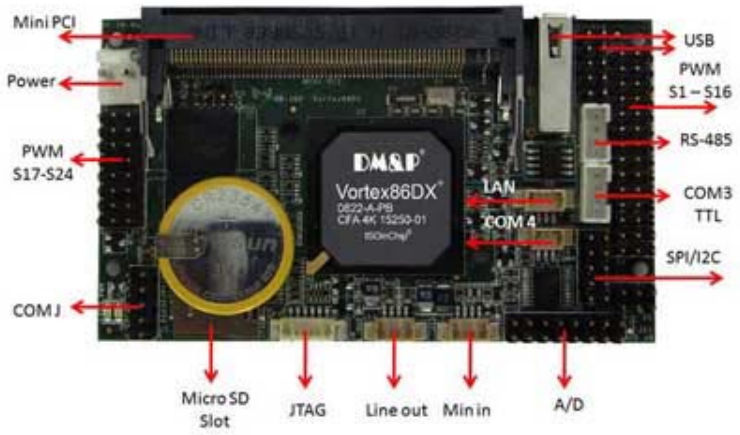


Figura 3.3: *Sistema hardware RoBoard*

Se describen las características de este sistema:

- Basado en arquitectura Von Neumann. Interfaz depuración JTAG
- Tiempo de puesta en marcha del orden de segundos
- CPU de 32 bits con unidad de coma flotante (FPU) de 32 bits, velocidad reloj 1 GHz, buses de 32 bits. Repertorio de instrucciones: Pentium 1, sin soporte MMX ni SSE
- Consumo energético: 5 Wattios . Tamaño: 96 x 56 mm.
- I/O : SPI half-duplex, I2C, salidas PWM, UART, USB, A/D 10 bits a través de conversor externo via SPI, sin bus CAN, I/O para audio
- Sin soporte actual en MATLAB® /SIMULINK®, es posible crear dicho soporte
- Precio por unidad: 180€

El alto coste por unidad, el sencillo repertorio de instrucciones sin respaldo hardware para DSP, el alto consumo energético y la ausencia de múltiples interfaces I/O como el bus CAN, puertos DAC o A/D integrados hacen que este sistema no sea apropiado para los objetivos de esta tesis.

3.4. PC 104

Figura 3.4: *Sistema hardware basado en el estándar PC 104*

Se describen las características de este sistema:

- Basado en arquitectura Von Neumann.
- Tiempo de puesta en marcha del orden de segundos
- CPU de 32 bits con unidad de coma flotante (FPU) de 32 bits, velocidad reloj (600 MHz - 1 GHz), buses de 64 bits. Repertorio de instrucciones: Intel Atom, con soporte MMX y SSE
- Consumo energético: 20 - 50 Wattios . Tamaño creciente según interfaces I/O
- I/O : variable, mediante placas de expansión apilables
- Con soporte actual en MATLAB® /SIMULINK®, xPC Target® toolbox, difícil crear soporte adicional
- Precio por unidad: 350 - 600 € por módulo apilable

El altísimo coste por unidad, el gran tamaño necesario para disponer de multitud de interfaces I/O, el altísimo consumo energético y la naturaleza comercial del soporte para SIMULINK® hacen que este sistema no sea apropiado para los objetivos de esta tesis.

3.5. Raspberry Pi

Figura 3.5: *Sistema hardware Raspberry Pi*

Se describen las características de este sistema:

- Basado en arquitectura Von Neumann.
- Tiempo de puesta en marcha del orden de segundos
- CPU de 32 bits con unidad de coma flotante (FPU) de 32 bits, velocidad reloj (700 MHz - 1 GHz), buses de 32 bits. Repertorio de instrucciones: ARMv6, con equivalencia MMX y SSE
- Consumo energético: 5 Wattios . Tamaño: 85.60 x 56 x 21 mm.
- I/O : Cantidad muy reducida; I2C, SPI, USB, Ethernet, UART
- Con soporte actual para MATLAB® /SIMULINK® version 2012b, blockset orientado sólo a multimedia. Posibilidad de crear soporte adicional, el blockset no funciona bajo el paradigma de Tiempo-Real
- Precio por unidad: 35 €

La necesidad de disponer de un sistema operativo de escritorio, como linux, debido a que los drivers de vídeo y audio no están liberados, hace que el sistema sólo pueda comportarse en Tiempo-Real utilizando el modo Kernel, que imposibilita entornos multitarea y un determinismo absoluto. También el alto consumo energético y la cantidad tan reducida en número de interfaces de I/O hacen que el sistema no sea apropiado para los objetivos de esta tesis.

3.6. BeagleBoard

Figura 3.6: *Sistema hardware BeagleBoard*

Se describen las características de este sistema:

- Basado en arquitectura Von Neumann. Opción de depuración via JTAG
- Tiempo de puesta en marcha del orden de segundos
- CPU de 32 bits con unidad de coma flotante (FPU) de 32 bits, velocidad reloj (720 MHz), buses de 32 bits. Repertorio de instrucciones: ARMv7, con equivalencia MMX y SSE
- Consumo energético: 5 Wattios . Tamaño: 76.2 x 76.2 x 16 mm.
- I/O : 3x I2C,3x SPI, USB, Ethernet, 2x UART, CAN Bus, niveles de tensión 1.8V, capacidades multimedia, video mediante FSMC
- Con soporte actual para MATLAB®/SIMULINK® version 2012b, blockset orientado sólo a multimedia. Posibilidad de crear soporte adicional, el blockset no funciona bajo el paradigma de Tiempo-Real
- Precio por unidad: 70 €

El alto consumo energético, el alto precio por unidad, la necesidad de disponer de un sistema operativo de escritorio, como linux y el reducido repertorio de interfaces de I/O así como los bajos niveles de tensión de éstas, hacen que el sistema no sea apropiado para los objetivos de esta tesis.

3.7. BeagleBone Black

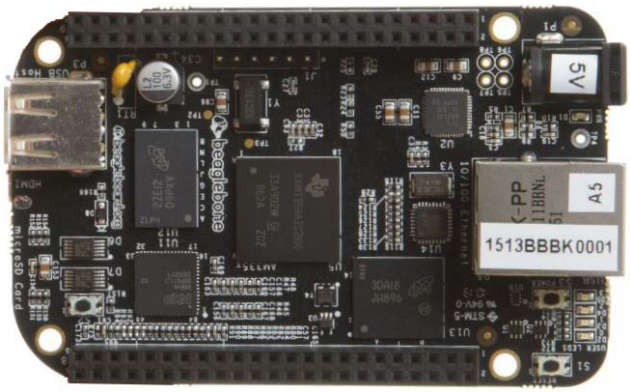


Figura 3.7: *Sistema hardware BeagleBoard*

Se describen las características de este sistema:

- Basado en arquitectura Von Neumann.
- Tiempo de puesta en marcha del orden de segundos
- CPU de 32 bits con unidad de coma flotante (FPU) de 32 bits, velocidad reloj (1 GHz), buses de 16 bits. Repertorio de instrucciones: ARMv7, con equivalencia MMX y SSE
- Consumo energético: 5 Wattios . Tamaño: 86.40 x 53.3 mm.
- I/O : USB, Ethernet, A/D de 1.8V máximo, I2C, SPI, CAN Bus, 4x PWM,4x UART, capacidades multimedia, video mediante FSMC
- Sin soporte para MATLAB®/SIMULINK®. Posibilidad de crear soporte inicial y adicional.
- Precio por unidad: 50 €

El alto consumo energético, el alto precio por unidad, la necesidad de disponer de un sistema operativo de escritorio, como linux y la limitación de su memoria DDR3 a tan sólo un ancho de banda de 16 bits (1,6 GB/s máximo) **??** hacen que la potencia computacional de su procesador no pueda ser aprovechada. El sistema no es apropiado para los objetivos de esta tesis.

3.8. Sistemas basados en FPGA

Las FPGA (del inglés "Field Programmable Gate Array") son dispositivos electrónicos basados en lógica combinacional que poseen la propiedad de poder ser reconfigurados, son circuitos basados en puertas lógicas.

Debido a estar basados en combinaciones de circuitos en lugar de ser un sistema basado en procesador, pueden realizar transformaciones de datos en cuestión de nanosegundos y recurrir a paralelizar el trabajo a desempeñar pues se puede crear un circuito distinto para cada transformación a realizar. Cada puerta lógica implementada en una FPGA consume cierto número de celdas lógicas, número que es limitado en cada modelo de FPGA.

Como principales ventajas:

- Velocidad de procesamiento
- Alta capacidad de paralelizado
- Tiempos de puesta en marcha del orden de nanosegundos

Como principales desventajas:

- El consumo energético es proporcional a la complejidad del sistema lógico combinacional implementado.
- Resulta muy complicado reutilizar funcionalidades, cada parte del circuito lógico necesita utilizar sus propios recursos, el número de celdas lógicas disponibles decrece rápidamente con la complejidad del sistema. Este defecto se está paliando en la actualidad reconfigurando el circuito lógico contenido en la FPGA mientras ésta se encuentra funcionando, con la consiguiente latencia (hay que esperar a que se terminé de reconfigurar) e incremento de la complejidad del circuito electrónico y conocimientos necesarios para llevar esto a cabo.
- No hay soporte para trabajar con tipos de datos en punto flotante y operaciones matemáticas complejas. Trabajar con este tipo de datos y funciones consume celdas lógicas a un ritmo vertiginoso.

- Las interfaces de I/O complejas como SPI, I2C, USB, CAN Bus, A/D, DAC, PWM, Timers... se deben implementar manualmente en código VHDL/Verilog o adquirirlas de forma comercial. Consumen gran cantidad de celdas lógicas.

- Implementar bucles de realimentación en una FPGA conlleva serios problemas de rendimiento, cada bucle de realimentación depende del contexto y debe implementarse de forma personalizada, la figura 3.8 ilustra este problema.

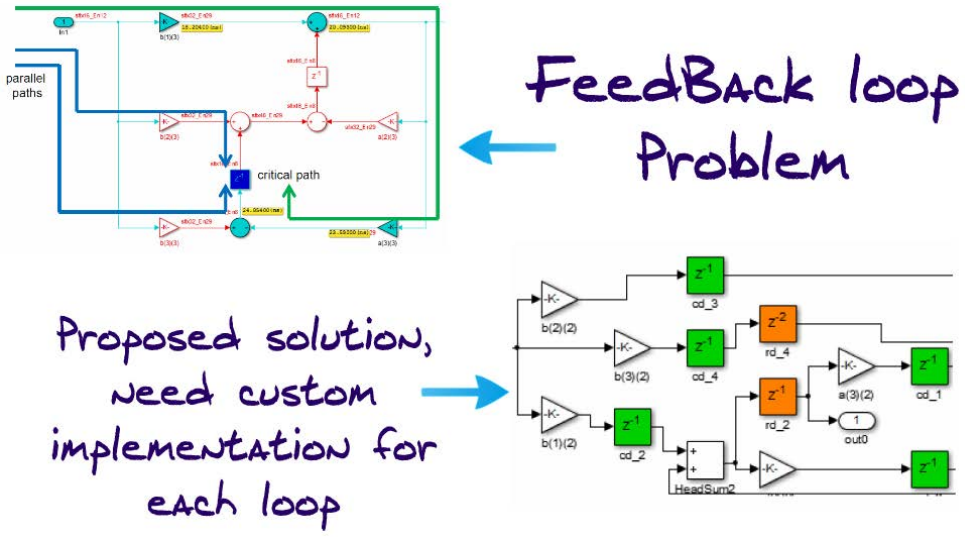


Figura 3.8: *Solución al problema de las retroalimentaciones en una FPGA*

- Las operaciones condicionales incrementa el consumo de celdas lógicas), por cada opción disponible se debe implementar el circuito electrónico, por lo tanto no se racionaliza el uso del número de celdas lógicas disponibles al hacer uso de operaciones condicionales.

- No se recomienda el uso de tipos de datos de 32 bits, ni aún números enteros; pues a mayor número de bits mayor es el consumo de celdas lógicas. Es práctica habitual utilizar tipos de datos en punto fijo de 16 bits, pero proporcionan poca precisión para números decimales.

- La presencia de diferentes tiempos de muestreo en el modelo de control supone que estos tiempos tienen que ser gestionados por el programador mediante un planificador temporal personalizado, la figura 3.9 muestra la solución para emplear diferentes tiempos de muestreo en una FPGA.

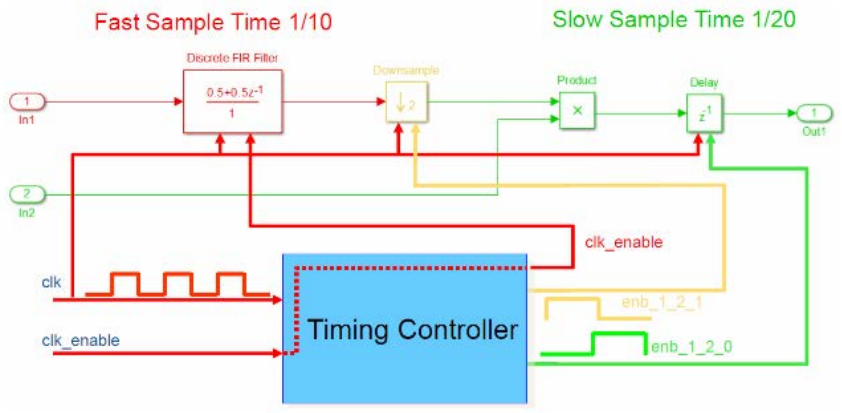


Figura 3.9: *Solución para utilizar diferentes tiempos de muestreo en una FPGA*

- Muchos de los inconvenientes anteriores hacen que resulte extremadamente difícil generalizar un lenguaje basado en modelos gráficos para programar un sistema basado en FPGA, sin incurrir constantemente en pérdidas del rendimiento o consumo excesivo de celdas lógicas.
- Son sistemas costosos, las que se encuentran a menor precio poseen una cantidad de celdas lógicas limitada.
- Los requisitos funcionales del controlador deben definirse antes de comenzar la implementación, pues un pequeño cambio en el modelo de control repercute en una implementación completamente diferente del sistema FPGA, prácticamente deben volver a plantearse qué estrategias de implementación deben seguirse debido al cambio introducido: modificar las paralelizaciones, las secuencias en serie... No son sistemas aptos para ajustarse a un modelo de control que evoluciona rápidamente con el tiempo.

Un sistema de control basado en FPGA da lugar a un dispositivo muy especializado y de alto rendimiento. Por contra y debido a los inconvenientes expuestos con anterioridad, no resulta adecuado como sistema RCP debido al alto nivel de detalle, conocimientos y especialización necesarios para trabajar con una FPGA, si se quieren obtener buenos resultados; no se encuentra al alcance de un público multidisciplinar.

En la realización de este apartado se ha contado con la asistencia de Yuri Zharikov, ingeniero especialista en implementaciones sobre FPGA [57].

Respecto a la utilización de FPGA, existe otro procedimiento que está siendo acuñado: Implementar procesadores en la propia FPGA. Aunque, y tras severas consultas a especialistas en este campo, los procesadores que se pueden implementar en una FPGA adolecen de ser muy sencillos e ineficientes. Esto es debido a que la tecnología en que se basa una FPGA no es la apropiada para implementar físicamente un procesador, lo que redunda en un rendimiento muy pobre y una cantidad de memoria disponible muy reducida, así como unos anchos de banda y anchos de buses muy por debajo de los que se encuentra disponible en MCU bastante baratos.

También en contra, las FPGA necesarias para implementar en ellas sistemas basados en procesador son bastante caras, para el rendimiento tan bajo que se obtiene; en palabras del citado especialista, llevar a cabo esta labor se puede considerar un desperdicio de recursos.

Las FPGA encuentran su mayor utilidad en el prototipado de circuitería puramente digital, y aún así su rendimiento, debido a la tecnología en que se basan, es menor que un circuito integrado estático que integre las funcionalidades sintetizadas en la FPGA.

Por todos los citados aspectos, y debido al alto nivel de complejidad, conocimientos y tiempo necesarios para llevar a cabo una implementación eficiente sobre FPGA, éstas se han descartado en la utilización como base hardware de un sistema ARCP. No resulta posible generalizar un lenguaje de programación gráfico para estos dispositivos pues el objetivo de la eficiencia está directamente relacionado con cuál es la mejor manera de implementar una u otra realimentación del sistema, y existen infinitas posibilidades de llevar a cabo una realimentación determinada y sólo algunas de ellas son eficientes. También las realimentaciones de señales en una FGPA requieren de una máquina de estados de complejidad creciente respecto al número de realimentaciones cuyas transiciones tiene que gestionar.

3.9. MCU STM32F103

Figura 3.10: *Placa con MCU STM32F103ZE del fabricante ST Microelectronics®*

Se describen las características de este sistema:

- Basado en arquitectura Harvard. Interfaz depuración JTAG y SWD.
- Tiempo de puesta en marcha del orden de milisegundos
- CPU de 32 bits sin unidad de coma flotante (FPU), velocidad reloj (72 MHz), buses de 64 bits. Repertorio de instrucciones: ARMv7, con equivalentes MMX
- Consumo energético: 0.12 Wattios . Tamaño: 50 x 80 mm.
- I/O : 112x GPIO (3V, tolerante a 5V), 16x A/D 12 bits, 2x DAC 12 bits, 16x salida PWM, 5x entrada PWM, 11x Timers, 2x I2C, 3x SPI, 1x CAN Bus, 5x UART, 1x USB, 1x Lector SD. Capacidad de vídeo por FSMC
- Con soporte actual para MATLAB®/SIMULINK®, blockset gratuito de funcionalidad básica de Aimagin Ltd. Es posible dotar al sistema de soporte adicional
- Precio por unidad MCU: 3 - 5 € Precio por placa (figura 3.10): 9 €

El estar basado en una arquitectura Harvard, en la que la memoria de programa no puede corromperse al no estar compartida con la memoria de datos, su tiempo de puesta en marcha de milisegundos y sus interfaces de depuración, hacen que este sistema sea viable para cubrir las necesidades impuestas por el nivel de seguridad SIL 4. La gran capacidad, número y flexibilidad de sus interfaces I/O y su soporte básico para SIMULINK y su bajo coste por unidad hacen que este MCU sea adecuado para proponer un sistema ARCP. No obstante, la ausencia de unidad FPU, el limitado ancho de banda de sus buses y su reducida velocidad de reloj hacen que adolezca de una capacidad computacional demasiado ajustada.

3.10. MCU STM32F4

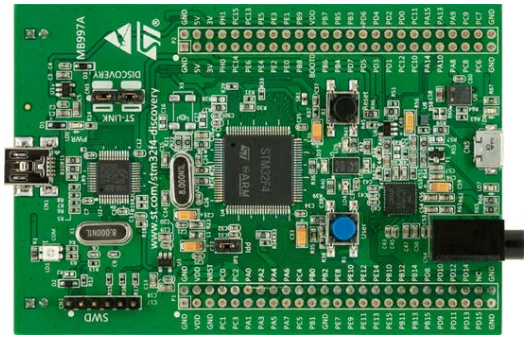


Figura 3.11: *Placa STM32F4 Discovey del fabricante ST Microelectronics®*

Se describen las características de este sistema:

- Basado en arquitectura Harvard. Interfaz depuración JTAG y SWD. Depurador integrado en la placa.
- Tiempo de puesta en marcha del orden de milisegundos
- CPU de 32 bits con unidad de coma flotante (FPU) de 32 bits, velocidad reloj (168 MHz), buses de 128 bits. Memoria caché: 1 KB instrucciones y 64 KB para datos y FPU. Repertorio de instrucciones: ARMv7, repertorio completo de la CPU ARM Cortex-A8
- Consumo energético: 0.3 Wattios . Tamaño: 66 x 97 mm.
- I/O : 140x GPIO (3V, tolerante a 5V), 16x A/D 12 bits, 2x DAC 12 bits, 31x salida PWM, 6x entrada PWM, 12x Timers, 3x I2C, 3x SPI, 2x CAN Bus, 6x UART, 1x USB, 1x Lector SD, 1x Ethernet, 1x Audio DAC. Capacidad de vídeo por FSMC
- Con soporte actual para MATLAB®/SIMULINK®, blockset gratuito de funcionalidad básica de Aimagin Ltd. Es posible dotar al sistema de soporte adicional
- Precio por unidad MCU: 3 - 9 € Precio por placa (figura 3.11): 15 €

El estar basado en una arquitectura Harvard, en la que la memoria de programa no puede corromperse al no estar compartida con la memoria de datos, su tiempo de puesta en marcha de milisegundos y sus interfaces de depuración, hacen que este sistema sea viable para cubrir las necesidades impuestas por el nivel de

seguridad SIL 4. La gran capacidad, número y flexibilidad de sus interfaces I/O, su soporte básico para SIMULINK, sus buenas prestaciones computacionales y su bajo coste por unidad (que también incluye el depurador) hacen que este MCU sea adecuado para proponer un sistema ARCP.

3.11. Conclusiones Elección Hardware

Contando con las características necesarias de portabilidad y seguridad del sistema de control necesario, la elección del hardware sobre el que diseñar y construir un sistema Avanzado de Prototipado Rápido para Control (ARCP) recae sobre el microcontrolador STM32F4 del fabricante ST Microelectronics®. Dado que en este hardware no es imperativo emplear un sistema operativo, todos los recursos hardware se encuentran accesibles, desde la gestión completa de todas las fuentes de interrupción (IRQ) hasta el uso de los canales de acceso directo a memoria (DMA); es una arquitectura orientada a controlar sistemas mecatrónicos que exijan altos niveles de seguridad, de ahí su naturaleza basada en una arquitectura Harvard y el número tan reducido de componentes con el fin de maximizar la fiabilidad. De esta forma el código autogenerado por la toolbox de funcionalidades avanzadas para SIMULINK® puede ser el más optimizado y completo posible, evitando completamente recurrir a la programación manual del sistema de control.

El resto de sistemas hardware analizados como Roboard, Raspberry Pi, BeagleBoard... adolecen de imitar la arquitectura computacional de un ordenador PC, obligando a utilizar un sistema operativo de escritorio, limitando completamente el acceso a la reprogramación de las fuentes IRQ y canales DMA, así como introduciendo grandes latencias en el acceso a los recursos hardware debido a utilizar drivers con bajo nivel de privilegios y obligando a recurrir al modo kernel del sistema operativo como única alternativa para ejecución en Tiempo-Real.

Respecto al uso de sistemas basados en FPGA, estos dan lugar a dispositivos muy especializados y de alto rendimiento. Aunque la escasa flexibilidad en su programación y los serios problemas derivados del uso de bucles realimentados y tipos de datos de 32 bits y en coma flotante imposibilitan su programación rápida y eficiente mediante modelos basados en gráficos. Las limitadas posibilidades de I/O también son un factor determinante para rechazar las FPGA como base de un sistema ARCP.

Capítulo 4

Diseño Basado en Modelos a Código Fuente

El diseño basado en modelos (MBD) es un procedimiento de programación orientado originalmente a la realización de simulaciones, proveyendo de un lenguaje basado en gráficos sobre el que modelar matemáticamente el comportamiento de un sistema, sea cual sea su naturaleza.

La programación mediante MBD se asemeja al planteamiento sobre papel de un modelo matemático o esquema de funcionamiento de un programa informático, trabajando en el más alto nivel de abstracción, sin tener en cuenta los detalles más minuciosos. La figura 4.1 muestra la analogía entre plantear un modelo gráfico y programarlo de forma textual; con las técnicas software adecuadas el modelo gráfico puede generar código fuente con calidad de producción que reproduzca el comportamiento definido mediante grafos.

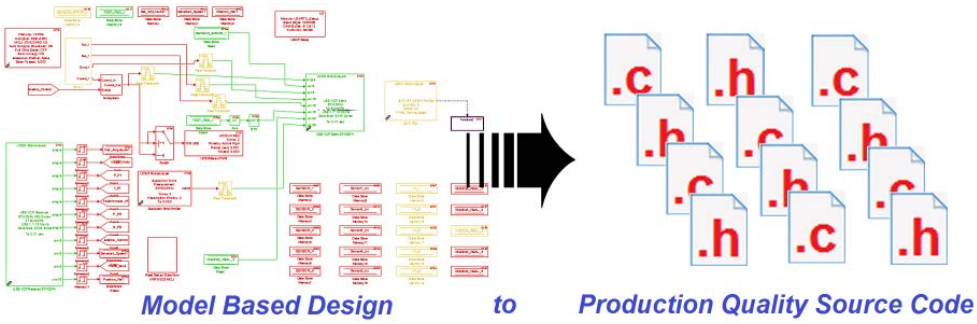


Figura 4.1: *Analogía entre el esquema en grafos y el código fuente generado. Ambos describen el mismo comportamiento, son el mismo programa*

4.1. Elección del software basado en modelos

El paquete software elegido que proporciona un entorno de diseño basado en modelos gráficos (MBD) para llevar a cabo los planteamientos propuestos en esta tesis es SIMULINK®, herramienta software adicional incluida en las distribuciones del software de cálculo matemático MATLAB®; ambos son programas de la compañía MathWorks®.

SIMULINK® posee la capacidad, mediante software adicional, de generar código fuente en lenguaje C para diferentes arquitecturas computacionales, así como permitir el añadir soporte para arquitecturas hardware adicionales, soporte que puede ser originado por el usuario. El código fuente generado está diseñado originalmente para generar programas que funcionan bajo el paradigma de Tiempo-Real en modalidad monotarea. Las expansiones software o toolboxes que permiten generar este tipo de código fuente son SIMULINK Coder® y Embedded Coder®.

SIMULINK también resulta muy interesante para ser utilizado en laboratorios de investigación por su alto nivel de integración con el resto de toolboxes disponibles para MATLAB; el conjunto de éstas tales como 'System Control Toolbox', 'SimMechanics', 'SimRF'... resultan muy útiles para cubrir las necesidades propias de un amplio espectro de ramas de la ingeniería.

Otros entornos MBD son LabVIEW® de la compañía National Instruments® y ScicosLab. LabVIEW® se ha descartado en esta tesis por la imposibilidad de añadir soporte personalizado para arquitecturas hardware distintas de las ofertadas por National Instruments®. ScicosLab por su parte presenta grandes dificultades en la integración de soporte hardware personalizado, debido al mecanismo adoptado para la generación de código fuente y a sus limitadas capacidades en cuanto al soporte nativo respecto a tipos de datos flotantes y en punto fijo.

4.2. Formalismo en el Diseño Basado en Modelos

La analogía mostrada con anterioridad, entre esquemas gráficos manuales y los lenguajes gráficos, permite que resulte más sencillo y natural transferir ideas y conceptos a un lenguaje gráfico que a un lenguaje textual. Debido a esta interesante propiedad, los entornos software MBD han sido utilizados casi desde sus inicios como lenguajes de programación síncronos, pues también resulta sencillo establecer en ellos los formalismos necesarios para evitar ambigüedades de entendimiento. La figura 4.2 muestra un sencillo ejemplo programado en lenguaje gráfico en el que se multiplican dos variables, el formalismo inherente al lenguaje gráfico, en este caso SIMULINK®, permite establecer el orden de la ejecución de forma unívoca; se pone a disposición del usuario la posibilidad de realizar las operaciones y la escritura/lectura de las entradas/salidas en el orden que se desee, dejando constancia para la posteridad del orden deseado de ejecución, no dejando lugar a ambigüedades. Concepto éste muy importante en sistemas de control. En el caso de SIMULINK® los errores de interpretación son evitados estableciendo prioridades para cada elemento de programación basado en gráficos.

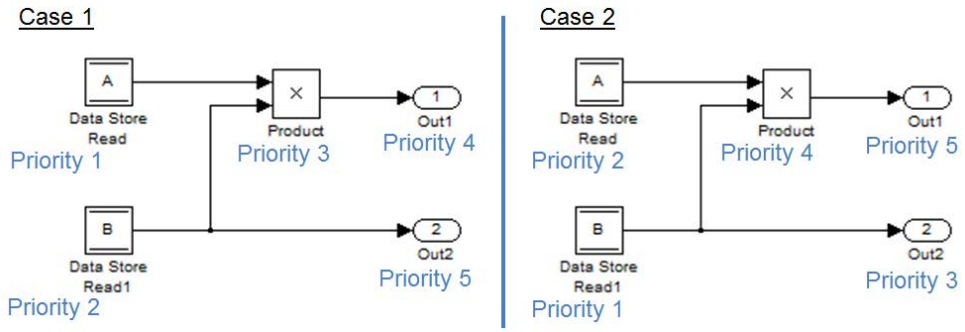


Figura 4.2: *Formalismo del lenguaje gráfico SIMULINK®. Ambos casos generan un código fuente distinto y unívoco en base a las prioridades establecidas.*

Los ejemplos de la figura 4.2 generan el código en lenguaje C:

- Caso 1:

```
void case1_step(void)
{
  untitled_Y.Out1 = untitled_DWork.A * untitled_DWork.B;
  untitled_Y.Out2 = untitled_DWork.B;
}
```

- Caso 2:

```
void case2_step(void)
{
  untitled2_Y.Out2 = untitled2_DWork.B;
  untitled2_Y.Out1 = untitled2_DWork.B * untitled2_DWork.A;
}
```

Estos son ejemplos sencillos, pero en los casos reales con programas más complejos, el formalismo de estos lenguajes permite que éstos sean deterministas.

4.3. Generación Automática de Código Fuente

El proceso de generación de código fuente en lenguaje C desde el lenguaje gráfico de MATLAB®/SIMULINK® comprende una serie de etapas, mostradas de forma esquemática en la figura 4.3. Estas etapas tienen su base en el formalismo del lenguaje gráfico y en las preferencias del objetivo hardware que se hayan seleccionado. Es dicho formalismo el que permite describir de manera textual el modelo gráfico. A partir del modelo gráfico y el conjunto de funciones compiladas (denominadas .mex) que lo compone se obtiene una estructura de información, estructura en lenguaje C denominada "Simulink.RTW Model", que define el comportamiento del modelo de forma jerarquizada, pues tiene en cuenta el formalismo del lenguaje. A partir de esta estructura de información, el núcleo de MATLAB® comienza a escribir el código fuente (en lenguaje C) unívoco correspondiente al modelo gráfico. Dicha estructura de información obtiene sus datos de las funcionalidades de cada bloque gráfico, funcionalidades escritas en diferentes lenguajes. Los posibles errores detectados en cada etapa detienen el proceso de generación de código, y como norma general se muestra en forma de mensaje textual la localización del error, sea cual sea el lenguaje en el que se haya producido.

La organización en la escritura del código fuente en lenguaje C corre a cargo del lenguaje de script denominado "Target Language Compiler" (TLC), cada bloque gráfico tiene asociado su archivo TLC y mediante instrucciones propias de este lenguaje se accede a las opciones de cada bloque para escribir el código fuente de una u otra manera. Sirve también para ubicar el código y las llamadas a funciones en uno u otro sitio de cada archivo fuente, lo que permite que unas funcionalidades gráficas formen parte del algoritmo y otros bloques gráficos representen funciones asíncronas como IRQ, creación de tareas o sólo aporten código que se ejecute una vez al inicio del programa.

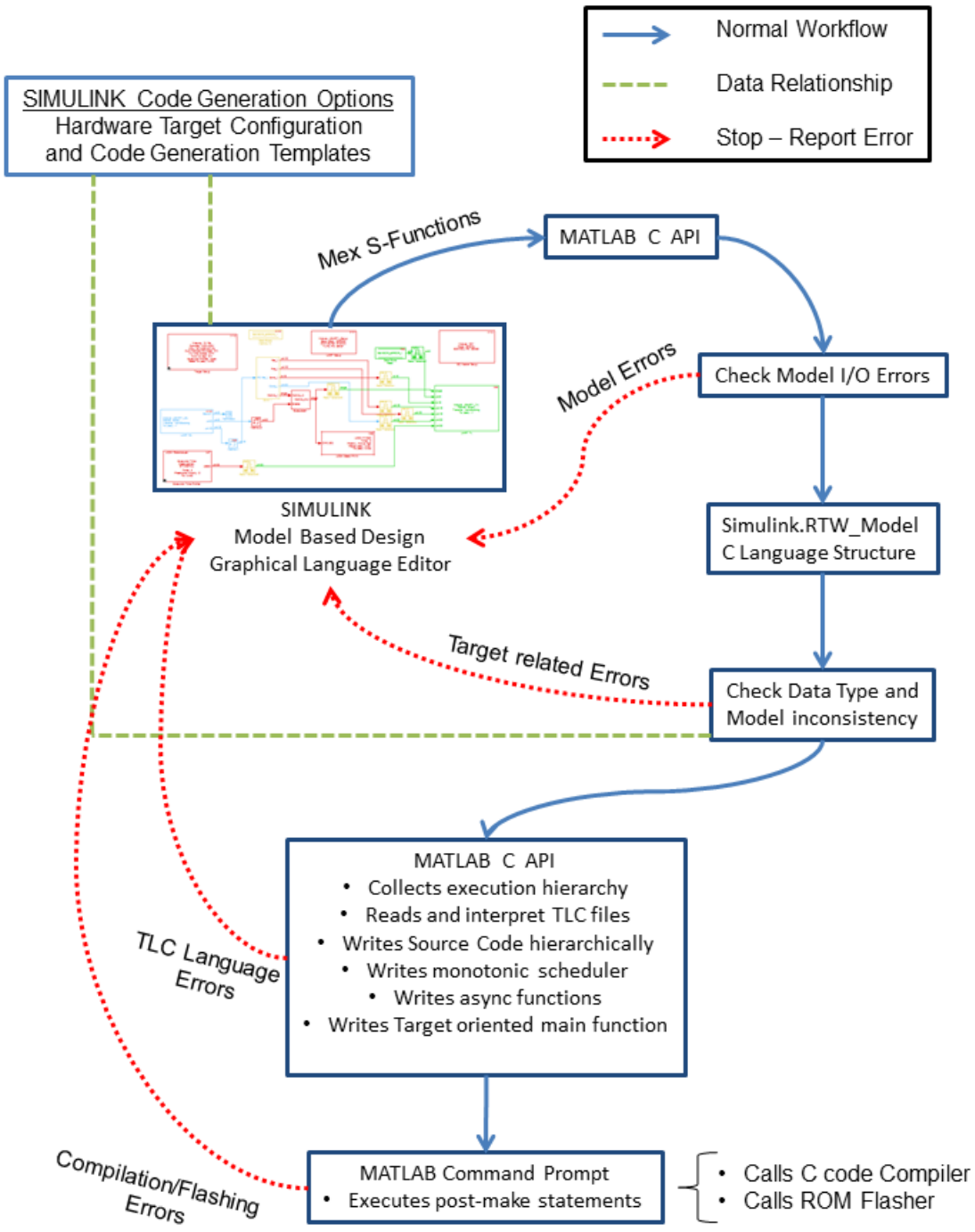


Figura 4.3: *Etapas en la generación de código fuente a partir de un modelo gráfico en MATLAB®/SIMULINK®.*

4.4. Funciones y lenguajes para crear un bloque

En esta sección se describen las funcionalidades y lenguajes necesarios para crear un nuevo bloque para SIMULINK®. La figura 4.4 ilustra las funcionalidades que conforman un bloque y la relación entre ellas.

4.4.1. MEX S-Functions

Una función MEX es un programa en lenguaje C que utiliza la interfaz de programación para aplicaciones (del inglés API, "Application Programming Interface") en C de MATLAB®. Sirve para crear nuevas funcionalidades, ya sea para MATLAB® o SIMULINK®, se compilan y ejecutan en código máquina, aunque requieren del núcleo de ejecución de MATLAB®.

En SIMULINK® todos los bloques gráficos tienen en su función .mex su base de funcionamiento. Dicha función define el número (que puede ser dinámico) de entradas/salidas, el soporte de tiempo-discreto y/o tiempo-continuo, el número y tipo de opciones de usuario que posee el bloque y la funcionalidad que tiene el bloque en el modo de simulación. En definitiva, la función .mex define el comportamiento del bloque.

Si se deseara que un bloque gráfico también realizará funciones en el modo de simulación además de generar código fuente, sería necesario añadir la funcionalidad para simulación en el código que compone el comportamiento del bloque. Por tanto una función .mex para un bloque de SIMULINK® contiene diferentes llamadas a funciones escritas en lenguaje C:

- $mdlCheckParameters(SimStruct*S)$ Comprueba los errores en las opciones de usuario.
- $mdlInitializeSizes(SimStruct*S)$ Gestiona el número y tipo de puertos I/O.
- $mdlInitializeSampleTimes(SimStruct*S)$ Gestiona el tiempo de muestreo, si es discreto, continuo o asíncrono.
- $mdlOutputs(SimStruct*S, int_T tid)$ Opcional. Proporciona la funcionalidad en modo simulación.
- $mdlTerminate(SimStruct*S)$ Gestiona las acciones al terminar la ejecución.
- $mdlRTW(SimStruct*S)$ Escribe lo necesario en Simulink.RTW Model para generar código fuente.

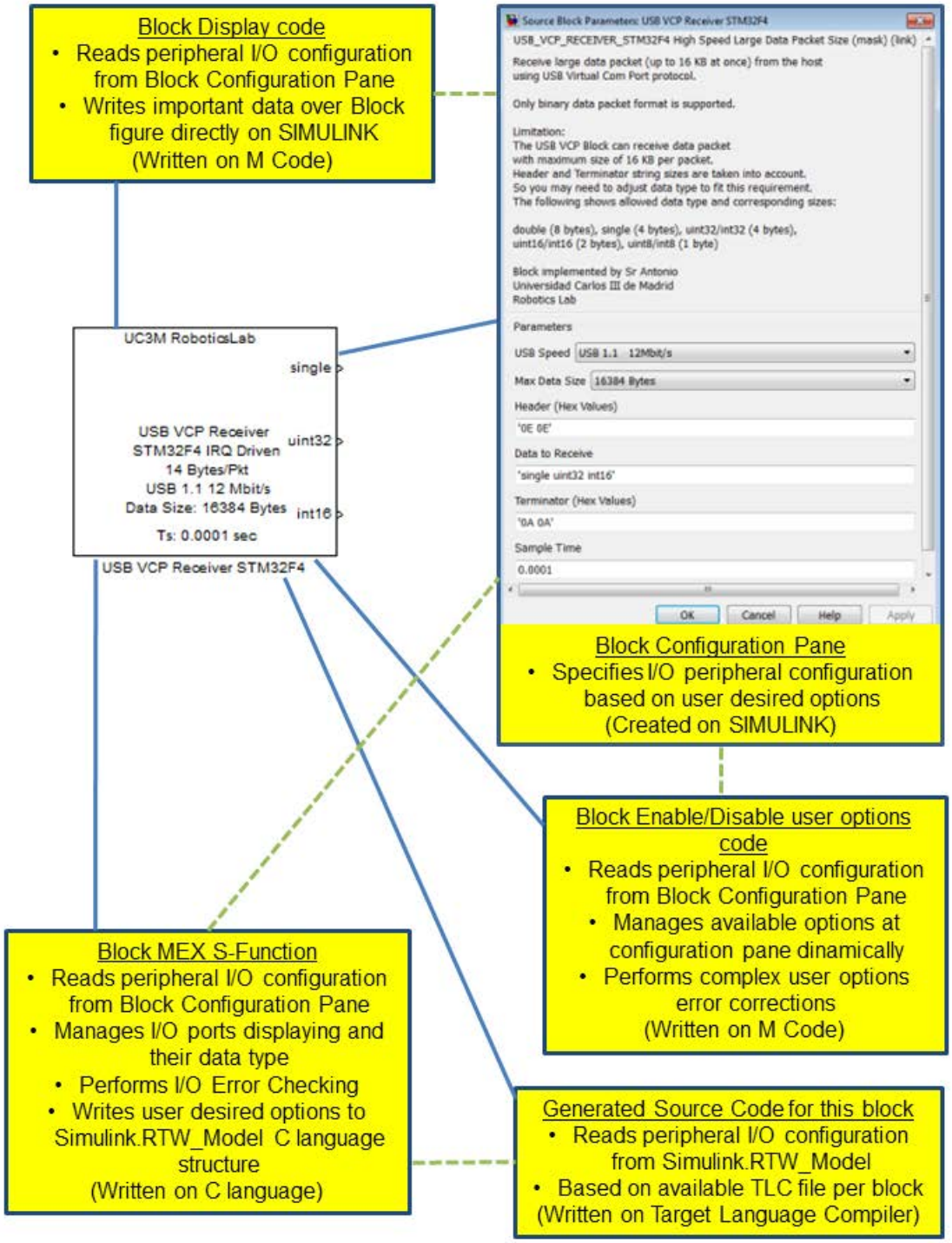


Figura 4.4: *Funcionalidades que conforman un bloque de soporte hardware para SIMULINK®. En azul se señalan las distintas funcionalidades. En verde se indican las relaciones de datos entre funcionalidades.*

4.4.2. Panel de Configuración

Cada bloque funcional ya sea de la librería estándar o de una librería personalizada de SIMULINK® tiene asociado el código fuente que puede generar, éste código fuente no siempre es estático, sino que algunas de sus partes pueden ser distintas en base a las opciones que el usuario haya seleccionado en el panel de configuración desplegable, en el caso de que el bloque de programación gráfica tenga un panel de configuración asociado. Este panel de configuración permite elegir y/o cambiar las variables y datos de inicialización del bloque gráfico; en realidad se están cambiando variables y datos del código fuente asociado al bloque gráfico. Se puede interpretar que cada bloque gráfico es el intermediario entre el código fuente en lenguaje C y el entorno software basado en modelos.

La figura 4.5 muestra un ejemplo de panel de configuración para el bloque gráfico de recepción USB del MCU STM32F4, también se relaciona dónde tienen su origen cada una de las opciones disponibles para el usuario en el panel de configuración, en el editor de máscaras de SIMULINK®. En este editor también se especifica el código que ofrece el aspecto visual del bloque, en "Icon and Ports"; el texto mostrado en el panel de configuración se especifica en "Documentation".

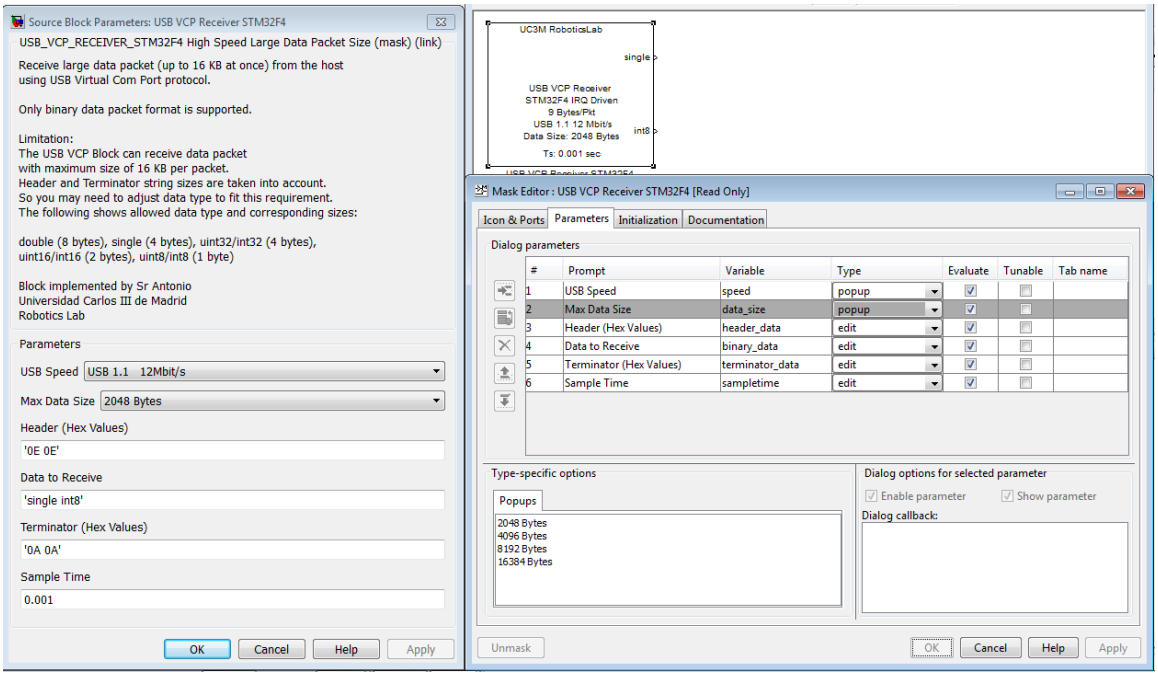


Figura 4.5: *A la izquierda: Panel de Configuración bloque USB Recibir. A la derecha: Editor del panel de configuración en SIMULINK®. Arriba: Bloque de recepción por USB del MCU STM32F4*

4.4.3. Aspecto Visual

El aspecto visual de un bloque gráfico se define en lenguaje M, mediante funciones de dibujado de MATLAB® y tiene acceso a leer las opciones asociadas al panel de configuración del bloque, aunque el acceso a dichas opciones se hace de manera radicalmente diferente a cómo se hace utilizando la API en C de MATLAB® por parte de la función MEX. El resultado de las funciones de dibujado al ser llamadas desde un bloque de SIMULINK®, tienen su efecto sobre el recuadro del bloque.

4.4.4. Funciones de Callback

Las funciones de callback en un bloque gráfico de SIMULINK® sirven para inspeccionar y corregir dinámicamente los errores cometidos por el usuario al introducir las opciones en el panel de configuración, también sirve para habilitar o deshabilitar distintas opciones del panel de configuración de forma dinámica; habilitar una opción del panel de configuración puede dar lugar a disponer de más opciones, como resultado de haber habilitado la anterior. La figura 4.6 muestra un ejemplo en el que al habilitar la opción inicial, surgen dos opciones más y también se crea un nuevo puerto de salida en el bloque.

