CÁTEDRA DE PROYECTO FINAL / INGENIERÍA EN ELECTRÓNICA

DISEÑO, IMPLEMENTACION Y ANALISIS DE SISTEMAS DE CONTROL SOBRE FPGA

APLICADO A ROBOTICA

PROYECTO

Versión 1.8
02/10/2022

AUTOR
Gonzalo Miguel Berardo
Legajo 26.499
gonzalo_b@msn.com

TUTOR
Ing. Elbo Hugo Morales
hugom@frm.utn.edu.ar

DIRECTOR
Ing. Jorge Alberto Abraham
jabraham@frm.utn.edu.ar

RESPONSABLE DE CÁTEDRA
Esp. Ing. Antonio Álvarez Abril
antonioalvarezabril@yahoo.com.ar

LABORATORIOS DE INVESTIGACIÓN
Laboratorio de Robot Móvil y Manipuladores Autónomos (LARMMA)
Ing. Elbo Hugo Morales
hugom@frm.utn.edu.ar

Laboratorio de Computación Reconfigurable (LCR)
Dr. Ing. Rodrigo Gonzalez
hrodralez@frm.utn.edu.ar

AÑO ACADÉMICO
2022

HISTORIAL DE VERSIONES

Version #	Implementado por	Fecha de Revisión	Aprobado por	Fecha de Aprobación	Razón
1.1	Gonzalo Berardo	16/06/21			
1.2	Gonzalo Berardo	30/06/21			
1.3	Gonzalo Berardo	07/07/21			
1.4	Gonzalo Berardo	02/08/21			
1.5	Gonzalo Berardo	17/09/21			
1.6	Gonzalo Berardo	03/09/22			
1.7	Gonzalo Berardo	21/09/22			
1.8	Gonzalo Berardo	02/10/22			

INFORMACIÓN DEL PROYECTO

Proyecto	Diseño, implementacion y analisis de sistemas de control sobre FPGA aplicado a robotica
Empresa / Organización	---
Fecha de preparación	15/03/21
Cliente / Laboratorio	Laboratorio de Computación Reconfigurable (LCR) y Laboratorio de Robot Móvil y Manipuladores Autónomos (LARMMA)
Patrocinador	Ing. Antonio Alvarez Abril
Gerente / Lider de Proyecto	Gonzalo Miguel Berardo

INFORMACIÓN INTERNA DEL PROYECTO

Nombre del integrante 1	Gonzalo Miguel Berardo
Legajo	26.499
e-mail	gonzalo_b@msn.com
Responsable Revisión Cátedra	Ing. Ana Lattuca
Fecha de entrega Anteproyecto	15/03/21
Fecha de aprobación del Anteproyecto	
Responsable de la Cátedra	Ing. Antonio Alvarez Abril

INDICE

1 INTRODUCCIÓN

1.1 Executive summary

The design, implementation and analysis of control systems on FPGA (Field Programmable Gate Array) are presented, using a 5 degrees of freedom robot manipulator as a case study. For the work, PID (Proportional Integral Derivative) control algorithms are synthesized in hardware for different numerical representations: integers, fixed point and floating point. The impact on the response in the time domain and the use of hardware resources (area / frequency) is analyzed, in order to obtain valid results without falling to the use of expensive FPGAs.

The work is a continuation of two research articles carried out for the Reconfigurable Computing Laboratory (LCR) of the Mendoza Regional National Technological University. In [1] a position control is implemented that operates with integers and yields an error greater than 12% in steady state. This situation is not tolerable so in [2] the use of fixed point is explored. The procedure for defining the number of bits in Qmn format and the impact of different types of rounding is shown.

Additionally, a wireless HMI (Human Machine Interface) is made to control and parameterize the robot, carried out by means of a SoC (System on a Chip) that incorporates dual core microprocessor and WiFi connectivity.

It includes mathematical modeling of the plant, time and frequency domain responses, PID tuning mechanisms, hardware synthesis of communication cores, control and management using VHDL as hardware description language. Design and manufacture of PCBs (Printed Circuit Board) for stages of data acquisition, signal conditioning, power and electrical supply.

Keywords— PID, Fixed Point, Floating Point, FPGA, VHDL, HDL-Coder, IP-Cores.

1.2 Resumen ejecutivo

Se presenta el diseño, la implementación y el análisis de sistemas de control sobre FPGA (Field Programmable Gate Array), empleando como caso de estudio un robot manipulador de 5 grados de libertad. Para el trabajo se sintetiza en hardware algoritmos de control PID (Proportional Integral Derivative) para diferentes representaciones numéricas: enteros, punto fijo y punto flotante. Se analiza el impacto sobre la respuesta en el dominio del tiempo y el uso de recursos de hardware (área/frecuencia), a fin de obtener resultados aceptables sin caer al empleo de FPGAs costosas.

El trabajo es una continuación de dos artículos de investigación realizados para el Laboratorio de Computación Reconfigurable (LCR) de la Universidad Tecnológica Nacional Regional Mendoza. En [1] se implementa un control de posición que opera con enteros y arroja un error mayor al 12% en estado estacionario. Esta situación no es tolerable por lo que en [2] se explora el uso de punto fijo. Se muestra el procedimiento para definir la cantidad de bits en formato Qmn y el impacto de los distintos tipos de redondeo.

Adicionalmente se realiza un HMI (Human Machine Interface) inalámbrico para control y parametrización del robot, llevado a cabo, por medio de un SoC (System on a Chip) que incorpora microprocesador dual core y conectividad WiFi.

Se incluye, modelado matemático de la planta, respuestas en el dominio del tiempo y de la frecuencia, mecanismos de sintonización de PIDs, sintetización en hardware de núcleos de comunicación, control y gestión usando VHDL como lenguaje descripción de hardware. Diseño y fabricación de PCBs (Printed Circuit Board) para etapas de adquisición de datos, acondicionamiento de señales, potencia y alimentación eléctrica.

Palabras Clave— PID, Punto Fijo, Punto Flotante, FPGA, VHDL, HDL-Coder, IP-Cores.

2 DESCRIPCIÓN DEL PROYECTO

Un manipulador es un robot industrial multifuncional reprogramable con varios grados de libertad, capaz de manipular materias, piezas, o herramientas según trayectorias programadas. La incorporación de los manipuladores en ambientes de trabajo, trae como ventajas la liberación del hombre a trabajos peligrosos, desagradables o monótonos y arroja como resultado un aumento de la productividad y calidad de los procesos.

El manipulador posee 5 grados de libertad, los cuales están denotados por sus similitudes con el ser humano, estos son: cintura, hombro, codo, muñeca elevación y muñeca giro. En la Fig.1 se muestra la vista general del robot indicando los movimientos relativos a cada grado de libertad.

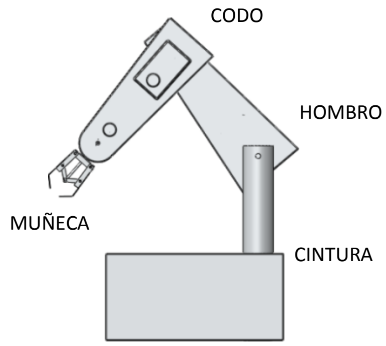


Fig. 1 - Robot Manipulador.

El sistema se diseña con el objeto de controlar la posición angular de las articulaciones. Esto se logra empleando algoritmos PID. Los resultados se modulan por ancho de pulso para gobernar las velocidades de los motores, de tal modo que variando coherentemente dicha velocidad, se intenta controlar la posición del robot.

Al mismo tiempo, se resuelve el estudio cinemático donde se relaciona la posición y orientación del extremo del robot, con un sistema de coordenadas de referencia.

Para lograr que cada articulación inicie y termine su movimiento en tiempo real, se implementa un controlador en una arquitectura tipo paralela, por medio de procesos dentro de una FPGA. Este es un dispositivo de hardware reconfigurable que, mediante un lenguaje de descripción de hardware como VHDL, permite desarrollar diversos circuitos. Siendo esto posible gracias a programas CAD (Computer Aided Design) llamados sintetizadores.

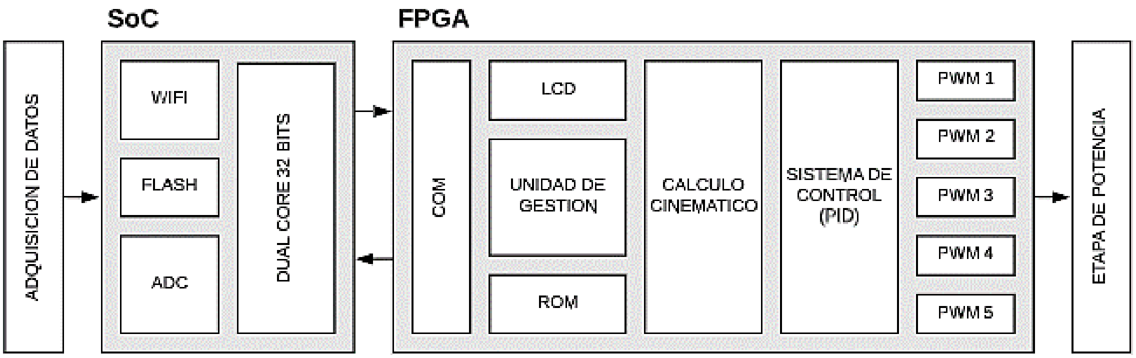


Fig. 2 - Bloques del proyecto.

En la imagen anterior, se ilustra los procesos sintetizados restantes, denominados LCD para visualización en campo de variables, ROM para almacenar valores e instrucciones específicas de funcionamiento, COM para comunicación serial de datos y Unidad de Gestión que actúa como árbitro entre los núcleos.

El SoC se encarga de realizar las conversiones analógicas a digital y trasmitirlas hacia la FPGA por I2C (inter integrated circuits). El módulo WiFi se configura como punto de acceso y se monta un Servidor Web para la parametrización del robot por parte del usuario. De este modo, por medio de un dispositivo Smart es posible establecer una comunicación inalámbrica para tomar acciones de control y monitoreo. Las etapas de Adquisición de Datos y Potencia se encargan de acondicionar las señales eléctricas a fin de lograr un correcto funcionamiento global.

2.1 Objetivo General

- Diseñar un sistema de control de posición para un robot manipulador.
- Evaluar diferentes representaciones numéricas.
- Implementar los diseños sobre una FPGA usando VHDL.
- Crear un HMI.
- Generar documentación detallada para próximos proyectos.

2.2 Objetivo Particular

- Aplicar métodos de identificación de sistemas para modelado de plantas.
- Usar procedimientos para el ajuste de parámetros de PIDs.
- Simular lazos de control para análisis en el dominio tiempo y de la frecuencia.
- Poner a prueba las herramientas existentes para el uso de operaciones aritméticas con enteros y reales para HDL.
- Describir IP Cores (Intellectual Property Core) de SPI (Serial Peripheral Interface) y UART (Universal Asynchronous Receiver-Transmitter) necesarios para comunicación. Además núcleos para modulación PWM (Pulse Width Modulation), Control PID y LCD (Liquid Crystal Display), entre otros.
- Empleo de los siguientes software para objetivos específicos:
  - MATLAB & Simulink - MathWorks.
  - SolidWorks CAD (Computer Aided Design) - Dassault Systemes
  - Altera Quartus II & modelSim

- Estudio de las siguientes herramientas de software
  - System Identification Toolbox - MATLAB - MathWorks
  - PID Controller for Simulink - MATLAB - MathWorks
  - Control System Toolbox Root Locus Design GUI - MathWorks
  - Generate Verilog and VHDL code for FPGA and ASIC designs using HDL-Coder – MathWorks

- Realizar PCB necesarias.

3 JUSTIFICACIÓN DEL PROYECTO

3.1 ANTECEDENTES DEL PROYECTO

El proyecto es la culminación de dos trabajos realizados como Investigador Académico en los laboratorios LARMMA (Laboratorio de Robots Móvil y Manipuladores Autónomos) y LCR (Laboratorio de Computación Reconfigurable), de la Universidad Tecnológica Nacional, Facultad Regional Mendoza.

El trabajo desarrollado para LARMMA titulado “Controlador de Posición Basado en FPGA para Robot Manipulador” publicado en el **24º Congreso Argentino de Control Automático** presenta el diseño de un controlador implementado en FPGA, para el control de posición de un robot manipulador con 5 grados de libertad. El desarrollo incluye para cada articulación la implementación de dos cores (núcleos), un PID y un PWM, para gobernar a cada motor. La comunicación entre los núcleos y las etapas externas se efectúa a través de un núcleo SPI. Finalmente un core central, actúa como unidad de gestión. En la implementación en hardware se utiliza un kit de desarrollo de Digilent basado en un FPGA Spartan-3E de Xilinx junto con su software ISE, mientras que la descripción se realiza en VHDL. Se presentan los resultados de la síntesis y el procedimiento de sintonización del PID.