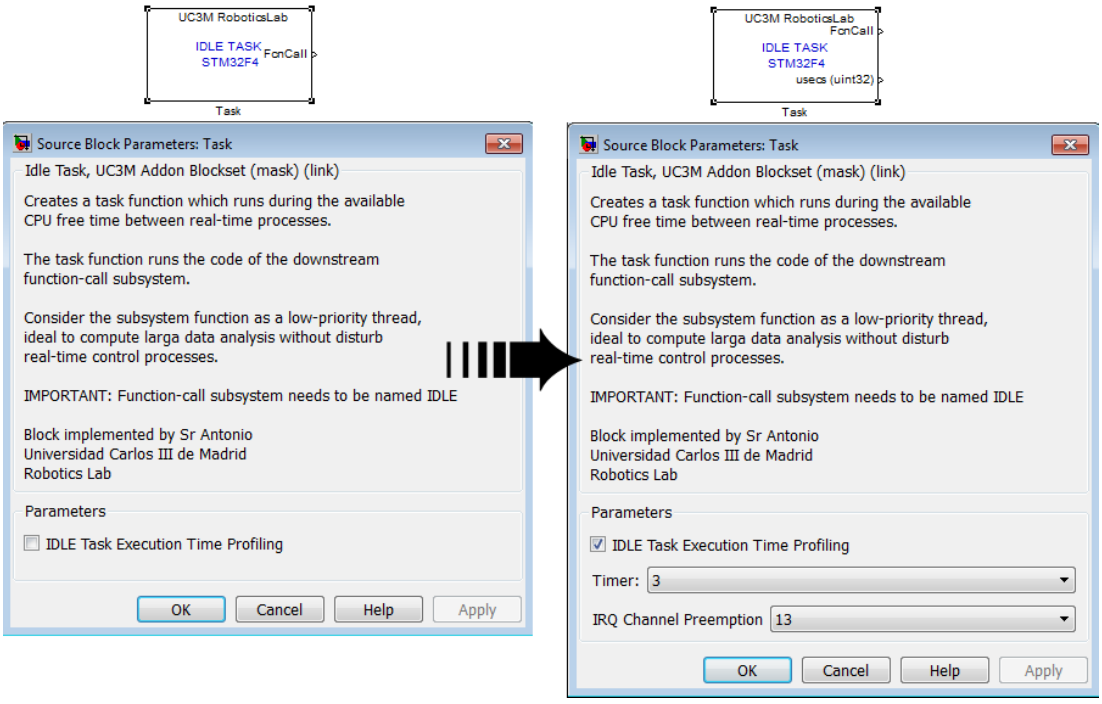


Figura 4.6: *Efecto de una función de callback, habilitar una opción da lugar a otras dos opciones.*

4.4.5. Escritura del Código Fuente. Target Language Compiler

El código fuente en lenguaje C generado por un bloque gráfico de SIMULINK® está orquestado por un lenguaje de script denominado "Target Language Compiler" (TLC). A través del lenguaje TLC se accede a los campos de la estructura "Simulink.RTW Model" relacionados con el bloque gráfico al que está asociado el fichero en lenguaje TLC, y dependiendo de las opciones que se eligieran, se generará un código fuente en lenguaje C unívoco acorde a dichas opciones.

Los ficheros en lenguaje TLC se estructura de forma que permitan generar las diferentes partes del código fuente necesarias, tales como código de inicialización, código de iteración, código asíncrono, código de finalización... por lo que en un fichero TLC se pueden encontrar las siguientes llamadas a funciones, siempre escritas en lenguaje de script TLC:

- $\%functionBlockInstanceSetup(block, system)void$ Organiza la escritura del código fuente necesario para la inicialización de la funcionalidad del bloque gráfico, lo hace cada vez que esté presente ese bloque.
- $\%functionBlockTypeSetup(block, system)void$ Organiza la escritura del código fuente necesario para la inicialización de la funcionalidad del bloque gráfico, lo hace una sola vez aunque el bloque esté presente en el modelo gráfico varias veces.
- $\%functionStart(block, system)Output$ Lleva a cabo la inicialización de variables y llamadas a funciones para inicializar el periférico. Todas las funciones "Start" de los ficheros TLC escribirán el código fuente en la función "initialize model()".
- $\%functionOutputs(block, system)Output$ Organiza la escritura del código fuente necesario para llevar a cabo la funcionalidad deseada del bloque, teniendo en cuenta la modalidad temporal del bloque; síncrona o asíncrona.
- $\%functionTerminate(block, system)Output$ Organiza la escritura del código fuente necesario para llevar a cabo la funcionalidad deseada al terminar la ejecución, es útil incluir escritura de código fuente si se tiene en cuenta la funcionalidad avanzada de poder detener la ejecución del modelo y llevarlo a un estado seguro, o incluso detener solamente partes del modelo.

Además de los ficheros TLC individuales y propios para cada bloque gráfico, ***es necesario disponer de un fichero TLC orientado a la arquitectura hardware para la que se esté generando el código fuente***. Este ***fichero TLC principal*** define cómo

se organiza el código fuente del fichero que contiene la función "main", las rutas donde se incluyen los ficheros de encabezado .h y cuáles se han de incluir. También orquesta cómo se han de estructurar, en qué ficheros .c y .h se van a escribir la funciones de inicialización y de ejecución del modelo. En definitiva, el fichero TLC principal contiene las plantillas de generación de código; el disponer de estas plantillas es condición indispensable para los niveles de seguridad SIL 3 y SIL 4.

4.5. Trazabilidad del código

El proceso de generación de código fuente crea un informe en lenguaje HTML en el que mediante hipervínculos se relaciona el código en lenguaje C con los bloques gráficos de los que procede, la relación puede ser seguida de forma bidireccional tal y como ilustra la figura 4.7; de esta manera el código fuente puede ser analizado y revisado de forma que se garantice la univocidad del mismo. También se incluyen automáticamente comentarios respecto a que función realiza cada parte identificable del código fuente, relacionando también con los bloques gráficos de los que procede. Se identifican de forma bidireccional el uso y ubicación de las variables y el tipo de datos que utilizan.

Las funcionalidades avanzadas añadidas como soporte adicional a SIMULINK® también tienen cabida en este informe de trazabilidad, siempre que las nuevas funcionalidades hagan uso de la estructura "Simulink.RTW Model". El informe de trazabilidad es condición exigida para que un lenguaje basado en gráficos pueda generar un código fuente con calidad de producción.

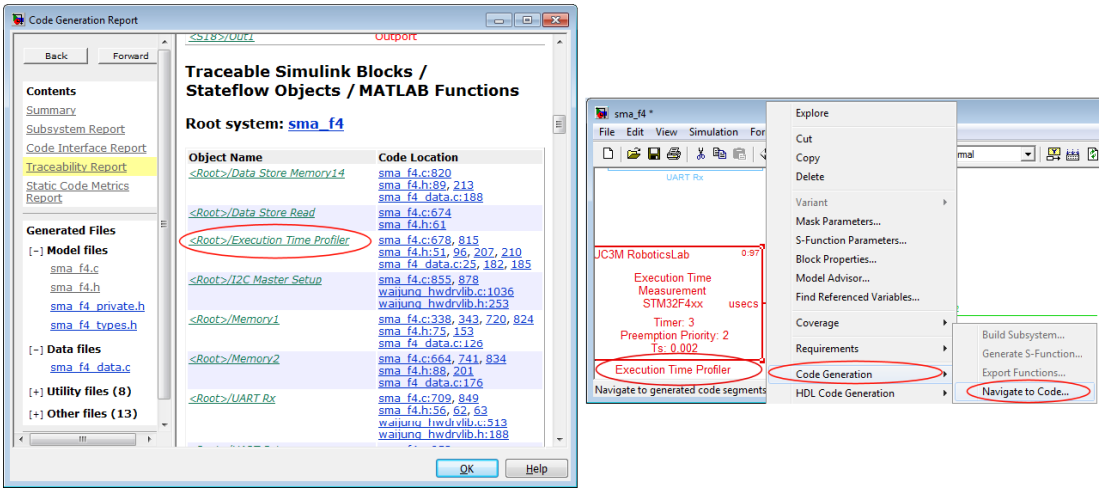


Figura 4.7: *A la izquierda: Trazabilidad desde el código fuente al modelo gráfico. A la derecha: Trazabilidad desde el modelo gráfico al código fuente.*

4.6. Modalidades de Tiempo-Real. Planificador Temporal

Las capacidades de generación automática de código fuente de MATLAB®/SIMULINK® permiten utilizar hasta tres modalidades diferentes para implementar código en Tiempo-Real a partir del modelo gráfico:

- **Monotarea con tiempo de muestreo único**: La ejecución del código se realiza a intervalos de tiempo regulares, marcados por una IRQ. Se ejecuta la totalidad del modelo programado de forma gráfica, todos los bloques deben tener establecido el mismo tiempo de muestreo. No hay planificador temporal.

- **Monotarea con múltiples tiempos de muestreo**: La ejecución del código se realiza a intervalos de tiempo regulares, marcados por una IRQ. Cada bloque gráfico puede tener un tiempo de muestreo diferente siempre que sean múltiplos del anterior intervalos de tiempo regular (tiempo base). El planificador temporal tiene la función de ejecutar cada bloque gráfico con el tiempo de muestreo que le corresponde, respetando el orden de ejecución establecido mediante el formalismo de SIMULINK®.

- **Multitarea con un tiempo de muestreo diferente por tarea**: El planificador temporal está ubicado dentro de la IRQ temporal (la IRQ que marca los intervalos de tiempo regulares), esta modalidad presupone que el sistema hardware soporta IRQ que pueden interrumpirse a sí mismas; dado que esta no es una situación generalizada, esta plantilla se suele modificar para que cada tiempo de muestreo dé lugar a una nueva tarea dentro de un entorno gobernado por un Sistema Operativo en Tiempo Real (RTOS). No es una plantilla de generación de código que encuentre un uso muy difundido, puesto que existen mecanismos más sencillos en SIMULINK® para crear tareas en Tiempo-Real.

Los mecanismos anteriores no permiten utilizar tiempo-continuo ni más de un planificador temporal, estas son características del sistema ARPC presentado en esta tesis, explicadas en la sección 5.2. Poder utilizar más de un planificador temporal permite que las distintas tareas en Tiempo-Real no necesiten que sus tiempos de muestreo sean múltiplos de un mismo tiempo base, y también permite que en una misma tarea haya partes de código que se ejecuten a distintos intervalos temporales, sin agrupar las funcionalidades de esos bloques en nuevas tareas.

Capítulo 5

Sistema Avanzado de Prototipado Rápido para Control

El concepto de Sistema Avanzado de Prototipado Rápido para Control (ARCP) difiere del concepto clásico de Sistema de Prototipado Rápido para Control (RCP). La concepción clásica define un sistema RCP como el conjunto de herramientas software y hardware que permiten probar en poco tiempo la viabilidad de un modelo de control diseñado mediante un lenguaje basado en gráficos, mientras que el concepto de sistema ARCP define el conjunto de herramientas software y hardware que permiten crear un controlador optimizado con calidad de producción en poco tiempo mediante un lenguaje basado en gráficos.

La diferencia de características y posibilidades a disposición del usuario para ambos conceptos resulta significativamente diferente:

Para el caso de un sistema RCP tradicional, dado que la intención es tan sólo comprobar la viabilidad del modelo de control, se pone a disposición del usuario un repertorio de funcionalidades básicas que permiten gestionar los periféricos de I/O y dar lugar a una tarea en Tiempo-Real sólo con soporte de tiempo discreto; no tiene en cuenta las profundas optimizaciones que pueden tener cabida en la arquitectura computacional hardware del sistema RCP, pues éstas características se tendrán en cuenta en la etapa de implementación final, en la que el hardware será diferente.

Para el caso de un sistema ARPC como el propuesto en esta tesis, existe una doble intención, además de validar la viabilidad del modelo de control, se pone a disposición del usuario un repertorio de funcionalidades básicas y avanzadas que

permiten gestionar los periféricos de I/O, gestionar distintas tareas en Tiempo-Real haciendo uso de soporte para tiempo discreto y para tiempo continuo, gestionar las fuentes de IRQ, gestionar las rutinas de atención a errores graves del sistema, utilizar capacidades multimedia de audio y vídeo, y de forma transparente al usuario, tener en cuenta las mejores optimizaciones que tienen cabida en la arquitectura hardware empleada en el sistema ARCP, de forma que el controlador obtenido tenga calidad de producción, llevando a cabo en una sola etapa la citada doble intencionalidad del sistema ARCP, utilizarlo como sistema de pruebas en el laboratorio y como sistema de implementación final al mismo tiempo utilizando el mismo lenguaje basado en gráficos.

El sistema ARPC dota al usuario de funcionalidades básicas y avanzadas; la figura 5.1 ilustra de forma esquemática el abanico de posibilidades cubiertas por el sistema ARCP, que serán detenidamente explicadas en posteriores apartados.

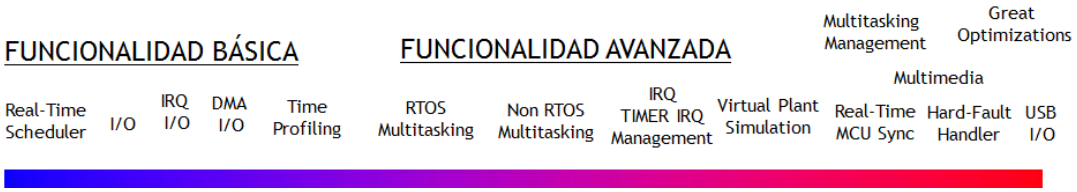


Figura 5.1: *Esquema de las funcionalidades cubiertas por el sistema ARCP.*

5.1. Funcionalidades Básicas

Las funcionalidades básicas en SIMULINK® para el MCU STM32F4 están cubiertas por la toolbox de la compañía Aimagin Ltd®, revisada en el apartado 2.4.6. El nivel de funcionalidad cubierto por esta toolbox es el mostrado en la figura 5.2.

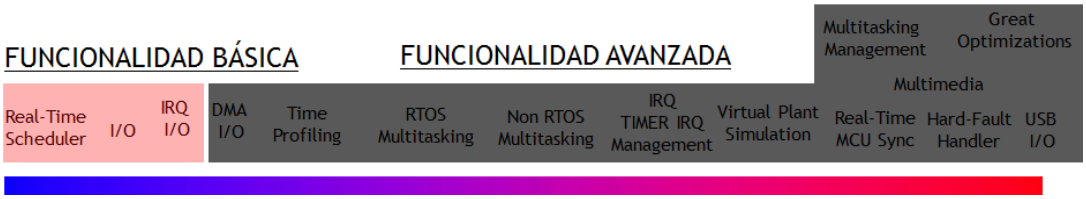


Figura 5.2: *Esquema de las funcionalidades básicas (en rojo) cubiertas por la toolbox de Aimagin Ltd®. En color oscuro se encuentran sombreadas las funcionalidades que son consideradas no básicas.*

Estas funcionalidades básicas cubren la configuración del MCU de la familia STM32F4 elegido y la generación de código fuente en tiempo-real monotarea con múltiples tiempos de muestreo, así como la configuración y utilización de los siguientes periféricos; en definitiva permite utilizar los recursos de I/O y gestiona los medios de comunicación UART y CAN por medio de IRQ:

- Conversor A/D en modo básico con 12 bits de resolución; aunque en el MCU STM32F4 existen tres conversores A/D para gestionar los 16 canales disponibles, mediante la toolbox de funcionalidad básica sólo es posible utilizar uno de ellos.
- I/O mediante puertos digitales.
- I/O mediante puerto SPI e I2C
- Salidas del conversor DAC
- Salidas PWM
- I/O por IRQ de los puertos UART
- I/O por IRQ del bus CAN
- Lectura de imágenes desde cámaras con interfaz DCMI
- I/O para tarjeta de memoria SD

También permite utilizar algunas características adicionales:

- Uso del reloj calendario
- Forzar reseteo del sistema
- Utilizar la unidad de cálculo de chequeo de redundancia cíclica (CRC)

Esta toolbox ha sido diseñada con la intención de resultar fácil de utilizar, no requiere un conocimiento profundo del hardware a la hora de utilizar los bloques de configuración de los periféricos; como ejemplo del caso opuesto se puede citar la toolbox de ST Microelectronics® revisada en el apartado 2.4.8.

5.1.1. Correcciones a las funcionalidades básicas

La toolbox para MATLAB®/SIMULLINK® de de Aimagin Ltd.® provee de las funcionalidades básicas para el MCU STM32F4. Debido a su importancia, no es tolerable que las funcionalidades básicas tengan defectos o limitaciones severas que impidan el correcto funcionamiento en todas las situaciones.

Originalmente la I/O digital de propósito general (GPIO) utilizaba direccionamiento a nivel de bits, pero para determinadas combinaciones no funcionaba bien. En el desarrollo de esta tesis este error de funcionamiento ha sido corregido utilizando las funciones de escritura/lectura de puertos GPIO suministradas por ST Microelectronics®.

La gestión de la IRQ de recepción del bus CAN tampoco funcionaba bien, al llegar dos paquetes de datos sin haberse atendido el primero se producía un cuelgue del sistema. Ha sido corregido permitiendo la acumulación de varios paquetes de datos y la acumulación de solicitudes de IRQ por parte del bus CAN.

El puerto I2C requiere de un tiempo de espera, denominado "timeout", de forma que si transcurrido dicho tiempo no ha terminado la transacción de datos, ésta se dé por terminada, porque con toda seguridad ha acontecido un problema de cableado entre MCU y dispositivo con interfaz I2C. La implementación del tiempo de espera ha sido reimplementada pues no actuaba y consumía recursos adicionales. Ahora utiliza consultas al valor del timer del sistema.

La corrección de estos fallos forma parte de la toolbox de funcionalidades avanzadas y modifica los archivos necesarios de la toolbox de funcionalidades básicas.

5.1.2. Añadidos a las funcionalidades básicas. Compilación Automática

Originalmente las toolboxes básica y avanzada sólo generaban código fuente en lenguaje C; formaba parte de las responsabilidades del usuario el compilar el código fuente y grabarlo en el MCU. Para solucionar esto, se obliga a MATLAB® a ejecutar código en lenguaje M adicional antes de dar por terminado el proceso de generación de código fuente. El código M adicional llama por línea de comandos a los programas necesarios para compilar y grabar el modelo de MCU seleccionado. En esta tesis se estudió cómo compilar en el entorno Keil Uvision® por línea de comandos y cómo grabar el resultado en el MCU, por medio de la utilidad ST-LINK Utility. Se añadió esta opción al bloque de SIMULINK® de configuración del MCU, así como la posibilidad de modificar las opciones de compilación para Keil®. Aimagin Ltd.® en colaboración con la UC3M otorgó esta capacidad para el compilador gratuito GNU ARM y para IAR EWARM®.

5.2. Funcionalidades Avanzadas

Las funcionalidades avanzadas de las que se ha dotado al sistema ARCP presentado en esta tesis están estrechamente relacionadas con las características a nivel de arquitectura interna de cada equipo computacional soportado. Unas u otras características u optimizaciones sólo podrán estar disponibles si la arquitectura hardware provee de los recursos necesarios para llevarlas a cabo. Además de este requisito, es necesario también dotar al mecanismo de generación de código fuente de MATLAB® de nuevas características que no están previstas por defecto.

Las funcionalidades avanzadas que se han integrado en el sistema ARCP se muestran sombreadas en azul en la figura 5.3.

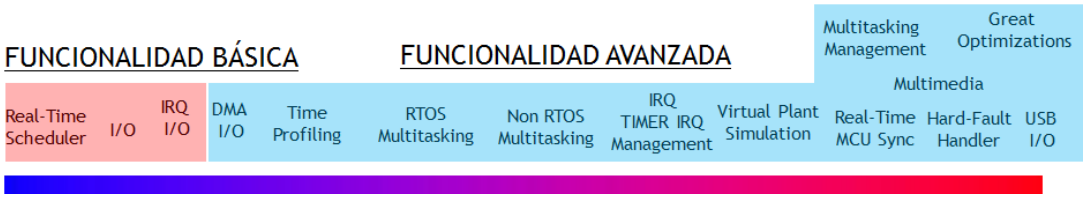


Figura 5.3: *Esquema de las funcionalidades avanzadas (en azul) cubiertas por la toolbox presentada en esta tesis.*

Estas posibilidades avanzadas dotan al usuario del entorno de programación basado en gráficos de una gran libertad de acción, permitiendo resolver un mismo problema de múltiples formas diferentes, con la garantía de estar creando controladores embebidos cuyo código fuente se encuentra optimizado de cara al rendimiento y a los requisitos exigidos por las normativas MISRA C 2004, IEC 60880 (sistemas control energía nuclear) e IEC 26262 (sistema de control para autómovil) .

Las características de optimización del rendimiento hacen un uso intensivo de las transacciones de datos con acceso directo a memoria (DMA) y de las distintas ubicaciones de memoria RAM en los procesadores soportados. Asimismo otras características hacen uso de las nuevas propiedades incorporadas al mecanismo de generación de código de MATLAB®. En sucesivos apartados irán estudiándose estas características avanzadas detenidamente; para empezar conviene explicar cómo está configurada la arquitectura computacional del MCU STM32F4 y cómo se puede obtener un mayor rendimiento a la misma.

5.2.1. Arquitectura MCU STM32F4

La arquitectura del MCU STM32F4 se muestra, de forma simplificada, en la figura 5.4.

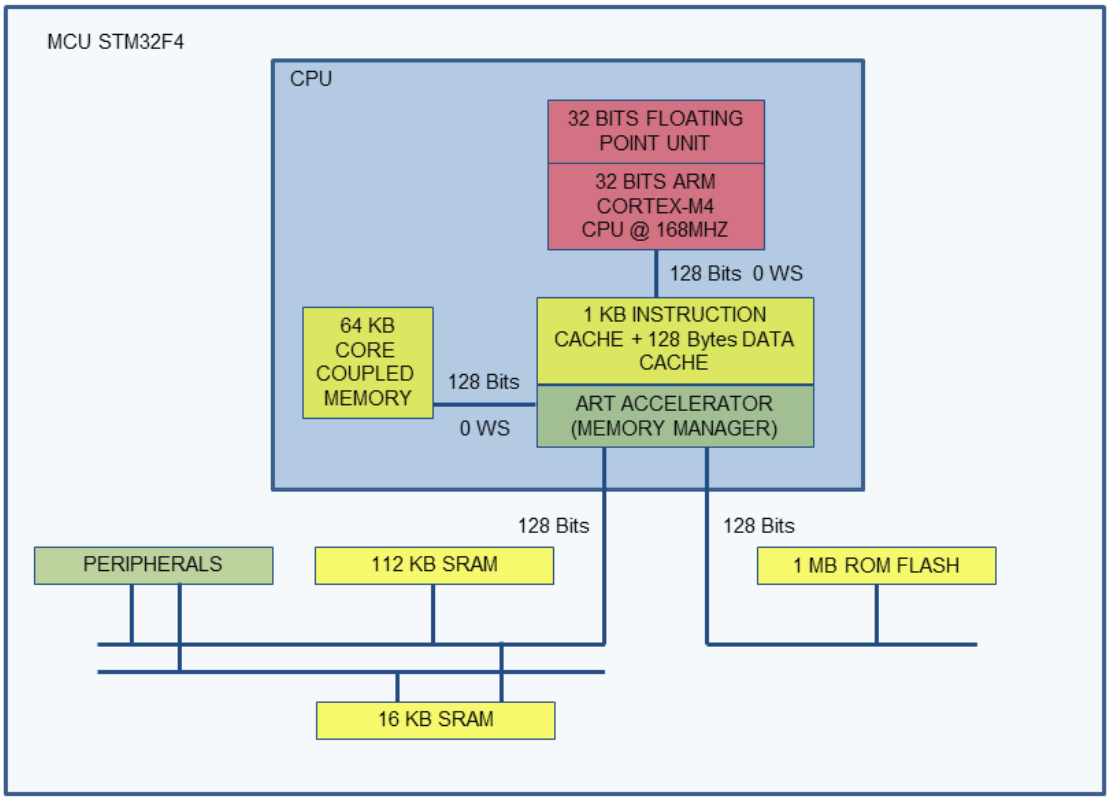


Figura 5.4: *Esquema simplificado de la arquitectura interna del MCU STM32F4*

En este MCU la CPU es un ARM-Cortex M4F de 32 Bits con encolamiento de 4 instrucciones, la figura 5.5 ilustra el concepto de encolar instrucciones. Incluye 1 KB de memoria caché (esto depende del fabricante del MCU, no de la CPU utilizada) y unidad de coma flotante (FPU) con soporte para números decimales en precisión simple de 32 Bits. Destaca la distribución de su memoria RAM, en total están presentes 192 KB de RAM, memoria que se encuentra distribuida en 3 secciones, una de ellas integrada dentro de la CPU. La memoria de programa, memoria ROM, alcanza un tamaño de 1 MB. Todos los buses de memoria poseen un ancho de 128 bits; respecto a los ciclos de espera, a 168 MHz tanto la memoria caché como la memoria acoplada al núcleo (en inglés *Core Coupled Memory*, CCM) no tienen ciclos de espera, lo que hace que estas memorias dispongan de un ancho de banda de 2.625 GB/s, mientras que la memoria ROM y RAM externas a la

CPU requieren 5 ciclos de espera, haciendo que el ancho de banda para cada una de estas memorias alcance 448 MB/s, recordando que la memoria ROM y RAM son accedidas con buses diferentes. El gestor de memoria (ART Accelerator) de ST Microelectronics® se encarga de alimentar la memoria caché e identificar los datos en memoria RAM referenciados por las instrucciones que está recuperando, a su vez recoge estos datos de la memoria RAM sin necesitar la intervención de la CPU.

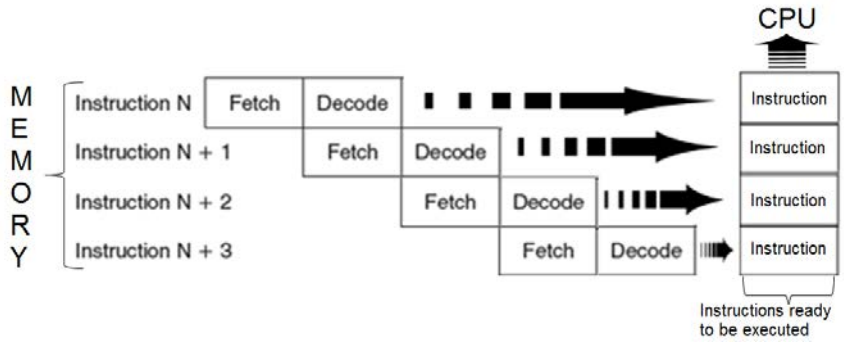


Figura 5.5: *Encolamiento de 4 instrucciones en la CPU del MCU STM32F4*

Estas características no son comunes entre los distintos microcontroladores que pueblan el mercado, en cambio si son más habituales entre controladores hardware más especializados, de los que se espera un mayor rendimiento, se trata de dispositivos orientados a tratamiento digital de señales (del inglés Digital Signal Processor, DSP) en coma flotante. El rendimiento que se puede obtener de un MCU STM32F4 es cercano al que proporciona un MCU MPC555, a un precio muy inferior.

Las funcionalidades avanzadas hacen uso de estas características respecto de la memoria RAM, así como lograr que MATLAB®/SIMULINK® puedan generar código orientado a este tipo de arquitecturas.

Las funcionalidades de carácter avanzado propuestas en esta tesis se fundamentan en el soporte hardware disponible en cada arquitectura soportada, dado que se prescinde de emplear capas de abstracción software tales como RTOS en beneficio de la predectibilidad, determinismo y sencillez del sistema ARCP que redundan en una mayor fiabilidad. Estas funcionalidades avanzadas engloban desde exprimir el uso de las distintas secciones de memoria hasta el poder disponer de un mecanismo que permita la multitarea sin necesidad de sistema operativo. En las siguiente secciones se irán detallando cada una de estas funcionalidades desarrolladas en esta tesis.

5.2.2. Optimizaciones en el uso de la memoria RAM. Uso de la memoria CCM

En toda arquitectura computacional las instrucciones que resultan más lentas en su ejecución son las relativas a la realización de cálculos con números decimales en coma flotante, incluso en las arquitecturas que cuentan con FPU; también trabajar con flotantes de 64 bits resulta más lento que hacerlo con homólogos de 32 bits.

En la arquitectura del MCU STM32F4, existen 64 KB de memoria integrados en la CPU (véase la figura 5.4), esta memoria no es accesible por los periféricos del MCU, pero la CPU posee la peculiaridad de poder ejecutar instrucciones que residan en este espacio de memoria. Al hacerlo se consigue que la CPU ejecute las instrucciones contenidas en estos 64 KB y el gestor de memoria recupere las variables referenciadas por estas instrucciones de alguna de las otras secciones de memoria RAM, logrando que ambos ejercicios de direccionamiento se realicen de forma simultánea, maximizando el rendimiento. MATLAB® no posee mecanismos innatos para ubicar datos y/o instrucciones en diferentes secciones de la memoria RAM.

Por tanto, la toolbox de funcionalidades avanzadas presentada en esta tesis permite que todas las instrucciones del repertorio "Digital Signal Processing" (DSP) de la librería de SIMULINK® se ejecuten desde la memoria CCM; así como otros usos para esta memoria. Las funciones DSP están basadas en cálculos iterativos sobre números en coma flotante, por tanto son las más lentas en su ejecución, ubicar estos bucles de código máquina en una memoria muy rápida y de fácil acceso para la CPU hará que se ejecuten mucho más deprisa.

Para lograr estos objetivos se debe modificar el proceso estándar de generación de código fuente de MATLAB®/SIMULINK®, es necesario que las funciones DSP generen su código fuente en forma de una función reutilizable, en lugar de escribir directamente su código dentro de la función que ejecuta la funcionalidad descrita en el modelo gráfico. Para ubicar código máquina ejecutable en la memoria CCM, es necesario indicar al linkador qué funciones en lenguaje C irán ubicadas en la memoria CCM, no se pueden ubicar partes de funciones, sino funciones completas. Esta funcionalidad no requiere la intervención directa del usuario, sino que se realiza de forma automática.

La figura 5.6 ilustra dónde se ubican las modificaciones añadidos al proceso de generación de código fuente de MATLAB®; el primer añadido respecto a la figura 4.3 viene tras la generación de la estructura "Simulink.RTW Model", una nueva

función revisa por completo la estructura con la finalidad de localizar los identificadores propios de las funciones DSP, a continuación establece para cada funcionalidad DSP que el código a generar para ella sea código reutilizable, por lo que MATLAB generará el código que le corresponda como una función en lenguaje C independiente, también se establece que el tipo de datos más complejo a utilizar en los cálculos DSP sean flotantes de 32 bits y se modifica la llamada al lenguaje TLC de cada función DSP, para que se añada el prefijo " attribute ((section".ccm"))" a la declaración de la función DSP, de forma que el linkador ubique estas funciones en la memoria CCM. Se ubicarán en el momento del arranque del MCU, ya que al principio todo el código ejecutable se encuentra en la memoria ROM.

La definición de dónde se encuentra la sección de memoria correspondiente al mnemónico "ccm" se ha realizado en el código de arranque y configuración de los relojes escrito en ensamblador, necesario para toda CPU basada en ARM.

Por motivos de seguridad y para mantener la integridad del código ejecutable generado a partir del código fuente, se ha llevado a cabo un script para revisar los resultados de la compilación del código, comprobando si los códigos objeto ubicados en la memoria CCM superan los 64 KB disponibles, si este fuera el caso, el programa volvería a compilarse ubicando una de las funciones ubicadas en dicho espacio de memoria en la ROM convencional, si el resultado siguiera siendo negativo, se eliminarían dos funciones... (el script es válido para Keil Uvision®). Por lo general, las funciones DSP ocupan poco espacio, por ejemplo un filtro paso banda de orden 128 ocupa 237 bytes de código máquina, aunque resulta pesado computacionalmente porque el contenido de la función se ejecuta 128 veces. Los resultados en tiempo de cómputo para el citado ejemplo reflejan 710 us ejecutándose desde la memoria flash y 110 us ejecutándose desde la memoria CCM. Por tanto, el rendimiento ha pasado de 6 millones de instrucciones en coma flotante (MFLOPS) a 41 MFLOPS al utilizar la memoria CCM para esta función DSP. Estos cálculos de rendimiento no tienen en cuenta las multiplicaciones por cero, puesto que se eliminan en la etapa de compilación. En el apartado 8.2 se presentan más pruebas experimentales respecto a las ventajas de emplear la memoria CCM.

Otra funcionalidad que se ha trasladado a la memoria CCM, es la memoria de pila de la CPU, lo que ha permitido que el cambio de contexto de la CPU se realice de forma mucho más rápida. Más adelante, en las secciones relativas a multitarea se explicará en qué consiste el cambio de contexto de una CPU, importante de cara a mejorar el determinismo del sistema.

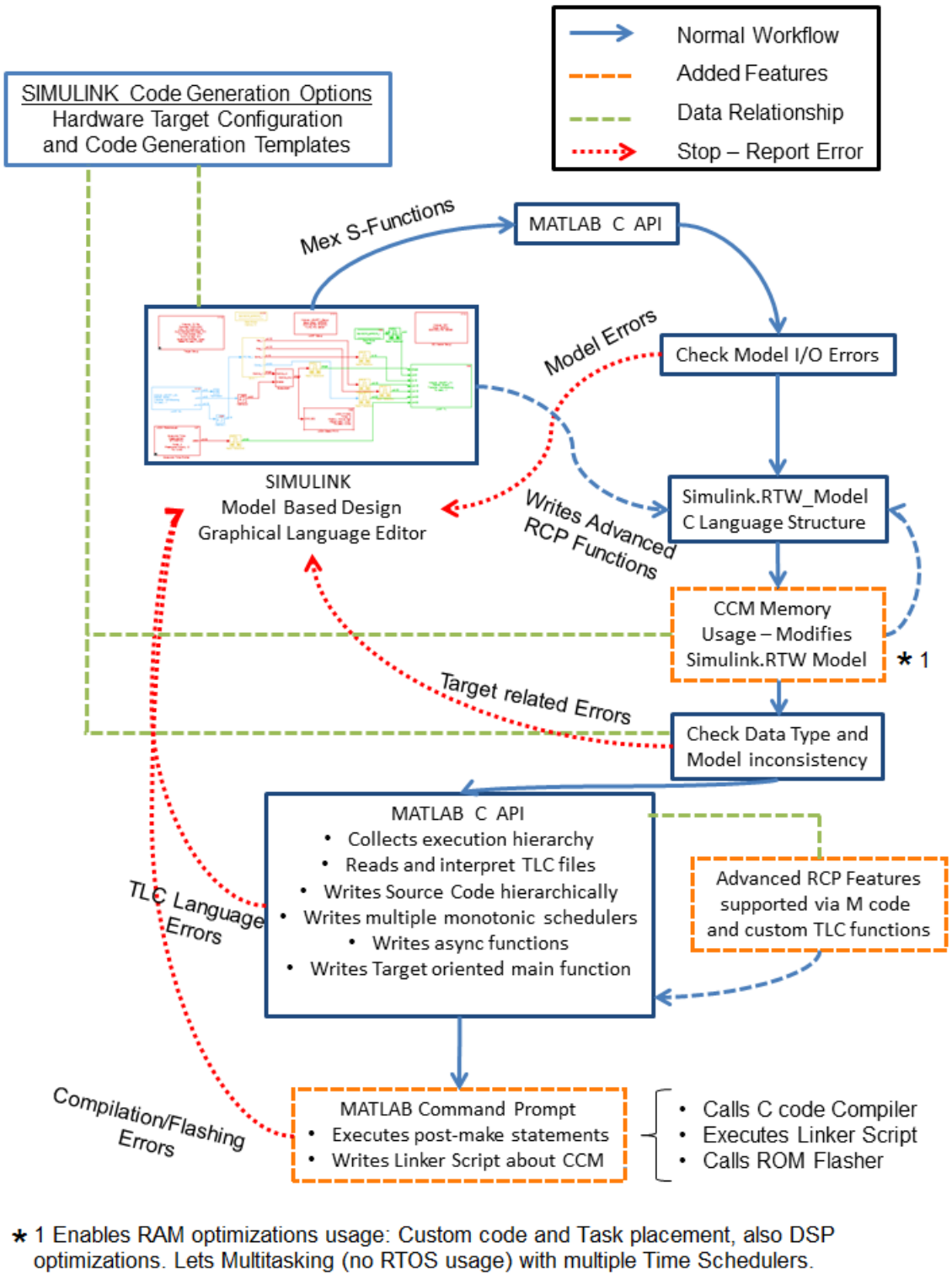


Figura 5.6: *Nuevo esquema con soporte a funcionalidades avanzadas en la generación de código fuente a partir de un modelo gráfico en MATLAB®*

Los siguientes bloques gráficos han sido diseñados para proveer de un alto nivel de abstracción y simplificar el uso de la memoria CCM; se pueden ubicar las tareas en tiempo-real y también se puede ubicar el código de cualquier función en dicha memoria. También se puede deshabilitar la autodetección de las funciones DSP, de forma que éstas no se ubiquen automáticamente en la memoria CCM. La figura 5.7 ilustra los bloques gráficos que permiten realizar estas acciones.

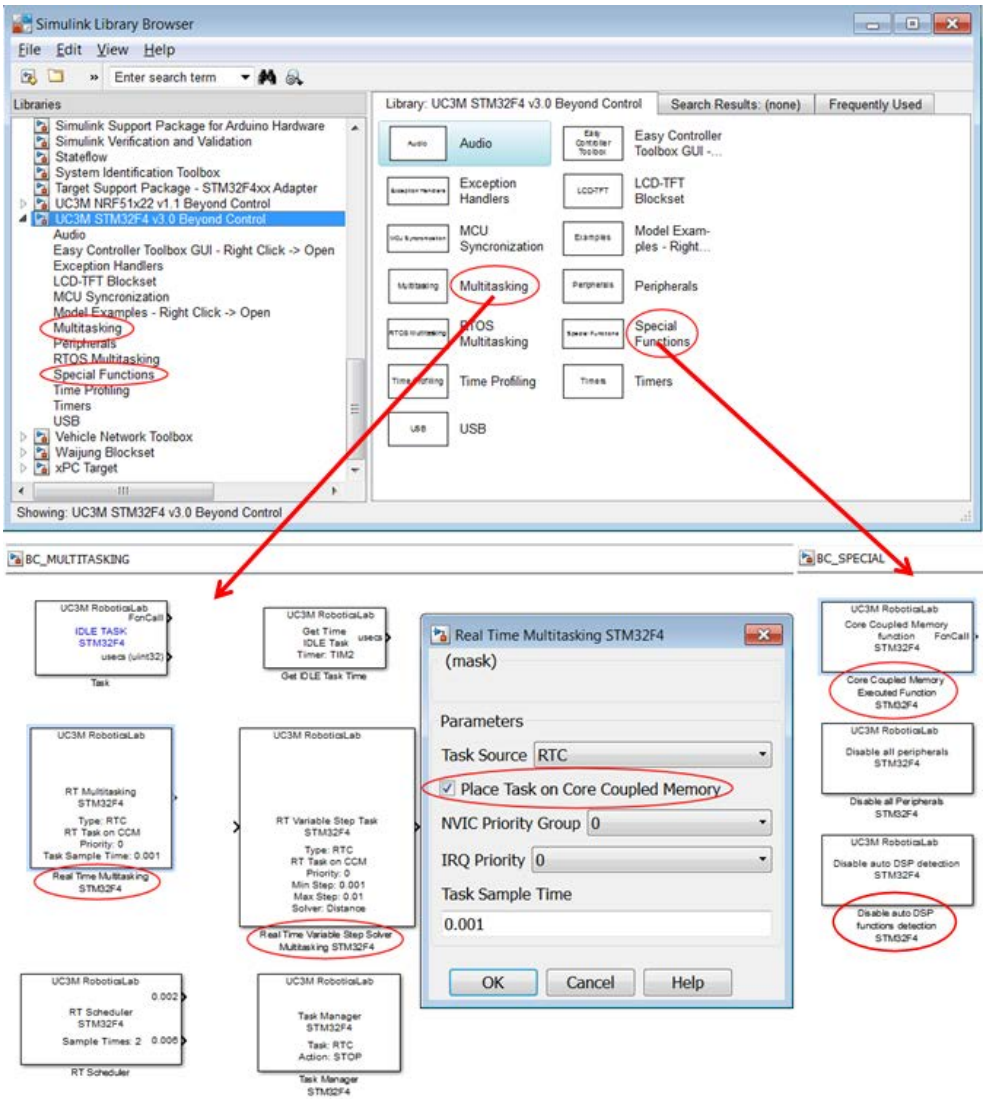


Figura 5.7: *Funcionalidades avanzadas de programación gráfica que hacen uso de la memoria acoplada al núcleo (CCM)*

5.2.3. Periféricos con propiedades DMA

El acceso directo a memoria (del inglés *Direct Memory Access*, DMA) es una propiedad por la que un dispositivo hardware, en el caso de un MCU suele ser un periférico de I/O, puede realizar movimientos de lectura/escritura en la memoria RAM sin recurrir al concurso de la CPU del sistema; de esta manera las transacciones de datos que realice el periférico mediante DMA no interrumpirán la ejecución del programa en la CPU.

En el caso de que el bus de memoria sea único, la transacción de datos tendrá lugar en los momentos en que la CPU no esté utilizando la memoria. En el caso del MCU STM32F4, se dispone de un complejo mecanismo de arbitraje con una serie de buses conectados en forma de matriz, por lo que es posible que un periférico realice transacciones por DMA mientras la CPU continua con un funcionamiento ininterrumpido. La figura 5.4 ilustra dos ubicaciones de memoria para la que los periféricos de I/O tienen acceso, una de 112 KB y otra de 16 KB, por tanto ambas memorias RAM son factibles de ser utilizadas mediante DMA. Con el fin de aprovechar al máximo las capacidades que brinda este MCU, se utilizará en primera instancia la memoria RAM de 16 KB como memoria dedicada a transacciones DMA de los periféricos I/O, mientras que los 112 KB restantes serán utilizados como memoria RAM de propósito general.

La toolbox de funcionalidad básica de Aimagin Ltd.® utiliza IRQ para gestionar la I/O de los periféricos de comunicaciones UART, Ethernet y bus CAN. En la toolbox de funcionalidades avanzadas presentada en esta tesis, estos puertos de comunicación ahora utilizan transacciones DMA y sus fuentes de IRQ son opcionales, dado que el usuario que plantea el modelo gráfico puede desear que la CPU ejecute un subsistema cuando se termine de recibir un paquete de datos, función asíncrona que debe utilizar IRQ. Pero ya no poseé un carácter obligatorio, por lo que resulta posible que estos puertos de comunicación reciban datos y los ubiquen en memoria sin intervención de la CPU, para que sean leídos llegado el momento temporal correspondiente. También se aplican transacciones DMA para enviar los paquetes de datos, el programa tan sólo escribe en el vector correspondiente en la memoria RAM los valores a enviar y da la orden al periférico de iniciar un envío mediante DMA, sin ser necesario ejecutar funciones adicionales. A nivel técnico, una transacción de datos mediante DMA requiere que el vector de datos se encuentre alineado en memoria, no puede estar segmentado. Para ubicarlos en una u otra sección de memoria, se procede de manera idéntica a como se han ubicado funciones en la memoria CCM.

Los periféricos soportados que hacen uso de DMA son los siguientes en orden de prioridad: Triple ADC, USB, UART, bus CAN, Ethernet, Video FSMC, Audio I2S. Al agotarse la memoria de 16 KB, se procede a utilizar el bloque de 112 KB para transacciones DMA.

5.2.4. Optimización del conversor A/D. Triple DMA-ADC

La toolbox de funcionalidad básica de Aimagin Ltd.® provee de acceso a los 16 canales A/D de forma muy básica, de los 3 conversores ADC presentes en el MCU STM32F4 sólo utiliza uno de ellos. En esta tesis esta situación ha sido corregida mediante un nuevo bloque gráfico para SIMULINK®, el número de conversores a utilizar se configura automáticamente dependiendo del número de canales a leer, cada 5 canales se utiliza 1 conversor más, de forma que el tiempo necesario para realizar una lectura permanezca inferior a 1 us. También es posible aumentar el número de bits resultante de la conversión, se puede elevar a 13 o a 14 bits haciendo uso de las técnicas de sobremuestreo y diezmado (en inglés *Oversampling and Decimation*). Siempre que se seleccione un número de bits mayor de 12, se utilizarán los 3 conversores pues de otra manera no sería posible llevar a cabo las técnicas de sobremuestreo.

Esta funcionalidad A/D mejorada utiliza transacciones DMA, el resultado de las conversiones es guardado automáticamente en el bloque de memoria RAM de 16 KB, pues tan sólo son necesarios 32 bytes para los 16 canales A/D, ahorrando tiempo a la CPU pues no es necesario ejecutar las funciones de recuperación de datos del conversor A/D. La figura 5.8 muestra el bloque gráfico en SIMULINK®.

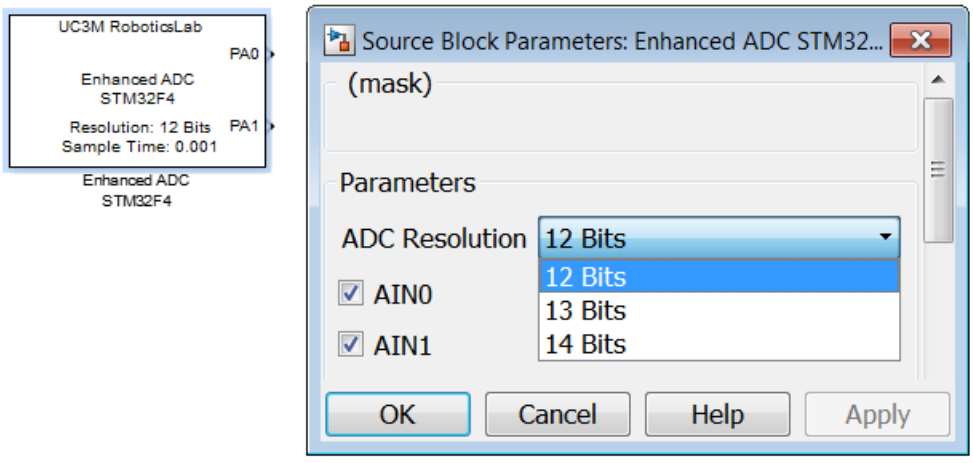


Figura 5.8: *Modos de funcionamiento y gestión de las comunicaciones a través de un puerto USB®*

5.2.5. Comunicaciones USB de alto rendimiento

El sistema ARCP propuesto en esta tesis utiliza el puerto de comunicaciones bus serie universal (del inglés Universal Serial Bus, USB®) como medio principal de intercambio de información entre MATLAB® y el controlador hardware por la facilidad y rapidez en su conexión y utilización. El puerto USB se utiliza con el propósito de monitorizar el comportamiento del controlador enviando y recibiendo información, o también con la finalidad de ser utilizado como medio de comunicación en la implementación final [58].