El segundo trabajo llevado a cabo para el Laboratorio LCR titulado “Consideraciones numéricas al implementar un sistema de control sobre FPGA” publicado en **IEEExplore** y en la **XVI Reunión de Trabajo en Procesamiento de la Información y Control.** El trabajo describe las consideraciones numéricas al momento de hacer un sistema de control sobre FPGA. Se toma como caso de estudio la implementación del control de posición de un robot manipulador. El sistema se realiza usando dos representaciones numéricas, enteros y punto fijo. Se muestra el procedimiento para definir la cantidad de bits de la parte entera y fraccionaria. También se analiza el impacto de los distintos tipos de redondeo. Se concluye que, si no existe requerimiento de frecuencia, optar por HDL-Coder es una buena opción, por su facilidad de uso y escasa ocupación de área. Encontrando las mejores prestaciones para los redondeos floor y simplest. Además, en la optimización se ve que una estrategia de síntesis que pondere el área, da la mayor frecuencia, aunque sea poco intuitivo.

3.2 ESTADO ACTUAL

Cuando se planteó recuperar el robot manipulador, para el inicio del proyecto, se ausentaban componentes que formaban parte de su estructura original, como el bus de comunicación, el simulador, la PC IBM, tampoco se disponía de información de la parte técnica, solo constaba de un manual de usuario destinado a la puesta en marcha y operación del manipulador. A faltar estos componentes es que se decide reemplazar la electrónica de control analógico por un sistema de control digital, a la vez de implementar todo lo necesario para su manipulación.

3.3 BENEFICIOS DEL PROYECTO

Con el proyecto se ven beneficiados los laboratorios LARMMA y LCR de la Universidad Tecnológica Nacional Facultad Regional Mendoza ya que queda como material didáctico para el aprendizaje de múltiples conceptos. Adicionalmente es utilizado en la Catedra de Sistemas de Control de la carrera de Ingeniería Electrónica como Practica de Laboratorio para los conceptos de control automático.

4 PLANTEAMIENTO DEL PROBLEMA

En todo sistema de control es necesario realizar operaciones aritméticas a fin de obtener un resultado. Estas operaciones consumen más o menos recursos de acuerdo al tipo de datos elegido. A su vez, el tipo de datos es seleccionado de acuerdo a dos parámetros impuestos por la aplicación, resolución y rango dinámico. La resolución es la mínima distancia entre dos números que el sistema es capaz de representar, mientras que el rango dinámico es la diferencia entre el número más grande y el más pequeño en una representación determinada. Por lo tanto, el estudio se centra en escoger el formato que cumpla con las especificaciones de diseño y que a su vez, presente el menor uso de hardware a fin de mantener una relación de costo-beneficio adecuado.

El tipo datos a usar son enteros o reales, y los últimos en una de sus dos representaciones computacionales: punto fijo o flotante. Si se utiliza VHDL (Very High Speed Integrated Circuit Hardware Description Language) para el diseño, las operaciones con enteros se realizan de forma sencilla, gracias a librerías existentes. En cambio sí se opera con reales la complejidad del diseño crece.

Las herramientas de diseño como ISE de Xilinx o Quartus II de Altera presentan módulos configurables, IP-Cores, que permiten operar en punto flotante, tanto de simple (32 bits) como de doble precisión (64 bits) de acuerdo al estándar IEEE-754 [4]. Sin embargo, no presentan esta opción para punto fijo, ni tampoco hay una norma que regule su formato.

Para aquellas aplicaciones en donde no se requiere de un rango dinámico amplio (en tal caso es más adecuado emplear punto flotante), se puede optar por trabajar con punto fijo. Para lo cual, existen varias opciones, entre ellas: hacer uso de los tipos de datos ufixed y sfixed de la versión 2008 de VHDL [5], o usar HDL-Coder de Matlab [6].

5 ALCANCE

5.1 DEFINICIÓN DE ETAPAS

A continuación se enumeran las etapas del proyecto:

- ✓ Relevamiento electromecánico del robot manipulador.
- ✓ Diseño e implementación de etapas de potencia y Adquisición de datos.
- ✓ Modelado matemático de la planta.
- ✓ Diseño del sistema de control de posición.
- ✓ Sintetización en hardware de los procesos ilustrados en Fig. 2.
- ✓ Programación de software y firmware del SoC, según Fig. 2.
- ✓ Integración y prueba de funcionamiento hibrida (FPGA-SoC).
- ✓ Análisis del uso de diferentes representaciones numéricas y tipos de redondeo.
- ✓ Diseño e implementación de interfaz hombre maquina en Web Server.
- ✓ Prueba de funcionamiento global.
- ✓ Documentación de proyecto.

5.2 ENTREGABLES PRINCIPALES

5.2.1 Diagramas y Placas Electrónicas

Se entrega los diagramas electrónicos correspondientes a las etapas de potencia, adquisición de datos e interconexiones entre la PFGA y el SoC. Además se incluye un relevamiento electromecánico del robot manipulador y la placa PCB MainBoard que alberga a todos los dispositivos involucrados del proyecto y es donde convergen todas las señales eléctricas.

La placa MainBoard adicionalmente cuenta con un filtro EMI para la protección contra señales espurias y una fuente conmutada Step Down para estabilizar y controlar la tensión en 3.3V. Para el control de motores se incluye dos drivers L298N que permite controlar tanto la velocidad como sentido de giro.

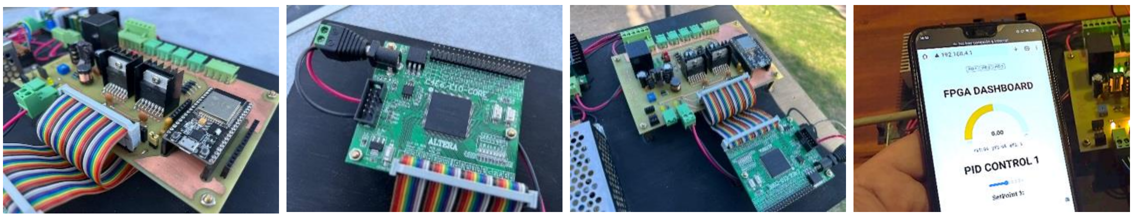


Fig. 3 - Fotografías del proyecto.