Resulta beneficioso disponer del mayor ancho de banda disponible y optimizarlo para que el puerto USB represente la menor carga de trabajo posible para la CPU. Tradicionalmente, las posibilidades de gestionar la comunicación del puerto USB® comprenden dos modalidades, descritas en la figura 5.9, denominadas modo 1 y modo 2. El tamaño máximo de los paquetes de datos que admite el puerto USB® es de 64 Bytes, aunque es posible enviar paquetes de datos más largos utilizando el modo 2, modalidad en la que al concluir el envío de un paquete de 64 bytes, acontece una IRQ para que la CPU rellene el buffer del puerto USB con los siguientes 64 bytes de datos.

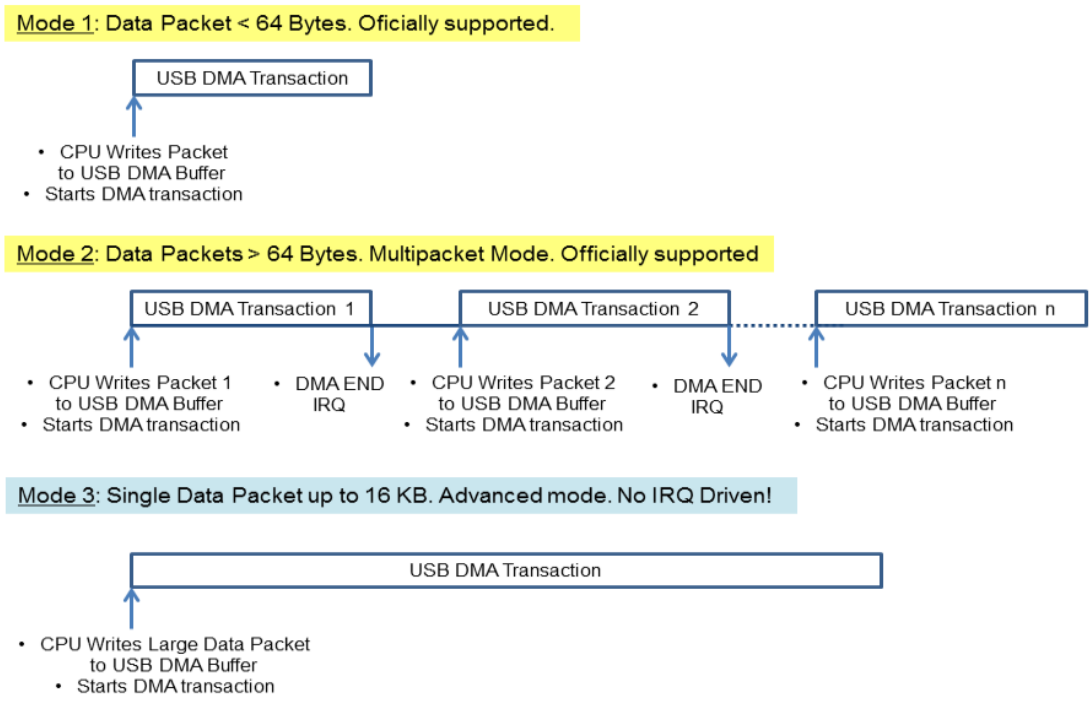


Figura 5.9: *Modos de funcionamiento y gestión de las comunicaciones a través de un puerto USB®*

Los modos 1 y 2, mostrados en la figura 5.9, representan lo dictado por el estándar USB® 1.1/2.0 . Los dispositivos USB requieren de la intervención de dos partes, el dispositivo anfitrión (Host), en este caso un computador PC y el dispositivo que implementa el medio de comunicación (Device). Sólo los dispositivos "Device" que no excedan las características dictadas por el estándar serán reconocidos por el dispositivo "Host".

Estos modos tienen la ventaja de permitir que el resto de dispositivos conectados al bus tomen el testigo del mismo y realicen sus transacciones de datos, por contra el modo 2 recurre a numerosas interrupciones cuando tiene que transmitir una gran cantidad de datos, reduciendo el ancho de banda aprovechable y sobrecargando la CPU del MCU STM32F4, exigiendo parte del tiempo que debería emplearse para atender los procesos en Tiempo-Real.

En esta tesis de propone emplear un nuevo modo de comunicación, no basado estrictamente en el estándar, el modo 3. Modalidad de comunicación via USB® en la que la CPU del sistema rellena el buffer de transmisión con un paquete de datos de hasta 16 KB y da la orden de comenzar una transcacción DMA por parte del puerto USB®, sin necesitar una gestión basada en IRQ. El puerto USB enviará los datos del buffer (ubicados en la memoria RAM de 16 KB, siempre que sea posible) mientras la CPU continua trabajando los procesos en Tiempo-Real. Generalmente en sistemas de control no se utilizan paquetes de datos tan grandes, por lo que la memoria RAM de 16 KB presente en el MCU STM32F4 suele servir muy bien a los propósitos de transacciones DMA.

Para poder utilizar un dispositivo de comunicaciones USB® CDC (del inglés *Communication Device Class*) que no cumpla estrictamente con los estándares, es necesario seguir un proceso de inicialización alternativo, este proceso se encuentra ilustrado en la figura 5.10. Al conectar el CDC al computador Host, se producirá una IRQ en el MCU relativa al puerto USB®, en la que habrá que informar al Host de las características del dispositivo CDC, éstas serán características comprendidas en el estándar, en caso contrario al dispositivo CDC no se le permitiría iniciar informando de un error de tipo 10 (información visible en el administrador de dispositivos del Host). Una vez iniciado el dispositivo CDC, se puede comenzar con la vía alternativa para poder utilizar el modo 3; se procede a dormir el periférico USB y se reconfigura el tamaño de sus búfferes de transmisión/recepción, se despierta y la solicitud del computador Host provocará una IRQ en el dispositivo CDC para gestionar el despertar del periférico USB®, se envía la información actualizada sobre el tamaño de los búfferes y se activa el flag del modo 3, para permitir que el controlador embebido pueda comenzar con las transacciones de datos por

DMA de está forma tan optimizada. El dispositivo CDC, al despertar, no tiene que volver a ser reconocido puesto que esto ya se ha hecho con anterioridad (se deben mantener los valores de identificación básicos PID y VID), esta es la clave para implementar cualquier tipo de dispositivo USB® con las características que se deseen; obligar a que sea reconocido como un dispositivo estándar y a continuación modificar sus propiedades sin que se produzca la desconexión del dispositivo USB®. Si se modificasen las propiedades del dispositivo USB® sin dormirlo, se produciría automáticamente la desconexión del mismo y al despertarlo tendría que volver a ser reconocido y al no ser un dispositivo estándar, no se le permitiría iniciar.

El dispositivo USB® CDC utilizado en la toolbox de funcionalidades avanzadas para el MCU STM32F4 integra la información y funciones necesarias para ser reconocido por el ordenador Host como un puerto serie estándar (COM). La configuración del ancho de banda en baudios no se aplica, esta opción tan sólo se encuentra disponible para que el puerto serie COM sea compatible con programas ideados para comunicarse con un puerto COM basado en RS-2332, en el que si es necesario especificar el ancho de banda en baudios. De esta forma, la interfaz de comunicaciones por USB® del sistema ARCP es compatible con todos los programas, librerías y lenguajes de programación que puedan comunicarse con un puerto serie COM basado en RS-232, con la ventaja de que el ancho de banda disponible es el seleccionado para el medio USB®, que puede ser USB 1.1 (12 Mbits/s) o USB 2.0 (480Mbit/s).

Las pruebas para la modalidad USB 1.1 reflejan los siguientes resultados:

- En MATLAB® se alcanzan los 2 KHz y 1.28 MB/s en modo normal y 3 KHz y 1.28 MB/s en modo acelerador
- Con un programa en lenguaje C se alcanzan los 10 KHz y 1.28 MB/s

Para la modalidad USB 2.0 se obtienen estos resultados:

- En MATLAB® se alcanzan los 2 KHz y 6.32 MB/s en modo normal y 3 KHz y 6.32 MB/s en modo acelerador
- Con un programa en lenguaje C se alcanzan los 10 KHz y 6.32 MB/s

La modalidad USB 1.1 alcanza el techo marcado por el ancho de banda, llegando a 1.28 MB/s máximos. En USB 2.0, pese a que los datos se transfieren a una velocidad de reloj superior, el periférico USB y la CPU del MCU STM32F4 no pueden gestionar más de 6.32 MB/s de datos. Para los propósitos del sistema ARCP propuesto, la modalidad de 12 Mbits/s y sus envíos/recepciones de hasta 10 KHz son más que suficientes.

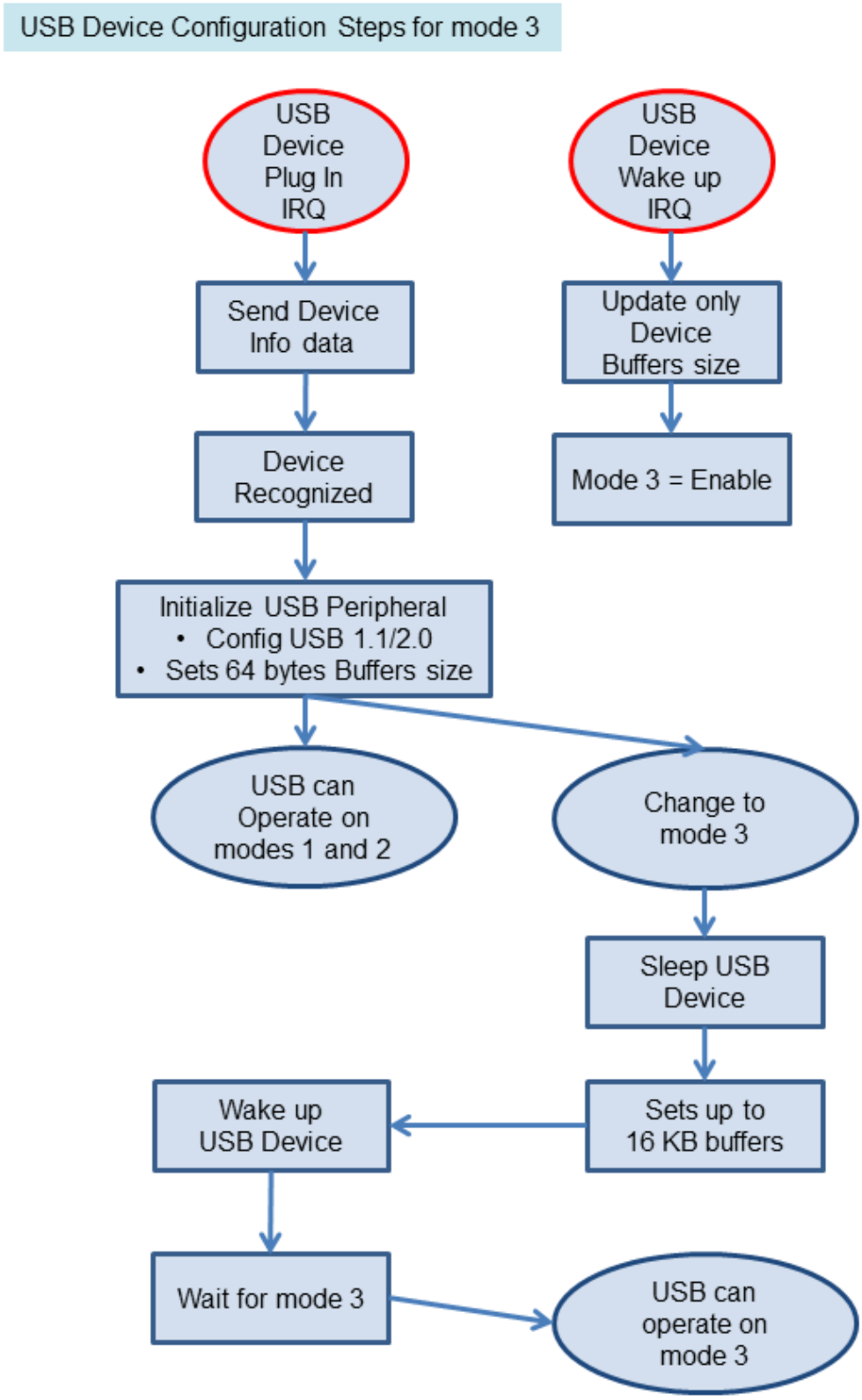


Figura 5.10: *Proceso de inicialización necesario para poder utilizar la modalidad de transacciones de datos propuesta en esta tesis*

Además de la comunicación vía USB con la modalidad de Puerto Serie Virtual, se ha analizado y llevado a cabo la comunicación USB a través de la modalidad de dispositivo de interfaz humana HID (del inglés, 'Human Interface Device'). Esta modalidad presenta grandes ventajas, no requiere software controlador específico para el dispositivo implementado y resulta mucho más sencillo preparar, iniciar y llevar a cabo las funciones de comunicación. Desde el punto de vista de la gestión de IRQ, sólo el dispositivo Host (ordenador PC, por ejemplo) incurre en una IRQ al conectar el dispositivo, pero si el MCU con dispositivo USB HID se desconecta, no acontece ninguna IRQ en éste para tratar esta situación.

Pese a las simplificaciones a nivel técnico de USB HID, el autor de esta tesis aún no ha logrado romper la barrera de los 64 bytes por paquete al emplear modalidad de comunicación HID, resulta útil y necesario poder enviar/recibir paquetes mayores de 64 bytes.

La librería software utilizada para dar soporte a la modalidad HID en MATLAB® es 'HIDAPI' [39]. El software empleado para detectar y depurar el dispositivo USB HID del MCU STM32F4 es 'USB HID Analyzer'.

En la figura 5.11 se muestra la detección del dispositivo, el nombre del mismo, los paquetes de datos recibidos y el tamaño configurado para los bufferes.

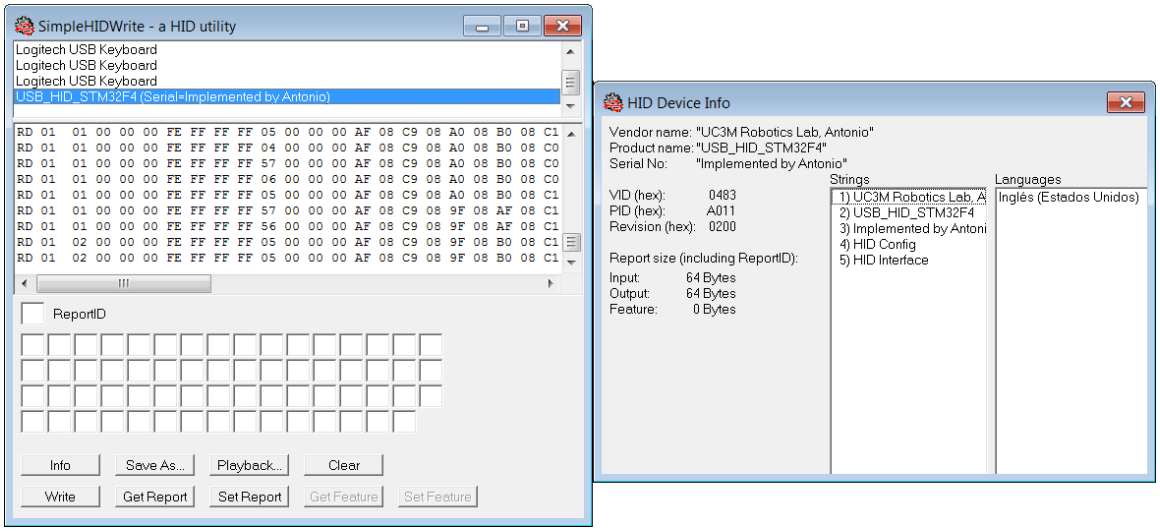


Figura 5.11: *Identificación e información del dispositivo USB HID del MCU STM32F4*

[39] http://www.signal11.us/oss/hidapi/ , librería HIDAPI, última consulta Septiembre 2014

El medio de comunicación mediante USB HID permitirá dotar al sistema ARCP basado en PlayStation 2® (véase apartado 7.1.5) de la capacidad de comunicarse también por vía USB con el MCU STM32F4; mientras que sería muy complejo escribir un controlador de dispositivo USB específico para el citado sistema.

La figura 5.12 muestra la librería de funcionalidades para puerto USB puestas a disposición del lenguaje gráfico de SIMULINK® por la toolbox de funcionalidades avanzadas.

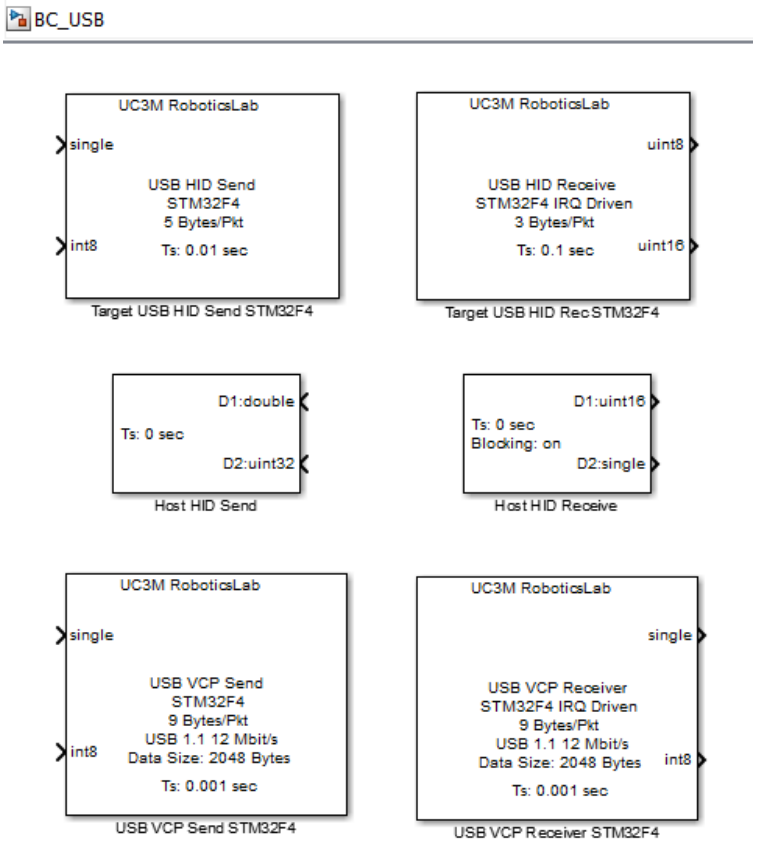


Figura 5.12: *Bloques de programación gráfica disponibles en la toolbox de funcionalidades avanzadas para manejar el puerto USB tanto del MCU STM32F4 como del ordenador PC*

5.2.6. Reprogramación de los manejadores de excepción

Los manejadores de excepción (en inglés, 'Exception Handlers') son funciones de código que no se ejecutan en condiciones normales, pero sí se ejecutan cuando acontece un error grave que impide el correcto funcionamiento del programa, con la intención de poder devolver la ejecución a un estado normal.

Estos manejadores de excepciones se ejecutan al acontecer un fallo imprevisto tal como pueda ser el desbordamiento del puntero de pila del procesador, el salto a una dirección de memoria incorrecta, una incorrecta configuración de un periférico, una configuración incorrecta de los relojes del sistema...

La toolbox de funcionalidades avanzadas adopta dos posturas respecto a los manejadores de excepciones:

- Primero: En caso de que el usuario no haya reprogramado los manejadores de excepción, el proceso de generación de código modificado, ilustrado en la figura 5.6, escribirá unos manejadores de excepción genéricos que provocan un reseteo del sistema en caso de ejecutarse cualquiera de ellos.
- Segundo: La toolbox de funcionalidades avanzadas pone a disposición del usuario la posibilidad de reprogramar qué se ejecutará en el caso de que se produzca una excepción; se puede optar por alguna las funciones especiales como deshabilitar todos los periféricos y resetear el MCU, así como activar una salida digital para cortar el suministro de energía del sistema...

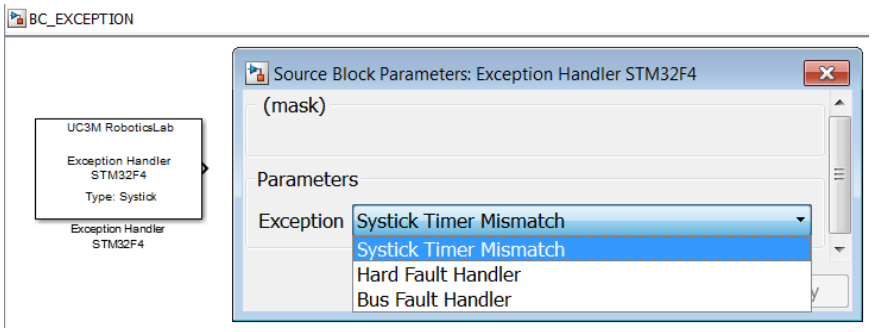


Figura 5.13: *Bloque de programación gráfica disponible en la toolbox de funcionalidades avanzadas para reprogramar los manejadores de excepción*

5.2.7. Soporte para Tiempo-Continuo

La ejecución en tiempo-continuo difiere de la ejecución periódica en que cada iteración del programa no necesita esperar a marcas temporales para ejecutarse; la ejecución en tiempo-continuo permite que el modelo vuelva a ejecutarse inmediatamente, lo que conlleva frecuencias de iteración del modelo de decenas de KHz. Con la ventaja de evitar los límites de una ventana temporal, una iteración podría tomar más tiempo del previsto sin incovenientes.

Para que la ejecución se pueda considerar en tiempo-continuo, la evolución en los estados del sistema mecatrónico que se está controlando debe ser inapreciable o mínima para el tiempo que necesita cada iteración del algoritmo. Ejemplificando, se puede considerar ejecución en tiempo-continuo (utilizando un MCU STM32F4) el controlar un motor de combustión interna a 20.000 revoluciones por minuto si cada iteración del algoritmo se encuentra por debajo de 50 microsegundos.

Originalmente el soporte de tiempo-continuo en el proceso de generación de código fuente de MATLAB® permite dar soporte a las librerías de SIMULINK® basadas en tiempo-continuo, utilizando el contador de marcas temporales por IRQ como resolución temporal. Ejemplificando, si un modelo gráfico se configura para ejecutarse en tiempo-real cada 1 ms, tanto los bloques gráficos basados en tiempo-continuo como los generadores de funciones, sólo puedan actualizar los valores de sus salidas cada 1 ms, sin importar el número de veces que se ejecuten en ese intervalo de 1 ms, lo que conlleva que a nivel de código fuente un modelo en tiempo-continuo tenga un comportamiento idéntico a uno en tiempo-discreto.

Entonces se necesita dotar al código fuente de un mecanismo que provea de una resolución temporal mucho mayor que aquella establecida por la IRQ que marca el ritmo de los procesos en tiempo-discreto. La toolbox de funcionalidades avanzadas de esta tesis integra una nueva función que facilita un valor temporal con resolución de microsegundos, sin necesitar recursos hardware adicionales a los utilizados para gestionar una tarea en tiempo-discreto. Esta nueva función recupera el valor actual del timer de sistema (SysTick Timer) en microsegundos y suma el valor acumulado en el contador de intervalos temporales usado para tiempo-discreto, obteniéndose el valor absoluto de tiempo transcurrido en microsegundos. La nueva función está optimizada y ubicada en la memoria CCM, para que se ejecute lo más rápido y distorsione lo mínimo el cálculo del tiempo absoluto. La función ocupa 24 Bytes de código máquina.

Las funcionalidades avanzadas permiten crear una tarea que se ejecute en tiempo-continuo, además de múltiples tareas en tiempo-discreto simultáneamente. Estas últimas pueden desalojar a la tarea en tiempo-continuo (ésta posee la menor prioridad). Combinar ambas modalidades de tareas permite utilizar el 100 % de los recursos computacionales disponibles y resolver así situaciones difíciles de manejar sólo en tiempo-discreto.

La figura 5.14 muestra una ejemplificación de las bondades de poder utilizar una tarea en tiempo-continuo. La tarea mostrada en la figura es una máquina de estados con tres ramas de ejecución, la forma en que se ejecuta es idéntica al concepto de un autómata industrial, con la ventaja de que las actualizaciones de las condiciones para pasar de un estado a otro se están realizando constantemente, en lugar de a intervalos temporales fijos. También puede considerarse como otra forma de llevar a cabo una ejecución multitarea. De esta manera se pueden resolver problemas en los que una tarea debe quedarse esperando a algún evento originado por otra tarea distinta, sin necesidad de recurrir a implementar esto una tarea en tiempo-real, que no podría quedarse esperando a un evento, pues debería terminar su ejecución dentro de su plazo temporal.

Como ejemplo práctico real, este tipo de ejecución en tiempo-continuo multitarea es necesaria para recuperar los datos de encoders industriales con interfaz SSI (del inglés *Serial Synchronous Interface*) cuya salida de datos no es potencia de 2, y por tanto no resulta sencillo emplear periféricos de un MCU para llevar a cabo esta labor; labor en la que los datos se actualizan rápidamente y no podrían ser recuperados por una tarea de ejecución periódica ni por una máquina de estados con actualización periódica. Este tipo de situaciones no pueden resolverse fácilmente con un sistema RCP convencional, en cambio no plantean mayores problemas al utilizar el sistema ARCP propuesto en esta tesis.

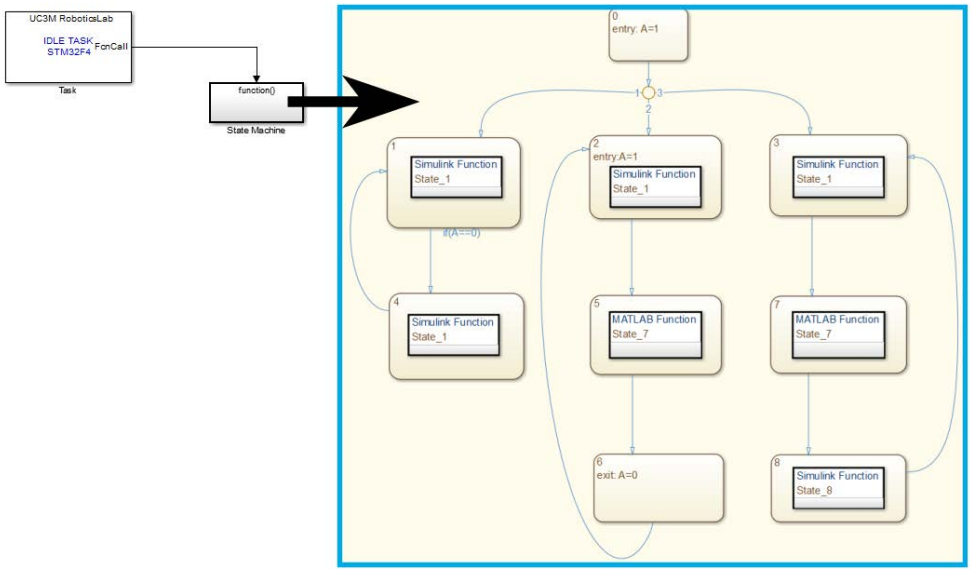


Figura 5.14: *Tarea con tres ramas de ejecución en forma de máquina de estados ejecutándose en tiempo-continuo.*

5.2.8. Soporte para paso de ejecución variable

Las funcionalidades avanzadas incluyen el soporte para crear tareas en tiempo-real cuyo intervalo temporal de ejecución es variable, se conoce de manera más común como tareas con paso de ejecución variable.

La práctica totalidad de los sistemas RCP comerciales permiten disponer solamente de tareas en tiempo-real con paso de ejecución fijo, son tareas que vuelven a ejecutarse a intervalos de tiempo regulares, sin importar cómo hayan evolucionado las circunstancias de su entorno. La figura 5.15 muestra un ejemplo de una tarea en tiempo-real con paso de ejecución fijo, en la imagen se observan unos puntos azules sobre una línea azul, estos puntos representan iteraciones de la tarea. La línea azul representa la referencia o consigna de control de la tarea. Para referencias que evolucionan a baja frecuencia, el intervalo temporal entre iteraciones de la tarea permitirá un buen rendimiento de la misma, en cambio cuando la consigna de control evolucione a frecuencias mayores la tarea no podrá ofrecer un buen rendimiento, puesto que los intervalos temporales de su ejecución se encontrarán excesivamente distanciados unos de otros.

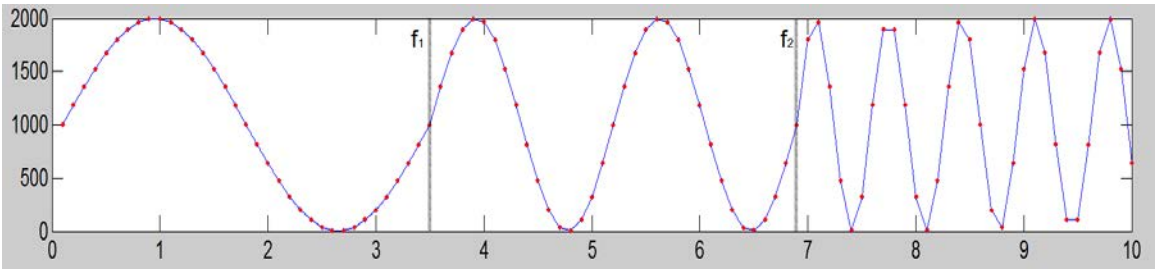


Figura 5.15: *Tarea ejecutándose con paso de ejecución fijo, cada punto rojo indica una iteración de la tarea mientras que la onda senoidal, de frecuencia variable, es la consigna de control de la misma*

En cambio, si se emplea un método que permita variar dinámicamente el lapso de tiempo entre iteraciones de la tarea en tiempo-real, será posible variar automáticamente la frecuencia a la que se ejecuta la tarea, de forma que ésta pueda ofrecer siempre un buen rendimiento. La figura 5.16 muestra cómo cambia dinámicamente la frecuencia de ejecución de una tarea conforme evoluciona su consigna de control en el MCU STM32F4. El experimento de la figura 5.16 emplea un método basado en calcular la distancia (diferencia entre dos medidas) en la magnitud ofrecida por la consigna de control, si la distancia entre dos iteraciones empieza a ser grande, se reprograma automáticamente el lapso temporal del timer que gobierna la tarea para intentar reducir esa distancia, y por ende muestrear mejor la consigna de control. El efecto de este bloque gráfico para crear tareas con paso de ejecución variable puede verse en la figura 5.16 . También es posible utilizar la frecuencia

de la señal de consigna como medio para controlar la frecuencia de iteración de la tarea, técnica que se denomina *zero crossing*. De momento se han desarrollado dos técnicas, pero pueden implementarse técnicas más sofisticadas.

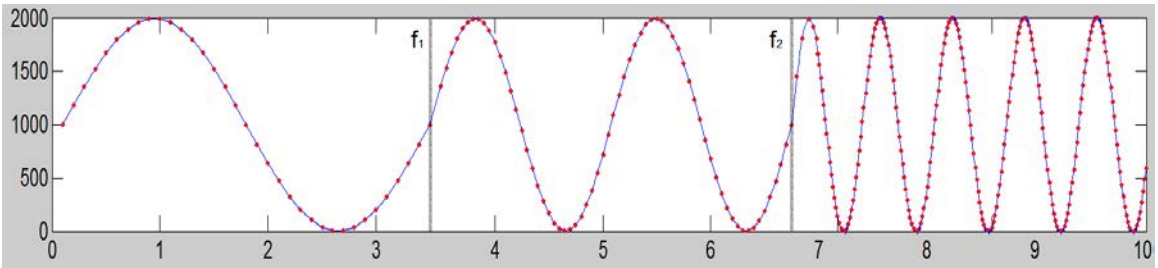


Figura 5.16: *Tarea con paso de ejecución variable, cada punto rojo indica una iteración de la tarea mientras que la onda senoidal, de frecuencia variable, es la consigna de control de la misma*

Como efecto secundario adicional, emplear tareas en tiempo-real con capacidad de ejecutarse con paso variable, permite ajustar dinámicamente la carga computacional del controlador, de forma que la tarea de tiempo libre, la que permite ejecución en tiempo-continuo, puede disponer de mayores recursos computacionales o si fuera necesario, se le pueden arrebatar temporalmente parte de esos recursos para emplearlos en las tareas de tiempo-real cuando éstas así lo requieran, manteniendo así una calidad óptima en el control. La figura 5.17 muestra el bloque gráfico que habilita esta funcionalidad y su panel de configuración.

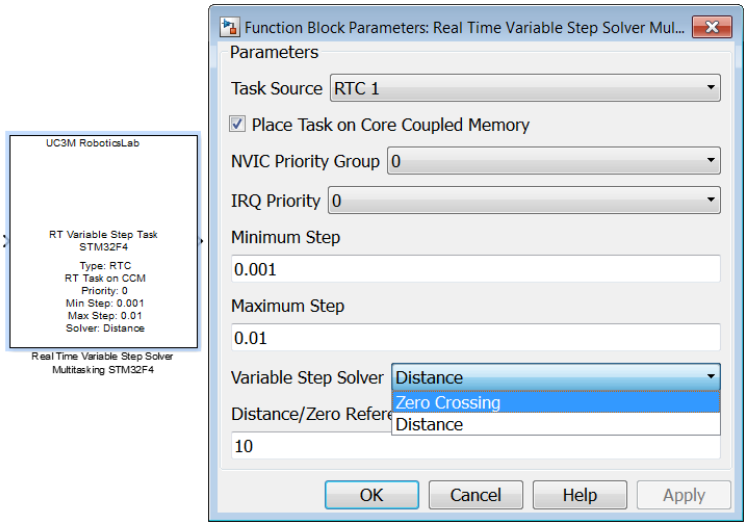


Figura 5.17: *Creación de tareas en Tiempo-Real con paso de ejecución variable mediante un bloque de programación gráfica en SIMULINK®*

5.2.9. Multitarea sin Sistema Operativo

El concepto de multitarea en computación significa ejecutar diferentes programas o tareas, más comúnmente denominadas procesos, en el mismo periodo de tiempo. Los distintos procesos se ejecutan de forma concurrente, compartiendo el tiempo disponible, una tarea comienza su ejecución mientras otra aún se encuentra ejecutándose, en lugar de hacerlo de forma secuencial. El paradigma de ejecución multitarea no conlleva necesariamente que las distintas tareas se ejecuten en el mismo instante temporal, no conlleva el concepto de paralelismo, de hecho un sistema computacional con una única CPU que disponga de un sólo núcleo de ejecución tan solo puede ejecutar una única tarea para un instante temporal cualquiera. El paradigma de ejecución multitarea resuelve este problema utilizando un planificador temporal que permitirá que la CPU del sistema permute su ejecución entre las distintas tareas, dando la sensación de cara al exterior de que las distintas tareas se están ejecutando en paralelo. La figura 5.18 ilustra el concepto de multitarea, en la parte superior de la imagen se muestra el comportamiento del que se aprecia del sistema, y en la parte de abajo se muestra lo que realmente sucede; el planificador temporal administra el tiempo disponible entre las distintas tareas.

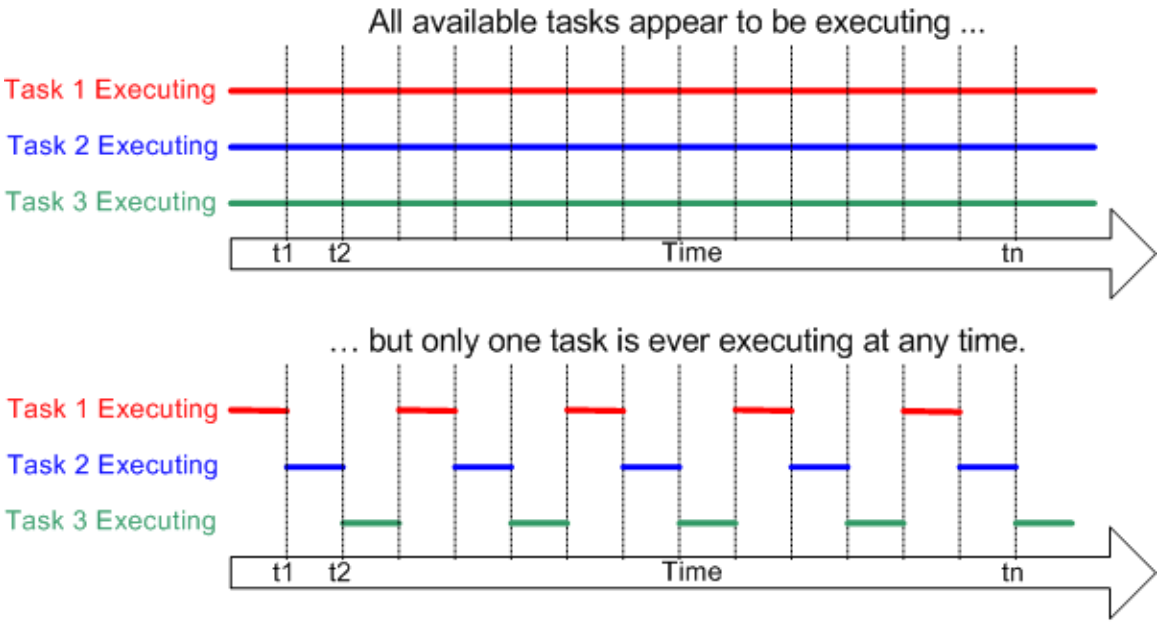


Figura 5.18: *Ejecución de varias tareas bajo el paradigma de Multitarea. Fuente: Documentación FreeRTOS*

Existe dos topologías en el paradigma multitarea:

1. **Multitarea cooperativa**: Es un tipo de multitarea en el que la tarea en ejecución debe ceder voluntariamente el testigo de la ejecución a otra tarea, todas las tareas deben ceder tiempo en algún momento para que se puedan seguir ejecutando el resto de tareas, de lo contrario se rompería el esquema multitarea. Con este esquema no es posible comenzar a ejecutar una tarea determinada en un instante de tiempo deseado. Es un modelo recomendado en sistemas que no han de gobernar componentes electromecánicos y están basados en interfaces de usuario en las que una precisión en plazos de ejecución del orden de microsegundos o milisegundos no sea necesario.

2. **Multitarea apropiativa**: Es un tipo de multitarea en el que la ejecución de la tarea en curso puede quedar interrumpida por otra tarea más prioritaria a la que le ha llegado el momento de ejecutarse. Con este esquema es posible comenzar a ejecutar determinadas tareas en el instante temporal deseado. Debido a basar su funcionamiento en el conocimiento del transcurso del tiempo, este modelo requiere que el sistema computacional pueda proveer de un mecanismo que marque el paso del mismo, este mecanismo es una IRQ que acontecerá cada determinado intervalo de tiempo. Es el modelo utilizado para gobernar sistemas electromecánicos dado que puede garantizar plazos de ejecución temporales del orden de microsegundos para cada tarea.

La figura 5.19 ilustra el concepto de multitarea apropiativa. Se muestran tres tareas con distinta prioridad y que se ejecutan a distintos intervalos temporales, el conocimiento sobre el paso del tiempo procede de la IRQ temporal. Gráficamente resulta sencillo apreciar como unas tareas interrumpen la ejecución de otras que ya se encontraban ejecutándose y aún no habían terminado.

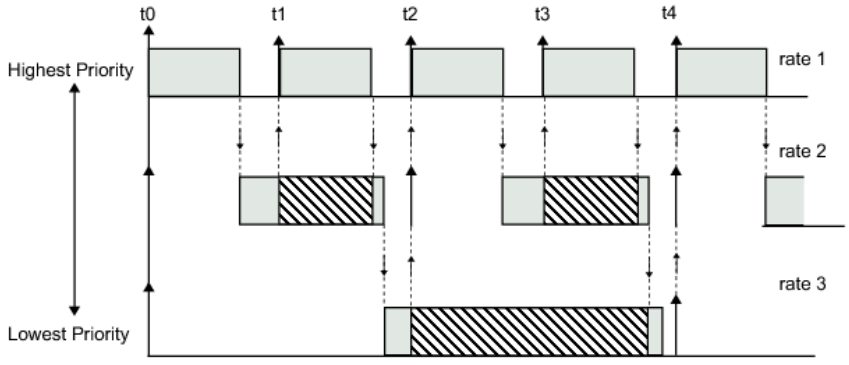


Figura 5.19: *Ejecución de varias tareas bajo el paradigma de Multitarea Apropiativa*

Tradicionalmente la ejecución multitarea apropiativa en sistema embebidos se ha realizado utilizando Sistemas Operativos en Tiempo-Real (RTOS, del inglés "Real Time Operating System"). **La ejecución multitarea introduce un concepto denomidado cambio de contexto**. El cambio de contexto implica que se debe guardar y restaurar el estado interno a nivel de registros de la CPU/FPU, cada vez que se tenga que ejecutar una tarea distinta a la que ya se está ejecutando o se tenga que retomar la ejecución de la tarea que había quedado suspendida, de hecho el cambio de contexto es necesario para poder retomar la ejecución de la tarea suspendida donde se había quedado sin incurrir en corrupciones de datos, puesto que la tarea retomada necesitará los valores que estaban presentes en los registros de la CPU/FPU para realizar sus cálculos correctamente.

Este proceso, el del cambio de contexto, se realiza automáticamente por hardware en el caso de IRQ; pero en el caso de la alternancia de tareas en un entorno multitarea gobernado por un RTOS, el cambio de contexto debe realizarse por software después de atender al planificador temporal. **El cambio de contexto en un RTOS consume una cantidad de tiempo considerable**, del orden de microsegundos, puesto que se contabiliza también el tiempo necesario para ejecutar una llamada al planificador temporal, que es quién decide qué tarea debe ejecutarse en un instante temporal dado. Por lo que el cambio de contexto en un RTOS está compuesto de la ejecución del planificador temporal, comúnmente denominado "Scheduler", más la realización de las operaciones de guardado y restaurado del cambio de contexto mediante software. La figura 5.20 muestra de forma esquemática los efectos de múltiples cambios de contexto sobre el tiempo de ejecución total, los cambios de contexto se señalan en color negro, el resto de colores indican diferentes procesos.

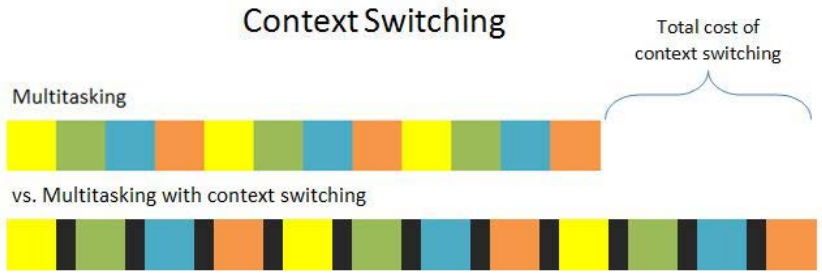


Figura 5.20: *Efectos sobre el tiempo total de ejecución del cambio de contexto en un RTOS. Fuente: Documentación FreeRTOS*

Utilizar un RTOS como base para un sistema con ejecución multitarea conlleva ventajas:

- Posibilidad de gestionar gran cantidad de tareas sin consumir recursos de I/O adicionales por tarea
- Existen RTOS gratuitos

También lleva implícito una serie de inconvenientes:

- Latencia considerable en el cambio de contexto por software
- Distinta implementación del RTOS para cada plataforma hardware
- Misma base temporal para todas las tareas, los tiempos de muestreo de cada tarea deben ser múltiplos de un tiempo base
- Se corre el peligro de colmar la memoria de pila de la CPU. Por lo que se debe detectar por software el posible peligro de intentar lanzar una tarea que no terminó su iteración anterior, alargando aún más el tiempo necesario para realizar el cambio de contexto.
- En relación con los lenguajes de programación gráficos basados en modelos y los RTOS, se suele considerar que cada tiempo de muestreo de cada bloque gráfico se corresponde con una tarea diferente, produciendo una fragmentación de los algoritmos que se han diseñado, generalmente no es esta la intención del programador.

Emplear un RTOS es útil cuando el número de tareas apropiativas es elevado, pues permite gestionarlas sin consumir recursos adicionales relacionados con las capacidades de I/O.

En cambio, **si el número de tareas apropiativas en tiempo-real necesarias no es muy elevado, se puede optar por emplear un mecanismo que provea de multitarea en tiempo-real de forma diferente** a como lo realiza un RTOS. **Es posible utilizar una fuente distinta de IRQ temporal por cada tarea** si el sistema computacional puede permitirse estos recursos y el número de tareas no sea excesivamente elevado. Utilizando el mecanismo hardware de cambio de contexto por IRQ, se ahorra el tiempo necesario para ejecutar el complejo planificador temporal de un RTOS y el posterior cambio de contexto por software. En un RTOS estas dos secciones son secciones críticas, en las que las fuentes de IRQ deben deshabilitarse, por lo que existe la probabilidad de desatender o atender con retraso en el tiempo a las fuentes de IRQ externas como la contabilización de pulsos digitales o la entrada

PWM, circunstancia no deseada en el ARCP propuesto en esta tesis, en el que se desea garantizar el mayor determinismo posible.

La figura 5.20, en la parte superior, muestra el comportamiento del mecanismo de ejecución multitarea mediante múltiples IRQ temporales, en las que no se ejecuta el cambio de contexto por software. El retraso en el inicio de la ejecución de las tareas es el mínimo posible. **El utilizar diferentes temporizadores para cada tarea, permite que los tiempos de muestreo no requieran ser múltiplos de un mismo tiempo base (generalmente este tiempo base se obtiene del timer de sistema, denominado SysTick Timer) y que cada tarea posea su propio mecanismo hardware para el cambio de contexto**; **también se ha añadido en esta tesis como soporte adicional en SIMULINK® la posibilidad de que cada tarea contenga su propio planificador temporal basado en el modelo de generación de código "Monotarea con múltiples tiempos de muestreo"** explicado en el apartado 4.6. Este planificador temporal es el configurado por defecto en la generación de código en MATLAB y resulta computacionalmente muy sencillo comparándolo con el planificador de un RTOS. De esta forma el poder disponer de un planificador temporal en cada tarea hace que se reproduzca el modelo "Monotarea con múltiples tiempos de muestreo" de forma que al tener varias tareas se obtiene automáticamente un modelo "Multitarea con múltiples tiempos de muestreo" distinto del provisto por defecto en MATLAB/SIMULINK®. A diferencia del éste último modelo de generación de código fuente, no es necesario una tarea por cada tiempo de muestreo, sino que en una misma tarea puede haber distintas partes que se ejecutan a diferentes intervalos temporales. Mecanismo que no es posible directamente en MATLAB/SIMULINK® y que se ha integrado como un nuevo bloque de programación gráfica denominado "Task Scheduler" que también hace uso de las funcionalidades añadidas al esquema modificado de generación de código fuente mostrado en la figura 5.6.

En el MCU STM32F4, con multitarea por IRQ es posible alcanzar el siguiente número de tareas:

- 1 Tarea en Tiempo-Real utilizando el Systick Timer
- 1 Tarea en Tiempo-Real utilizando el reloj calendario RTC
- 13 Tareas en Tiempo-Real adicionales utilizando los Timers, se pierden las I/O PWM de los timers involucrados
- 1 Tarea en tiempo-continuo, se ejecuta constantemente, suele denominarse "Idle Task", ideal para ejecutar código basado en tiempo-continuo o cálculos intensivos

En total en el MCU STM32F4 se puede disponer de 15 tareas preemptivas en tiempo-real y 1 tarea con ejecución continuada, en total 16 tareas posibles. Las dos primeras tareas, que utilizan el Systick Timer y el RTC timer no consumen recursos de I/O, el resto inhiben parte de las I/O PWM, aquellas asociadas a cada timer.

Este mecanismo de multitarea apropiativa mediante IRQ posee ventajas:

- **No hay secciones críticas en las que sea necesario deshabilitar las IRQ, se puede atender el 100 % del tiempo a fuentes externas de IRQ**
- **El puntero de pila de la CPU no puede rebosar, ya que las IRQ no pueden interrumpirse a si mismas en las arquitecturas basadas en ARM**. No es necesario comprobar esta circunstancia en el cambio de contexto. Esto también facilita sobremanera el detectar si una tarea se ha quedado "atascada" y no puede concluir su ejecución
- Permite distintas bases temporales para cada tarea
- La facilidad en la implementación permite portar el mecanismo multitarea fácil y rápidamente a otras arquitecturas computacionales
- Asegura una latencia mínima en el cambio de contexto
- Ofrece un comportamiento muy similar a un RTOS convencional

También presenta inconvenientes:

- El número de tareas es limitado
- Aumentar el número de tareas en tiempo-real puede incurrir en una pérdida de recursos de I/O

En el caso de que 16 tareas no sean suficientes, se ha provisto soporte en SIMULINK® para el RTOS de Erika Enterprise® (véase el apartado 2.5.1 como recordatorio de dicha compañía).

En las siguientes dos páginas se ejemplifica de forma visual el nuevo concepto y mecanismo de generación de código para multitarea en tiempo-real propuesto en esta tesis, en el que cada tarea puede contener su propio planificador temporal.

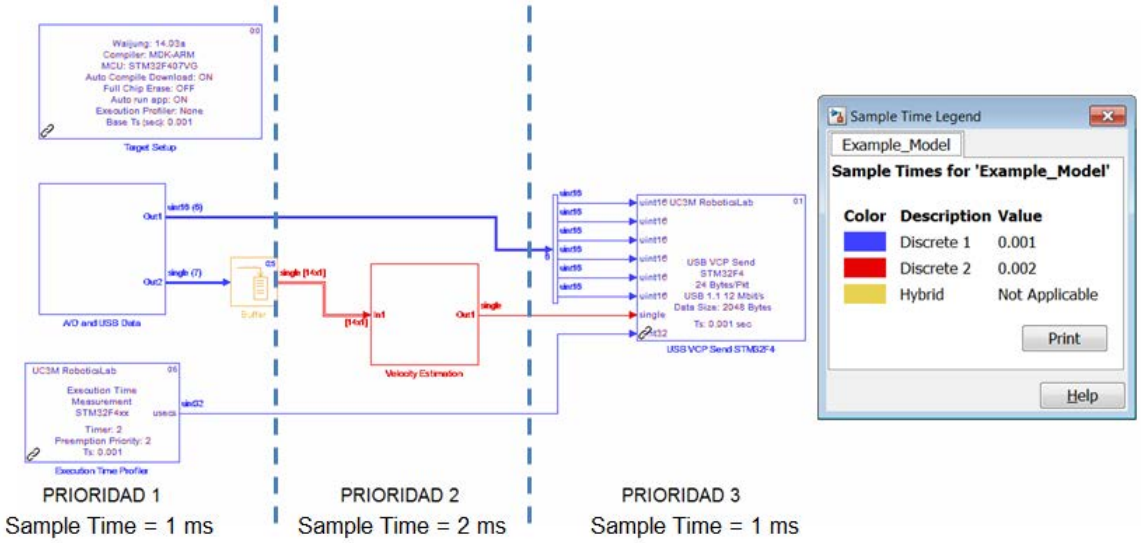


Figura 5.21: *Programa gráfico de ejemplo para ilustrar los modelos de generación de código en MATLAB®.*

La figura 5.21 ilustra un programa para el MCU STM32F4 diseñado en base a grafos, en el que los bloques gráficos de color azul poseen un tiempo de muestreo de 1 ms, se deben ejecutar cada milisegundo. Los bloques de color rojo poseen un tiempo de muestreo de 2 ms.

El orden en la ejecución del programa es el determinado por las prioridades establecidas, mostradas en la figura mediante texto. Por tanto, en el modelo de la figura 5.21 se desea que comience la ejecución por los bloques de la izquierda siguiendo hacia la derecha; cuando corresponda debido a su tiempo de muestreo, deberán ejecutarse los bloques de color rojo y a continuación seguirán los ubicados más a la derecha, los de prioridad 3. Aunque este sea el orden de ejecución definido por el programador, el modelo de generación de código multitarea predeterminado en MATLAB® no puede llevarlo a cabo, no puede respetar el orden en la ejecución de los bloques, pues agrupa éstos en tareas basándose sólo en el tiempo de muestreo. El modelo de generación de código fuente multitarea de MATLAB® genera una tarea por cada tiempo de muestreo, además este modelo requiere de la adaptación a un RTOS, adaptación que debe ser realizada manualmente o añadiendo el soporte de programación gráfica necesario por el usuario.

La figura 5.22 ilustra el resultado del código fuente generado por los modelos de generación de código predeterminados de MATLAB y cómo funcionaría el programa utilizando el nuevo modelo multitarea sin sistema operativo en tiempo-real propuesto en esta tesis. En éste último modelo, se genera un código fuente verdaderamente multitarea respetando las necesidades y prioridades establecidas por el programador. Ahora cada tarea posee su propio planificador temporal y esto le permite gestionar diferentes tiempos de muestreo, haciendo posible que cada tarea se comporte a nivel técnico como el modelo monotarea con diferentes tiempos de muestreo, con la salvedad de que ahora es posible recurrir a multitud de 'monotareas' para llevar a cabo un sistema multitarea.

El nuevo modelo multitarea tampoco exige que las tareas compartan un tiempo de muestreo que sea múltiplo de un tiempo base; cada tarea puede tener el tiempo de muestreo que el programador considere.

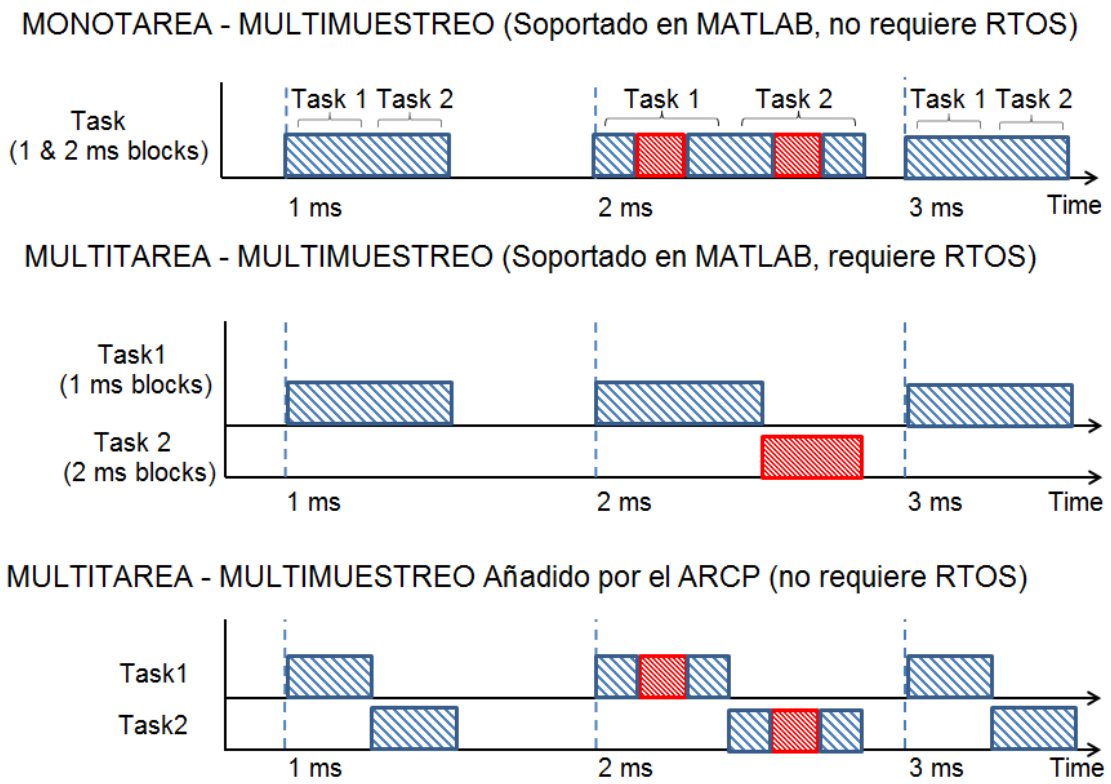


Figura 5.22: *Resultado del código fuente generado por los modelos de generación de código predeterminados de MATLAB® y del generado por el nuevo modelo multitarea propuesto en esta tesis. Las secciones azules y rojas se corresponden con sus homólogas de la figura 5.21*

La figura 5.23 muestra los bloques de programación gráfica que brindan la oportunidad al usuario de manejar y gestionar tareas en tiempo-real y en tiempo-continuo con un gran nivel de abstracción. Empezando por el bloque de la esquina superior izquierda, éste permite utilizar el tiempo libre entre iteraciones de las tareas en tiempo-real o bien se puede emplear sólo esta funcionalidad dando lugar a un controlador que funciona en tiempo-continuo. A su derecha se presenta el bloque que permite obtener una medición del tiempo de ejecución por cada iteración del bloque anterior. En el medio de la figura se muestran los bloques que permiten crear tareas en tiempo-real, o incluso tareas en tiempo-real con paso de ejecución variable. En la parte más baja de la figura se muestran el bloque que permite generar un planificador temporal en cada tarea en tiempo-real, 'RT Scheduler' y a su derecha el bloque que permite gestionar las tareas, brindando la posibilidad de detenerlas, reanudarlas o deshabilitarlas.

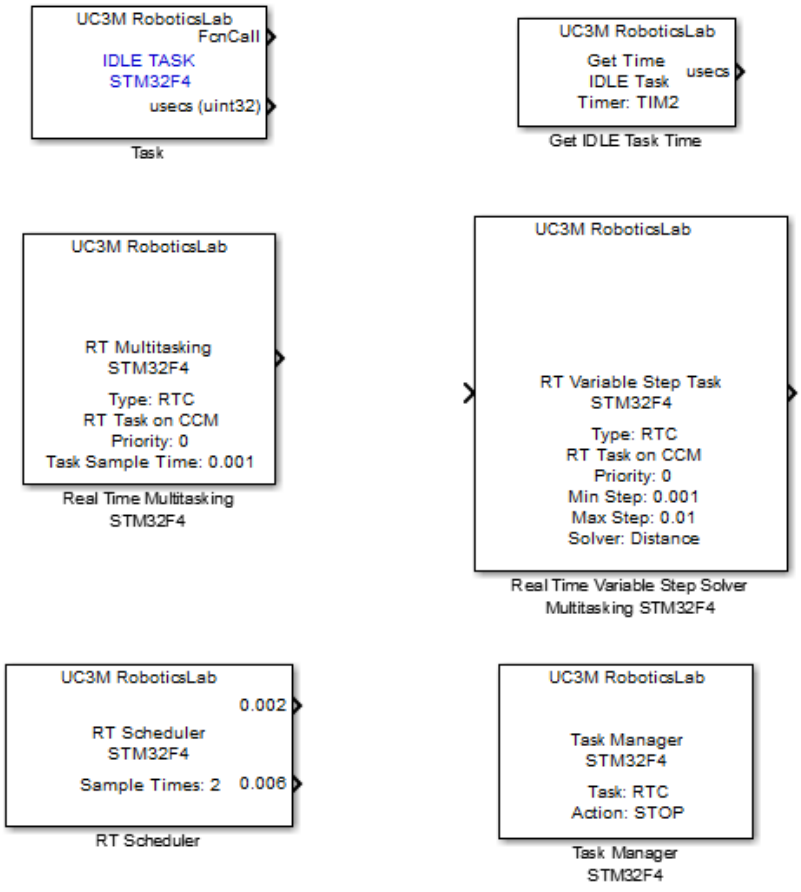


Figura 5.23: *Librería con todas las funcionalidades relativas a multitarea para el MCU STM32F4*

5.2.10. Sincronización de múltiples controladores

Un aspecto que se ha tenido en cuenta en el sistema ARCP propuesto en esta tesis, es contar con un sistema de control distribuido, en el que un complejo sistema pueda ser gobernado no por un sólo controlador, sino por una multitud de controladores todos ellos programados bajo el mismo paradigma basado en modelos gráficos.

El poder sincronizar todos los relojes de tiempo real de los controladores involucrados, de forma que marquen el paso del tiempo al unísono, conlleva la ventaja de que cada controlador puede conocer de antemano el estado en que se encuentra cualquier otro controlador del sistema distribuido. Por lo que la transmisión y recepción de datos entre ellos resulta más sencilla puesto que podrán coordinarse de forma que no "colisionen" a la vez paquetes de datos de múltiples controladores que quieren transmitir en el mismo instante temporal. Ahora es posible que los controladores compartan un canal de transmisión/recepción de voz única, en el que varios controladores no pueden comunicarse a la vez como es un canal de datos inalámbrico. La figura 5.24 ilustra cómo esta funcionalidad de sincronización para sistemas de control distribuidos solventa el problema descrito. El MCU número 5 es el maestro sincronizador, el resto de MCUs sincronizan su timer de sistema con el del maestro. Ahora es posible evitar que varios MCU transmitan datos al mismo tiempo, cosa que no podrían hacer con un medio inalámbrico o cableado basado en UART, SPI, I2C...

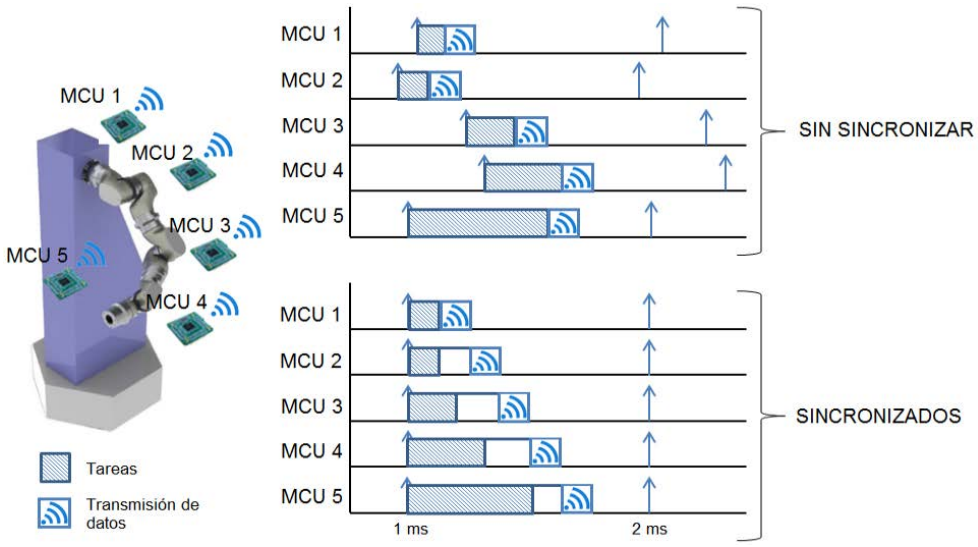


Figura 5.24: *Ejemplo de utilización de la funcionalidad de sincronización de controladores*

Se ha integrado la capacidad de sincronizar mediante una señal digital, al recibir el primer paquete de datos por UART o CAN, o al recibir el primer paquete de datos de forma inalámbrica. Los pasos a seguir para lograr el mecanismo de sincronización son los siguientes:

1. Inicializar los periféricos del sistema
2. No habilitar las IRQ del Systick timer y del resto de timers
3. Comprobar la señal de sincronización (leer el puerto digital o consultar los flags de recepción de UART, CAN, o Wireless) en el caso de tratarse de un controlador esclavo o enviar la señal de sincronización en caso de ser controlador maestro
4. Comprobar si se ha agotado el tiempo de espera (se lee el valor del Systick timer), en caso negativo volver al paso 3, si ya se ha sincronizado o se ha agotado el tiempo de espera pasar al punto 5
5. Resetear los valores del Systick Timer y del resto de timers
6. Habilitar las IRQ del Systick timer y del resto de timers

La máxima desviación temporal observada ha sido de 1 microsegundo utilizando sincronización mediante señales digitales. En el caso de UART, CAN o Wireless la desviación temporal se corresponde con el tiempo necesario para enviar el paquete de datos de menor tamaño posible, que está en torno a unos pocos microsegundos. La figura 5.25 ilustra el bloque de programación gráfica de SIMULINK® que permite gestionar y utilizar la funcionalidad avanzada de sincronizar múltiples controladores, útil en sistemas de control distribuidos.

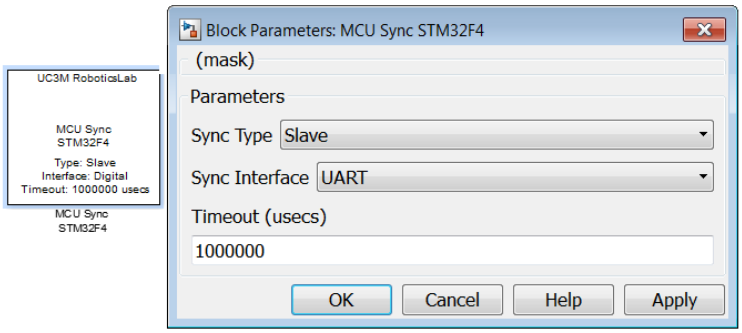


Figura 5.25: *Funcionalidad gráfica para sincronizar los relojes de tiempo-real de múltiples controladores en sistemas de control distribuidos*

5.2.11. Control total de los timers. Medida de los tiempos de ejecución

Al sistema ARCP se le ha dotado de las funcionalidades necesarias para gobernar con total libertad todas las características de los timers del MCU, accesibles desde el lenguaje gráfico de SIMULINK®.

Desde poder medir el tiempo de ejecución necesario para cada iteración de las tareas preemptivas como medir el tiempo de ejecución de cada iteración de la tarea de tiempo continuo o *Idle Task*; la precisión en la medición del tiempo abarca desde 100 ns hasta microsegundos, configurable por el usuario. También es posible hacer funcionar con libertad un timer y leer el valor de tiempo devuelto en cualquier momento (ideal para medir el tiempo de ejecución de una parte del algoritmo), resetearlo o detenerlo para crear con total libertad tiempos de espera reconfigurables o bucles complejos con necesidades temporales que varíen dinámicamente.

Además también se pueden definir desde SIMULINK® rutinas IRQ independientes al detectar un flanco de bajada y otra IRQ distinta para el flanco de subida de una señal, así como medir el tiempo transcurrido entre ambos flancos. Como ejemplo estas últimas son funcionalidades requeridas para manejar señales digitales MAF (del inglés, *Mass Air Flow*) muy frecuentes en control de motores de combustión interna.

La figura 5.26 muestra los bloques de programación gráfica que permiten utilizar con libertad las posibilidades que brindan los timers.

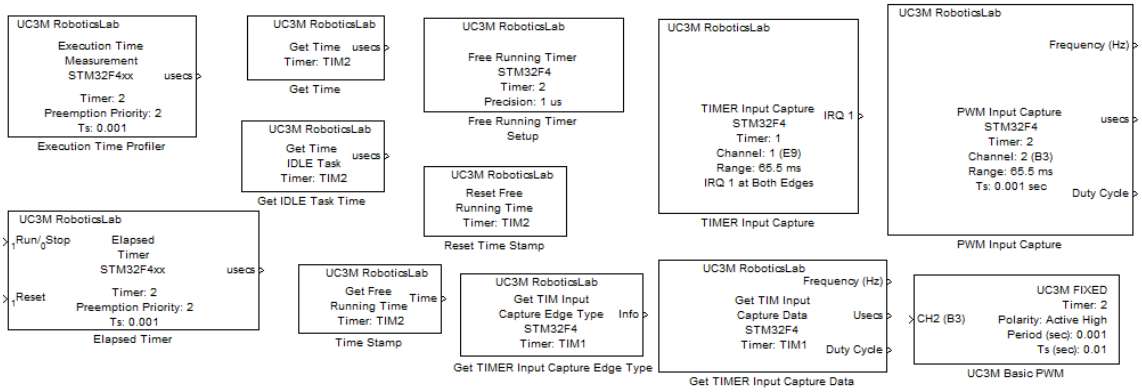


Figura 5.26: *Bloques creados para SIMULINK® del sistema ARPC para gestionar los timers*

5.2.12. Mezclador de audio

El MCU STM32F4 posee la capacidad computacional y los recursos de I/O necesarios para disponer de un medio de reproducción de audio. Así mismo la placa STM32F4Discovery utilizada dispone de un DAC de audio de 16 bits de doble canal (estéreo) rutado a un conector de tipo mini-jack.

La reproducción de audio es una de las capacidades multimedia explotadas en el sistema ARCP propuesto en esta tesis y proporciona un feedback adicional. Integrado como funcionalidades accesibles desde el lenguaje gráfico de SIMULINK®. Para la reproducción de audio se ha dotado al MCU STM32F4 de la funciones necesarias para mezclar hasta 4 muestras de audio a 16 bits estéreo sin comprimir.

Las funciones del mezclador que pueden interpretarse como el driver de audio, no deben interferir con los procesos y recursos de I/O que se estén utilizando para atender las tareas de tiempo-real. Esto es diferente a cómo se entiende un driver de audio para un computador de propósito general, en el que los drivers de audio y vídeo poseen la mayor prioridad, aún mayor que el núcleo del sistema operativo y suelen ejecutarse en modo kernel; el modo kernel de un sistema operativo se encuentra detallado en el apartado 2.5.3. Para evitar que la reproducción de audio perjudique a los procesos de tiempo-real, ésta se realiza maximizando la utilización de los recursos DMA y ajustando la prioridad de la IRQ del DMA con la menor prioridad posible. La figura 5.27 ilustra el funcionamiento del mezclador de audio y cómo se envían los datos resultantes de la mezcla a través de un buffer DMA al periférico I2S (los periféricos SPI del MCU STM32F4 pueden realizar las funciones de periféricos de transmisión de audio digital).

Teniendo presente la figura 5.27, se puede describir el funcionamiento del mezclador de audio. Las muestras de audio se ubican en la memoria ROM o pueden generarse por software. Para reproducir audio es recomendable utilizar la tarea de tiempo-continuo pues es la que menos prioridad posee. Una vez que se ha ordenado reproducir una muestra, se transfieren hasta 4 KB de datos de audio al buffer DMA. Antes se comprueba si hay más de 1 sonido por reproducir, en caso afirmativo se realiza la mezcla del audio de las dos muestras, si hubiera una tercera muestra, se volvería a mezclar el resultado de la primera mezcla con la tercera muestra y así sucesivamente hasta 4 muestras. Mezclar audio es computacionalmente sencillo pero muy intensivo. Si sólo se tiene que reproducir una muestra de audio no será necesario mezclar nada, por tanto reproducir una sola muestra de audio casi no tiene impacto en los recursos computacionales del MCU. Una vez iniciado el proceso de reproducción de audio, el buffer DMA provocará una IRQ

cuando se haya enviado la mitad del buffer, y se comprobará si quedan datos de audio por mezclar o enviar (en el caso de una sola muestra) y se escribirán estos datos en la primera mitad del buffer. El final del buffer también origina otra IRQ que volverá a comprobar si quedan datos de audio por enviar y/o mezclar, y se escribirán éstos en la segunda mitad del buffer. De forma que el buffer DMA circular siempre contenga datos de audio para alimentar al periférico I2S conectado al DAC de audio donde se convierten en ondas analógicas, hasta que se terminen de enviar las muestras de audio. Estas IRQ poseen la menor prioridad, en caso de no ser atendidas en el momento en que se producen, sigue quedando un remanente de 2 KB que otorga tiempo suficiente para que se pueda atender estas IRQ y no se produzcan efectos de entrecortado del audio. La figura 5.28 muestra los bloques de programación gráfica del sistema ARCP para gobernar la reproducción de audio.

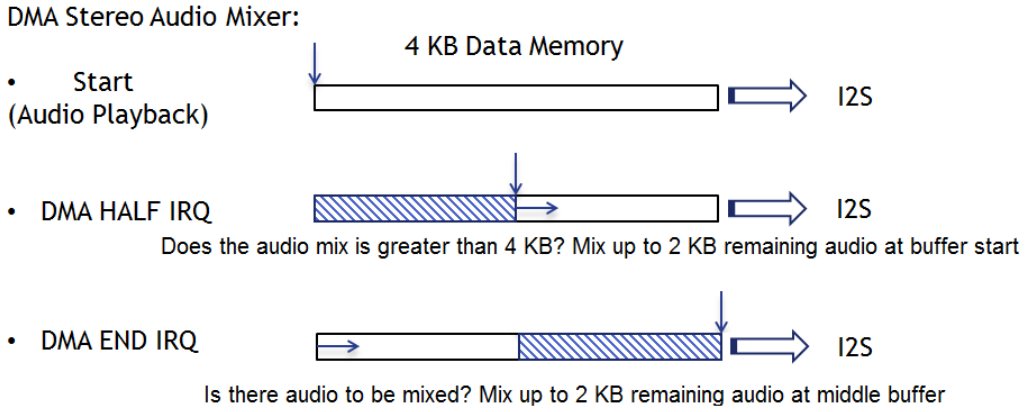


Figura 5.27: *Descripción esquemática el funcionamiento del mezclador de audio*

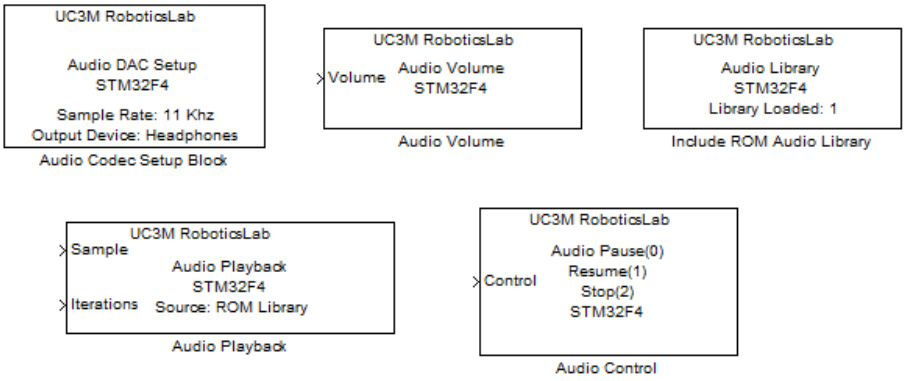


Figura 5.28: *Bloques creados para SIMULINK® del sistema ARPC para reproducir audio*

5.2.13. Capacidades de vídeo

Al igual que sucede en el apartado anterior con las capacidades de audio del MCU STM32F4, éste dispone de los recursos hardware necesarios para dar lugar a un dispositivo de vídeo embebido. Pero, debido a las prioridades de control electromecánico del sistema ARCP, la funcionalidad de vídeo debe llevarse a cabo de manera diferente a cómo se entiende en un computador de propósito general; en el caso del sistema ARCP las funcionalidades orientadas a gráficos no deben suponer un impedimento o interferencia para las tareas de control.

La figura 5.29 ilustra el esquema de un MCU conectado a una pantalla LCD-TFT que contiene su propia memoria de vídeo.

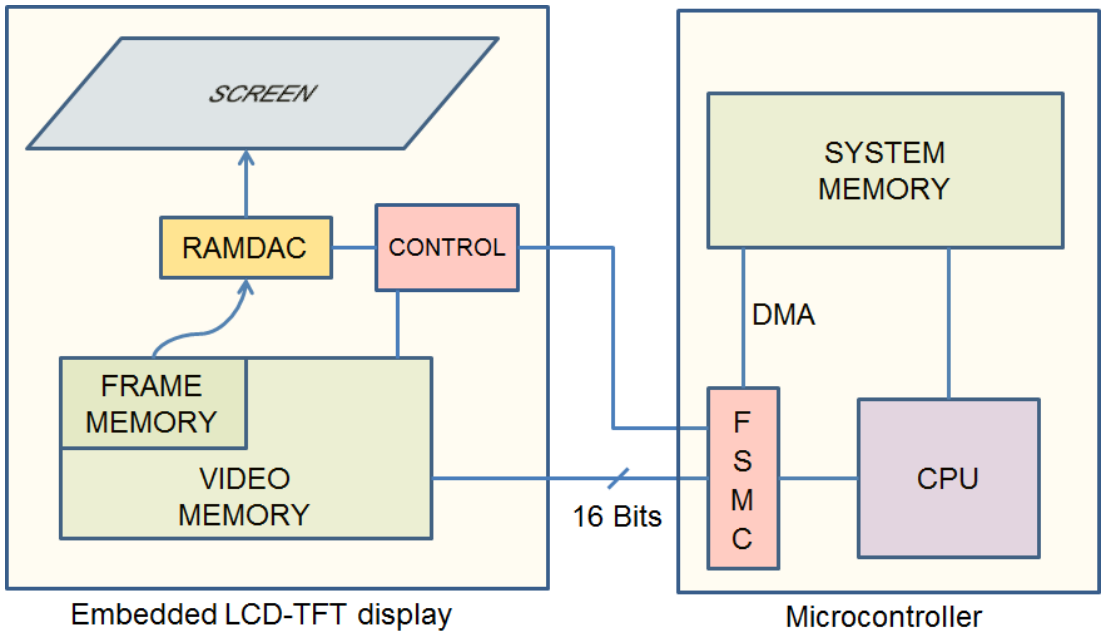


Figura 5.29: *Esquema de un MCU actual gobernando un dispositivo de imagen con memoria de vídeo incluída*

Antes de continuar profundizando en las funcionalidades gráficas otorgadas a la toolbox de funcionalidades avanzadas, conviene explicar brévemente en qué consiste un adaptador de vídeo. El propósito de un adaptador de vídeo es el de mostrar información en forma de gráficos en una pantalla o monitor. Los gráficos pueden ser imágenes, vectores o textos. Dichos gráficos se van ubicando en una memoria denominada memoria de vídeo, de forma que acaban componiendo la imagen deseada. En la memoria de vídeo se ubica la memoria de cuadro, que es la memoria cuyo contenido acabará mostrándose en la pantalla o monitor, el sobrante de

la memoria de vídeo no ocupada por la memoria de cuadro suele utilizarse para realizar en ella operaciones de transformaciones de gráficos, o incluso para contener éstos mismos. Una vez que la memoria de cuadro contiene la información de la imagen deseada; se realizará la conversión de los datos digitales de la memoria de cuadro transformándolos en señales eléctricas analógicas para atacar los transistores de película delgada (TFT, del inglés *Thin Film Transistor*) o la entrada de un monitor de forma que se muestre la imagen en el dispositivo final, pantalla o monitor. Actualmente existen monitores capaces de admitir directamente los datos digitales de la memoria de cuadro, pero respecto a MCU se empleará el esquema de la figura 5.29.

Como medio de visualización de datos en formato gráfico para MCU, se utilizan pantallas LCD-TFT que contienen su propia memoria de vídeo con interfaz de conexión por puerto paralelo de 16 bits. Esta interfaz es un estándar de facto para conectar módulos de memoria RAM externa a MCU o sistemas computacionales en general, de hecho la memoria de vídeo de estas pantallas se considera un módulo de memoria RAM externo al MCU. La interfaz que permite gobernar módulos de memoria RAM externos es el controlador de memoria estática flexible, más conocido por sus siglas FSMC (del inglés 'Flexible Static Memory controller'). Este periférico permite mapear en el espacio de direcciones del procesador los módulos de memoria externos, lo que en este caso facilita el uso de canales DMA entre la memoria interna del MCU y la memoria externa. Dichas memorias externas necesitan además del bus de datos y direcciones, que es un bus de 16 bits, unas señales adicionales de control y sincronización, que también son provistas y gobernadas por el periférico FSMC.

Como librería de tratamiento, acceso y funcionalidades gráficas se utiliza la librería gráfica para sistemas embebidos de la compañía SEGGER [59]. Pese a tratarse de una librería comercial, puede utilizarse de forma gratuita juntamente con MCUs del fabricante ST Microelectronics® debido a un acuerdo entre ambas compañías; siempre que se distribuya de forma compilada (librería de funciones) y no en forma de código fuente. Esta librería de funciones gráficas se denomina EmWin.

Respecto al funcionamiento de la librería EmWin, ésta guarda las órdenes de dibujado en una lista. En esta tesis se ha modificado la gestión de esta lista de órdenes de dibujado para comprobar qué órdenes de la misma se superponen unas a otras, de forma que se eliminen de la lista aquellas órdenes gráficas que queden ocultas, de forma que no se pierda tiempo ejecutando funciones que no tendrán efecto alguno.

La modificación anterior, pese a resultar útil, supone una de las pequeñas modificaciones en cuanto a rendimiento y funcionamiento realizadas a la librería EmWin. Esta librería, originalmente puede realizar el envío de los datos gráficos a la memoria de vídeo de dos maneras:

1. Puede dibujar directamente enviando la información de cada píxel a la memoria de vídeo mediante el periférico FSMC. De esta forma no se requiere el uso de la memoria interna del MCU, pero se aprecia como se construye la pantalla línea a línea. Debido al uso intensivo de la CPU, puesto que no existe un buffer de memoria que enviar a través de un canal DMA, la tasa de refresco de la pantalla es muy baja y el efecto de construcción de la imagen, denominado 'tearing' mostrado en la figura 5.30, omnipresente.

2. Puede utilizar la memoria interna del MCU para crear una o dos copias, doble o triple búfer [60], de la memoria de cuadro, de forma que la librería gráfica trabaje sobre estas memorias y una vez terminada de construir la imagen, enviar estos datos a la memoria de cuadro. De esta forma se minimizan los efectos de '*tearing*' pero requiere dedicar ingentes cantidades de memoria RAM del MCU de forma exclusiva para la funciones gráficas.

Figura 5.30: *Ilustración del efecto de 'tearing' al dibujar en una pantalla*

En esta tesis se propone una aproximación diferente, eliminar el efecto de '*tearing*' sin recurrir al uso del método del doble/triple búfer:

1. El procedimiento para el dibujado de la pantalla en esta tesis consiste en una modificación de la primera técnica explicada en la página anterior. Ahora la librería gráfica en lugar de enviar los datos de la imagen píxel a píxel a través del periférico FSMC, los envía a un pequeño búfer ubicado en la memoria de sistema del MCU, el tamaño del búfer es configurable entre 2 y 32 KB, pero con 6 KB ofrece una buena tasa de refresco. El contenido de éste búfer se envía mediante transacciones DMA del periférico FSMC, estas transacciones a través del bus de 16 bits son lentas, pero ahora ya no consumen tiempo de CPU. El concepto detrás de la utilización de este búfer es el mismo que el empleado para el mezclador de audio, explicado en el apartado anterior. También se evita que se pueda percibir cómo se construye la pantalla, deshabilitando el RAMDAC de las pantallas LCD-TFT embebidas, cada modelo de pantalla posee un mecanismo diferente para evitar el refresco continuo de la pantalla, gobernados desde los pines de control conectados al FSMC. Una vez terminadas todas las transacciones DMA, se da la orden de realizar un refresco de pantalla y la nueva imagen surge de repente, sin efectos de '*tearing*'.

Respecto al rendimiento, el MCU STM32F4 es capaz de producir un refresco de aproximadamente 60 imágenes por segundo, imágenes a 320x240 con profundidad de color de 16 bits. Utilizando tanto el mecanismo original de EmWin como el mecanismo modificado. La diferencia se encuentra en el tiempo de cómputo que le queda libre al procesador, con el método original de Emwin, se consume aproximadamente el 99 % de los recursos computacionales, pues existen largos períodos de latencia entre envíos píxel a píxel del periférico FSMC. Mientras que con la técnica modificada se consume aproximadamente el 35 % de las capacidades computacionales del MCU STM32F4, no se pueden lograr tasas de refresco mayores a color de 16 bits porque el ancho de banda del FSMC encuentra su límite en ese punto.

Las capacidades gráficas integradas en la toolbox de funcionalidades gráficas emplean color de 16 bits, en concreto la modalidad RGB 565, donde 5 bits corresponden al color rojo, 6 bits al color verde y los restantes 5 bits a la componente azul. Se ha dado más relevancia al color verde, debido a que el ojo humano tiene centrada la captación del espectro de color en el verde. Haber utilizado color de 8 bits habría reducido a la mitad el tiempo en el envío de la información de los píxeles, pero no todas las pantallas embebidas soportan paletizado en su RAMDAC. En total se ha dado soporte para 32 modelos de pantallas, completamente configurables y utilizables desde SIMULINK®.

La figura 5.31 ilustra los bloques para funcionalidades gráficas embebidas disponibles en la toolbox de funcionalidades avanzadas. En el apartado 8.3 se muestra el sistema ARCP de esta tesis con interfaz gráfica embebida, controlando un sistema de actuación basado en aleaciones con memoria de forma (SMA, del inglés *Shape Memory Alloy*).

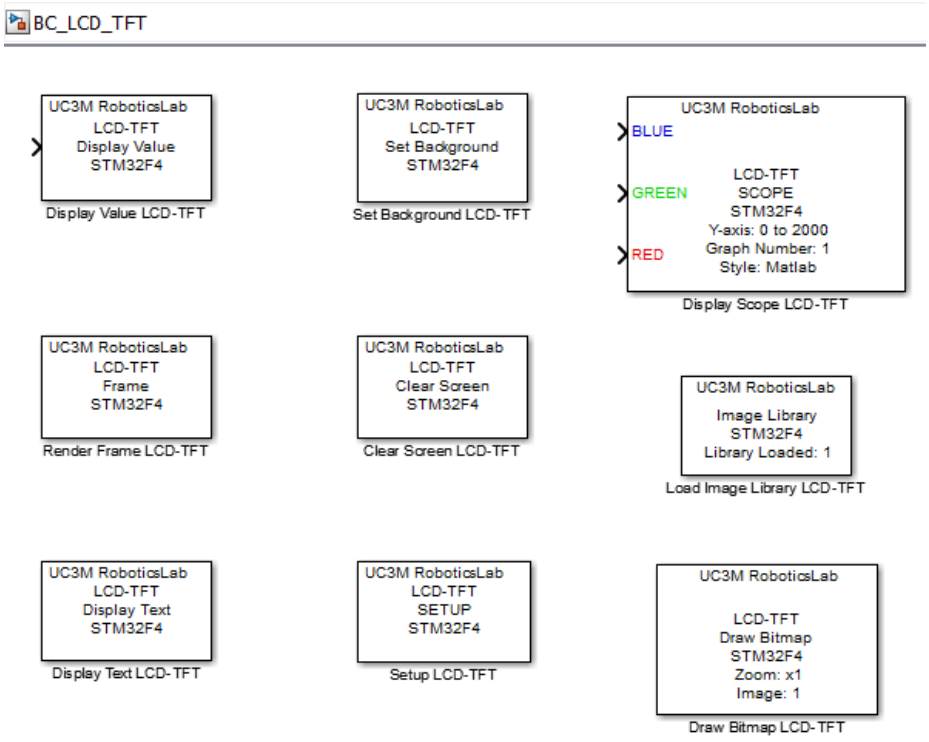


Figura 5.31: *Bloques creados para SIMULINK® del sistema ARPC para visualizar datos en pantallas embebidas LCD o TFT*

5.3. Carácter multidisciplinar del sistema ARCP

La toolbox de funcionalidades avanzadas ha sido diseñada pensado en la facilidad para poder manejarla, no requiere del usuario un conocimiento profundo de la arquitectura hardware que está manejando. Asi mismo, las explicaciones necesarias en algunos casos, de forma que se pueda aprovechar y entender perfectamente la intención y funcionalidad de las distintas capacidades, se muestran en forma de texto en el panel de configuración del bloque de programación gráfico.

De esta manera se ha logrado el objetivo de poner al alcance de un público multidisciplinar el uso del sistema ARCP.

También se ha facilitado la instalación y uso de la herramienta software, el anexo 2 muestra el proceso de instalación automatizado y algunas otras facilidades.

Capítulo 6

Verificación de la calidad del software generado por el ARCP

A diferencia del concepto tradicional de sistemas RCP cuya generación de código fuente está orientada para su puesta en marcha en prototipos. El sistema ARCP propuesto en esta tesis está orientado a producir código fuente con calidad de producción. El código fuente generado con el concepto tradicional también puede ser utilizado en la etapa de producción, pero requiere mayor intervención por parte del usuario. Con un RCP tradicional es necesario inspeccionar manualmente, o apoyado por herramientas de verificación, el código para detectar posibles errores graves o defectos que impidan al código fuente ajustarse a las normas dictadas por el estándar de seguridad exigido.

El proceso de verificación del código fuente, que redunda en la calidad del software embebido, requiere inevitablemente de tiempo y coste económico; la calidad del software es directamente proporcional al tiempo y al coste empleados. El tiempo, y por ende el coste, puede reducirse drásticamente empleando un software MBD con soporte en la programación mediante un lenguaje gráfico, como ya se ha visto ampliamente en pasados apartados de esta tesis. Asi mismo, el tiempo adicional necesario para verificar el código fuente generado puede reducirse igualmente si todo el código relativo al soporte del hardware ya ha sido verificado y corregido con anterioridad, de forma que los nuevos proyectos no necesiten invertir tiempo en verificar la adecuación del código fuente que maneja las posibilidades del hardware embebido.

La figura 6.1 ilustra la relación entre la calidad del software y los requisitos de tiempo y coste al utilizar un planteamiento de desarrollo convencional; tanto

la calidad, tiempo y coste forman un triángulo equilátero (lado izquierdo de la figura), mientras que el utilizar un sistema ARCP cuyo código fuente ya ha sido verificado, hace que la relación de las variables pase a formar un triángulo isósceles (lado derecho de la figura).

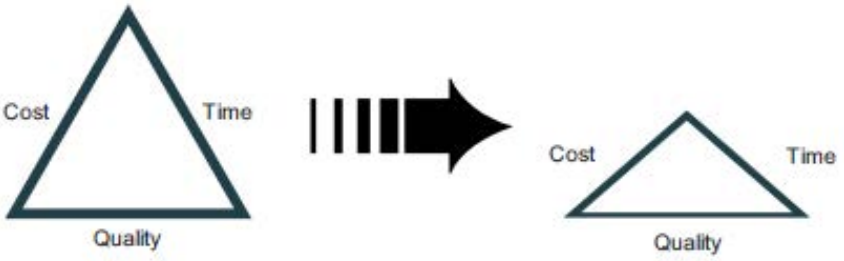


Figura 6.1: *Relación entre la calidad del software, el tiempo y el coste.*

Pese a las ventajas en necesidades de tiempo y personal que se pueden obtener empleando un sistema ARCP, siempre será necesario un mínimo de tiempo. Respecto a la verificación del código fuente de un modelo de control generado con un sistema ARPC; la complejidad en la adecuación del código vendrá determinada por el código de los algoritmos y no del relacionado con el hardware, esta complejidad siempre dependerá de las opciones de generación de código seleccionadas y de las opciones de comprobación de cálculos habilitadas en los bloques gráficos de SIMULINK®. De forma que si el proyecto acometido requiere un código de mayor o menor seguridad en los algoritmos, esto será responsabilidad del usuario que emplea SIMULINK®, pues deberá conocer las opciones disponibles a este respecto para asegurarse que la etapa de verificación de código le exigirá menos tiempo.

El código fuente generado por la toolbox de funcionalidades avanzadas presentada en esta tesis, ha sido verificado siguiendo la norma MISRA C 2004 [61], y las normativas de seguridad IEC 61508 [62, 63, 64, 65] (relativo a seguridad de sistemas electrónicos programables), IEC 26262 (relativo a seguridad de la electrónica embarcada en el automóvil) e IEC 60880 (relativo a seguridad de dispositivos que hacen uso de energía de origen nuclear). Respecto a estas normativas, MISRA hace hincapié en aspectos de la programación que aportan seguridad tales como el cumplimiento en el manejo de los rangos en vectores/matrices, el impedimento en realizar operaciones de desplazamiento de bits en números negativos, la comprobación al realizar operaciones matemáticas de forma que se impide que una variable rebase su capacidad... el resto de normativas, además de hacer uso de las mismas normas que MISRA y añadir alguna más como la necesidad de justificar mediante comentarios la inclusión de código que no pueda ser alcanzado (”*unreachable code*”) en ciclos de ejecución normales, exigen que los dispositivos e

implementaciones hardware se diseñen siguiendo ciertas características.