5.2.2 Dashboard Control

El uso de la FPGA permite realizar, en tiempo real, los cálculos de los algoritmos PID para cada articulación del robot, todo esto gracias a su gran capacidad de cómputo que permite realizar múltiples tareas de manera concurrente. Por lo tanto, toda la información procesada por la FPGA es visualizada en una página web, por medio de la implementación de un WebServer en el SoC ESP32. La visualización web de los diferentes parámetros del robot se agrupa en un dashboard control para facilitar la interacción con el robot.

El dashboard junto a todos sus códigos fuentes quedan a disposición como parte de los entregables principales del proyecto. Dentro del apartado Hardware Complementario del presente informe, se brinda mayor información de lo mencionado.

La tabla siguiente reúne los entregables principales del proyecto.

Tabla 1: Entregables

Entregables	**Descripción**
Diseño Electrónico	Diagramas, esquemas y circuitos eléctricos
Código Fuente	Códigos Fuentes de software, firmware y descripción de hardware
Placa de Control	Implementación de placa de control que integra todas las etapas del .proyecto (Adquisición de datos, potencia, Fpga y Soc).
Interfaz HMI	App para la interacción entre el robot y el usuario
Documentación	Análisis, modelados matemáticos, investigaciones, publicaciones, etc.

5.3 LÍMITES O FUERA DE ALCANCE

El trabajo aporta las bases para la implementación del modelado cinemático de la planta bajo estudio. El mismo se llevará a cabo por el Laboratorio de Robot Móvil y Manipuladores Autónomos (LARMMA) de la Universidad Tecnológica Nacional, Facultad Regional Mendoza.

6 FACTIBILIDAD ECONOMICA

6.1 FLUJO DE CAJA

Para la viabilidad del proyecto y desarrollo del flujo de caja se analizan los siguientes puntos:

1. Se estima producir y vender 500 unidades anuales a US$50 cada una durante los 2 primeros años y a US$60 a partir del 3er año cuando el producto se haya consolidado en el mercado. Las proyecciones de ventas muestran que a partir del 6to año éstas se podrían incrementar en un 20%. Para llevar adelante el proyecto es necesario invertir en:

Terrenos	US$12.000
Obras Físicas	US$13.000
Maquinarias	US$48.000

Una de las maquinarias, cuyo valor es de US$10.000, debe reemplazarse cada 8 años por otra similar.

2. La maquinaria usada se podría vender en US$2.500.
3. Los costos de fabricación para un volumen de hasta 550 unidades anuales son los siguientes:

Mano de obra	US$2,00
Materiales	US$3,50
Costos indirectos	US$0,50

Sobre este nivel de producción es posible importar directamente los materiales a un costo unitario de US$3,20.

4. Los costos fijos de fabricación se estiman en US$2.000, sin incluir depreciación. La ampliación de la capacidad hará que estos costos se incrementen en US$200.
5. Se pagarán comisiones del 2% sobre ventas a los vendedores.
6. Los gastos de administración y ventas se estiman en US$800 anuales, los primeros 5 años, y en US$910 cuando se incrementa el nivel de operación.
7. La legislación vigente permite depreciar las obras físicas en 20 años y todas las maquinarias en 10 años. Los activos intangibles se amortizan linealmente en 5 años.

CONCEPTO	1	2	3	4	5	6	7	8	9	10
Obra Física inicial	US$3.000	US$3.000	US$3.000	US$3.000	US$3.000	US$3.000	US$3.000	US$3.000	US$3.000	US$3.000
Obra Física Amp.						US$600	US$600	US$600	US$600	US$600
Maq. inicial a)	US$3.800	US$3.800	US$3.800	US$3.800	US$3.800	US$3.800	US$3.800	US$3.800	US$3.800	US$3.800
Maq. inicial b)	US$1.000	US$1.000	US$1.000	US$1.000	US$1.000	US$1.000	US$1.000	US$1.000		
Maq. Reemp.									US$1.000	US$1.000
Maq. Amp.						US$800	US$800	US$800	US$800	US$800
Depreciación Total	US$7.800	US$7.800	US$7.800	US$7.800	US$7.800	US$9.200	US$9.200	US$9.200	US$9.200	US$9.200

8. La amortización de intangibles corresponde al 20% anual del total de activos intangibles posibles de contabilizar, incluyendo el costo del estudio del proyecto. Los gastos de puesta en marcha ascienden a US$2.000, dentro de los que se incluye el costo del estudio de viabilidad, que asciende a US$800.

	0	2	3	Incremento	5	6	Incremento
Costos Variables	US$ -3.000	US$ -3.000	US$ -3.000	US$ 0	US$ -3.000	US$ -3.420	US$ -420
Costos Fabricación Fijos	US$ -2.000	US$ -2.000	US$ -2.000	US$ 0	US$ -2.000	US$ -2.200	US$ -200
Comisiones Ventas	US$ -500	US$ -500	US$ -600	US$ -100	US$ -600	US$ -720	US$ -120
Gastos Administración y venta	US$ -800	US$ -800	US$ -800	US$ 0	US$ -800	US$ -910	US$ -110
COSTO TOTAL	US$ -6.300						US$ -850
6 meses	50%						
TOTAL INVERSION CT	US$ -3.150			US$ -50			US$ -425

9. El valor libro es el saldo por depreciar del activo que se vende al termino del 8vo año. Como éste tuvo un costo de US$10.000 y se desprecia en 10 años, su valor libro corresponde a US$2.000.
10. Ajuste de gastos no desembolsables: Para anular el efecto de haber incluido gastos que no constituían egresos de caja, se suma la depreciación, la amortización de intangibles y el valor libro. La razón de incluirlos primero y eliminarlos después obedece a la importancia de incorporar el efecto tributario que estas cuentas ocasionan a favor del proyecto.