Como ejemplo en los sistemas de control nuclear se exige la presencia de los controladores por duplicado, funcionando en paralelo, y a su vez se deben comprobar las decisiones tomadas por el hardware para evaluar las posibles discrepancias entre controladores idénticos, además de añadir elementos de seguridad pasivos que otorguen tiempo suficiente para permitir una desconexión y retorno a la seguridad de forma manual.

La figura 6.2 ilustra las normativas y estándares de seguridad más comunes para sistemas de control embebidos.

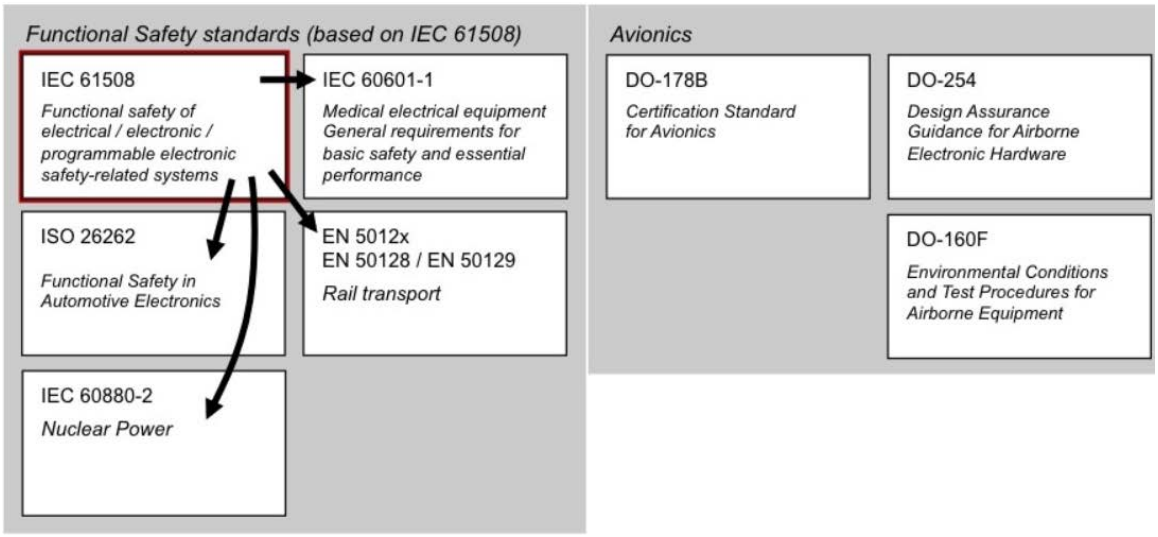


Figura 6.2: *Relación de normativas y estándares de seguridad para sistemas electrónicos de control.*

En el proceso de verificación llevado a cabo para el sistema ARCP propuesto se ha puesto más énfasis en las normas MISRA C 2004 [61]. Respecto a las normativas basadas en IEC 61508 se han tenido en cuenta para familiarizarse con los procesos y metodologías adecuados, de cara a posibles usos y proyectos futuros en los que se exija el cumplimiento de determinada normativa. La verificación formal debe contar con el respaldo de una entidad acreditada como el grupo TÜV o DEKRA[66, 67] cuyos intereses no se encuentren comprometidos con el código fuente objeto del proyecto a inspeccionar.

6.1. Herramientas para verificar código

Las herramientas software para verificación de código son programas que facilitan la labor de detección de errores y defectos de programación, también comprueban que el código fuente se adecue a las exigencias de las normativas que se especifique. Están basados en interfaces gráficas en las que el usuario especifica el código fuente a comprobar y las reglas para comprobarlo. Tras la revisión del código, estos programas verificadores señalarán según un código de importancia los errores y posibles defectos localizados.

Los errores que son capaces de detectar este tipo de herramientas son desbordamiento en cálculos aritméticos, desbordamiento en el direccionamiento de vectores, divisiones entre cero, sentencias if-else/switch incompletas...

Para llevar a cabo estas tareas la herramienta comprueba cada sentencia de código teniendo en cuenta todos los posibles valores y rangos en el caso de vectores, de forma que para cada sentencia de código puede descubrir si existe un error explícito o un defecto de programación que podría llevar a un posible error en tiempo de ejecución.

Las herramientas software para verificación de código utilizadas en esta tesis doctoral han sido MathWorks PolySpace® e IBM Rational Logiscope®, éste último ha servido para corroborar los resultados reflejados por Polyspace®. Emplear varias herramientas verificadoras de código para contrastar los resultados es una técnica habitual [65, 64, 63]. La herramienta software IBM Rational Logiscope® fue adquirida por la compañía francesa Kalimetrix® a finales del año 2012; por lo que el software ahora se denomina Kalimetrix Logiscope®. El funcionamiento de estas herramientas de verificación de código se basa en análisis semántico abstracto [68, 62].

Polyspace® provee de una interfaz gráfica, mostrada en la figura 6.3. La imagen de arriba muestra la interfaz de configuración, en ella se eligen los ficheros de código fuente a analizar y las reglas a comprobar. La imagen de abajo ilustra los resultados de la comprobación, se muestran a la izquierda los defectos encontrados, la localización de estos en el código fuente y comentarios y sugerencias para solucionar los problemas.

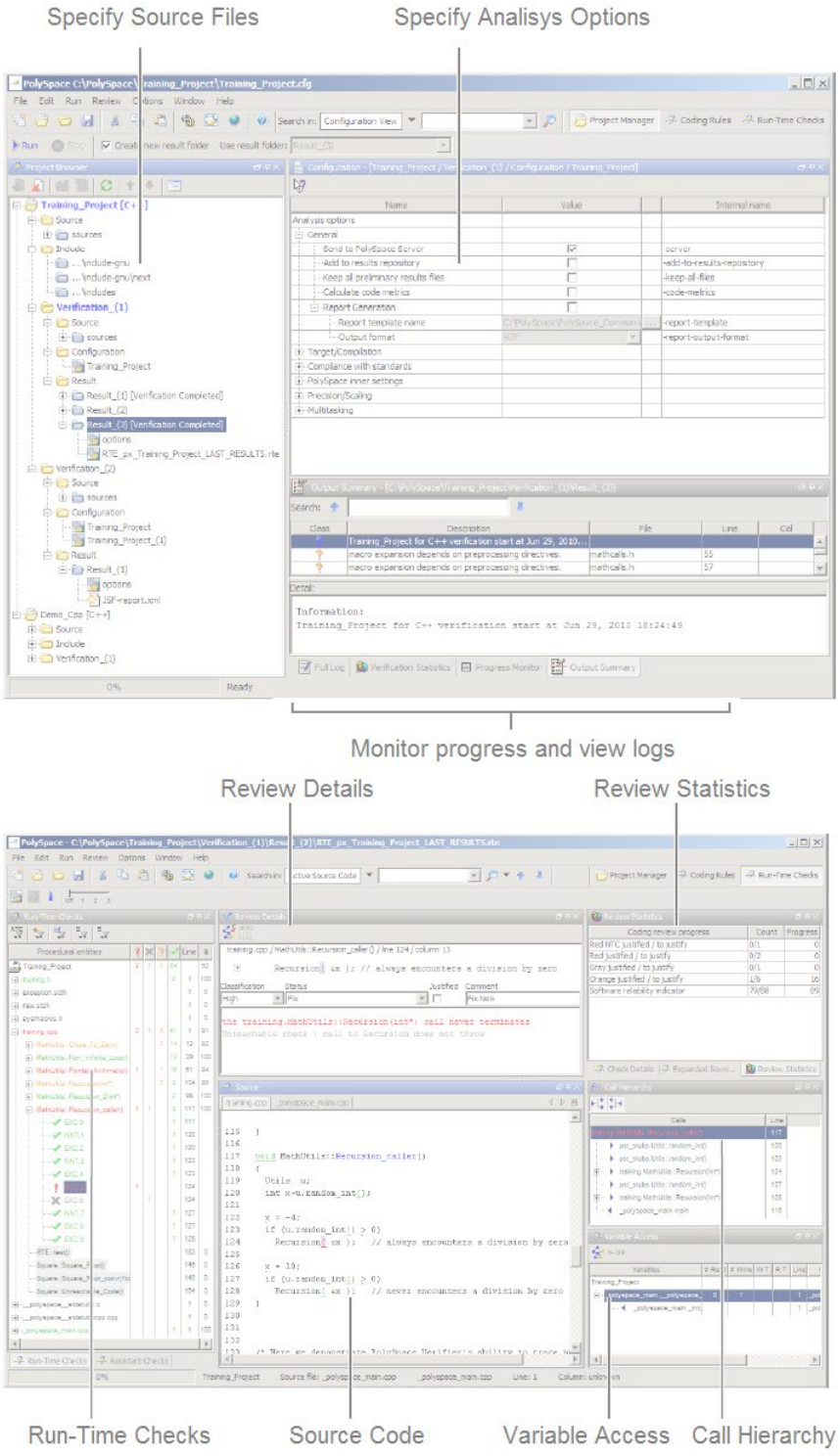


Figura 6.3: *Interfaz del programa verificador de código Polyspace®*

Respecto a IBM Rational Logiscope, se ha utilizado la versión 6.6, incluida en el paquete de evaluación por 30 días de IBM Rational Rhapsody®. Las características funcionales son similares a las ofrecidas por Polyspace®, aunque los errores detectados y los informes generados emplean leyendas diferentes.

Respecto a la detección de errores y defectos de programación, Polyspace® emplea un código de colores para diferenciarlos en categorías:

- Color Rojo: Indica código que contiene errores, ya sea por desbordamiento en el direccionamiento de un vector/puntero, un error aritmético como una división entre cero o una potencial división entre cero, código que provoca que la ejecución caiga en bucles infinitos en los que no se ejecuta prácticamente nada...
- Color Gris: Indica código que no puede alcanzarse, código que en un ciclo de ejecución normal no puede llegar a ejecutarse, como los manejadores de excepciones, las sentencias 'default' en los 'switch-case'. Como recomendación, este tipo de situaciones deben justificarse con comentarios que expliquen el por qué de la circunstancia.
- Color Naranja: Indica código que puede contener errores o su correcto funcionamiento no puede comprobarse. En un MCU todas las funciones de configuración de periféricos se resaltan en color naranja, dado que escriben en registros mapeados en memoria siendo esto direccionamiento de memoria absoluto. En las opciones de Polyspace® pueden deshabilitarse ciertas comprobaciones como la anterior.
- Color Verde: Indica el código fuente que no contiene errores ni ambigüedades. Codigo que se ejecutará correctamente en todas las situaciones.

La revisión del código generado por la toolbox de funcionalidades avanzadas ha sido realizada a partir del código fuente generado por modelos sencillos, en lo que se integraban las funcionalidades a evaluar. A continuación, se muestra en la figura 6.4 el resultado de revisar un modelo para STM32F4 que contiene el bloque de configuración de la CPU y la inicialización y uso de un puerto serie UART (toolbox de funcionalidades básicas de Aimagin® y del puerto USB (toolbox de funcionalidades avanzadas). Los resultados de la figura muestran que el código contenía un error grave, subsanado con la posibilidad de resetar el sistema si fracasa la configuración del reloj de sistema y/o de disparar un pulso digital para activar un circuito electrónico apropiado para desconectar la alimentación y salidas del MCU.

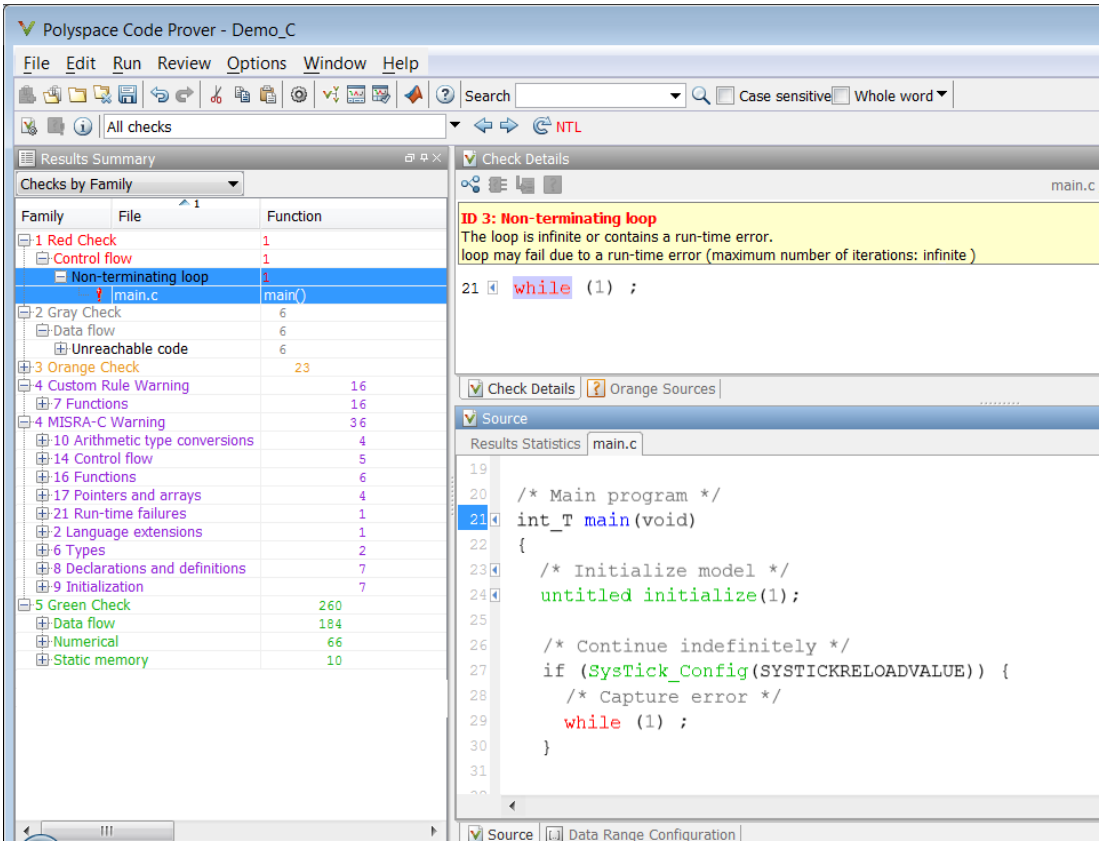


Figura 6.4: *Resultados del progreso en verificación y correción del código en un modelo para STM32F4 en Polyspace®*

Los defectos de color gris se deben a manejadores de excepción que no contenían código. Los defectos de color naranja se debieron a las funciones de configuración de los periféricos del MCU, que escriben en direcciones de memoria absolutas; se ha eliminado este tipo de comprobaciones en las opciones de Polyspace® porque en un MCU este tipo de operaciones son normales. Los defectos de color morado representan partes del código que no se adecuan a las normas MISRA, principalmente son variables sin inicializar, sentencias 'if' sin 'else', 'switch' sin caso 'default', variables inicializadas sin utilizar y divisiones que no comprueban si el denominador es cero. La mayor parte de los defectos relativos a la norma MISRA se deben al código de Aimagin®. El código señalado en color verde no tiene ningún problema y es seguro.

Gracias a estas herramientas de verificación de código fuente, se ha podido mejorar la seguridad y estabilidad de los programas generados desde SIMULINK®,

ya que localiza de manera sencilla y eficiente los posibles defectos, con lo que resulta sencillo corregirlos en los ficheros .TLC que orquestan la generación automática del código fuente.

La figura 6.5 resalta de forma gráfica el número de errores y defectos localizados y cual es el porcentaje del programa contenido en el código fuente que ha podido ser analizado, se excluyen las funciones que no son llamadas nunca, aunque se debe exceptuar las llamadas a IRQ que pueden ser especificadas en las opciones de Polyspace® de forma que las tenga en cuenta en su proceso de revisión.

Esto representa un añadido en lo que ha calidad de software se refiere, ya que las funcionalidades avanzadas han sido programadas con el entorno de desarrollo Keil Uvision® y depuradas con depurador en circuito en el mismo entorno; y posteriormente el código ha sido revisado con Polyspace® e IBM Rational Logiscope® para adecuar el estilo de la programación a uno preventivo respecto a errores y defectos de programación.

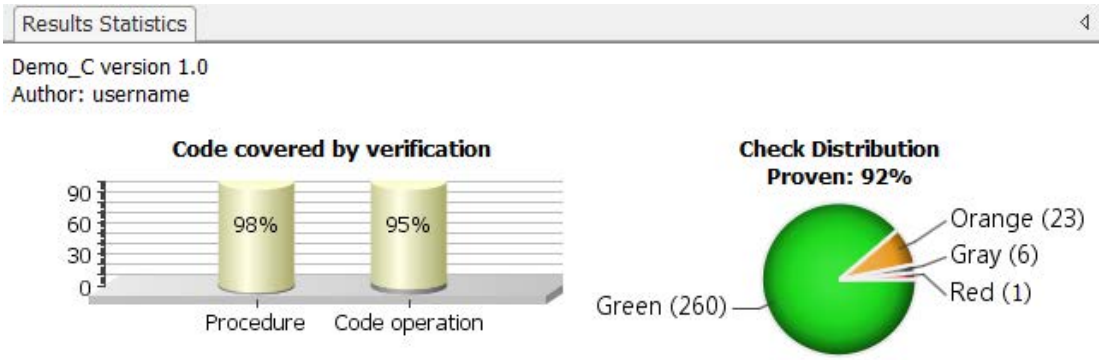


Figura 6.5: *Resultados de la comprobación de código en Polyspace® de forma gráfica*

Capítulo 7

Hardware adicional soportado

En apartados anteriores se ha indicado que los planteamientos propuestos como funcionalidades avanzadas debían poder portarse a diferentes arquitecturas computacionales sin grandes dificultades. Debido a que las funcionalidades avanzadas no requieren la presencia de ningún tipo de sistema operativo, pueden ser adaptadas a prácticamente cualquier MCU y otras arquitecturas que no precisen de sistema operativo. El soporte de la toolbox de funcionalidades avanzadas se ha extendido a sistemas como PlayStation 2® y el MCU NRF51x22 de Nordic®, aunque en estos casos se ha tenido que cubrir también las funcionalidades básicas. Este apartado no es tan exhaustivo como los dedicados al MCU STM32F4 y las funcionalidades avanzadas que se han integrado en esta tesis, en cambio se proporciona información sobre cómo programar el sistema multiprocesador de PlayStation 2®, las características de cada sistema y las interfaces de comunicación que se han empleado.

7.1. PlayStation 2®

PlayStation 2® es un sistema computacional orientado para su uso lúdico desarrollado por Sony Computer Entertainment®. Salió al mercado el 4 de marzo del año 2000. A finales del año 2009 el desarrollo de software para este sistema dejó de requerir licenciamiento oficial, por lo que desde ese momento programar para este sistema y distribuir software para el mismo es completamente gratuito. A día de hoy, y debido al gran número de unidades vendidas, alrededor de 161 millones, resulta muy fácil encontrar una PlayStation 2® del tamaño más reducido al precio de 15 - 20 €.

7.1.1. Arquitectura

Pese a ser un sistema cuya edad es de 14 años, a fecha de escritura de esta tesis, su capacidades computacionales siguen siendo notables debido a lo peculiar en el diseño de su arquitectura Von Neumman. Es un sistema basado en 5 procesadores, 4 de ellos son procesadores de propósito general, mientras que el procesador restante está dedicado únicamente a la realización de operaciones gráficas. Los 4 procesadores de propósito general poseen su propio espacio de memoria RAM dedicado, por lo que cada uno de ellos puede ejecutar un programa en perfecto paralelismo con los demás. La figura 7.1 ilustra de forma visual la arquitectura de un sistema PlayStation 2®.

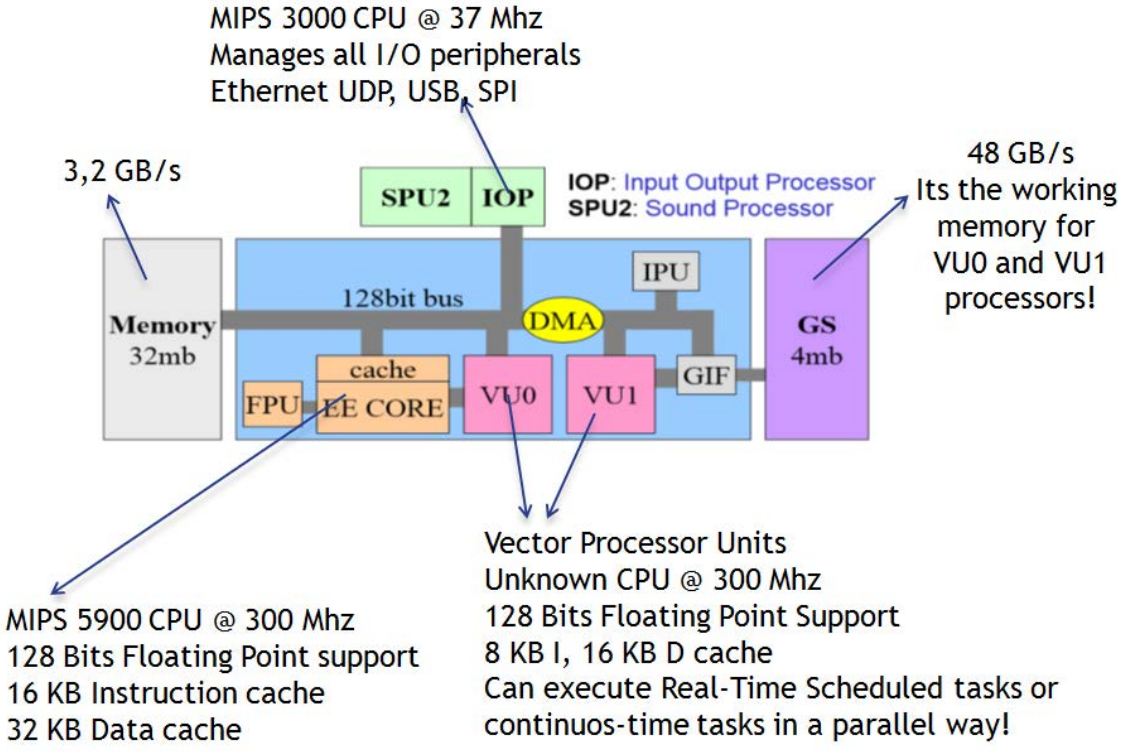


Figura 7.1: *Esquema de la arquitectura de PlayStation 2®. Se muestran las características de los procesadores y el ancho de banda de los diferentes espacios de memoria*

Teniendo presente la figura 7.1 se puede estudiar lo mostrado en la imagen. El procesador principal denominado "EE Core" está basado en una CPU MIPS 5900 [**?**] de 128 bits corriendo a 300 Mhz, como características posee una unidad FPU con soporte de números flotantes de hasta 128 bits, 16 KB como memoria caché de instrucciones y 32 KB como memoria caché para datos. Esta CPU principal puede

acceder al bloque de memoria de 32 MB, a los 4 MB de la memoria de vídeo y comunicarse con la CPU de I/O, la CPU basada en MIPS 3000, accediendo a los 2 MB disponibles para el MIPS 3000. Esta última CPU de 32 bits corriendo a 37 Mhz está heredada del sistema PlayStation 1® y tiene a su alcance 2 MB de memoria; CPU diseñada en el año 1994, su función en PlayStation 2® es la de gestionar los periféricos de I/O tales como el puerto Ethernet, USB, SPI (los mandos de control y las tarjetas de memoria de PlayStation 2® son periféricos SPI, en dichos puertos pueden conectarse sensores que admitan este protocolo como ejemplo). La CPU principal y la de I/O se comunican mediante lectura/escritura en los 2 MB de memoria del MIPS 3000.

Los otros dos procesadores de 128 bits que corren a 300 MHz, denominados unidades de cálculo vectorial, tienen acceso a la de memoria de vídeo (bloque morado a la derecha de la figura 7.1) y a la memoria principal de 32 MB. Originalmente estos procesadores estaban destinados únicamente a ser programados en ensamblador para realizar tareas muy puntuales, las últimas versiones del SDK proveen de compiladores de lenguaje C para estos procesadores; pese a que su repertorio de instrucciones es muy específico para cálculo vectorial, también poseen instrucciones para operaciones aritméticas con números enteros y flotantes así como instrucciones de comparación y salto a otras direcciones de memoria, por lo que se pueden utilizar como procesadores de propósito general, aunque no fuera está la intención original. Los procesadores VU0 y VU1 son prácticamente idénticos, cada uno de ellos puede tener acceso exclusivo a una sección de 2 MB de la memoria de vídeo, aunque el procesador VU1 tiene que compartir estos 2 MB con la memoria de cuadro (la memoria dedicada a mostrar la imagen en pantalla).

7.1.2. Programando el sistema multiprocesador

El arranque del sistema PlayStation 2® está inicialmente gobernado por el procesador MIPS 3000; en dicho arranque se detecta si existen extensiones del programa básico de I/O (BIOS) en cualquiera de las tarjetas de memoria, es una lectura mediante el puerto SPI. Mientras tanto el procesador principal se queda en 'espera', dado que su rutina inicial es saltar a una dirección de memoria y ejecutar un bucle 'while' infinito. El procesador MIPS 3000 ubicará la BIOS extendida, la última versión oficial es la 3.10 del año 2006, a su vez esta BIOS oficial admite extensiones desde la tarjeta de memoria que permiten cargar cualquier programa, en la memoria del procesador principal, cambiando la dirección de memoria inicial a la que salta este procesador; el MIPS 3000 reseteará el procesador principal y éste empezará a ejecutar la BIOS puesto que la nueva instrucción de salto inicial apunta al programa de la BIOS extendida.

Este mecanismo de quedarse en espera, mientras otro procesador carga en el espacio de memoria de otro procesador el programa a ejecutar, es el utilizado para cargar programas personalizados sobre el procesador VU0 y VU1, y también para cargar un programa de gestión de I/O personalizado para el MIPS 3000, como puede ser poder controlar el periférico Ethernet y/o leer desde el puerto USB.

Aunque el sistema consta de 4 procesadores, más un repertorio de funciones gráficas contenidas en la BIOS utilizadas por el procesador que realiza las funciones gráficas (GPU, del inglés 'Graphics Processing Unit'); todos ellos se programan como si de un único proyecto en lenguaje C se tratara, con la excepción de que las funciones que componen el programa destinado a otros procesadores que no son el principal deben ir precedidas en su declaración del identificador ' IOP', ' VU0' o ' VU1'; además el punto de entrada principal para los programas de estos procesadores irá precedido de este identificador más "main', señalando que se trata de la función 'main' del programa de estos procesadores. Estos programas orientados cada uno a un procesador distinto pueden compartir el bloque de memoria de 32 MB, por lo que se pueden utilizar las misma variables globales todos los procesadores, con la precaución de gestionar los accesos de lectura/escritura, mediante flags por ejemplo; aún así el sistema de direccionamiento de buses no permite que dos procesadores accedan a la misma memoria al mismo tiempo. En el caso de las variables locales a las funciones o aquellas precedidas del identificador 'section xx' se ubicarán en la de memoria RAM correspondientes a cada procesador en lugar de emplear el bloque de 32 MB.

Una vez definidos e implementados en lenguaje C los programas para los distintos procesadores, la compilación dará lugar a ficheros de extensión .irx que contendrán los programas compilados y las instrucciones en código máquina necesarias para que el procesador principal ubique los programas orientados a cada procesador y haga que estos inicien su ejecución. El fichero ejecutable resultante con extensión .elf no contiene los programas para cada procesador, sólo contiene el programa para el procesador principal. La carga de estos ficheros .irx se realiza con la función en lenguaje C provista en el SDK denominada 'SifExecModuleBuffer'. Antes de llamar a esta función es necesario indicar sobre que procesador se quiere cargar el código máquina contenido en el fichero .irx, utilizando la función 'SifInitIopHeap' o 'SifInitVu1Heap' dependiendo del procesador sobre el que se desee cargar un programa. A continuación, una vez cargado el programa, se debe resetear el procesador sobre el que se acaba de volcar código máquina llamando a la función 'SifIopReset', 'SifVu0Reset' o 'SifVu1Reset'.

De esta forma generan el código fuente los bloques para SIMULINK® implementados, de forma que la toolbox para PlayStation 2® puede hacer uso de todos los procesadores, creando tareas que se ejecutan en paralelo.

7.1.3. Multitarea en cada procesador

En el sistema PlayStation 2® se puede emplear el mismo mecanismo para lograr la ejecución de múltiples tareas preemptivas, con la ventaja de poder llevar este mecanismo a cada procesador.

El procesador principal posee 16 timers que generan sus correspondientes fuentes de IRQ, cada timer puede dar lugar a una nueva tarea preemptiva y el tiempo libre que quede entre ellas puede utilizarse por la tarea denominada ”Idle Task’ que se ejecuta en tiempo-continuo, o bien puede utilizarse solamente la tarea en tiempo-continuo tal y como se hace en el MCU STM32F4, de forma que el usuario de ambas toolboxes para SIMULINK® no perciba diferencia alguna en cómo debe programar múltiples tareas en ambos sistemas.

Los procesadores VU0 y VU1 sólo cuentan con un timer cada uno, por lo que en estos procesadores sólo se dispone de una tarea en tiempo-real, una tarea preemptiva y la tarea en tiempo-continuo. Aún así se cuenta con la ventaja de que estas tareas en tiempo-real se ejecutan en perfecto paralelismo a las tareas en tiempo-real del procesador principal.

7.1.4. Driver Ethernet UDP

Debido a la inexistencia de funciones adecuadas para manejar el periférico Ethernet mediante transacciones UDP en el SDK (del inglés *Software Development Kit*) de PlayStation 2®, que son las admitidas por el MCU STM32F4 para poder conectar ambos sistemas entre sí, dotando a PlayStation 2® de múltiples interfaces I/O, se debe buscar una alternativa.

Como ya se ha explicado anteriormente, el periférico Ethernet se encuentra gobernado por el procesador MIPS 3000. Por lo que cualquier programa que haga uso de la comunicación Ethernet mediante UDP, deberá contener entre sus ficheros el programa para MIPS 3000 con extensión .irx . En algunos videojuegos comerciales del año 2006 en adelante, se encuentra un fichero denominado 'udp.irx'. Haciendo uso de un desemsamblador-decompilador podrán recuperarse las funciones en lenguaje C que permiten manejar el periférico Ethernet mediante transacciones UDP. El decompilador utilizado es 'Hex-Rays'.

udp.irx

```
1   .sdata
2
3   .globl size_udp_irx
4   size_udp_irx:   .word 54445
5
6   .data
7
8   .balign 16
9
10  .globl udp_irx
11  udp_irx:
12
13
14      .byte 0x7f, 0x45, 0x4c, 0x46, 0x01, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
15      .byte 0x80, 0xff, 0x08, 0x00, 0x01, 0x00, 0x00, 0x00, 0xb4, 0x50, 0x00, 0x00, 0x34, 0x00, 0x00, 0x00
16      .byte 0x8c, 0x9a, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x34, 0x00, 0x20, 0x00, 0x02, 0x00, 0x28, 0x00
17      .byte 0x0a, 0x00, 0x09, 0x00, 0x80, 0x00, 0x00, 0x70, 0x74, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
18      .byte 0x00, 0x00, 0x00, 0x00, 0x1b, 0x00, 0x00, 0x00, 0x1b, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00
19      .byte 0x04, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x90, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
20      .byte 0x00, 0x00, 0x00, 0x00, 0xc0, 0x99, 0x00, 0x00, 0x7c, 0xcc, 0x00, 0x00, 0x07, 0x00, 0x00, 0x00
21      .byte 0x10, 0x00, 0x00, 0x00, 0xff, 0xff, 0xff, 0xff, 0xb4, 0x50, 0x00, 0x00, 0xc0, 0x19, 0x01, 0x00
22      .byte 0xa0, 0x8c, 0x00, 0x00, 0x20, 0x0d, 0x00, 0x00, 0xbc, 0x32, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
23      .byte 0xd0, 0xff, 0xbd, 0x27, 0x44, 0x00, 0xaa, 0x8f, 0x21, 0x40, 0xc0, 0x00, 0xff, 0xff, 0x42, 0x31
24      .byte 0x21, 0x30, 0x40, 0x00, 0x40, 0x00, 0xa2, 0x8f, 0x21, 0x18, 0xa0, 0x00, 0x18, 0x00, 0xa3, 0xa3
25      .byte 0x1c, 0x00, 0xa2, 0xa7, 0x4c, 0x00, 0xa3, 0x8f, 0x50, 0x00, 0xa2, 0x8f, 0x48, 0x00, 0xa5, 0x8f
26      .byte 0x21, 0x48, 0xe0, 0x00, 0x18, 0x00, 0xa7, 0x27, 0x28, 0x00, 0xbf, 0xaf, 0x19, 0x00, 0xa8, 0xa3
27      .byte 0x1a, 0x00, 0xa9, 0xa7, 0x10, 0x00, 0xa3, 0xaf, 0x14, 0x00, 0xa2, 0xaf, 0x36, 0x16, 0x00, 0x0c
28      .byte 0x1e, 0x00, 0xaa, 0xa7, 0x28, 0x00, 0xbf, 0x8f, 0x00, 0x00, 0x00, 0x00, 0x08, 0x00, 0xe0, 0x03
29      .byte 0x30, 0x00, 0xbd, 0x27, 0x02, 0x00, 0x83, 0x90, 0x03, 0x00, 0x82, 0x90, 0x01, 0x00, 0x85, 0x90
30      .byte 0x00, 0x1a, 0x03, 0x00, 0x00, 0x00, 0x86, 0x90, 0x21, 0x10, 0x43, 0x00, 0x00, 0x2c, 0x05, 0x00
31      .byte 0x21, 0x10, 0x45, 0x00, 0x00, 0x36, 0x06, 0x00, 0x08, 0x00, 0xe0, 0x03, 0x21, 0x10, 0x46, 0x00
32      .byte 0x42, 0x43, 0x02, 0x3c, 0x55, 0x53, 0x42, 0x34, 0x80, 0xff, 0x03, 0x24, 0x00, 0x00, 0x82, 0xac
33      .byte 0x0c, 0x00, 0x83, 0xa0, 0x08, 0x00, 0xe0, 0x03, 0x0d, 0x00, 0x80, 0xa0, 0x42, 0x53, 0x02, 0x3c
34      .byte 0x55, 0x53, 0x42, 0x34, 0x08, 0x00, 0xe0, 0x03, 0x00, 0x00, 0x82, 0xac, 0x80, 0xff, 0x02, 0x24
35      .byte 0x06, 0x00, 0x03, 0x24, 0x0c, 0x00, 0x82, 0xa0, 0x0e, 0x00, 0x83, 0xa0, 0x14, 0x00, 0x80, 0xa0
36      .byte 0x04, 0x00, 0x80, 0xac, 0x08, 0x00, 0x80, 0xac, 0x0f, 0x00, 0x80, 0xa0, 0x10, 0x00, 0x80, 0xa0
```

Figura 7.2: *Fichero UDP.IRX abierto con el decompilador Hex-Rays*

La figura 7.2 muestra el contenido del fichero udp.irx, se observa que contiene

instrucciones en código máquina en un formato de 128 bits, esta longitud de bits corresponde con código máquina para el procesador principal, el MIPS 5900. Una vez decompilado se recupera el código en lenguaje C correspondiente a este fichero, 'Hex-Rays' está preparado para decompilar código para PlayStation 2® por lo que identifica las funciones como 'SifExecModuleBuffer' que permiten cargar módulos de programa para el resto de procesadores.

Una de las llamadas a esta función tiene como argumento el puntero a un gran array de datos de 32 bits, estos datos de 32 bits contienen a su vez el código máquina para el procesador MIPS 3000 (es el único procesador de 32 bits en esta arquitectura).

Este array de datos de 32 bits se vuelve a decompilar con 'Hex-Rays' de la misma manera en que se ha realizado con el fichero UDP.IRX, aunque seleccionando como plataforma objetivo PlayStation 2®-MIPS 3000.

Se obtendrán las funciones en lenguaje C contenidas en este código de 32 bits, entre ellas se localizan las funciones de inicialización del periférico Ethernet para funcionar mediante transacciones UDP, incluyendo las funciones que permiten definir los paquetes de datos. Son estas funciones las que se utilizan como código fuente para el bloque gráfico de SIMULINK® que permite utilizar el puerto Ethernet con el paquete de datos deseado, tanto para enviar como para recibir.

7.1.5. PlayStation 2® se encuentra con SIMULINK®

La figura 7.3 muestra un modelo de control programado desde SIMULINK® con la toolbox de funcionalidades avanzadas para PlayStation 2® ejecutándose en la misma. El MCU STM32F4 se comunica mediante Ethernet UDP con PlayStation 2® dotando a ésta de multitud de interfaces de I/O.

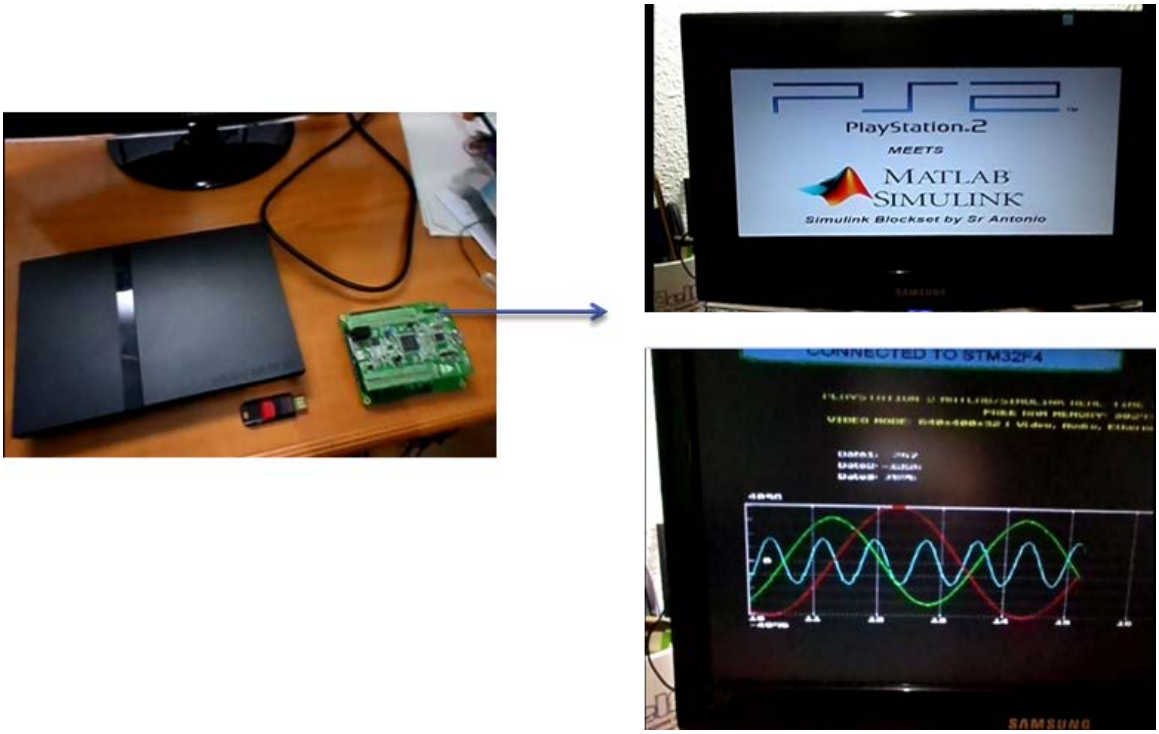


Figura 7.3: *Modelo de control programado en SIMULINK® ejecutándose en una PlayStation 2®. Todo el código fuente ha sido autogenerado por la toolbox de funcionalidades avanzadas*

La toolbox para SIMULINK® de funcionalidades básicas y avanzadas para este sistema computacional comprende el bloque de configuración hardware, utilizarlo sirve para indicar a MATLAB® el objetivo hardware para el que se quiere generar código fuente. También se ha dado soporte a las capacidades de gestión multitarea ya expuestas en el apartado 5; capacidades de vídeo para mostrar texto y gráficas, y el manejo del periférico Ethernet mediante transacciones UDP.

La figura 7.4 muestra la toolbox llevada a cabo en esta tesis que brinda soporte a PlayStation 2® en SIMULINK®. Mientras que la figura 7.5 muestra la librería de bloques con más detalle.

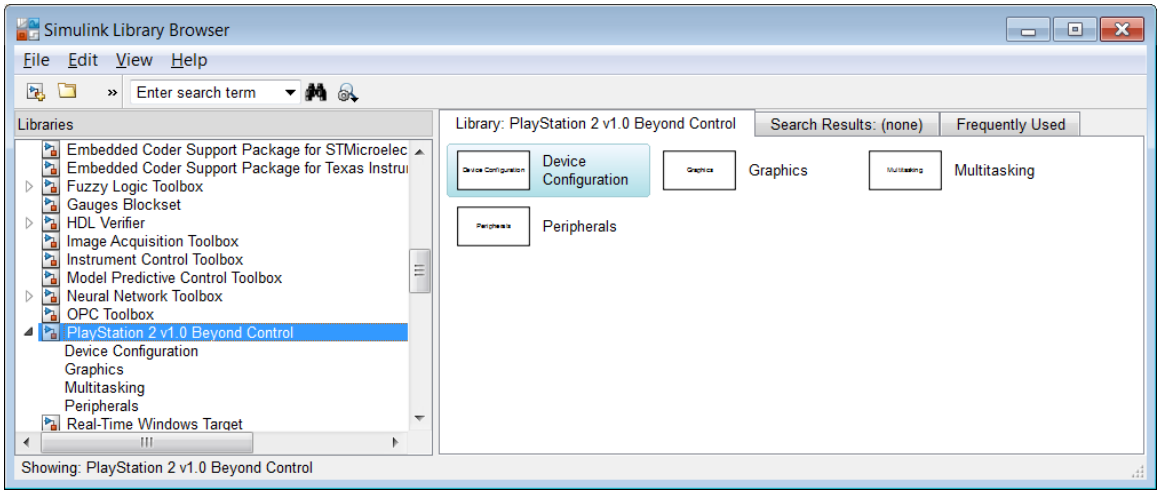


Figura 7.4: *Soporte de PlayStation 2® en SIMULINK®, contiene todas las funcionalidades avanzadas en cada categoría*

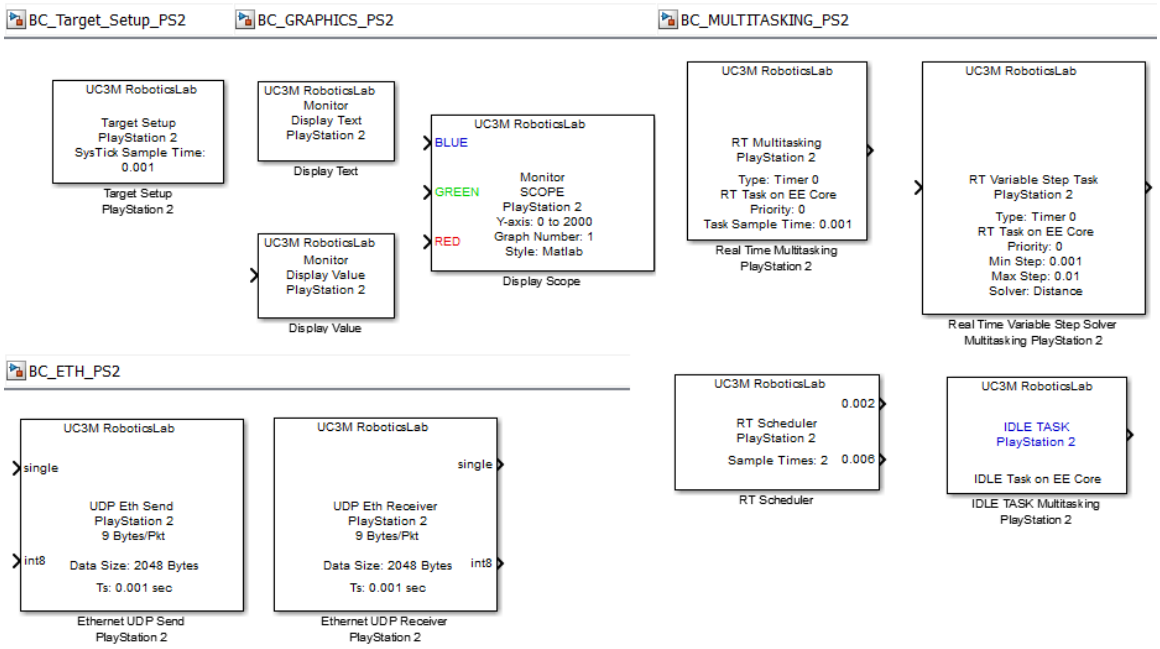


Figura 7.5: *Detalle de la librería de funcionalidades básicas y avanzadas desarrollada para PlayStation 2® en esta tesis*

7.1.6. PlayStation 2® en investigación

Respecto al uso del sistema PlayStation 2® como plataforma apta para su uso en investigación, se encuentra la conversión de este sistema en el controlador embebido en tiempo-real (considerando el reducido tamaño de la segunda versión de PlayStation 2®) de mayor capacidad computacional existente, trabajo llevado a cabo en esta tesis; también supone el controlador embebido con la mejor relación prestaciones/precio.

También se encuentran trabajos de investigación en los que este sistema es utilizado como medio de teleterapia y teleconsulta para pacientes con graves trastornos metabólicos [69], pierden masa muscular de manera totalmente anormal.

Trabajos científicos basados con visión por computador [70] y otros que proponían una metodología para poder programar los procesadores VU0 y VU1 [71, 72] cuando aún no estaba disponible el compilador de lenguaje C para los mismos.