    Para el análisis se considerará los desembolsos para cubrir; Impuesto al Valor Agregado, Ganancias Personas Jurídicas e Ingresos Brutos.
11. En el momento 0 se registra la inversión inicial (terrenos, obras físicas y maquinarias) por US$73.000, más la inversión relevante en activos intangibles por US$2.000.
12. En el año 8 se registra la inversión para reponer el activo vendido.
13. En el momento 5 (final del 5to año) se registra la inversión para enfrentar la ampliación de la capacidad de producción a partir del 6to año. El crecimiento de la producción para satisfacer el incremento de las ventas requeriría invertir US$9.000 en obras físicas adicionales y US$8.000 en maquinarias en el 5to año.
14. La inversión en capital de trabajo se registra en el año 0 y se estima como el equivalente a 6 meses del costo total desembolsable del 1er año. Es decir, se calcula como el 50% (medio año) de los costos anuales desembolsables; se registra primero en el momento 0 y luego el incremento en esta inversión en los momentos 2 y 5. La tabla siguiente muestra los pasos de cálculo.
15. La rentabilidad mínima exigida para el proyecto será del 12,5%.

#	CONCEPTO	0	1	2	3	4	5	6	7	8	9	10
1	Ingresos	-	$ 25.000	$ 25.000	$ 30.000	$ 30.000	$ 30.000	$ 36.000	$ 36.000	$ 36.000	$ 36.000	$ 36.000
2	Ventas de Activos	-	-	-	-	-	-	-	-	$ 2.500	-	-
3	Costos Variables	-	$ -3.000	$ -3.000	$ -3.000	$ -3.000	$ -3.000	$ -3.420	$ -3.420	$ -3.420	$ -3.420	$ -3.420
4	Costos de Fabricación Fijos	-	$ -2.000	$ -2.000	$ -2.000	$ -2.000	$ -2.000	$ -2.200	$ -2.200	$ -2.200	$ -2.200	$ -2.200
5	Comisiones por Ventas	-	$ -500	$ -500	$ -600	$ -600	$ -600	$ -720	$ -720	$ -720	$ -720	$ -720
6	Gastos de Administracion y Ventas	-	$ -800	$ -800	$ -800	$ -800	$ -800	$ -910	$ -910	$ -910	$ -910	$ -910
7	Depreciación	-	$ -7.800	$ -7.800	$ -7.800	$ -7.800	$ -7.800	$ -9.200	$ -9.200	$ -9.200	$ -9.200	$ -9.200
8	Amortizaciones Intangibles	-	$ -400	$ -400	$ -400	$ -400	$ -400	-	-	-	-	-
9	Valor libros	-	-	-	-	-	-	-	-	$ -2.000	-	-
	UTILIDAD BRUTA	-	$ 10.500	$ 10.500	$ 15.400	$ 15.400	$ 15.400	$ 19.550	$ 19.550	$ 20.050	$ 19.550	$ 19.550
10	Impuestos (38,5%)	-	$ -4.043	$ -4.043	$ -5.929	$ -5.929	$ -5.929	$ -7.527	$ -7.527	$ -7.719	$ -7.527	$ -7.527
	UTILIDAD NETA	-	$ 6.458	$ 6.458	$ 9.471	$ 9.471	$ 9.471	$ 12.023	$ 12.023	$ 12.331	$ 12.023	$ 12.023
	Depreciación	-	$ 7.800	$ 7.800	$ 7.800	$ 7.800	$ 7.800	$ 9.200	$ 9.200	$ 9.200	$ 9.200	$ 9.200
	Amortizaciones Intangibles	-	$ 400	$ 400	$ 400	$ 400	$ 400	-	-	-	-	-
	Valor libros	-	-	-	-	-	-	-	-		-	-
11	Inversión Inicial	$ -75.000	-	-	-	-	-	-	-	-	-	-
12	Inversion de Reemplazo	-	-	-	-	-	-	-	-	$ -10.000	-	-
13	Inversion por Ampliacion	-	-	-	-	-	$ -17.000	-	-	-	-	-
14	Inversion por Capital de Trabajo	$ -3.150		$ -50			$ -425					
	FLUJO DE CAJA	$ -78.150	$ 14.658	$ 14.608	$ 17.671	$ 17.671	$ 246	$ 21.223	$ 21.223	$ 11.531	$ 21.223	$ 21.223

RENTABILIDAD EXIGIDA Td	12,50%	**VAN**	$ 8.156,58	**TIR**	15%	**PRI**	6 años

- *El VAN al ser un valor positivo indica que conviene realizar la inversión, ya que se recupera lo invertido con un rendimiento equivalente a la tasa de descuento exigida Td = 12,50%.*
- Se observa que al ser el TIR > Td, *se recupera lo y además se gana más de lo que se esperaba ganar.*
- *El capital invertido se recuperará en un plazo de 6 años y 5 meses.*

6.2 INDICADORES ECONÓMICOS

Para conocer sobre la rentabilidad del proyecto es necesario primero definir algunos de los principales indicadores financieros tales como el Valor Actual Neto (VAN), la Tasa Interna de Retorno (TIR) y el Plazo de Recuperación (Pay-back).

6.2.1 Valor Actual Neto (VAN)

El VAN también llamado Valor Presente Neto (VPN), es la diferencia entre el valor presente de los ingresos futuros que percibirá una empresa y la cantidad que invierte para sacar adelante un proyecto. Si el resultado de esta operación es positivo, quiere decir que el negocio es viable.

Se dice que es un indicador dinámico ya que mide la viabilidad de la inversión actualizando los flujos netos de caja (FNC) a una determinada tasa de descuento (Td), que para el caso de estudio es de 12,5%.

$$VAN = -D_0 + \frac{FNC_1}{(1+Td)^1} + \frac{FNC_2}{(1+Td)^2} + \cdots + \frac{FNC_n}{(1+Td)^n} \quad (1)$$

$$\begin{aligned} VAN = -78.150 &+ \frac{14.658}{(1+0{,}125)^1} + \frac{14.608}{(1+0{,}125)^2} + \frac{17.671}{(1+0{,}125)^3} + \frac{17.671}{(1+0{,}125)^4} \\ &+ \frac{246}{(1+0{,}125)^5} + \frac{21.223}{(1+0{,}125)^6} + \frac{21.223}{(1+0{,}125)^7} + \frac{11.531}{(1+0{,}125)^8} \\ &+ \frac{21.223}{(1+0{,}125)^9} + \frac{21.223}{(1+0{,}125)^{10}} \end{aligned}$$

$$\boldsymbol{VAN = 8.156 \therefore Conviene\ realizar\ la\ inversi\acute{o}n.}$$

El VAN al ser un valor positivo indica que conviene realizar la inversión, ya que se recupera lo invertido con un rendimiento equivalente a la tasa de descuento exigida Td y además se gana más de lo que se esperaba ganar.