7.2. Microcontrolador NRF51x22

Se trata de un MCU de 32 bits de muy bajo consumo, un máximo de 20 mW, y tamaño muy reducido. Basado en una CPU ARM Cortex-M0 sin unidad FPU corriendo a 16 MHz fabricado por Nordic Semiconductor®; sus interfaces de I/O son las siguientes:

- 8 entradas A/D de 10 bits de resolución
- 3 salidas PWM
- 3 entradas PWM
- 1 puerto UART
- 2 puertos SPI
- 2 buses I2C
- 1 puerto Bluetooth 4.0 a 2Mbits/s
- Sensor de temperatura integrado

La compañía Aimagin® dejó descontinuado el soporte para este MCU y en esta tesis se ha retomado debido al pequeño tamaño y a su interfaz inalámbrica basada en Bluetooth 4.0 integrada en el propio MCU. El coste de adquisición de este MCU montado en una reducida PCB de tamaño 2x2 cm, con antena de radiofrecuencia impresa cuesta 5€. La figura 7.6 muestra la PCB en cuestión.

Figura 7.6: *Placa PCB con MCU NRF51822 y antena impresa para la interfaz inalámbrica*

7.2.1. Funcionalidades básicas añadidas

Las carencias en el soporte de funcionalidades básicas proporcionadas por Aimagin® ha obligado a dar soporte a las interfaces I/O cuyo soporte para MATLAB®/SIMULINK® faltaba.

Entre este soporte se encuentran los bloques gráficos de SIMULINK® para configurar y manejar el puerto SPI y para leer el sensor de temperatura.

7.2.2. Funcionalidades avanzadas

Respecto de las funcionalidades avanzadas, se han intentado portar la mayoría de las funcionalidades avanzadas disponibles en el MCU STM32F4, a excepción de aquellas que hacen uso de las características más avanzadas de la CPU ARM Cortex-M4F como son la posibilidad de utilizar la memoria CCM para albergar el código de tareas y funciones DSP, puesto que en el MCU NRF51x22 no tiene sentido, pues su CPU es muy sencilla y no dispone de los recursos hardware necesarios.

En cambio, se ha dotado al MCU NRF51x22 de capacidad multitarea, es posible utilizar hasta 2 tareas en tiempo-real y una tarea en tiempo-continuo. Las IRQ de los timers no se han utilizado en este caso como gestores de tareas en tiempo-real, pues sólo tiene 3 timers y se perderían todas las interfaces PWM rápidamente. Este MCU no tiene timer de sistema (Systick timer) en cambio posee dos relojes calendario que siempre están activos, aún cuando el MCU esté en modo durmiente. Son las IRQ de estos relojes calendario las que proveen el soporte hardware para gestionar las dos posibles tareas en tiempo-real preemptivas.

La tarea en tiempo-continuo, o la tarea que aprovecha el tiempo libre entre iteraciones de las tareas en tiempo-real también está disponible en este MCU, aunque sólo si no se utilizan las capacidades para dormir la CPU. Se ha integrado la capacidad de dormir la CPU cuando una tarea en tiempo-real concluya su ejecución, la CPU despertará cuando acontezca una IRQ ya sea debida a un timer o a un periférico cualquiera.

También se ha añadido el soporte para tareas en tiempo-real con paso de ejecución variable, la reprogramación de los manejadores de excepciones, la posiblidad de resetear el MCU en cualquier momento, configurar el watchdog, sincronizar los relojes de tiempo-real con otros controladores en sistemas distribuidos (en este caso también debido a un mensaje de radiofrecuencia)... todas aquellas funcionalidades avanzadas del MCU STM32F4 que son posibles con los recursos hardware

disponibles en este pequeño MCU. La figura 7.7 ilustra la toolbox de funcionalidades avanzadas, y básicas; que permite programar el MCU NRF51x22 con un gran nivel de abstracción por medio del lenguaje gráfico de SIMULINK®. La figura 7.8 muestra algunas de estas librerías de bloques gráficos con más detalle.

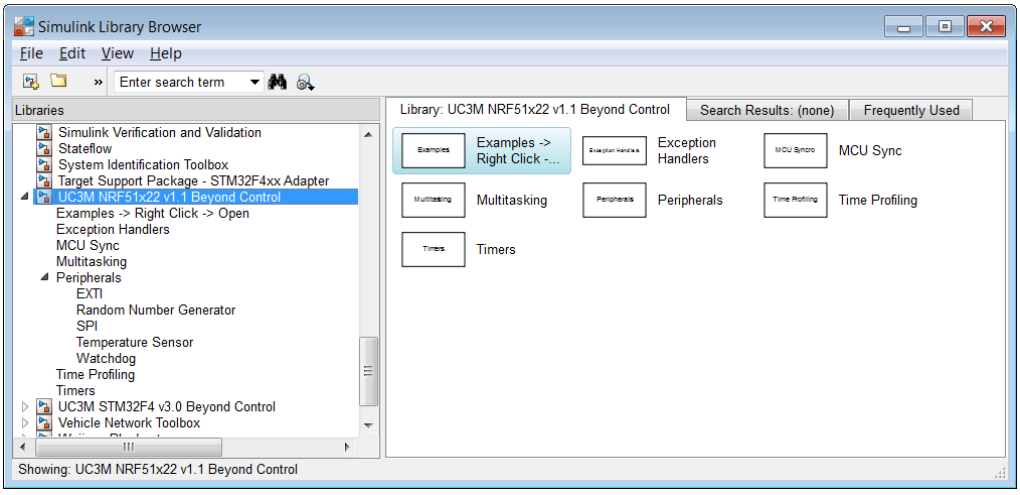


Figura 7.7: *Toolbox de funcionalidades avanzadas para el MCU NRF51x22*

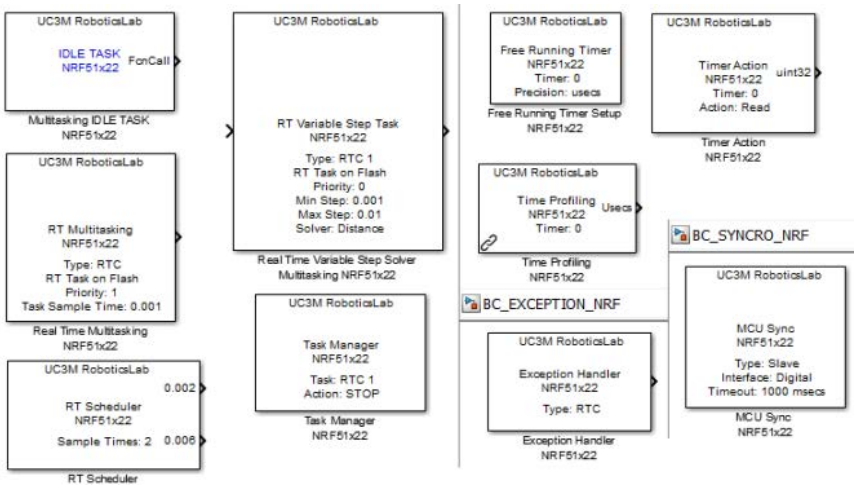


Figura 7.8: *Algunas de las librerías de bloques gráficos desarrolladas en esta tesis para el MCU NRF51x22*

Capítulo 8

Resultados experimentales. Validación del sistema ARCP

La mejor manera de refrendar el propósito del sistema ARCP es someterlo a intensa utilización experimental, de forma que pueda demostrar su utilidad y viabilidad. Este uso experimental ha sido realizado por distintos colectivos, tanto pertenecientes a la Universidad Carlos III de Madrid (UC3M) como personas ajenos a ella, tanto españoles como extranjeros. El sistema ARCP ha sido utilizado y está siendo utilizado como medio de puesta en práctica y verificación de teorías en tesis doctorales, tesis de máster y proyectos fin de carrera; además de los diferentes usos en proyectos nacionales y Europeos que tienen lugar en los laboratorios de investigación del departamento de Ingeniería de Sistemas y Automática de la UC3M.

En los apartados sucesivos se muestran detalladamente las diferentes puestas en práctica del sistema ARCP.

8.1. Plataforma de presión.

La plataforma de presión es un sistema compuesto de 256 sensores resistivos impresos sobre láminas de materiales flexibles. La técnica para producirlos está basada en impresión mediante 'screen-printing'. La figura 8.1 muestra en detalle unos de estos sensores de presión y la figura 8.2 muestra la plataforma al completo. Este dispositivo está diseñado y fabricado en la fundación CIDETEC, localizada en San Sebastián parque tecnológico de Miramón, siendo esta fundación un miembro partícipe en el grupo HYPER. CIDETEC solicitó a la UC3M la integración de esta plataforma haciendo uso del sistema ARCP.

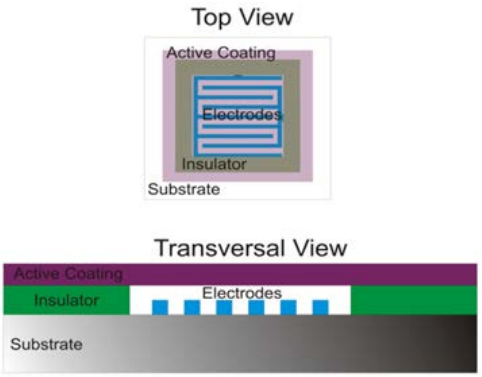


Figura 8.1: *Vistas superior y transversal de un sensor de presión*

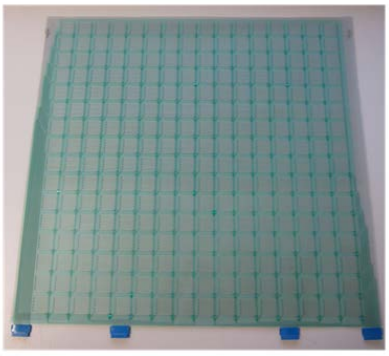


Figura 8.2: *Imagen de la matriz de 16 x 16 sensores (45 cm^2 $\times$ 45 cm^2)*

El principio de funcionamiento de estos sensores es sencillo, teniendo en cuenta la relación entre las dos superficies en contacto, a medida que ésta se incrementa, el valor de la impedancia decrece; debido a la aparición de nuevas rutas resistivas en paralelo. Desde el punto de vista de conversor ADC y del circuito de acondicionamiento, un valor de resistencia menor se traduce en un valor mayor de tensión a la salida del circuito de acondicionamiento. Los sensores empleados tienen una superficie de 2 cm^2, diseñado para una presión de 10 kg/cm^2, por tanto la presión de aplicación máxima para estos sensores es de 20 Kg. por sensor.

El circuito de acondicionamiento está diseñado para evitar los indeseados efectos debidos a 'cross-talking' entre diferentes rutas resistivas, en sistemas basados en matrices de sensores resistivos. En la literatura se encuentran diferentes técnicas [73], entre ellas la más difundida es la de 'grounding/virtual grounding'; cuyo objetivo es mantener el mismo nivel de tensión en los extremos de las impedancias parásitas de forma que no afecten a la medida analógica. El circuito electrónico de la figura 8.3 ilustra el esquema a seguir para eliminar la indeseable contribución explicada con anterioridad [74].

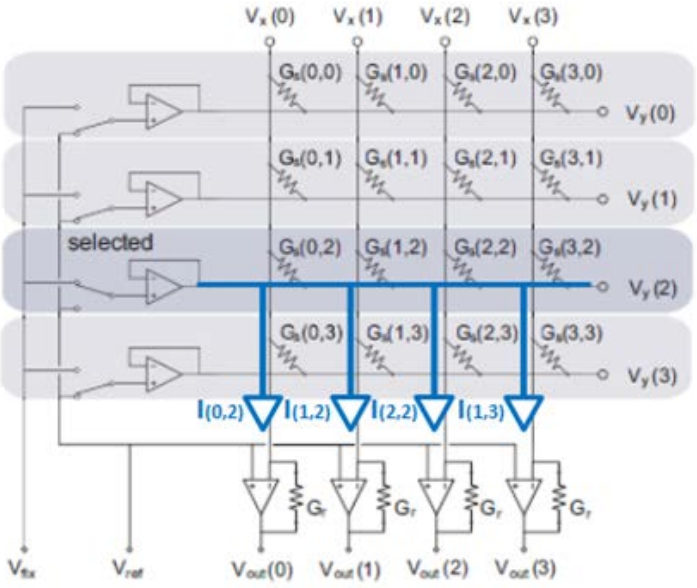


Figura 8.3: *Circuito electrónico de acondicionamiento de señal empleado para obtener los datos analógicos provenientes de los sensores*

El proceso de integración de la plataforma de sensores conlleva los pasos mostrados en la figura 8.4.

Con un método de trabajo tradicional, las etapas denominadas "*analog data acquisition and processing*" and "*output data via digital interfaces*", requerirían bastantes días de trabajo incluyendo la depuración mediante ejecución paso a paso. En cambio, al utilizar el sistema ARPC, estas etapas pudieron ser completadas en una mañana de trabajo, incluyendo la verificación del funcionamiento del sistema. Sin necesidad de depurar el código, puesto que todo el trabajo de depuración ya fue realizado en la creación de las toolboxes para SIMULINK®.

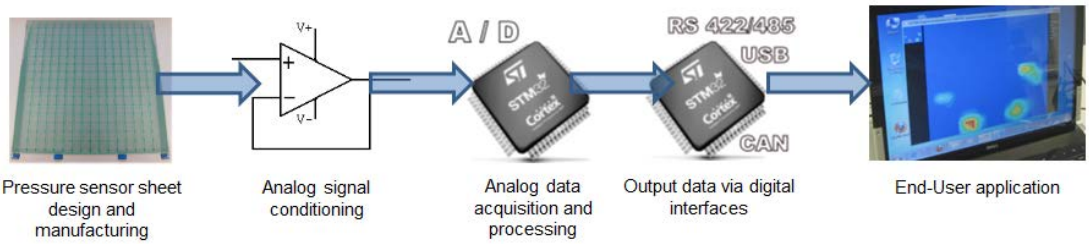


Figura 8.4: *Etapas de la integración, desde la alfombra de sensores hasta un sistema útil*

El modelo diseñado en lenguaje gráfico para el sistema ARCP, mostrado en la figura 8.6, realiza 10 barridos analógicos consecutivos a cada fila de sensores, aplicando un filtro paso bajo, una vez realizados el barrido de las 16 filas, se calcula el centro de gravedad y se envían este dato y la matriz entera por USB y UART; el sistema ARPC realiza estas operaciones a una frecuencia de 60 Hz, mientras tanto, en el tiempo libre entre iteraciones del barrido analógico, se van enviando mensajes por el bus CAN, ya que este bus no puede enviar grandes paquetes de datos de una sola vez. Por tanto se ha recurrido a utilizar la funcionalidad avanzada de poder emplear el tiempo libre entre iteraciones de una tarea en tiempo-real para algo útil. La figura 8.5 ilustra el aprovechamiento del tiempo por parte del sistema ARCP en la integración de la plataforma de sensores de CIDETEC.

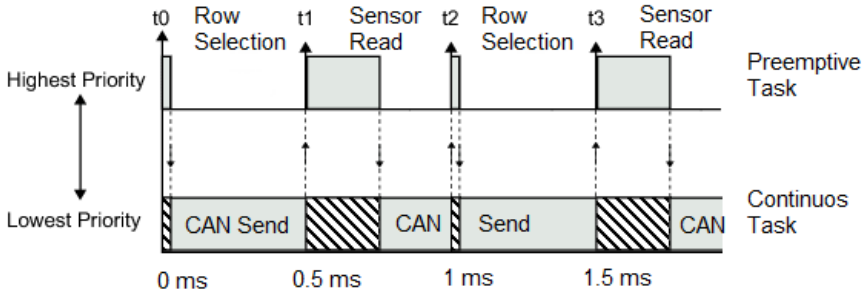


Figura 8.5: *Aprovechamiento del tiempo. Los intervalos de tiempo libre entre iteraciones de la tarea en tiempo-real se utilizan para enviar paquetes de datos por bus CAN*

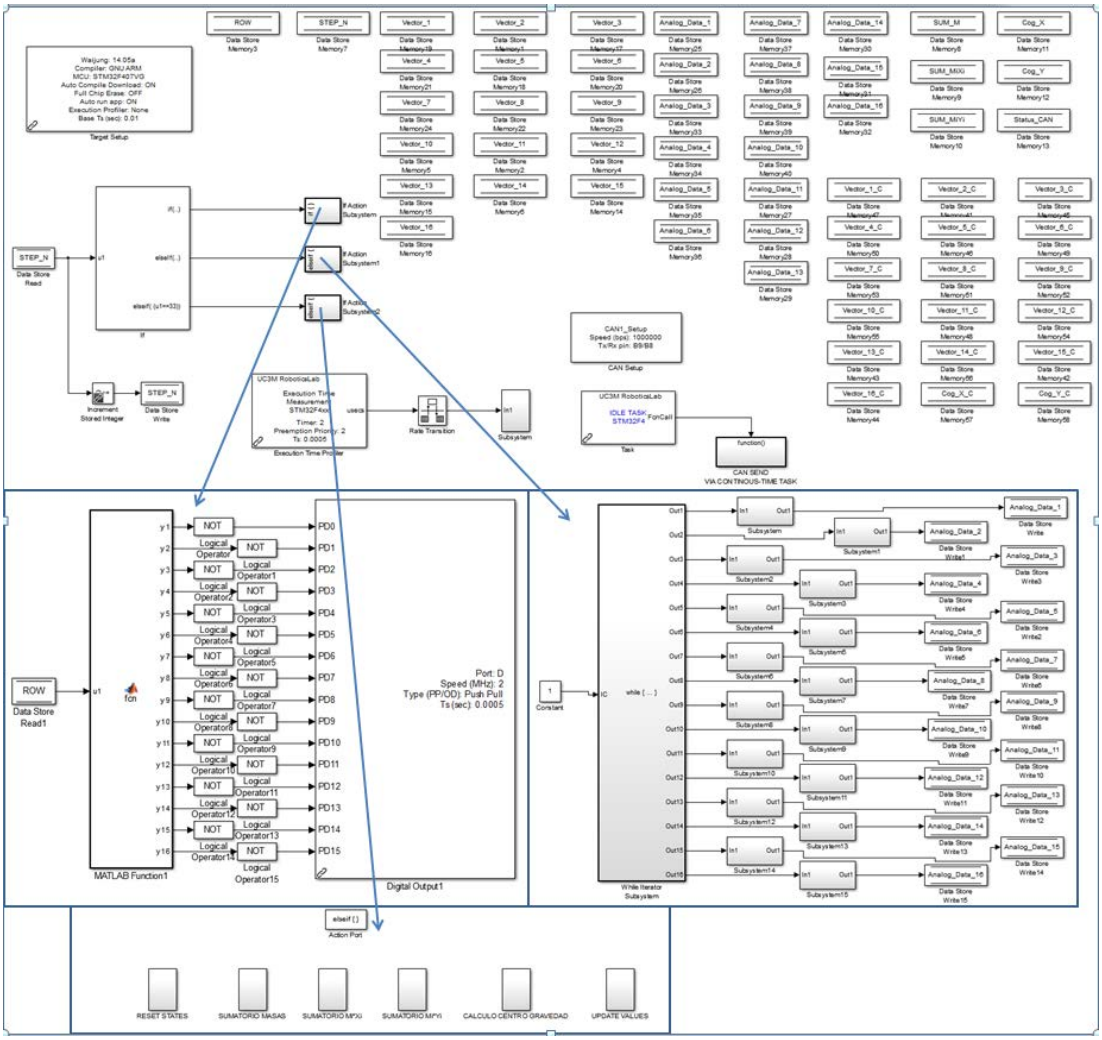


Figura 8.6: *Programa en SIMULINK para el MCU STM32F4 para gobernar la plataforma de presión*

Los resultados una vez integrada la plataforma con el sistema ARPC se ilustran en las figuras 8.7 y en la figura 8.9, ésta última muestra la interfaz gráfica en funcionamiento, desarrollada utilizando las librerías DirectX®, en la que muestra la distribución de la presión sobre la plataforma de sensores, con un código de colores y datos interpolados mediante un filtro bilineal. La figura 8.8 muestra el error máximo obtenido al leer los sensores.

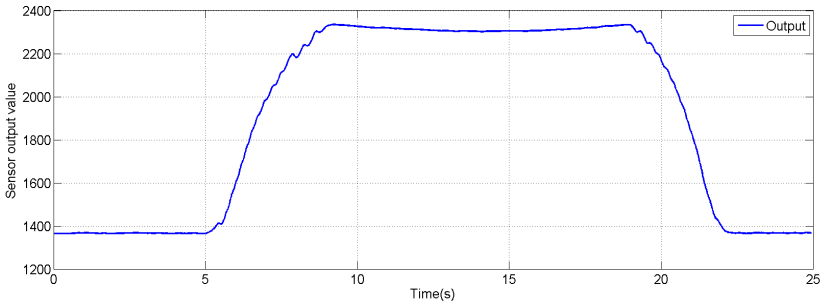


Figura 8.7: *Respuesta de un sensor de presión de la plataforma*

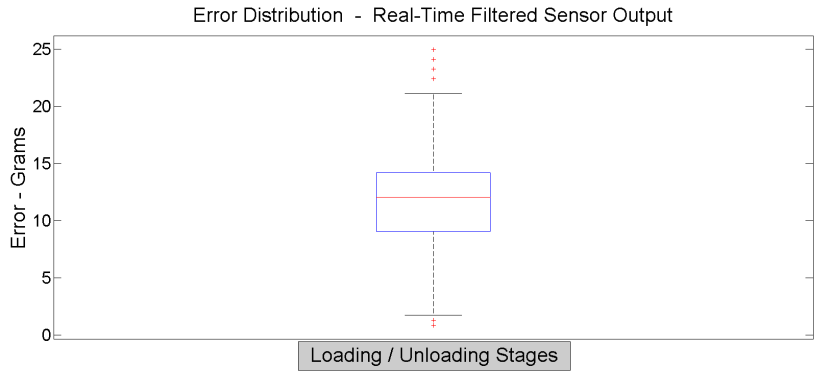


Figura 8.8: *Distribución del error*

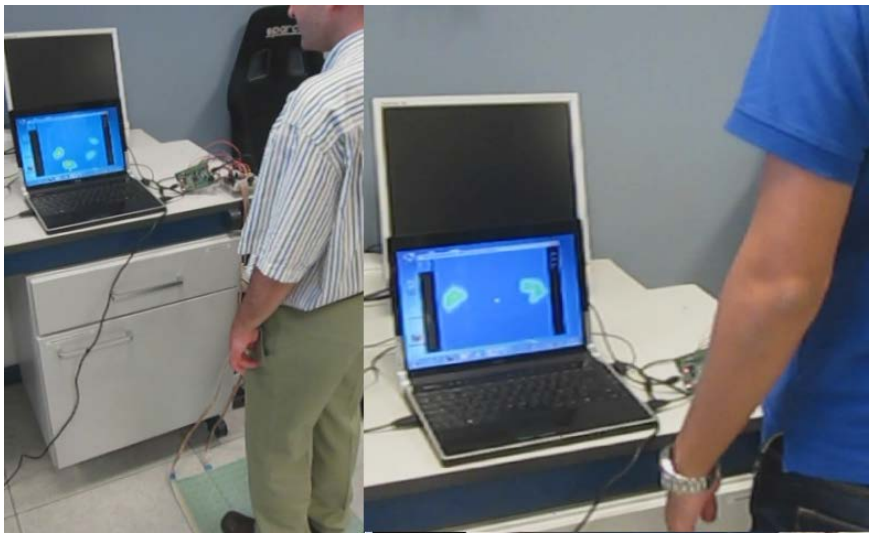


Figura 8.9: *Usuarios interactuando con el sistema completo de la plataforma de presión*

8.2. Control en posición/velocidad de actuadores basados en SMA.

Un actuador basado en aleaciones con memoria de forma (SMA, del inglés 'Shape Memory Alloy') consta de un filamento formado por una aleación metálica con la capacidad de acortar su longitud dependiendo de la temperatura a la que se someta a dicho filamento. Se encuentran disponibles en diferentes diámetros. La principal característica que los hace interesantes en su uso como sistemas de actuación es su buena relación peso/fuerza ejercida, un filamento de apenas unos gramos es capaz de levantar/mover una masa de 8 Kg.

Se trata de sistemas de actuación basados en efectos térmicos; el calor necesario para originar la contracción del actuador procede de una corriente eléctrica aplicada al mismo. La impedancia del actuador junto con la corriente que se fuerza a circular origina un consumo energético manifestado en forma de calor, que modifica la estructura cristalográfica del material, dando lugar a una reducción de la longitud. Esta reducción de la longitud puede alcanzar hasta el 4 - 5 $\%$ de la longitud total del filamento.

Controlar la posición del actuador es equivalente a decir que se debe controlar la contracción del filamento; para lograr este objetivo se debe controlar la magnitud de la corriente eléctrica aplicada. En los algoritmos de control en posición - velocidad probados en esta tesis la magnitud de la corriente eléctrica procede de una etapa de potencia de muy alta eficiencia energética (las pérdidas de potencia en el circuito conmutador son mínimas), optoacoplada al sistema ARCP que provee de una señal modulada en ancho de pulso (PWM, del inglés 'Pulse Width Modulate').

Para poder ensayar y experimentar distintos algoritmos de control para este tipo de actuadores, se hizo necesario llevar a cabo el diseño y construcción de un banco de pruebas en el que realizar dichos experimentos y ensayos. Se debe agradecer a Valeriu Baciu su tesis de máster en la que dió forma al actual banco de pruebas y al sencillo sistema de crimpado para los filamentos. La figura 8.10 muestra fotografías del banco de pruebas disponible en los laboratorios del departamento de Ingeniería de Sistema y Automática de la UC3M, en las mismas se han resaltado los componentes de forma que puedan ser identificados.

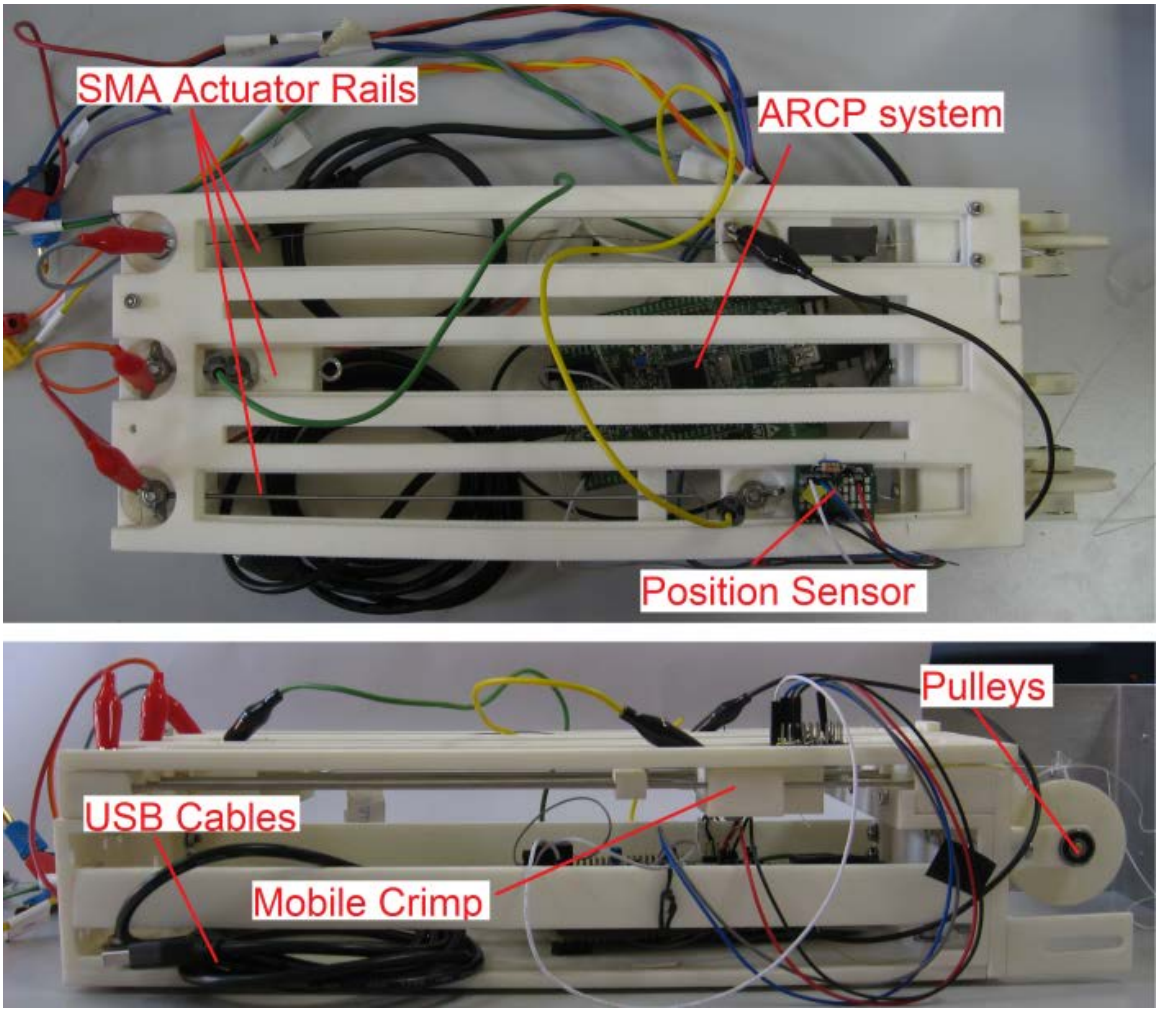


Figura 8.10: *Banco de pruebas para actuadores basados en SMA. Se señalan los principales componentes que lo forman*

El banco de pruebas posee tres carriles horizontales en los que ubicar sendos actuadores, así como sensores de posición basados en efecto hall con interfaces digitales (I2C - SPI) que cuentan con una precisión de 0.488 μm. El actuador al contraerse provoca el movimiento del carrito móvil (en la figura 8.10 se señala como 'mobile crimp'), el sensor de posición registra la evolución en la posición del carrito. A la derecha, conectados mediante filamentos de nylon a cada carrito móvil, se colocan pesos variables u otros sistemas que permitan aplicar cargas a los actuadores. El sistema ARPC y la electrónica de potencia, se ubican dentro de la estructura del banco de pruebas; al sistema ARCP están conectados todos los sensores y conmutadores de potencia. Los cables USB permiten adquirir datos o enviar órdenes al ARCP, así como reprogramar el algoritmo de control.

Los filamentos de SMA poseen un comportamiento altamente no lineal, comportamiento estudiado en profundidad en la tesis doctoral de Alejandro Martín Clemente [9].

Los algoritmos de control propuestos por el autor de esta tesis comprenden dos modalidades, un algoritmo basado en controlar continuamente la referencia en posición y otro algoritmo basado en controlar la referencia en velocidad observando si se ha alcanzado la posición deseada. La figura 8.11 ilustra el diagrama de bloques correspondiente al **algoritmo en posición**.

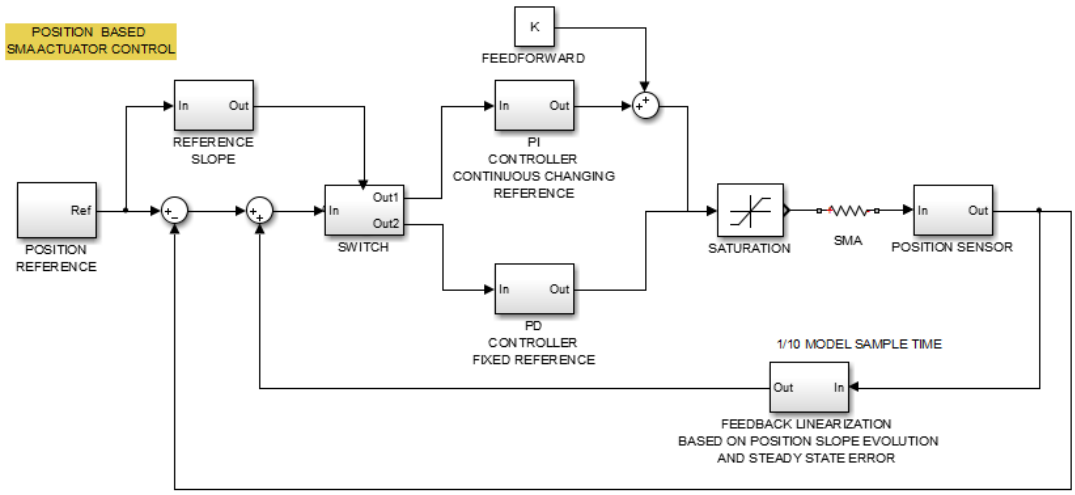


Figura 8.11: *Diagrama de bloques del algoritmo de control en posición para SMA*

Este algoritmo posee dos vías de ejecución; la correspondiente al controlador PI se utiliza para seguir referencias en posición que evolucionan constantemente (referencias senoidales, triangulares...), el valor integral ayuda a que la respuesta del actuador no se distancie de la señal de referencia dictada, la prealimentación otorga al actuador una respuesta inicial más rápida de forma que se evita la acumulación de excesivo valor integral que daría lugar a una respuesta en el actuador de carácter ondulatorio. El controlador PD se utiliza para mantener la posición cuando ésta no varía, la componente derivativa hace al sistema de actuación robusto frente a variaciones bruscas de la temperatura, ya que estas variaciones se traducen en variaciones bruscas registradas por el sensor de posición que provocarán una rápida aportación de corriente debido a la componente derivativa, necesaria para mantener la posición deseada. Ambos controladores se encuentran linealizados por retroalimentación, esta linealización está basada en una serie de sentencias condicionales que prestan atención a la evolución en la pendiente de la posición y a la distancia hasta la posición deseada, de forma que se detecta cuando el actuador

está aproximándose a la referencia demasiado rápido, el linealizador disminuirá en este caso el valor de entrada a los controladores progresivamente de forma que se evite la aparición de sobreoscilación o este linealizador detectará cuando los controladores están comenzando a tener problemas para llevar el actuador al punto deseado, debido a la evolución en el valor de la pendiente en la posición registrada por el sensor, de forma que se incremente progresivamente el valor de entrada a los controladores, permitiendo de esta forma llevar el actuador al punto deseado en forma y tiempo satisfactorios.

El linealizador por retroalimentación también tiene la función, inherente a su funcionamiento, de evitar que el valor de la componente proporcional tienda a cero cuando el actuador esté cerca o haya alcanzado la posición deseada.

Los resultados experimentales de este algoritmo de control se ilustran en la figura 8.12 para referencias de posición con variación constante; en la figura 8.13 se ilustra el rendimiento del mismo para referencias de posición fijas.

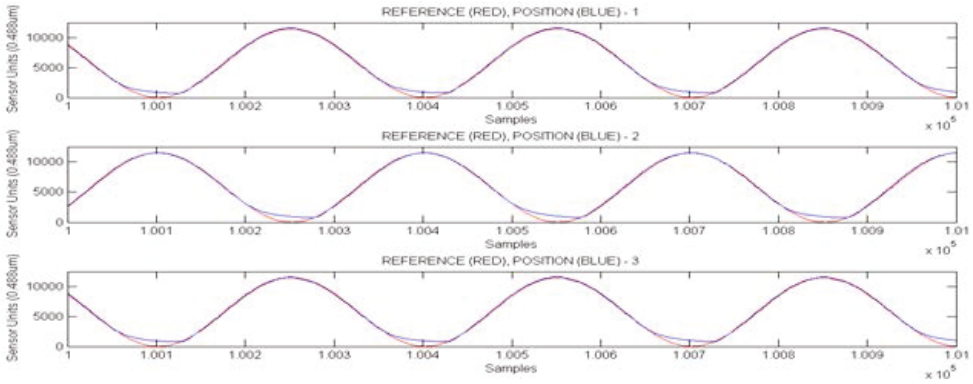


Figura 8.12: *Resultados control en posición para 3 actuadores simultáneos*

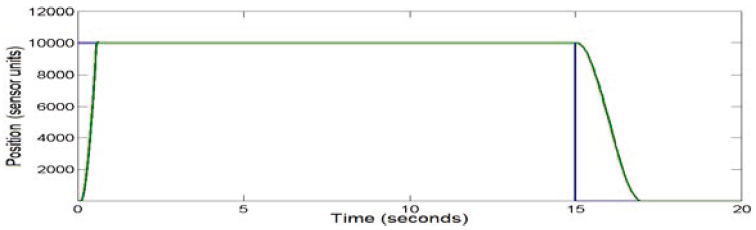


Figura 8.13: *Respuesta control en posición a entrada escalón*

El **algoritmo de control en velocidad** se muestra en forma de diagrama de bloques en la figura 8.14. El control en velocidad se basa en gran medida en el algoritmo de control en posición; el controlador está comprobando constantemente si el actuador basado en SMA ha alcanzado la posición deseada o está próximo a hacerlo, mientras tanto actúa el controlador en velocidad, basado en un PI con prealimentación, la señal de realimentación a este controlador en velocidad es una estimación suavizada de la velocidad calculada a partir de una serie de registros del sensor de posición; debido a que la velocidad es una variable derivada del valor en posición, ésta podría presentar variaciones bruscas tanto en sentido positivo como negativo, el estimador de velocidad intenta minimizar estas bruscas variaciones que serían contraproducentes para el control en velocidad. En este controlador el valor de prealimentación depende en gran medida de la referencia en velocidad. En definitiva, el algoritmo de control en velocidad intenta llevar el actuador de SMA a la posición deseada con la velocidad de contracción deseada.

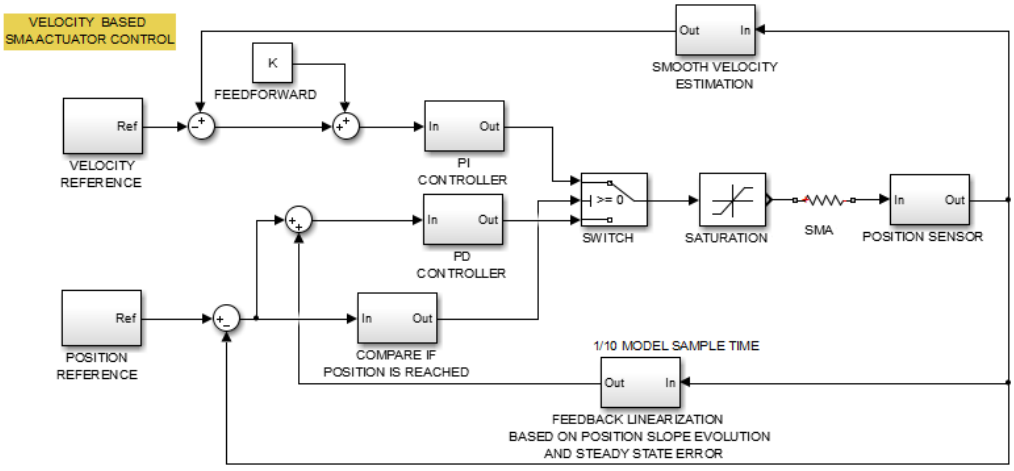


Figura 8.14: *Diagrama de bloques del algoritmo de control en velocidad para SMA*

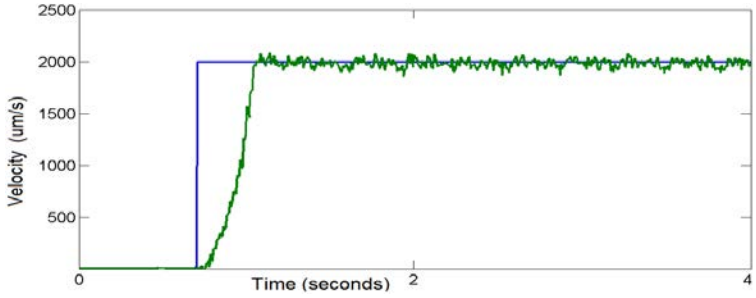


Figura 8.15: *Respuesta del control en velocidad para SMA*

Respecto al rendimiento de los algoritmos de control descritos, la medición en los tiempos de ejecución para el sistema ARPC utilizando el MCU STM32F4, seleccionando el compilador de Keil® (versión 5.10) con las opciones de compilación optimizadas para mejorar el tiempo de ejecución ofrecen los siguientes resultados; teniendo en cuenta que se incluye el tiempo de adquisición del dato del sensor de posición, que asciende a 14 microsegundos:

Tipo de Control	**Ejecución desde CCM**	**Tiempo (microsegundos)**
Posición	No	68
Posición	Si	42
Velocidad/Posición	No	72
Velocidad/Posición	Si	43

Los cálculos implicados en cada iteración de los algoritmos de control en posición o en velocidad emplean tipos de datos flotantes de precisión simple, flotantes de 32 bits. Utilizando tipos de datos en punto fijo de 32 bits con la configuración (1,12,32); configuración que significa datos con signo, 12 bits para representar la parte decimal y 19 bits para representar la parte entera, los tiempos de ejecución para el mismo MCU y opciones de compilación ascienden a los siguientes:

Tipo de Control	**Ejecución desde CCM**	**Tiempo (microsegundos)**
Posición	No	63
Posición	Si	40
Velocidad/Posición	No	67
Velocidad/Posición	Si	41

Como se observa en la tabla de resultados anterior, prescindir del uso de cálculos en coma flotante no tiene un impacto significativo en el MCU STM32F4, por lo que resulta más sencillo y ventajoso hacer uso de la unidad FPU integrada.

El programa en lenguaje gráfico que describe los controladores para actuadores basados en SMA utiliza funcionalidades básicas y las capacidades multitarea en tiempo-real provenientes de las funcionalidades avanzadas desarrolladas en esta tesis, así como los bloques para medir los tiempos de ejecución. Una tarea en tiempo-real para controlar los actuadores y otra tarea en tiempo-real para supervisar el correcto funcionamiento de los sensores.

8.2.1. Control en posición de SMA mediante compesador de histéresis Prandtl-Ishlinskii.

Este apartado corresponde con los estudios y análisis llevados a cabo en la tesis doctoral de Alejandro Martín Clemente [9], concluida en Enero de 2014, en el departamento de Ingeniería de Sistemas y Automática de la UC3M.

En dicha tesis se plantean modelos teóricos basados en compensadores de histéresis con la intención de controlar sistemas de actuación electromecánicos con comportamiento no lineal, la figura 8.16 muestra el esquema de control. Todos los ensayos experimentales y obtención de datos con la finalidad de validar los modelos teóricos a partir de los ensayos prácticos, fueron realizados mediante el uso intensivo del sistema ARCP. En concreto se utilizó para estos fines el banco de pruebas detalladamente explicado en el apartado 8.2 .

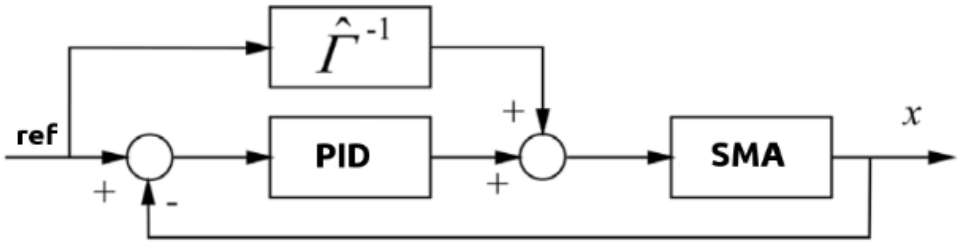


Figura 8.16: *Estrategia de control en prealimentación mediante el uso del compensador de Prandtl-Ishlinski*

8.3. Exoesqueleto ligero-flexible para la muñeca.

Se trata de un dispositivo de los denominados 'soft-exoskeleton', hace uso de los algoritmos de control y actuadores explicados en el apartado inmediatamente anterior. El dispositivo se muestra en la figura 8.17, figura en la que se han señalado los elementos principales. La particularidad de este dispositivo no sólo radica en el concepto de su estructura mecánica, o la casi ausencia de la misma. Sino que el controlador es completamente independiente y autonómo, en la manera en que provee de 'feedback' visual mediante una interfaz de usuario mostrada en una pantalla LCD embebida. No se necesita de un ordenador ni ningún otro dispositivo para poder operar el exoesqueleto; el controlador se muestra en la figura 8.18. La interfaz de usuario se diseña y programa mediante la toolbox de funcionalidades avanzadas para SIMULINK® y su soporte para pantallas embebidas.

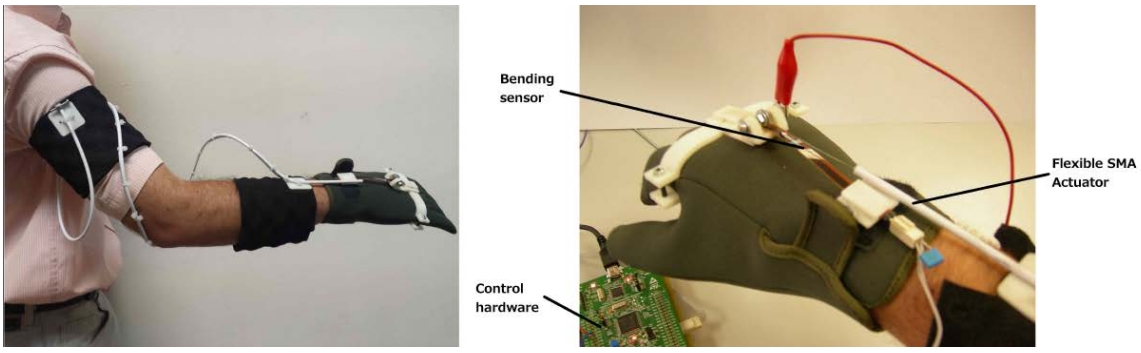


Figura 8.17: *Soft-Exoskeleton para rehabilitación de la articulación de la muñeca*

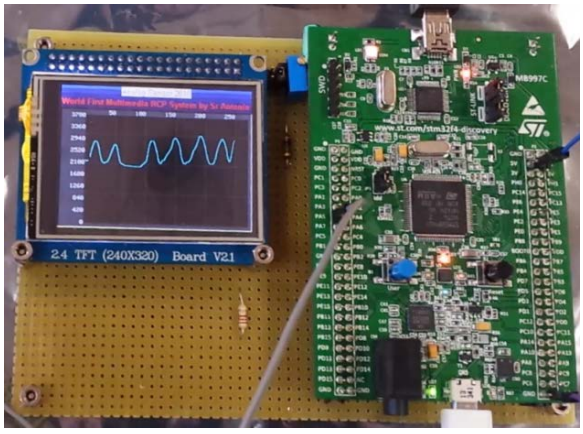


Figura 8.18: *Sistema ARCP con interfaz de usuario en pantalla LCD*

8.4. Mano robótica imprimible actuada con SMA.

Otro dispositivo que hace uso de los algoritmos de control para SMA explicados en el apartado 8.2. En este caso los actuadores basados en SMA, al igual que el sistema para rehabilitación de la muñeca del apartado anterior, emplean estos actuadores enfundados en un cable bowden (al estilo de un freno de bicicleta) [75], que a su vez contiene una funda de teflón para aliviar el rozamiento e impedir el cortocircuito que provocaría el contacto directo con la funda bowden metálica. Este enfundado permite dar cualquier forma al actuador basado en SMA y acelera el proceso de evacuación del calor, permitiendo que el filamento de SMA se enfríe más deprisa; por tanto aumenta la frecuencia en los ciclos de funcionamiento del actuador.

La figura 8.19 muestra el sistema mecánico de la mano robótica de plástico imprimible y el resto de componentes destacados, los MCU y sensores utilizados. Es un ejemplo de sistema de control distribuido, en el que se emplean 2 MCU NRF51822, uno de ellos está ubicado en la mano robótica, siendo el encargado de adquirir los datos analógicos de los sensores de posición y filtrarlos para a continuación enviarlos de forma inalámbrica con la interfaz Bluetooth 4.0 a 2 Mbits/s integrada en dichos MCUs. El restante MCU NRF51822 es el encargado de recibir los datos de posición enviados inalámbricamente y transferírselos al controlador principal, un MCU STM32F4. En el caso de utilizar sólo 3 dedos, podría obviarse la utilización del MCU STM32F4.

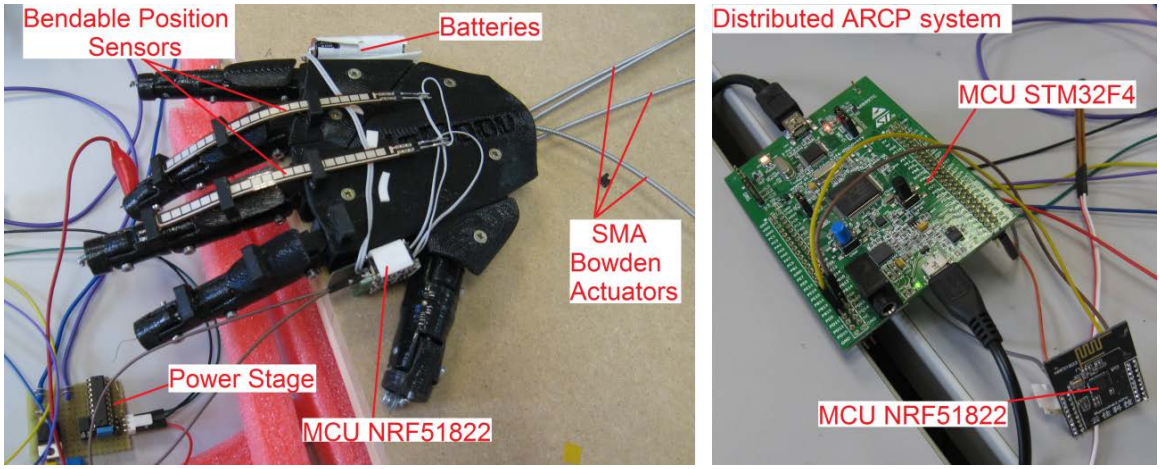


Figura 8.19: *Mano robótica imprimible y sistema ARCP distribuido que la controla*

Los resultados de control en posición de esta mano robótica se muestran en las figuras 8.20 , la recuperación a la posición inicial de los dedos la realiza otra funda bowden metálica ubicada de forma opuesta a la actuador SMA enfundado; la funda bowden de recuperación no se encuentra actuada, se aprovecha el efecto natural de recuperación de la funda gracias a que está constituida en forma de muelle.

Se han utilizado las nuevas funcionalidades multitarea integradas en esta tesis, así como las capacidades de vídeo embebidas integradas también por esta tesis. También se han probado las capacidades de sincronización para sistemas de control distribuido, dado que este sistema emplea MCU STM32F4 y MCU NRF51822.

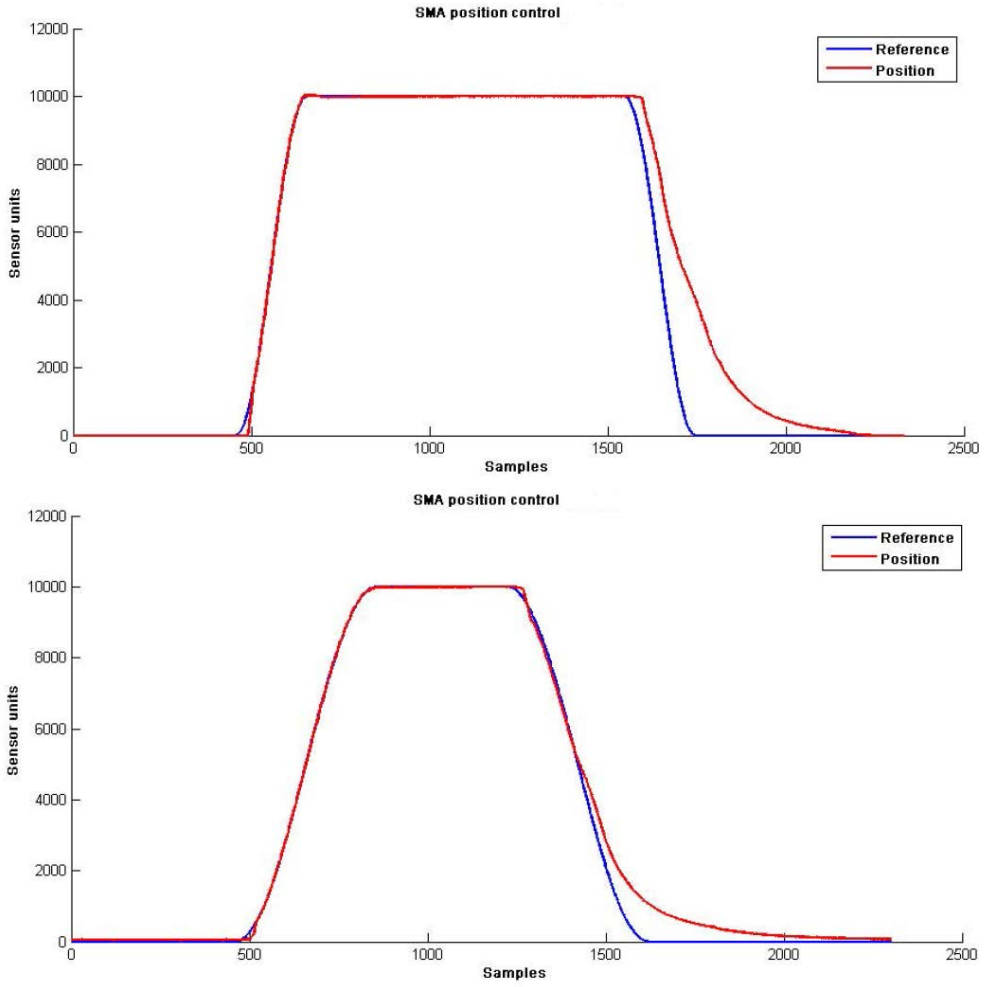


Figura 8.20: *Resultados del control en posición en la mano robótica imprimible*

8.5. Control y estabilización de un misil de prácticas. Ejército de Tierra Polaco

En el transcurso del período de tiempo necesario para la realización de la presente tesis doctoral, comprendido desde finales del año 2011 hasta estas fechas de 2014, se ha tenido la oportunidad de disponer de 'betatesters' de origen extranjero que han detectado errores en el sistema ARCP y a su vez lo han utilizado como sistema de control en los propósitos en que les ha sido necesario. Uno de estos 'betatesters' es un estudiante de ingeniería en armamento y construcción del ejército de tierra polaco.

La utilización del sistema ARCP mediante el MCU STM32F4 le ha permitido dotar a un misil de prácticas de control de los motores (4 motores brushless) de los alerones y estabilización mediante la lectura y tratamiento de los datos provenientes de sensores inerciales. El misil de prácticas no contiene cabezal explosivo, en su lugar al detectarse un fallo grave de funcionamiento o pérdida irrecuperable del control, éste despliega un paracaídas para que pueda ser recuperado con posterioridad. La figura 8.21 muestra el cohete de prácticas controlador por un MCU STM32F4 y la toolbox de funcionalidades avanzadas.El responsable de este sistema de prácticas es Marcin Chodnicki.

Figura 8.21: *Misil de prácticas gobernado por el sistema ARCP con MCU STM32F4. Fotografía cortesía del ejército de tierra polaco y del cabo Marcin Chodnicki*

8.6. Control en posición/velocidad de un robot orientado a uso educativo.

En los laboratorios-talleres del departamento de ingeniería de Sistemas y Automática existía un robot de dos grados de libertad, llamado Robot-Sidemar, controlado por un ordenador PC y una tarjeta de control de ejes PMAC del fabricante Delta Tau®; esta tarjeta controladora resulta muy costosa económicamente, es difícil de programar y entender, además de no disponer de soporte para ser programada mediante el lenguaje gráfico de MATLAB®/SIMULINK®.

El propósito era reemplazar el antiguo sistema de control basado en PMAC y acceder a los motores y sensores directamente, conectándolos al sistema ARCP con MCU STM32F4 propuesto en esta tesis, de forma que se puedan ensayar rápidamente modelos de control sobre el robot empleando MATLAB®/SIMULINK®. Esta integración fué realizada en la tesis de máster de Higor Alves y actualmente han sido integrados encoders absolutos de efecto hall y sensores inerciales al sistema de control compuesto por el ARCP y la toolbox de funcionalidades avanzadas. La figura 8.22 muestra el robot modular en cuestión, con el nuevo sistema de control basado en sistema ARCP con MCU STM32F4, en estadio de pruebas; en la figura se señalan los nuevos sensores incorporados al robot.

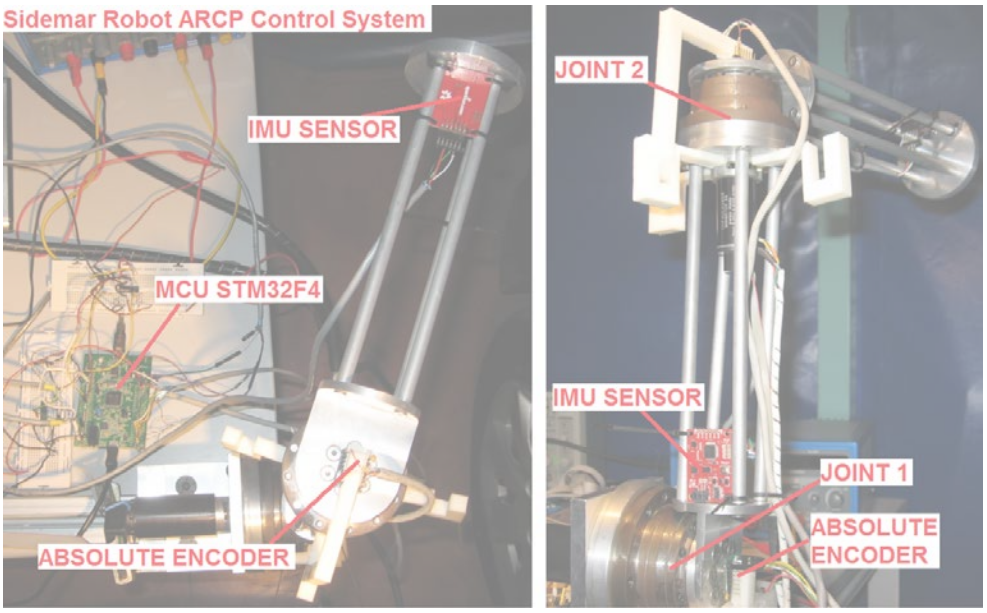


Figura 8.22: *Robot Sidemar con el nuevo sistema de control basado en ARCP*

Los resultados de control en posición del robot Sidemar se muestran en la figura 8.23, donde la señal amarilla es la posición registrada por encoders absolutos y la morada es la referencia en posición. La figura 8.24 **ilustra el proceso de verificación del modelo teórico del robot**, al contrastar el funcionamiento del modelo teórico en SIMMECHANICS® con la respuesta del robot real; esto es posible gracias a que la comunicación con el sistema ARCP no exige el funcionamiento en modo externo (véase apartado 2.5.3), y por tanto cualquier toolbox de MATLAB® puede desempeñar sus funciones junto con la comunicación USB del sistema ARCP.

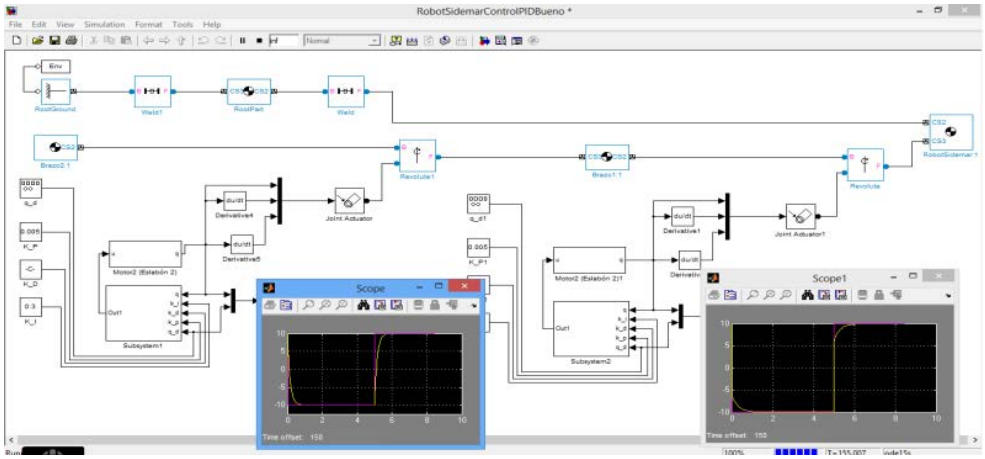


Figura 8.23: *Resultados del control en posición del robot Sidemar controlado por el sistema ARCP*

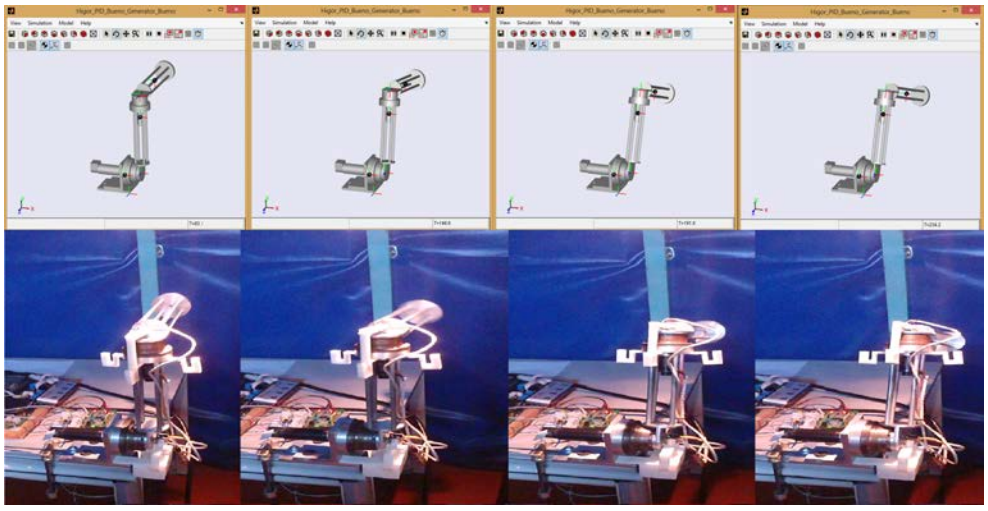


Figura 8.24: *Secuencia de funcionamiento del proceso de verificación del modelo teórico con el modelo real del robot*

8.7. Control y modelado de motores de ultrasonidos.

Este apartado corresponde con los estudios y análisis que están siendo llevados a cabo en la tesis doctoral de Dorin-Sabin Copaci, que se encuentra en progreso a fecha contemporánea a este documento, en el departamento de Ingeniería de Sistemas y Automática de la UC3M.

Los motores de ultrasonidos (USM) no poseen un funcionamiento basado en fundamentos electromagnéticos, sino que son sistemas de actuación rotatoria piezoeléctricos. Se alimentan con una señal ondulatoria de alta frecuencia que ataca un material piezoeléctrico donde se produce la vibración que transmite el movimiento al rotor del motor. Este tipo de motores poseen unas cualidades diferentes a los motores electromagnéticos:

- Permiten transmitir pares elevados a muy bajas revoluciones
- Son silenciosos
- Se pueden construir con dimensiones reducidas
- No se ven afectados por campos magnéticos, ni los generan

El uso de este tipo de motores eléctricos también conlleva desventajas:

- Su principio de funcionamiento basado en la vibración de un material cerámico en contacto constante con el rotor, produce un rozamiento tal que disipa una gran cantidad de energía en forma de calor
- El fuerte rozamiento del punto anterior hace decrecer las expectativas para el ciclo de vida de este tipo de motores
- Presentan problemas para poder mantener una velocidad de giro constante, se producen cambios bruscos de velocidad debido a variaciones en la intensidad del rozamiento entre vibrador y rotor
- Resultan costosos de adquirir debido a la baja demanda

Por tanto, trabajar con este tipo de motores directamente, diseñando algoritmos de control con la intención de paliar los problemas en el control en velocidad, sin contar con un modelo teórico que reproduzca el comportamiento del motor real; conllevará un rápido deterioro del USM. Una vez que el USM se encuentra cerca

del fin de su vida útil, el comportamiento y prestaciones del mismo cambian radicalmente.

Una alternativa, investigada en la tesis sobre control de motores USM, es el de utilizar el sistema ARCP propuesto en la presente tesis para obtener un modelo teórico preciso del motor USM disponible en el laboratorio. Y así trabajar los algoritmos de control sobre el modelo teórico, de forma que se prolongue la vida útil del caro motor USM disponible en el laboratorio.

Ayudándose del sistema ARCP se plantea un modelo de control que permita adquirir paquetes de datos a alta frecuencia (5 KHz), modelo mostrado en la figura 8.25. Este modelo, cargado en el sistema ARCP permite recibir órdenes vía USB y enviar grandes paquetes de datos de vuelta al PC para su análisis posterior con la toolbox de identificación de sistemas de MATLAB®. Las señales de excitación y de salida del motor USM son las mostrada en la figura 8.26.

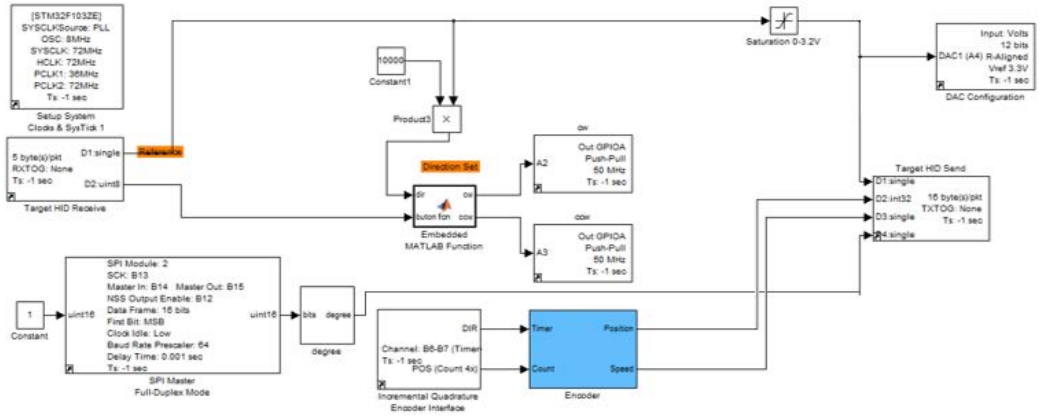


Figura 8.25: *Modelo de control para la obtención de datos de cara a la identificación del motor USM*

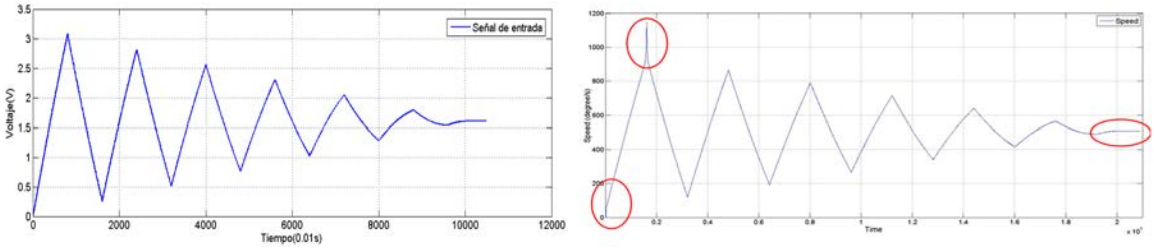


Figura 8.26: *A la derecha: Estimulo de entrada al modelo de identificación. A la izquierda: Respuesta del motor USM real. Se han resaltado las partes que han sido consideras como no lineales*

Los resultados del proceso de identificación del sistema a partir de los datos capturados del motor real con el sistema ARCP se muestran en la figura 8.27. En la figura se observa que el modelo teórico (señal azul) originado a partir de los datos experimentales, difiere respecto al motor real (señal roja) en que no presenta unas variaciones de la velocidad tan notables. Estas variaciones en la velocidad pueden ser modeladas como un ruido cuya distribución de potencia es constante y de magnitud inversamente proporcional a la pendiente de la referencia en velocidad. Añadiendo este término de ruido, el modelo teórico del motor USM podría llegar ser aún más aproximado al comportamiento real.

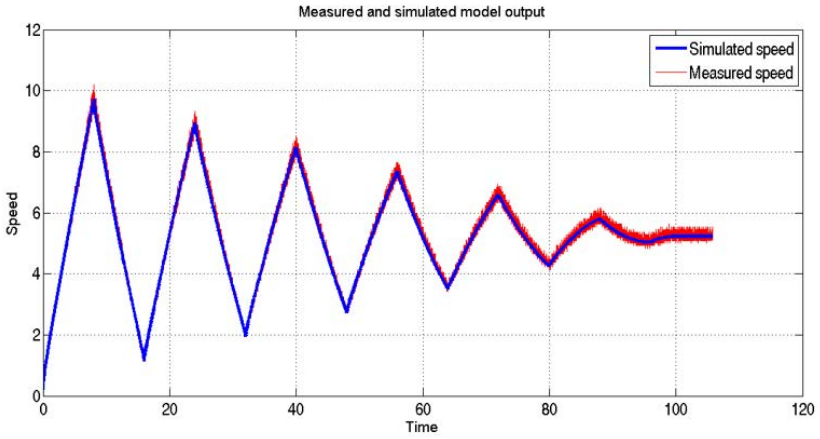


Figura 8.27: *Resultado de la identificación del motor USM a partir de los datos capturados con el sistema ARCP*

8.8. Adquisición y tratamiento de señales EMG.

El sistema ARCP está siendo utilizado en los laboratorios del departamento de Ingeniería de Sistemas y Automática de la UC3M, como medio para la adquisición y tratamiento de señales mioeléctricas (EMG, del inglés 'ElectroMyoGraphic') provenientes de los sistemas de actuación biológicos, músculos, del ser humano.

Se trata de un sistema desarrollado en la UC3M como alternativa a los caros sistemas comerciales disponibles para trabajar con señales EMG. La electrónica responsable de la captación de las señales EMG así como el sistema ARCP al que se conectan han sido ideados por el autor de la presente tesis y por Álvaro Villoslada Peciña, quien también recurre a los recursos proporcionados por el sistema ARCP para poder trabajar con un alto nivel de abstracción en su tesis doctoral. Ahora se está trabajando en los algoritmos de procesamiento de datos y reconocimiento de patrones.

Para llevar a buen término los citados propósitos, se están empleando los MCUs NRF51822 y STM32F4. De forma que la adquisición y tratamiento de estas señales no obligue a extender gruesos e incómodos cableados, desde los electrodos colocados sobre los músculos hasta la unidad de control, sino que los datos viajen de forma inalámbrica y con procesamiento mediante filtrado de señal previos a su entrada al MCU STM32F4. **Se trata de otro ejemplo de sistema de control distribuido, en este caso el sistema ARCP entra de lleno en el campo de la bioingeniería** combinándolo con el campo de la ingeniería industrial, pues el propósito final es mover actuadores en base a los resultados de los análisis de la señal EMG.

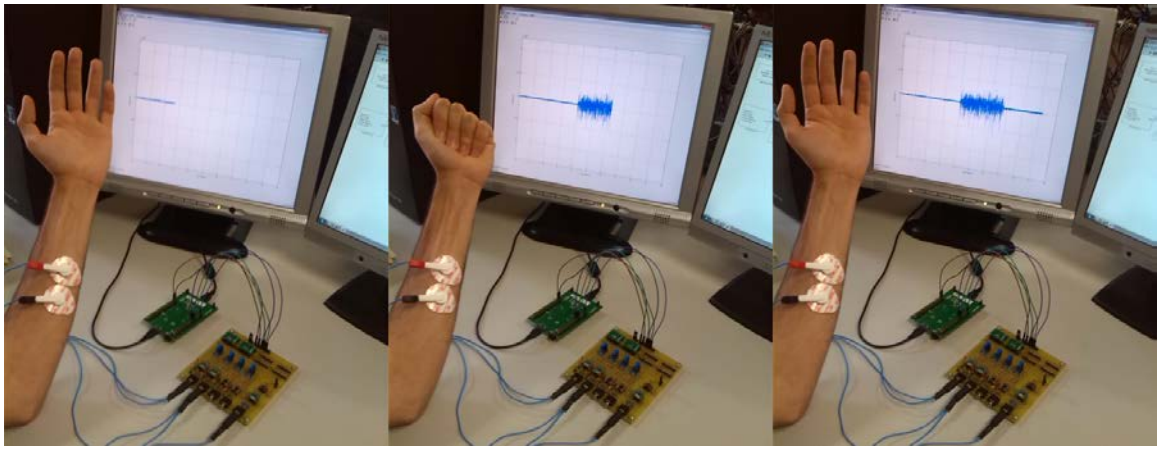


Figura 8.28: *Sistema ARCP adquiriendo y y tratando señales electromiográficas. En la figura se observa una secuencia completa desde estado de reposo, puño cerrado y vuelta al estado de reposo*

8.9. Controlador de un generador de energía eléctrica. Universidad de Almería.

Un estudiante, José Atienza García, de la universidad de Almería, ha completado su trabajo fin de carrera utilizando el sistema ARCP propuesto en esta tesis. Al comienzo de su TFG solicitó permiso (en conversación telefónica mantenida la mañana del Jueves 24 de Julio de 2014) al autor de la presente tesis doctoral para utilizar el conjunto de herramientas software/hardware que componen el sistema ARPC, en concreto para el MCU STM32F4; tras haber accedido a nuestras publicaciones en las que se muestra y explica el sistema ARCP.

El estudiante está ideando un controlador embebido para gestionar la potencia ofrecida por un generador de energía; que posea gran cantidad de interfaces de I/O, analógicas y digitales, de forma que con los recursos hardware de este MCU y el alto nivel de abstracción en la programación del lenguaje gráfico de SIMULINK® simplifiquen la labor de hacer realidad el controlador.

El sistema físico y electrónico de este TFG es el mostrado en la figura 8.29.

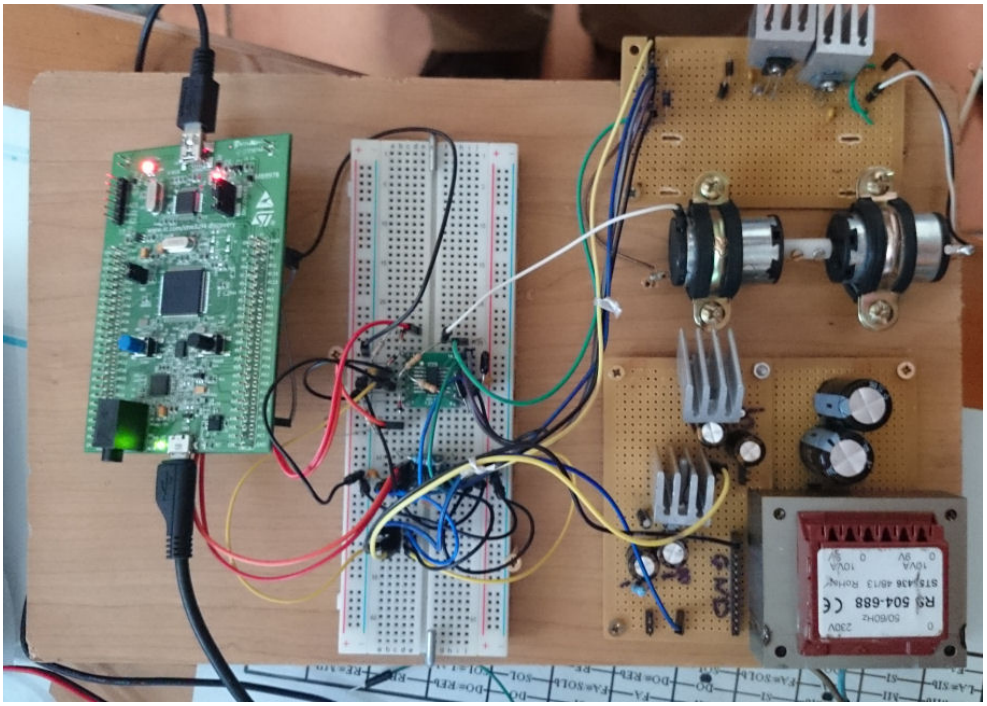


Figura 8.29: *Fotografía cortesía de la Universidad de Almería en la que el sistema ARCP propuesto en esta tesis está siendo utilizando para un TFG*

La siguiente figura 8.30 muestra los modelos de SIMULINK® utilizados por José Atienza García para llevar a cabo su trabajo fin de carrera.

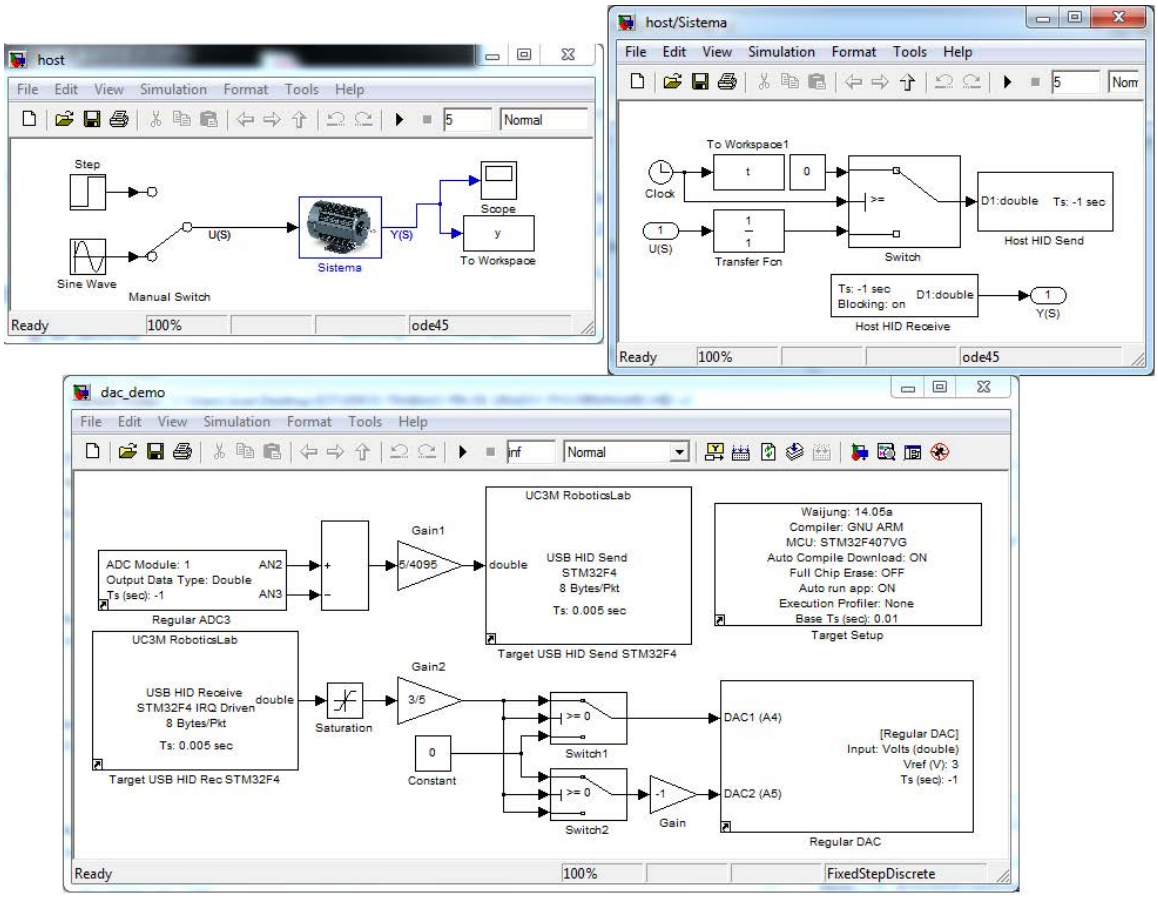


Figura 8.30: *Modelos de SIMULINK utilizados en el trabajo fin de carrera*

Capítulo 9

Conclusiones

Esta tesis presenta un sistema avanzado de prototipado rápido para control (ARCP) cuyo nivel de funcionalidad ha resultado ser superior a los sistemas de prototipado rápido para control (RCP) de cuya existencia se tiene conocimiento. Provee de mayor flexibilidad y posibilidades de programación al usuario haciendo uso del más alto nivel de abstracción, mediante lenguajes de programación basados en gráficos. El sistema se encuentra orientado a fusionar la etapa de experimentación con prototipos en el laboratorio y la etapa de implementación final, al permitir que ambos sistemas sean el mismo, mientras que esté no es el tópico general y clásico que se entiende sobre sistemas RCP.

El sistema ARCP ha sido evaluado en diferentes centros de investigación y en diversos proyectos, pudiéndose determinar la validez y solidez de los nuevos conceptos que se han acuñado.

Las nuevas funcionalidades destinadas a facilitar el diseño de sistemas de control distribuidos, las nuevas capacidades y conceptos sobre multitarea y multimedia; en definitiva todo aquello que ha modificado y ampliado las capacidades inherentes de generación automática de código fuente de MATLAB®/SIMULINK® han hecho que el sistema presentado haya superado la barrera impuesta por el estado del arte en lo que al ámbito del prototipado rápido para control se refiere.

La aceptación y difusión del sistema ARCP ha superado las expectativas iniciales, así como los ámbitos de aplicación cubiertos; actualmente se está utilizando no sólo como herramienta en el laboratorio para facilitar la integración y experimentación con sistemas mecatrónicos, sino que el sistema ARCP se está adoptando para

utilizarlo como elemento de control y análisis en tiempo-real en trabajos de bioingeniería. También se ha adoptado como herramienta útil y versátil en aspectos como la educación en ingeniería, siendo utilizado por estudiantes de doctorado y máster.

9.1. Aportaciones

El sistema ARCP aporta una serie de contribuciones en lo que a tecnologías de control se refiere:

- **El sistema ARCP aporta un mecanismo multitarea en tiempo-real** que respeta el orden de ejecución deseado por el usuario, **cuyo uso de capas de abstracción software es inexistente ya que elimina la necesidad de emplear un sistema operativo en tiempo-real**. **Haciendo el sistema completamente determinista** y más fiable al depender el programa de muchos menos elementos.

- **El sistema ARCP aporta también un nuevo concepto en comunicaciones en tiempo-real basadas en puerto USB®**; en el que **haciendo uso de una fuerte modificación de la clase CDC** (del inglés *Communication Device Class*) **se garantiza el plazo de los tiempos de ejecución para llevar a cabo las transacciones de datos**, habiendo aumentado la longitud de los paquetes de datos hasta los 16 KB en cada transmisión/recepción.

- **Se ha aportado el soporte para ejecución en tiempo-continuo, una modalidad de ejecución en que las tareas iteran constantemente en lugar de hacerlo a intervalos de tiempo periódicos**; añadiendo un nuevo mecanismo para medir tiempos a nivel de microsegundos en el que no se hace necesario recurrir a más recursos hardware de los ya utilizados por el soporte de ejecución en tiempo-real.

- **También se ha aportado el soporte para ejecutar tareas en tiempo-real con intervalos de tiempo variables, de forma que éstas puedan adaptar dinámicamente la carga computacional que suponen** dependiendo de cómo esté evolucionando el sistema que están controlando.

- **Se han asentado las bases para dotar a un sistema de control embebido de capacidades multimedia** (audio y vídeo) **de forma que dichas capacidades no perjudiquen el comportamiento en tiempo-real del sistema**. A diferencia de los dispositivos de audio y vídeo tradicionales que exigen mayor prioridad.

- **Se ha aportado un mecanismo de sincronización de los relojes de tiempo-real con el objetivo de sincronizar múltiples controladores en un sistema de control distribuido**. Esto conlleva la ventaja, como ejemplo, de que cualquier controlador puede conocer el estado de cualquier otro controlador del sistema, pudiendo compartir todos ellos con facilidad el mismo canal de comunicaciones, evitando la problemática de que se interrumpan entre ellos.
- **El sistema ARCP ha aportado al soporte del lenguaje gráfico todas aquellas funcionalidades de las que carecía, necesarias para que un lenguaje basado en gráficos llegue a ser tan versátil como un lenguaje basado en texto**. Este nuevo soporte permite añadir cualquier nueva funcionalidad que pueda llegar a considerarse.

9.2. Cumplimiento de los objetivos

El sistema ARCP propuesto en la tesis ha cumplido los objetivos de afianzar y reforzar las características de predictibilidad y fiabilidad del hardware/software de control al otorgar funcionalidades avanzadas sin hacer uso de capas de abstracción software, tales como sistemas operativos en tiempo-real; propiedades éstas que no habían sido logradas por ningún otro sistema RCP.

Las funcionalidades avanzadas son las que han permitido que el lenguaje gráfico llegue a ser tan versátil como un lenguaje textual, mientras que antes de la existencia de esta tesis se trataba de lenguajes de programación cuyas posibilidades se encontraban limitadas frente a los lenguajes basados en texto.

La sencillez de los nuevos conceptos acuñados en esta tesis respecto a multitarea en tiempo-real, multitarea con paso de ejecución variable, reprogramación de fuentes de interrupción, óptima utilización de las secciones de memoria disponibles, medición de los tiempos de ejecución... ha permitido no sólo aprovechar completamente los recursos disponibles en el MCU STM32F4 inicialmente previsto, sino que además ha sido posible trasladar en corto espacio de tiempo estas funcionalidades a otros dispositivos hardware tales como PlayStation 2® y otros MCU como el NRF51x22 de Nordic®. En cambio, de haber utilizado capas de abstracción software tales como sistemas operativos, el traslado de muchas funcionalidades no habría sido posible, y menos en períodos de tiempo realmente cortos. Este no es un objetivo que estuviera inicialmente previsto como parte de la tesis, pero ha servido para asentar la idea de que estos nuevos conceptos son fácilmente trasladables entre plataformas computacionales.

La nueva técnica multitarea con soporte de tiempo-continuo permite aprovechar al 100 % los recursos computacionales, así como la eliminación de secciones críticas en el soporte multitarea en tiempo-real, secciones en las que no es posible atender a fuentes de interrupción externa, ha otorgado efectos beneficiosos, se desperdicia menos tiempo de ejecución para atender el cambio entre una tarea y otra, y sigue resultando posible adquirir información sin fallos por medio de interrupciones externas y mantener el soporte multitarea en tiempo-real.

El concepto de sincronización de los relojes de tiempo-real entre múltiples controladores, permite compartir información sin provocar colisión de datos en los buses (que varios controladores quieran transmitir información a la vez), situación que permite extender el sistema de control hasta basarlo en el concepto de sistema de control distribuido.

Se han podido explorar las capacidades multimedia (audio y vídeo) de los controladores embebidos actuales y adaptarlas para un mejor uso y adecuación a sistemas de control críticos en seguridad, logrando de esta manera que un controlador embebido posea su propia interfaz gráfica y sonora, y que ésta no interfiera con las tareas de tiempo-real. Sin necesidad de recurrir a conectarlo a un ordenador de propósito general como es un PC para disponer de una interfaz de usuario.

Todas estas propiedades han podido ser preparadas para ser utilizadas desde el entorno basado en modelos de MATLAB®/SIMULINK®, compatibilizándolo con las versiones de 32 y 64 bits del mismo, desde su versión 2011b a 2014b.

Por último, la validez y estabilidad del sistema ARCP ha podido ser verificada por medio de numerosas aplicaciones prácticas, cuyo número continúa en aumento.

Las citadas aplicaciones prácticas ha sido llevadas a cabo por personal con formación universitaria o estudiantes de último curso, y también estudiantes de máster y doctorado. Pero no todos ellos cumplen el perfil de ser especialistas en sistemas hardware embebidos, corresponden con un público usuario de carácter multidisciplinar, en el que encontramos ingenieros mecánicos y bioingenieros entre otros. Por lo que el objetivo de comprobar el uso del sistema ARCP por un público multidisciplinar también ha podido ser cubierto.

9.3. Trabajos futuros

Debido a la rápida difusión y aceptación del sistema ARCP, éste está encontrando cabida en diferentes ámbitos, en los que su presencia no se había contemplado inicialmente; asimismo el conocimiento adquirido sobre cómo funciona el mecanismo de generación automática de código de MATLAB® y cómo modificarlo abre otros frentes objeto de estudio:

- **Educación en ingeniería**: Se encuentra en preparación el disponer del sistema ARCP para impartir prácticas de los estudios reglados de ingeniería industrial, utilizando en ARCP como medio para probar en la realidad la extensa teoría de control sobre maquetas con actuadores electromecánicos y en bioingeniería utilizándose para recuperar y tratar señales electromiográficas.
- **Programación orientada a gráficos mediante lenguaje basado en grafos**: Dado el conocimiento sobre programación orientada a gráficos del autor de esta tesis, y las destrezas adquiridas sobre generación automática de código y lenguajes de programación basados en grafos; el autor plantea el utilizar funcionalidades hardware del adaptador gráfico de un computador PC, mediante las librerías Microsoft DirectX® para dar soporte hardware al dispositivo Kinect® e integrar el mismo para su uso desde SIMULINK®. Haciendo uso de la herramientes hardware/software para mallados poligonales y reducción automática de la carga poligonal de una escena, así como utilización de funcionalidades orientadas a trabajo con vértices mediante el repertorio de funciones de *vextex shader*.
- **Ubicar variables de programa en diferentes secciones de la memoria**: Mejora del soporte de generación automática de código para poder ubicar variables, datos vectoriales y matriciales en diferentes secciones de la memoria RAM de un computador desde SIMULINK®, funcionalidad que aún no ha podido ser cubierta por el autor de esta tesis, y exigirá profundizar en estos aspectos del proceso de generación automático de código fuente.

Apéndice A

Anexo 1

En este anexo se incluye el análisis de más sistemas RCP comerciales. Sírvase este apartado como prolongación del estudio sobre el estado del arte.

Pese a tratarse de sistemas RCP y herramientas CASE que cuentan con una menor representación tanto en ámbitos profesionales empresariales e investigadores, no por ello merecen ser excluidos de un análisis pormenorizado debido a que presentan características interesantes y por ello pudieran llegar a tener una más notable representación en un futuro.

A.0.1. RT-Lab®

RT-Lab® es un complemento software para Matlab/Simulink® que además de permitir la programación mediante lenguaje gráfico de los sistemas RCP hardware extiende y amplia los dominios de simulación de Simulink®, a los ámbitos aeroespaciales y del automóvil, aunque a diferencia de dSPACE®, éste fabricante está más orientado al mundo aeronáutico, con la expansión del software RT-Lab® denominada Dinamo. Estos sistemas software y hardware pertenecen a la compañía OPAL-RT Technologies® fundada en 1997 [40].

Al igual que sucedía en el caso de dSPACE®, se observan dos vertientes en sus sistemas, en un primer lugar los grandes computadores en forma de "mainframes" orientados a simular en Tiempo-Real el modelo del sistema físico a controlar, además de ejecutar junto con la simulación el modelo de control que se haya diseñado en Simulink®. Son las simulaciones HIL. La figura A.1 muestra el aspecto de uno de estos "mainframes".

Figura A.1: *Mainframe OP7000 apto para su uso con la toolbox RT-Lab®*

La otra vertiente se centra en los sistemas RCP con multitud de interfaces I/O donde probar los modelos de control en un hardware real. Para simulaciones PIL.

Los sistemas operativos en Tiempo-Real utilizados por estos sistemas RCP son QNX y Linux Real-Time Red Hat, producto que pertenece a Red Hat Enterprise MGR® [41]; el uso de uno u otro dependerá de la arquitectura hardware del sistema RCP. Respecto a estos sistemas de prototipado rápido para control, esta compañía utiliza un estándar que ya se ha visto con anterioridad, el estándar PC 104, que suele ser el estándar más usual para sistemas RCP con la toolbox xPC Target® (revisado en la sección 2.3.1). El menor lapso de tiempo posible entre iteraciones del

[40] http://www.opal-rt.com/company/company-profile , OPAL-RT Technologies, Enero 2014
[41] http://es.redhat.com/products/mrg/realtime/ , Enero 2014

algoritmo de control en los sistemas RCP de RT-Lab es de 10 μs [42].

También, al igual que sucedía con xPC Target®, la toolbox RT-Lab® es compatible con tarjetas de expansión de I/O de distintos fabricantes, para buses PCI/PCI Express de ordenadores PC.

Los sistemas RCP basados en estándar PC 104 de RT-Lab® son compatibles con Matlab/Simulink® y con MATRIXx/SystemBuild® (un producto de modelado de sistemas similar al modelado en Matlab/Simulink® de National Instruments®). El aspecto físico de los sistemas PC 104 de RT-Lab es el mostrado en la figura A.2.