6.2.2 Tasa Interna de Retorno (TIR)

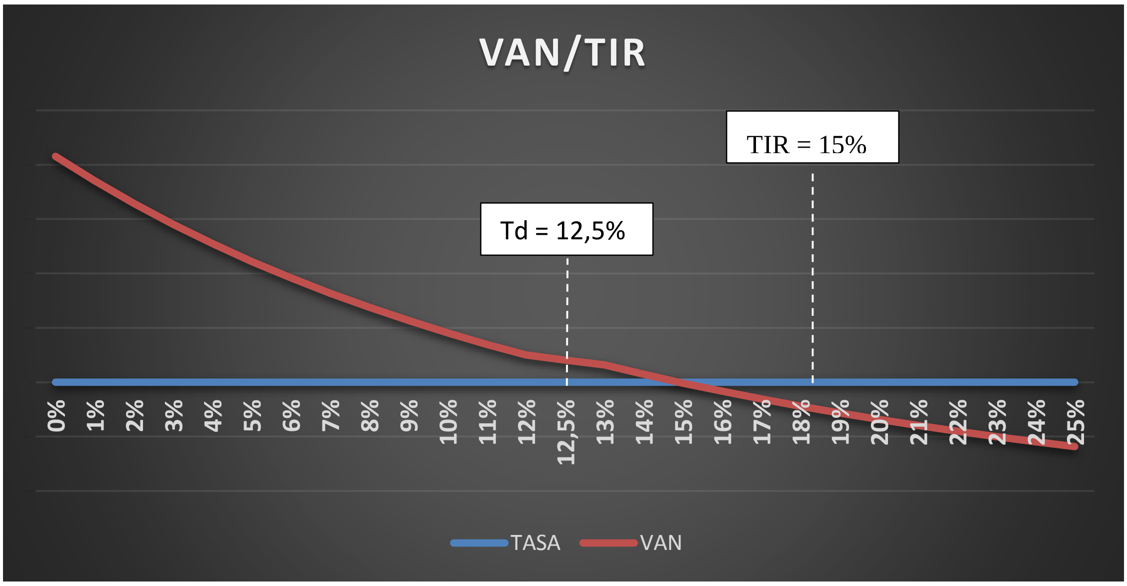
Calcula la tasa de retorno (rentabilidad) que generará una inversión. En otras palabras, arroja el valor de ganancia o pérdida que tendrá un proyecto para todas las partes involucradas. Está muy ligado al VAN, pues también se define como el valor de la tasa de descuento que hace posible que el Valor Actual Neto sea igual a cero.

$$0 = -D_0 + \frac{FNC_1}{(1+Td)^1} + \frac{FNC_2}{(1+Td)^2} + \cdots + \frac{FNC_n}{(1+Td)^n} \therefore Td = ? \quad (2)$$

$$\boldsymbol{Td = 15\% \therefore Se\ obtiene\ un\ 15\%\ sobre\ lo\ invertido.}$$

Se observa que al ser el TIR > Td, *se recupera lo invertido con un rendimiento equivalente a la tasa de descuento exigida Td y además se gana más de lo que se esperaba ganar.*

El siguiente grafico muestra la relación entre el VAN, el TIR y Td.



6.2.3 Plazo de Recuperación (Pay-back)

El Pay-back es una herramienta financiera de valoración de inversiones que permite determinar el plazo que demorará una empresa en recuperar el capital que invertirá en un proyecto. Considera que una inversión es más rentable que otra basándose solo en el argumento de cuál permitirá recuperar antes los recursos invertidos.

Entre las principales ventajas de determinar el Pay-back, se encuentran:

- Tiene una fórmula sencilla y fácil de aplicar. Se traen a valor presente todos los flujos futuros esperados y se van acumulando hasta recuperar el monto de la inversión.
- Brinda una idea inicial del nivel de liquidez que ofrecerá el negocio y el riesgo que supone la inversión.
- En épocas de crisis y volatilidad económica, es útil para aumentar la seguridad de los accionistas a la hora de realizar una inversión.
- Es un recurso sumamente valioso en dos casos: al momento de invertir en proyectos con un nivel de riesgo alto y en proyectos con vida limitada.

$$\boldsymbol{Payback = PRI = 6\ a\tilde{n}os\ y\ 5\ meses}$$

El capital invertido se recuperara en un plazo de 6 años y 5 meses.

6.2.4 Diferencias entre el VAN, la TIR y el Payback

A pesar de que tanto el VAN como la TIR analizan si es rentable invertir en un proyecto, también presentan notables diferencias. Una de las principales distinciones entre ambas es que el VAN arroja los resultados en unidades de valor monetario (soles, dólares, euros, etc.), mientras que la TIR expresa los resultados en forma de porcentaje.

Otra diferencia notoria es en la información que requieren para obtener los resultados: en el VAN se tiene que tomar en consideración los vencimientos de los flujos de caja, poniendo énfasis en los más próximos para evitar correr riesgos al realizar la inversión. Con respecto a la TIR, no considera este indicador para calcular la rentabilidad de un proyecto. Asimismo, el VAN compara proyectos para señalar cuál inversión es mejor que otra, mientras que la TIR no necesita recurrir a confrontaciones para indicar a qué tasa y en qué tiempo se recuperarán los recursos invertidos.

En cuanto al Pay-back, este difiere del VAN y se acerca al TIR, pues considera el flujo de caja para establecer el período de retorno de inversión (ROI) de un proyecto.

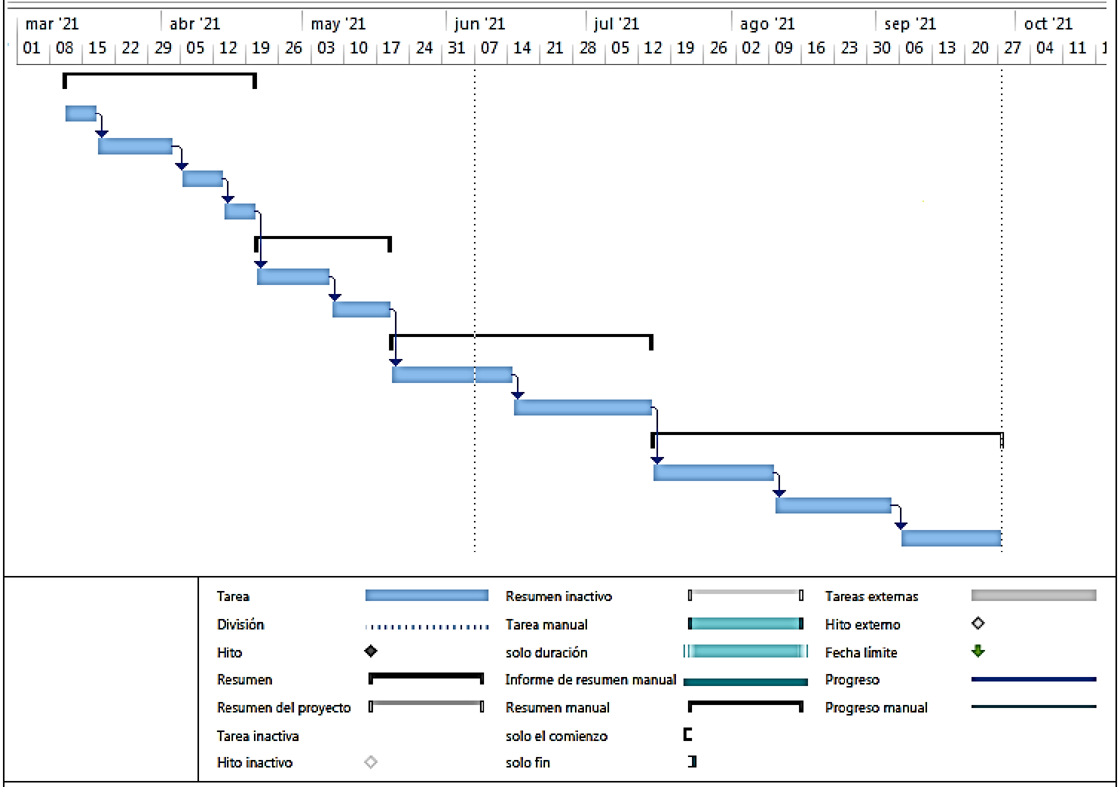
7 PLANIFICACIÓN DEL PROYECTO

7.1 DIAGRAMA DE GANTT

En la imagen siguiente se adjunta el diagrama de Gantt detallando las principales tareas del proyecto. Para mayor detalle ver la estructura de desglose de tareas (EDT).

DIAGRAMA DE GANTT

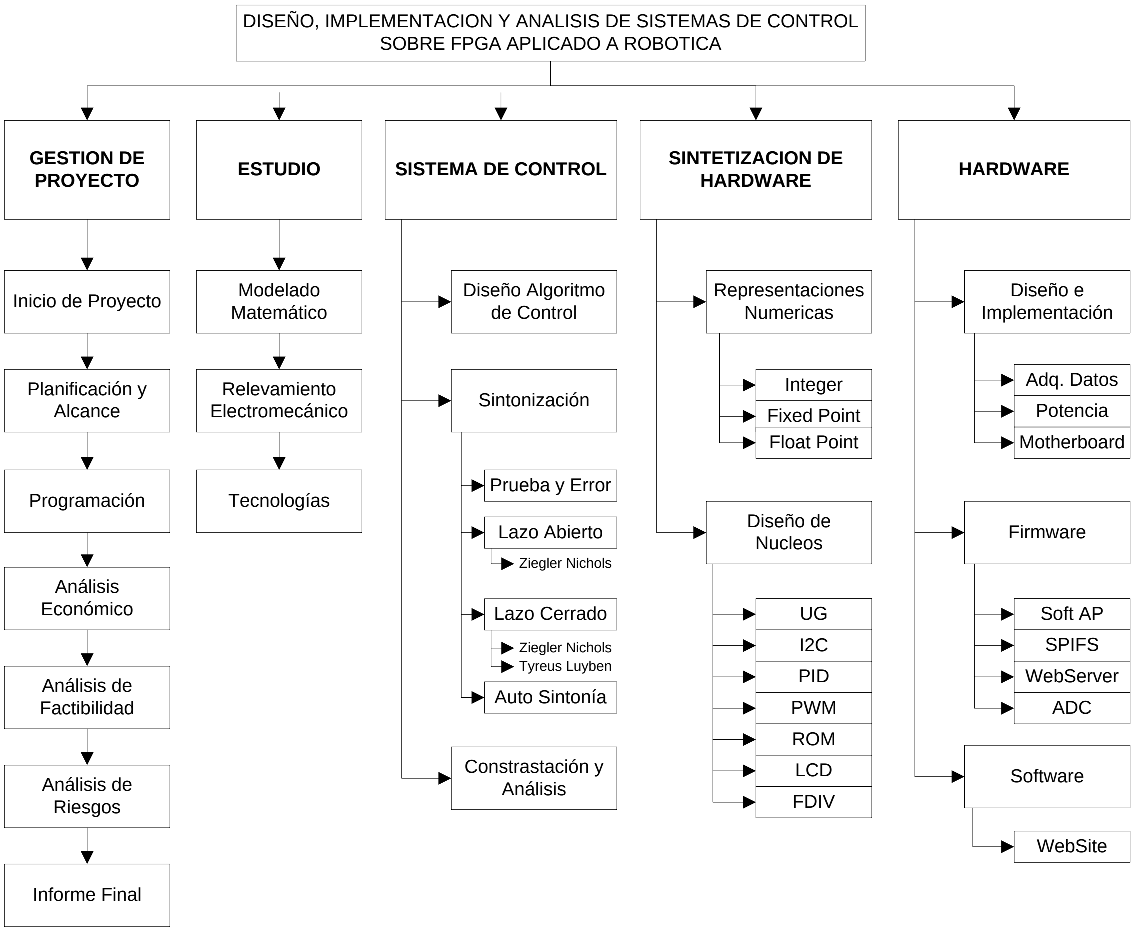
	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		**ESTUDIO**	**29 días**	**jue 11/03/21**	**mar 20/04/21**	
2		Relevamiento Electromecánico	5 días	jue 11/03/21	mié 17/03/21	
3		Modelado Matemático	12 días	jue 18/03/21	vie 02/04/21	2
4		Tecnologías	7 días	lun 05/04/21	mar 13/04/21	3
5		Fin de la Fase de Estudio	5 días	mié 14/04/21	mar 20/04/21	4
6		**SISTEMA DE CONTROL**	**21 días**	**mié 21/04/21**	**mié 19/05/21**	
7		Diseño Algoritmo de Control	12 días	mié 21/04/21	jue 06/05/21	5
8		Sintonización	9 días	vie 07/05/21	mié 19/05/21	7
9		**SINTETISIS DE HARDWARE**	**40 días**	**jue 20/05/21**	**mié 14/07/21**	
10		Representaciones Numericas	18 días	jue 20/05/21	lun 14/06/21	8
11		Diseño de Núcleos	22 días	mar 15/06/21	mié 14/07/21	10
12		**HARDWARE**	**53 días**	**jue 15/07/21**	**lun 27/09/21**	
13		Diseño e Implementación	18 días	jue 15/07/21	lun 09/08/21	11
14		Firmware	19 días	mar 10/08/21	vie 03/09/21	13
15		Software	16 días	lun 06/09/21	lun 27/09/21	14