Figura A.2: *Sistema RCP basado en estándar PC-104 de RT-Lab®*

De forma idéntica a cómo sucedía con el soporte para PC 104 de xPC Target®, las tarjetas de expansión apilables de distintos fabricantes cuentan con soporte en la toolbox RT-Lab®, la única diferencia es que se puede mantener el aspecto externo del chasis sin importar el número de tarjetas de expansión colocadas.

Respecto al uso en investigación de las tecnologías de RT-Lab®, la mayoría se encuentran localizadas en la ejecución conjunta de la simulación de la planta a controlar junto con el algoritmo que las controla en Tiempo-Real (simulaciones HIL), haciendo uso de unos u otros modelos de computadores "mainframes" [76], [77], [78]. Otros usos de este RCP en robots [79], [80], y control sobre plantas experimentales [81]. El carácter de las publicaciones sugiere que el uso para simulación Hardware-in-the-Loop es mucho más predominante que el uso para prototipar rápidamente el control y experimentar sobre la planta real. La capacidad de estos "mainframe" y sus toolboxes han alcanzado un buen nivel de perfección en lo que a simulaciones en Tiempo-Real de sistemas muy complejos se refiere.

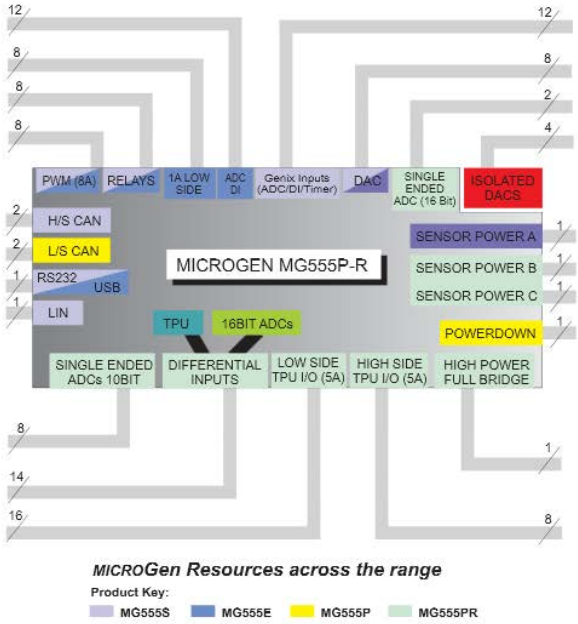
[42] http://www.opal-rt.com/product/rt-lab-professional-real-time-digital-simulation-software , RT-Lab, menor lapso de tiempo entre iteraciones, enero 2014

A.0.2. MicroGen555®

MicroGen555® es un sistema RCP diseñado con la intención de ser utilizado en pruebas de control para automóviles, aunque como el propio fabricante apunta, al disponer de multitud de interfaces de I/O puede utilizarse con fines de prototipado rápido para control más o menos universal. El fabricante es Add2® [43] con sede en Staffordshire en el Reino Unido.

Este RCP está basado en el microcontrolador (MCU) de Motorola/FreeScale® MPC555® [44] contando con 4 MB de RAM y 4 MB de ROM. La frecuencia de trabajo máxima de la CPU en la familia MPC555 es de 132 Mhz.

Sus características de I/O se muestran en la figuraA.3, donde mediante un código de colores se puede identificar la I/O propias de cada modelo de MicroGen555.



The Table below shows the IO and Comms ports available on the MG555

Model Number	Number of IO	Number of Communication Ports
MG555S	24	4
MG555E	57	4
MG555P	57	7
MG555PR	111	7

Figura A.3: *Relación de I/O para el sistema RCP Microgen555® y sus distintos modelos*

El soporte para lenguajes de programación gráficos se basa en Matlab/Simulink®, mediante el conjunto de bloques denominado "MicroGen555 blockset". Provee de funcionalidad para configurar el modelo exacto de MCU que incorpore el RCP MicroGen y de bloques para configurar y utilizar los periféricos de I/O. No dispone

[43] http://www.add2.co.uk/add2/about.html?ID=31 , Enero 2014

[44] http://www.freescale.com/webapp/sps/site/homepage.jsp?code=POWER_ARCH_5XXX , Información del MCU, Enero 2014

de soporte de funcionalidades avanzadas accesibles desde el lenguaje de programación gráfico. A efectos prácticos se consideran bloques de funcionalidad básica. La comunicación con Matlab® y la carga del modelo de Simulink® es a través de la interfaz por bus CAN, por lo que el ancho de banda para el intercambio de datos está limitado a 1 Mbit/s.

El aspecto físico de este sistema RCP es el mostrado en la figura A.4 con un peso de 1 Kg. y unas dimensiones de 240mm X 160mm X 56mm.

Figura A.4: *Aspecto físicio del sistema RCP Microgen555®*

Respecto a su uso en investigación, no se han podido localizar resultados destacables que utilicen este RCP. De todas formas, este sistema RCP se ha mostrado en este capítulo como elemento representativo de una multitud de sistemas RCP de características y formatos similares que no tienen buena acogida en el seno de la investigación, ya sea por disponibilidad, por la calidad del soporte para Matlab/Simulink® y por su relación coste/prestaciones.

A.0.3. HiLink® y Rapcon®

La plataformas HiLink® y Rapcon® ofrecen una interfaz de I/O entre las plantas físicas y Matlab/Simulink® para la implementación de sistemas de control en tiempo real mediante Processor-in-the-loop. Provee de bloques de programación gráfica para Simulink®. La plataforma interfaz de I/O permite plantear sistemas de control en tiempo real utilizando el ordenador PC como controlador. Orientado a su uso en aplicaciones educativas e industriales. El fabricante es Zeltom Educational and Industrial Control Systems®, con sede en los Estados Unidos y Turquía [45].

Debido a que estas plataformas sólo proporcionan capacidades de I/O a un ordenador PC, no pueden ser consideradas como sistemas de prototipado rápido en el sentido correcto del término, pues no se puede programar un controlador embebido en el hardware, tan sólo actúan como sistemas de adquisición de datos (DAQ); aunque no se debe obviar que puede ser de gran utilidad para probar algoritmos de control sobre la planta real.

Las capacidades básicas de I/O son las siguientes: 8 entradas analógicas de 12 bits, 2 salidas analógicas de 12 bits, 8 entradas digitales, 8 salidas digitales, 2 entradas de captura con timers, 2 entradas para encoders relativos, 2 salidas en frecuencia, 2 salidas PWM, 2 puentes H para motores DC. La figura A.5 muestra el aspecto físico de estos dispositivos y sus bloques de programación gráfica para Simulink®.

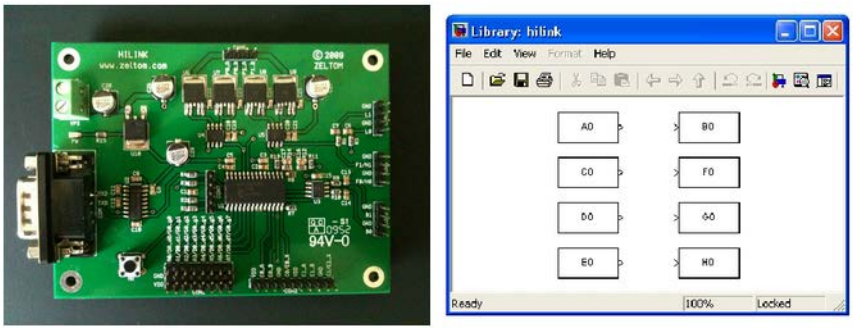


Figura A.5: *A la izquierda, interfaz de I/O Hilink®. A la derecha, sus bloques de programación gráfica para Simulink®*

Las capacidades básicas de I/O pueden extenderse con bloques para Simulink® que proporcionan más funcionalidades utilizando las capacidades disponibles del

[45] http://zeltom.com/aboutus , Enero 2014

hardware, expansiones de software que se venden por separado. El coste para instituciones educativas de una unidad Hilink es de 200 USD y de una unidad Rapcon 400 USD. Estos sistemas DAQ se basan en un microcontrolador.

La interfaz de comunicación de datos entre las I/O y el ordenador PC es un puerto serie RS-232 a 115200 baudios, con un ancho de banda aproximado (en el peor de los casos, 10 bits por cada byte) de 11520 Bytes/segundo o 12800 Bytes/segundo. El muestreo máximo establecido por el fabricante de 3,8 Khz, deja un máximo posible de 3 bytes de datos por paquete, teniendo en cuenta que los paquetes de datos necesitarán, siendo un puerto serie, un encabezado y un fin de mensaje, que también ocupan ancho de banda. Estos datos son para el hardware denominado Hilink®, para el hardware denominado Rapcon®, las capacidades de I/O son las mismas, su ancho de banda para el intercambio de datos con el PC es de 460800 baudios, en el mejor de los casos 51200 Bytes/segundo, que para su máximo muestreo de 15,2 Khz hacen 3 bytes de datos por paquete a esta frecuencia [46].

Este fabricante también provee de una interfaz puente RS-232 a USB, con las siguientes limitaciones, un máximo de 512 Hz en modalidad USB 2.0 y 1024 Hz en modalidad USB 3.0 .

Respecto al uso de estos sistemas en investigación encontramos experimentos de carácter médico, sobre cómo mejorar la introducción de agujas en tejido humano mediante sensores fuerza/par [82] y control de levitación mediante sistemas electromagnéticos [83].

[46] http://zeltom.com/products/hilink , afirmaciones sobre USB en 'Frequently Asked Questions', Enero 2014

A.0.4. MapleSIM®

MapleSim® es una herramienta software para modelado y simulación de sistemas multidominio (elementos mecánicos, eléctricos, sensores, bucles de control...) desarrollada actualmente por la compañía Maplesoft® [47]. El campo de aplicación e incluso la apariencia visual del entorno es muy similar a la que presenta Matlab® y Simulink®, tal y como se muestra en la figura A.6. La representación visual de los elementos mecánicos es igualmente similar a la aportada por SimScape® / SimMechanics®.

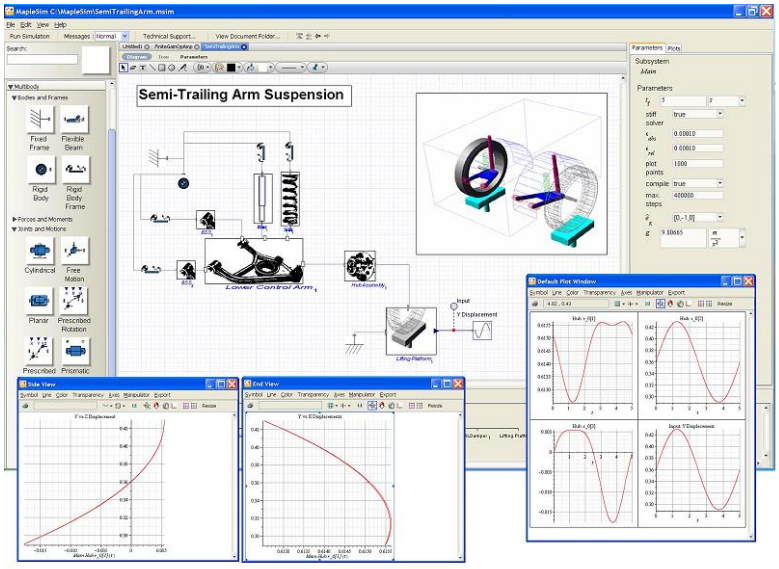


Figura A.6: *Entorno de trabajo en MapleSim®*

Este es un software que tiene su origen en una universidad en 1980, en concreto en el grupo de investigación "Symbolic Computation Group"de la universidad de Waterloo, Ontario, Canadá. Se ha de considerar desde 1988 como un software comercial, pues éste fue el año en que se fundó la compañía MapleSoft® que comercializa este software de modelado y simulación multidominio. En el año 2003 este software recibió su aspecto gráfico e interfaz de usuario actuales, en 2009 tanto el software como los derechos y propiedades intelectuales derivados de él fueron adquiridos por la compañía japonesa Cybernet Systems®.

Volviendo a los asuntos que conciernen a esta tesis doctoral, aquellos relacionados con las posibilidades de integrar la teoría de control sobre modelos de simulación y su volcado en forma de código fuente compilable para controladores

[47] http://www.maplesoft.com/company/about/index.aspx , Febrero 2014

embebidos en tiempo-real, si resulta posible utilizar o no este entorno de trabajo como base software para un sistema RCP. MapleSIM® ofrece de una manera muy semejante a Simulink® la capacidad de utilizar un lenguaje de programación del más alto nivel basado en gráficos, la figura A.7 ilustra un ejemplo de un controlador PID conectado al modelo eléctrico de un sencillo motor de corriente continua.

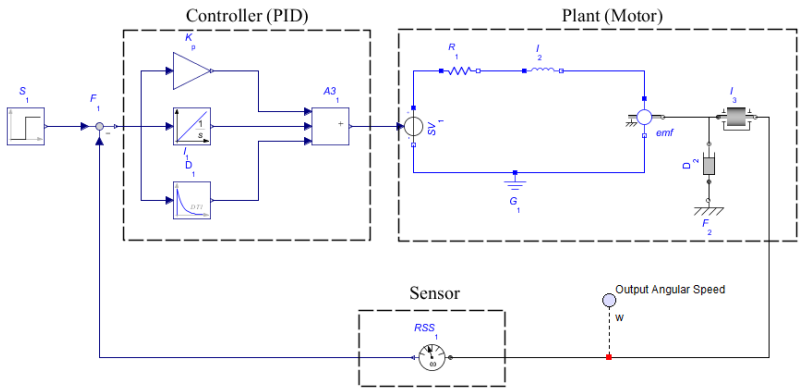


Figura A.7: *Ejemplo de controlador PID en MapleSim®*

El software provee de las funcionalidades para diseñar sistemas de control. Actualmente es posible generar el código fuente que realiza las funciones del modelo de control planteado, aunque de forma básica sin características avanzadas como soporte multitarea en Tiempo-Real con o sin RTOS, ni soporte a interrupciones. El único sistema RCP comercial soportado es la tarjeta de expansión DS1104 de dSPACE®.

Desde MapleSoft® se ofrece poca información sobre la posibilidad de adaptar la generación del código fuente a una variedad de arquitecturas computacionales. Recomiendan exportar los modelos de control a LabView® (CompactRIO®) o Simulink® (XPC Target® , dSPACE®) o bien contactar con el departamento adecuado de MapleSoft® para encargar la realización de una toolbox a medida.

Respecto al uso de este software en investigación, se encuentran trabajos sobre simulaciones tales como [84], la mayoría de ellos publicados por autores de la universidad de Waterloo y/o pertenecientes a Cybernet Systems® como los siguientes [84], [85], [86] y [87].

A.0.5. Wolfram®

Wolfram-Mathematica/System Modeler® es otra herramienta software para modelado y simulación de sistemas multidominio (elementos mecánicos, eléctricos, sensores, bucles de control...) desarrollada actualmente por la compañía Wolfram® [48]. Fundada por Stephen Wolfram, doctor en física teórica, a partir de la década de 1970 comienza a desarrollar lenguajes de procesamiento para álgebra, que culminarían en 1986 con la fundación de la compañía Wolfram® y en 1988 con la comercialización de Wolfram Mathematica®. Wolfram System Modeler®, la herramienta software de alto nivel de abstracción de esta compañía, permite realizar diagramas de control y simulaciones de sistemas multidominio, mediante el uso de lenguajes basados en gráficos de una forma muy similar a Simulink® o MapleSIM®. La figura A.8 ilustra el aspecto visual de este entorno de trabajo. System Modeler fué lanzado por primera vez en el año 2011, la versión actual corresponde a Mayo de 2012. Esta herramienta de programación basada en gráficos es el objeto de estudio en este apartado, como posible base software para sistemas RCP.

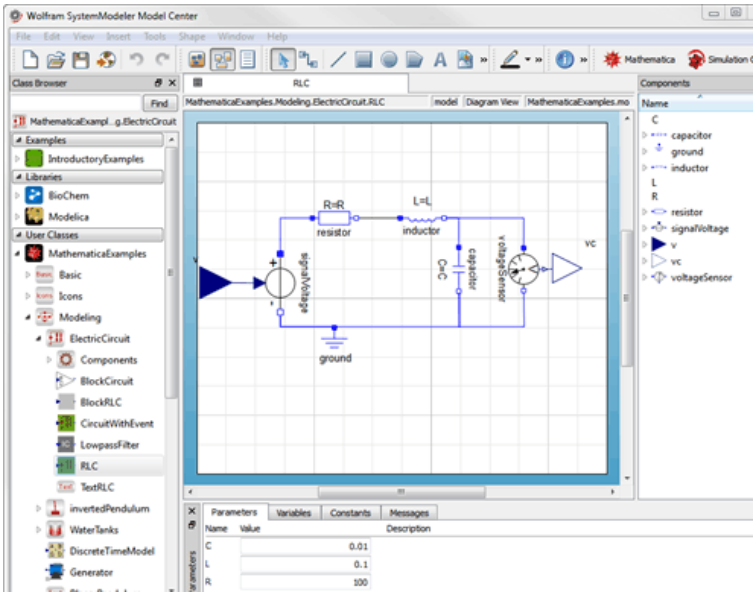


Figura A.8: *Entorno de trabajo en Wolfram System Modeler®*

El soporte para el lenguaje de programación basado en gráficos lo provee la librería gráfica de Modelica®. Esta es la explicación por la que tanto los bloques de

[48] http://www.wolfram.com/company/background.html, Febrero 2014

programación gráfica de MapleSIM® y de Wolfram System Modeler® sean idénticos, tanto en su apariencia como en sus opciones de configuración, a fin de verificar la veracidad de la anterior afirmación la figura A.9, correspondiente a Wolfram System Modeler®, puede compararse con la figura A.7 que corresponde a MapleSIM® y observar la exacta similitud. La versión actual de la librería de programación gráfica utilizada tanto por Wolfram System Modeler® como por MapleSIM® es Modelica Standard Library version $3{,}2{,}1$ [49].

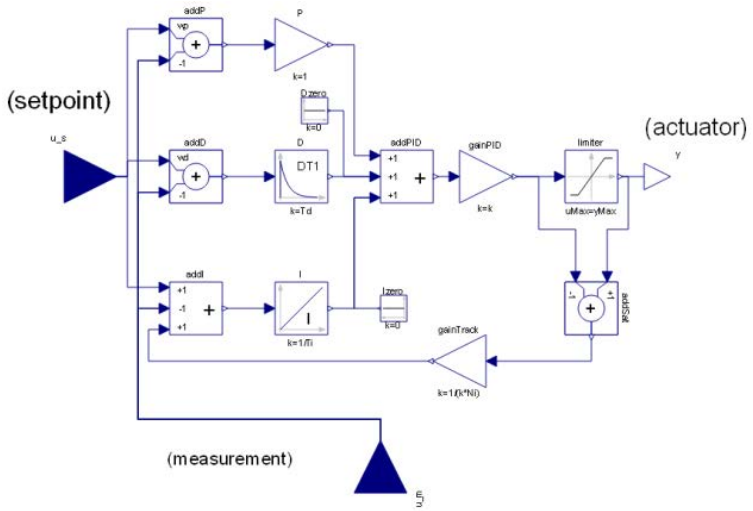


Figura A.9: *Ejemplo de controlador PID en Wolfram System Modeler®*

Las capacidades de generación de código fuente en lenguaje C son incipientes y limitadas, se puede generar código en forma de funciones, pero no de programas completamente independientes como es requisito necesario para un sistema RCP. Es posible diseñar un algoritmo de control en Wolfram System Modeler® y generar el código en lenguaje C de una función que realice las tareas y cálculos del algoritmo, recibiendo las entradas al algoritmo como argumentos de la función y devolviendo un puntero a una estructura con los resultados. No se ha encontrado constancia de sistemas RCP amparados por Wolfram®. Los desarrollos de Wolfram® en el campo de la computación embebida están siguiendo una trayectoria diferente, como es adaptar el software Wolfram® completo para ser ejecutado en computadores de tarjeta única como Raspberry PI añadiendo funciones en lenguaje y sintaxis propios de Wolfram® para manejar las interfaces de I/O [50]. No es

[49] $https://www.modelica.org/news_items/modelica-standard-library-3{,}2{,}1-released$, Febrero 2014

[50] $http://blog.stephenwolfram.com/2013/11/putting-the-wolfram-language-and-mathematica-on-every-raspberry-pi$, Febrero 2014

ejecución de código máquina nativo, aunque no deja de ser útil, no puede utilizarse bajo el paradigma de ejecución en Tiempo-Real.

Apéndice B

Anexo 2

En este anexo se incluyen los pasos necesarios para la instalación de la herramienta software del sistema ARCP.

B.1. Instalando la toolbox para SIMULINK® del sistema ARCP

La instalación manual de la herramienta software para MATLAB®/SIMULINK® del sistema ARCP propuesto en esta tesis comprende 14 pasos. Se trata, obviamente, de un proceso largo e incómodo en el que las posibilidad de caer en un error es muy alta.

Con el objetivo de facilitar la instalación de la herramienta software de cara al usuario final, se ha llevado a cabo un proceso de instalación automatizado; proceso que detecta automáticamente la versión del sistema operativo Windows® instalada, incluyendo la detección de si se trata de un sistema de 32 bits o 64 bits.

También se detecta automáticamente la versión de MATLAB® instalada en el sistema y a continuación precede con el copiado y registro de los componentes software necesarios para el sistema computacional PC en cuestión.

La figura B.1 muestra la ventana de presentación del instalador automatizado que permite obviar al usuario los 14 pasos requeridos por el proceso de instalación manual.

Como requisitos previos se exige tener instalado en el sistema alguna versión de MATLAB® desde la versión 2011b a la 2014b, cualquiera de las citadas inclusive.

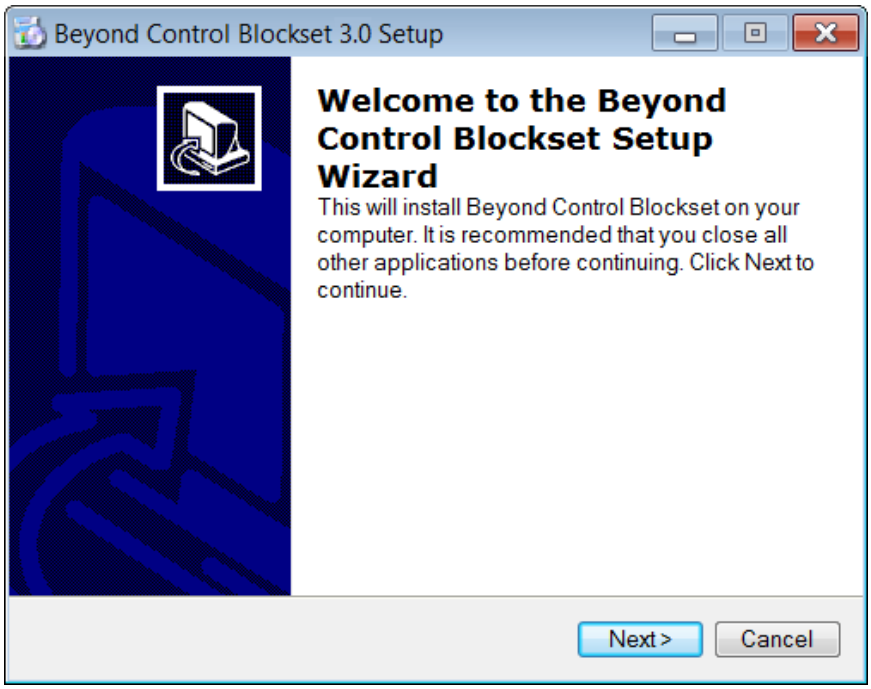


Figura B.1: *Ventana inicial del instalador de la toolbox para SIMULINK® Beyond Control Blockset desarrollada en esta tesis*

Una vez instalada, la toolbox es accesible desde el visor de librerías de SIMULINK®. Como punto de partida recomendado, se puede proceder a desplegar alguno de los ejemplos incluidos con la toolbox, librería de ejemplos mostrada en la figura B.2, que pueden ser abiertos desde el visor de librerías de SIMULINK®. Estos ejemplos consisten en modelos que ayudan a aprender a utilizar las distintas funcionalidades y capacidades de I/O del sistema ARCP, para cada arquitectura soportada.

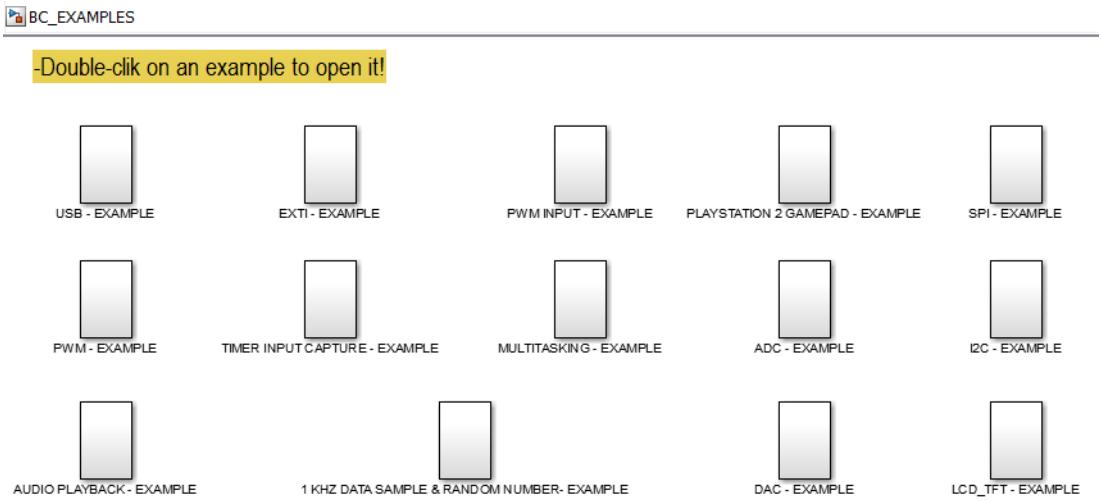


Figura B.2: *Modelos de ejemplo incluídos con la toolbox Beyond Control Blockset desarrollada en esta tesis*

Bibliografía

[1] G. Chindris and M. Muresan, "Deploying simulink models into system on-chip structures," in *ISSE 2006*.

[2] O. Pinzon-Ardila, L. Angel, and M. Useche, "xpc target an option for position control of robotic manipulators," in *LARC 2011*.

[3] X. Chen, X. Gong, H. Zhou, Z. Xu, Y. Xu, and C. Kang, "An economical rapid control prototyping system design with matlab/simulink and tms320f2812 dsp," in *IMECS 2010*.

[4] P. Prince, "Integration according to iec 61508," in *Programmable Electronics and Safety Systems: Issues, Standards and Practical Aspects (Ref. No. 2002/067), IEE Seminar on*, pp. 3/1–3/15, March 2002.

[5] W. Halang, "Automated control systems for the safety integrity levels 3 and 4," in *Object-Oriented Real-Time Dependable Systems, 2003. WORDS 2003 Fall. Proceedings. Ninth IEEE International Workshop on*, pp. 35–42, Oct 2003.

[6] W. S. Black, "Iec 61508 - what it doesn't tell you," *Computing Control Engineering Journal*, vol. 11, pp. 24–27, Feb 2000.

[7] M. Sveda, V. Hubik, V. Oplustil, P. Axman, and T. Kerlin, "Model based development of cots complex power-plant control system," in *Industrial Electronics, 2009. IECON '09. 35th Annual Conference of IEEE*, pp. 1621–1626, Nov 2009.

[8] J. Pons, *Wearable Robots: Biomechatronic Exoskeletons*. Wiley Online Library, 2008.

[9] A. I. Martín Clemente, "Modelado y control de sistemas no lineales de tipo sma," 2014.

[10] N. Halbwachs, *Synchronous programming of reactive systems*. Kluwer Academic Publishers, http://www-verimag.imag.fr/ halbwach/newbook.pdf, Última consulta Julio 2014.

[11] P. Raymond, *http://www-verimag.imag.fr/PEOPLE/Pascal.Raymond/edu/mosig/index.html*. Presentaciones sobre el lenguaje Lustre, Última consulta Julio 2014.

[12] P. Le Guernic, A. Benveniste, P. Bournai, and T. Gautier, "Signal–a data flow-oriented language for signal processing," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 34, pp. 362–374, Apr 1986.

[13] *http://www.netlib.org/linpack/*. Librería escrita en Fortran, Última consulta Septiembre 2014.

[14] *http://www.lcrc.anl.gov/installed-software/MATLAB*. Integración actual de la librería EISPACK, desarrollada en el centro de investigación estadounidense Argonne National Laboratory, Última consulta Septiembre 2014.

[15] *http://www.mathworks.com/*. MathWorks, Toolbox sistema RCP XPC Target, Última consulta Septiembre 2014.

[16] S. Chuan-xue, L. Jian-hua, J. Li-qiang, and W. Ji, "Development of can card driver module using s-function in xpc target," in *Intelligent Systems and Applications (ISA), 2010 2nd International Workshop on*, pp. 1–4, May 2010.

[17] P. Shi, L. Miao, G. Zou, and X. Jiao, "Development of uniform hardware driver for real-time windows and xpc target," in *Information and Computing Science, 2009. ICIC '09. Second International Conference on*, vol. 1, pp. 377–380, May 2009.

[18] T. Sugar, J. He, E. Koeneman, J. Koeneman, R. Herman, H. Huang, R. Schultz, D. Herring, J. Wanberg, S. Balasubramanian, P. Swenson, and J. Ward, "Design and control of rupert: A device for robotic upper extremity repetitive therapy," *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, vol. 15, pp. 336–346, Sept 2007.

[19] H. Zhang, S. Balasubramanian, R. Wei, H. Austin, S. Buchanan, R. Herman, and J. He, "Rupert closed loop control design," in *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*, pp. 3686–3689, Aug 2010.

[20] H. Zhang, H. Austin, S. Buchanan, R. Herman, J. Koeneman, and J. He, "Feasibility study of robot-assisted stroke rehabilitation at home using rupert," in *Complex Medical Engineering (CME), 2011 IEEE/ICME International Conference on*, pp. 604–609, May 2011.

[21] S. Godfrey, C. Schabowsky, R. Holley, and P. Lum, "Hand function recovery in chronic stroke with hexorr robotic training: A case series," in *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*, pp. 4485–4488, Aug 2010.

[22] S. Godfrey, R. Holley, and P. Lum, "Comparison of tone compensation and spring assistance for hand rehabilitation in hexorr," in *Engineering in Medicine and Biology Society,EMBC, 2011 Annual International Conference of the IEEE*, pp. 8535–8538, Aug 2011.

[23] O. Pinzon-Ardila, L. Angel, and M. Useche, "xpc target an option for position control of robotic manipulators," in *Robotics Symposium, 2011 IEEE IX Latin American and IEEE Colombian Conference on Automatic Control and Industry Applications (LARC)*, pp. 1–6, Oct 2011.

[24] K. H. Low, H. Wang, and M. Wang, "On the development of a real time control system by using xpc target: solution to robotic system control," in *Automation Science and Engineering, 2005. IEEE International Conference on*, pp. 345–350, Aug 2005.

[25] X. Wei, C. Jihong, and Y. Jin, "Design of servo system for 3-axis cnc drilling machine based on xpc target," in *Computing, Communication, Control, and Management, 2009. CCCM 2009. ISECS International Colloquium on*, vol. 1, pp. 447–450, Aug 2009.

[26] N.-V. Truong, "Hardware-in-the-loop approach to controller design and testing of motion control systems using xpc target," in *Intelligent and Advanced Systems (ICIAS), 2012 4th International Conference on*, vol. 1, pp. 117–121, June 2012.

[27] N.-V. Truong and D.-L. Vu, "Hardware-in-the-loop approach to the development and validation of precision induction motor servo drive using xpc target," in *Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*, pp. 159–163, May 2012.

[28] C. hoon Park, S.-K. Choi, Y. S. Son, and Y.-H. Han, "Development of 5kwh flywheel energy storage system using matlab/xpc target," in *Computer Science and Information Engineering, 2009 WRI World Congress on*, vol. 2, pp. 701–705, March 2009.

[29] M. Kanthi, I. S. V. Karteek, H. Mruthyunjaya, and V. I. George, "Real-time control of active ankle foot orthosis using labview and compact-rio," in *Biomedical Engineering (ICoBE), 2012 International Conference on*, pp. 296–299, Feb 2012.

[30] M. Rosol, A. Pilat, and A. Turnau, "Real-time controller design based on ni compact-rio," in *Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference on*, pp. 825–830, Oct 2010.

[31] S. Gadzhanov, A. Nafalski, and Z. Nedic, "An application of ni softmotion rt system in a motion control workbench," in *Remote Engineering and Virtual Instrumentation (REV), 2013 10th International Conference on*, pp. 1–9, Feb 2013.

[32] B. Al-Naami, J. Chebil, B. Trabsheh, and H. Mgdob, "Developing custom signal processing algorithm with labview fpga and compact rio to detect the aortic stenosis disease," in *Computers in Cardiology, 2006*, pp. 193–196, Sept 2006.

[33] P. Bilski, W. Winiecki, and T. Adamski, "Implementation of symmetric cryptography in embedded systems for secure measurement systems," in *Instrumentation and Measurement Technology Conference (I2MTC), 2011 IEEE*, pp. 1–6, May 2011.

[34] D. Besiris, V. Tsagaris, N. Fragoulis, and C. Theoharatos, "An fpga-based hardware implementation of configurable pixel-level color image fusion," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 50, pp. 362–373, Feb 2012.

[35] Z. Ghani, M. Hannan, and A. Mohamed, "Development of three-phase photovoltaic inverter using dspace ds1104 board," in *Research and Development (SCOReD), 2009 IEEE Student Conference on*, pp. 242–245, Nov 2009.

[36] Z. Ghani, M. Hannan, and A. Mohamed, "Renewable energy inverter development using dspace ds1104 controller board," in *Power and Energy (PECon), 2010 IEEE International Conference on*, pp. 69–73, Nov 2010.

[37] L. Qiao and W. Jie, "Implementation of a new nonlinear controller for dc-dc converter using matlab and dspace dsp," in *Industrial Electronics, 2005. ISIE 2005. Proceedings of the IEEE International Symposium on*, vol. 2, pp. 621–625 vol. 2, June 2005.

[38] I. Sefa, N. Altin, S. Ozdemir, and M. Demirtas, "dspace based control of voltage source utility interactive inverter," in *Power Electronics, Electrical Drives, Automation and Motion, 2008. SPEEDAM 2008. International Symposium on*, pp. 662–666, June 2008.

[39] Y. Luo, H. Li, and M. Shen, "Speed control of bldcm for industrial sewing machine based on dspace," in *Mechatronics and Automation, Proceedings of the 2006 IEEE International Conference on*, pp. 2127–2132, June 2006.

[40] M. Hu, J. Qiu, and C. Shi, "A comparative analysis of fuzzy pi and pi speed control in brushless dc motor based on dspace," in *Electrical Machines and Systems (ICEMS), 2011 International Conference on*, pp. 1–5, Aug 2011.

[41] S. Amamra, L. Barazane, M. Boucherit, and A. Cherifi, "Inverse fuzzy model control for a speed control induction motor based dspace implementation," in *Modern Electric Power Systems (MEPS), 2010 Proceedings of the International Symposium*, pp. 1–5, Sept 2010.

[42] S. Mat Isa, Z. Ibrahim, J. Lazi, M. Talib, N. Yaakop, and A. Abu Hasim, "dspace dsp based implementation of simplified fuzzy logic speed controller for vector controlled pmsm drives," in *Power and Energy (PECon), 2012 IEEE International Conference on*, pp. 898–903, Dec 2012.

[43] T. He and L. Peng, "Application of neuron adaptive pid on dspace in double loop dc motor control system," in *Computing, Control and Industrial Engineering (CCIE), 2010 International Conference on*, vol. 2, pp. 257–260, June 2010.

[44] C. Lapusan, V. Maties, R. Balan, O. Hancu, S. Stan, and R. Lates, "Rapid control prototyping using matlab and dspace. application for a planar parallel robot," in *Automation, Quality and Testing, Robotics, 2008. AQTR 2008. IEEE International Conference on*, vol. 2, pp. 361–364, May 2008.

[45] C. Rad, M. Manic, R. Balan, and S. Stan, "Real time evaluation of inverse kinematics for a 3-rps medical parallel robot usind dspace platform," in *Human System Interactions (HSI), 2010 3rd Conference on*, pp. 48–53, May 2010.

[46] K. Patil, T. Maxwell, S. Bayne, and R. Gale, "Vehicle development process for ecocar: The next challenge competition," in *Vehicle Power and Propulsion Conference (VPPC), 2010 IEEE*, pp. 1–6, Sept 2010.

[47] D. Di Domenico, G. Fiengo, and L. Glielmo, "On-board prototype of a vehicle localization system," in *Control Applications, 2007. CCA 2007. IEEE International Conference on*, pp. 438–443, Oct 2007.

[48] L. Guvenc, I. Uygan, K. Kahraman, R. Karaahmetoglu, I. Altay, M. Senturk, M. Emirler, A. Karci, B. Guvenc, E. Altug, M. Turan, O. Tas, E. Bozkurt, U. Ozguner, K. Redmill, A. Kurt, and B. Efendioglu, "Cooperative adaptive cruise control implementation of team mekar at the grand cooperative driving challenge," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 13, pp. 1062–1074, Sept 2012.

[49] S. Stanescu and P. Cristea, "Simple low level laser therapy device approach," in *Systems, Signals and Image Processing (IWSSIP), 2012 19th International Conference on*, pp. 44–47, April 2012.

[50] E. Enikov and G. Campa, "Mechatronic aeropendulum: Demonstration of linear and nonlinear feedback control principles with matlab/simulink real-time windows target," *Education, IEEE Transactions on*, vol. 55, pp. 538–545, Nov 2012.

[51] E. Enikov, V. Polyzoev, and J. Gill, "Low-cost take-home experiment on classical control using matlab/simulink real-time windows target," *American Society for Engineering Education*, vol. 55, pp. 538–545, Nov 2010.

[52] W. Dixon, D. Dawson, B. T. Costic, and M. De Queiroz, "A matlab-based control systems laboratory experience for undergraduate students: toward standardization and shared resources," *Education, IEEE Transactions on*, vol. 45, pp. 218–226, Aug 2002.

[53] W. Dixon, D. Moses, I. Walker, and D. Dawson, "A simulink-based robotic toolkit for simulation and control of the puma 560 robot manipulator," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 4, pp. 2202–2207 vol.4, 2001.

[54] H. Marchand, P. Bournai, M. Leborgne, and P. Le Guernic, "A design environment for discrete-event controllers based on the signal language," in *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*, vol. 1, pp. 734–739 vol.1, Oct 1998.

[55] A. Benveniste, P. Bournai, T. Gautier, M. Le Borgne, P. Le Guernic, and H. Marchand, "The signal declarative synchronous language: controller synthesis and systems/architecture design," in *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*, vol. 4, pp. 3284–3289 vol.4, 2001.

[56] A. Gamatie and T. Gautier, "Synchronous modeling of avionics applications using the signal language," in *Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE*, pp. 144–151, May 2003.

[57] *Yuri Zharikov Sánchez; contacto: yzhariko@ing.uc3m.es*. Ingeniero Especialista en sistemas basados en FPGA, Depatarmento de Tecnología Electrónica Universidad Carlos III de Madrid, Última entrevista Octubre 2014.

[58] A. Flores-Caballero, D. Copaci, M. D. Blanco, L. Moreno, J. Herrán, I. Fernández, E. Ochoteco, G. Cabañero, and H. Grande, "Innovative pressure sensor platform and its integration with an end-user application," *Sensors*, vol. 14, no. 6, pp. 10273–10291, 2014.

[59] *https://www.segger.com/emwin.html*. Librería Gráfica para sistemas embebidos de la compañía Segger, Última consulta Septiembre 2014.

[60] *https://www.segger.com/emwin-multiple-buffering.html*. Explicación del efecto gráfico 'tearing' y los conceptos de doble y triple búfer, Última consulta Septiembre 2014.

[61] *Polyspace User Guide*. The Mathworks, available at Matlab DVD or ISO image at doc folder, 2014.

[62] C. Hote, "Extension of static verification techniques by semantic analysis," in *Digital Avionics Systems Conference, 2005. DASC 2005. The 24th*, vol. 2, pp. 9 pp. Vol. 2–, Oct 2005.

[63] Z. Ding, H. Wang, and L. Ling, "Practical strategies to improve test efficiency," *Tsinghua Science and Technology*, vol. 12, pp. 250–254, July 2007.

[64] H. Wang, Z. Ding, and Y. Zhong, "Static analysis test platform construction for embedded systems," in *Audio, Language and Image Processing, 2008. ICALIP 2008. International Conference on*, pp. 808–812, July 2008.

[65] E. Denney, B. Fischer, and J. Schumann, "Adding assurance to automatically generated code," in *High Assurance Systems Engineering, 2004. Proceedings. Eighth IEEE International Symposium on*, pp. 297–299, March 2004.

[66] *TÜV Certification company, IEC 61508, IEC 26262, IEC 60880 and more*. TÜV Group, 2014.

[67] *DEKRA Certification company, IEC 61508, IEC 26262, IEC 60880 and more*. DEKRA Group, 2014.

[68] C. Hote, "Next generation testing technique for embedded software: abstract semantics analysis," in *Digital Avionics Systems Conference, 2004. DASC 04. The 23rd*, vol. 2, pp. 10.A.6–10.1–8 Vol.2, Oct 2004.

[69] M. Lotan, S. Yalon-Chamovitz, and P. Weiss, "Lessons learned towards a best practices model of virtual reality intervention for individuals with intellectual and developmental disability," in *Virtual Rehabilitation International Conference, 2009*, pp. 70–77, June 2009.

[70] Y. K. Ryu and C. Oh, "Rf signal synchronized low cost motion capture device," in *Microwave Conference, 2007. APMC 2007. Asia-Pacific*, pp. 1–4, Dec 2007.

[71] L. Howes, P. Price, O. Mencer, and O. Beckmann, "Fpgas, gpus and the ps2 - a single programming methodology," in *Field-Programmable Custom Computing Machines, 2006. FCCM '06. 14th Annual IEEE Symposium on*, pp. 313–314, April 2006.

[72] L. Howes, P. Price, O. Mencer, O. Beckmann, and O. Pell, "Comparing fpgas to graphics accelerators and the playstation 2 using a unified source description," in *Field Programmable Logic and Applications, 2006. FPL '06. International Conference on*, pp. 1–6, Aug 2006.

[73] T. D'Alessio, "Measurement errors in the scanning of piezoresistive sensors arrays," *Sensors and Actuators A: Physical*, vol. 72, no. 1, pp. 71 – 76, 1999.

[74] A. Iwashita and M. Shimojo, "Development of a mixed signal lsi for tactile data processing," in *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, vol. 5, pp. 4408–4413 vol.5, Oct 2004.

[75] A. Villoslada, A. Flores-Caballero, D. Copaci, D. Blanco, and L. Moreno, "High-displacement flexible shape memory alloy actuator for soft wearable robots," *Robotics and Autonomous Systems*, no. 0, pp.–, 2014.

[76] J. Makwana, A. Mishra, P. Agarwal, and S. Srivastava, "Sensorless control of switched reluctance motor drive: An analytical method," in *Advances in Engineering, Science and Management (ICAESM), 2012 International Conference on*, pp. 571–576, March 2012.

[77] C.-A. Rabbath, H. Desira, and K. Butts, "Effective modeling and simulation of internal combustion engine control systems," in *American Control Conference, 2001. Proceedings of the 2001*, vol. 2, pp. 1321–1326 vol.2, 2001.

[78] R. Maharjan and S. Kamalasadan, "Real-time simulation for active and reactive power control of doubly fed induction generator," in *North American Power Symposium (NAPS), 2013*, pp. 1–6, Sept 2013.

[79] H. ok Lim, Y. Suhara, and A. Takanishi, "Development of a biped locomotor applicable to medical and welfare fields," in *Advanced Intelligent Mechatronics, 2003. AIM 2003. Proceedings. 2003 IEEE/ASME International Conference on*, vol. 2, pp. 950–955 vol.2, July 2003.

[80] Y. Sugahara, T. Endo, H. ok Lim, and A. Takanishi, "Control and experiments of a multi-purpose bipedal locomotor with parallel mechanism," in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, vol. 3, pp. 4342–4347 vol.3, Sept 2003.

[81] M. Addel-Geliel, "Real-time implementation of constrained control system on experimental hybrid plant using rt-lab," in *Control and Automation, 2008 16th Mediterranean Conference on*, pp. 1060–1065, June 2008.

[82] T. Lehmann, M. Tavakoli, N. Usmani, and R. Sloboda, "Force-sensor-based estimation of needle tip deflection in brachytherapy," *Journal of Sensors*, vol. 1, no. 3, p. 10, 2013.

[83] D. A. Meshcheryakov, "Controller design for electromagnetic levitation system via time-scale separation technique," *14th International Student Olympiad on Automatic Control (Baltic Olympiad)*, pp. 86–88, 2011.

[84] N. Gachadoit and R. Renaud, "Modeling and design of an active suspension system with maple and maplesim," in *Mechatronics (MECATRONICS) , 2012 9th France-Japan 7th Europe-Asia Congress on and Research and Education in Mechatronics (REM), 2012 13th Int'l Workshop on*, pp. 425–432, Nov 2012.

[85] T.-S. Dao and J. Friebe, "Symbolic techniques for model-based design of hybrid electric vehicles," in *Electric Drives Production Conference (EDPC), 2012 2nd International*, pp. 1–10, Oct 2012.

[86] L. He, "Tip-over avoidance algorithm for modular mobile manipulator," in *Innovative Engineering Systems (ICIES), 2012 First International Conference on*, pp. 115–120, Dec 2012.

[87] T. Iwagaya and T. Yamaguchi, "Speed improvements for xil simulation based on symbolic-algebraic method," in *SICE Annual Conference (SICE), 2013 Proceedings of*, pp. 1338–1343, Sept 2013.