Se estima un total de 143 días de trabajo con jornadas de 6 horas diarias de lunes a viernes. El proyecto comienza el 6 de junio de 2021 y se estima terminar para finales de septiembre del mismo año.

7.2 ESTRUCTURA DE DESGLOSE DE TAREAS

A continuación se presenta la estructura de desglose de tareas (EDT) correspondiente al proyecto.



8 MATRIZ DE RIESGOS

Los riesgos en proyectos son eventos o condiciones inciertas que, en caso de ocurrir, tiene un efecto positivo o negativo sobre los objetivos de un proyecto.

Todo proyecto tiene asociados riesgos, por lo que es importante enumerarlos, analizarlos y formar un plan de acción para su reducción o mitigación, a fin de no comprometer el normal desarrollo del mismo.

8.1 IDENTIFICACIÓN DE RIEGOS

La primera etapa de la planificación de riesgos comienza con su identificación, los cuales pueden presentarse en la dimensión tecnológica, en la ambiental, en los recursos humanos, en la economía asociada, en el flujo del presupuesto, en cuestiones culturales, una mala planificación, etc.

Algunos de los riegos presentes en el proyecto son por ejemplo, los daños que puedan existir en equipos e instrumentación de laboratorio necesarios para el avance de las tareas. La probabilidad de contraer Covid-19 la persona encargada de llevar adelante el proyecto o la existencia de retrasos de tareas tercerizadas, como así también, la presencia de restricciones de circulación por la actual pandemia que nos toca atravesar. Cortes de energías, cierre de importaciones, Interrupciones forzadas del proyecto, insumos de baja calidad, cambios en el diseño e implementación, retrasos en envíos o entregas, cambios en el presupuesto y problemas personales son algunos de los riegos analizados.

8.2 ANÁLISIS DE RIEGOS

Para priorizar los riesgos se tienen que tener en cuenta cuál es el impacto del mismo sobre el proyecto y cuál es su probabilidad de ocurrencia dentro del contexto global de desempeño. El análisis de riesgo tiene como objetivo fundamental tratar de mitigar el impacto de los riesgos que son más dañinos sobre el proyecto y gran parte del análisis gira en torno a generar estrategias relacionadas con esto.

A continuación, se describe un análisis de riesgo cualitativo por medio de la herramienta de matriz de probabilidad/impacto que permite establecer prioridades en cuanto a los posibles riesgos de un proyecto en función tanto de la probabilidad de que ocurran como de las repercusiones que podrían tener sobre el proyecto en caso de que ocurrieran.

La matriz se compone de dos ejes: un eje vertical en donde se establecen los valores de probabilidad (entre 0 – imposible y 1 – siempre) y un eje horizontal en donde se establecen los valores del impacto del riesgo sobre los objetivos del proyecto (en donde 0 implica que ese riesgo no repercutiría en los objetivos y 1 que dificultaría en gran medida el cumplimiento de los mismos). Los valores obtenidos en las diferentes celdas de la matriz son el resultado de multiplicar la probabilidad de ocurrencia por el impacto del riesgo, indicando los valores más altos (máximo 1) los riesgos más críticos del proyecto y los más bajos los menos relevantes".

Tabla 12: Matriz de riesgos del proyecto

MATRIZ DE RIESGOS

RIESGO	Aparición probabilidad	Gravedad (Impacto)	Valor del Riesgo	Nivel de Riesgo
Daños en equipos e instrumentación	2	3	6	Apreciable
Contraer Covid-19	3	5	15	Muy grave
Retrasos tareas tercerizadas por Covid-19	4	3	12	Importante
Restricciones de circulación por cuarentena	3	1	3	Apreciable
Cortes de energías	2	5	10	Importante
Cierre de importaciones	2	5	10	Importante
Interrupciones forzadas del proyecto	2	5	10	Importante
Insumos de baja calidad	4	2	8	Apreciable
Cambios en el diseño e implementación	2	4	8	Apreciable
Retrasos en envíos o entregas	4	3	12	Importante
Cambios en el presupuesto	5	2	10	Importante
Problemas personales	1	5	5	Apreciable

LEYENDA

			GRAVEDAD (IMPACTO)				
			MUY BAJO	BAJO	MEDIO	ALTO	MUY ALTO
			1	2	3	4	5
APARICIÓN (probabilidad)	MUY ALTA	5	5	10	15	20	25
	ALTA	4	4	8	12	16	20
	MEDIA	3	3	6	9	12	15
	BAJA	2	2	4	6	8	12
	MUY BAJA	1	1	2	3	4	5

(rojo)	Riesgo muy grave. Requiere medidas preventivas urgentes. No se debe iniciar el proyecto sin la aplicación de medidas preventivas urgentes y sin acotar sólidamente el riesgo.
(naranja)	Riesgo importante. Medidas preventivas obligatorias. Se deben controlar fuertemente llas variables de riesgo durante el proyecto.
(amarillo)	Riesgo apreciable. Estudiar económicamente ai es posible introducir medidas preventivas para reducir el nivel de riesgo. Si no fuera posible, mantener las variables controladas.
(blanco)	Riesgo marginal.Se vigilará aunque no requiere medidas preventivas de partida.

La matriz de riesgo es una herramienta muy sencilla, la misma permite tener un panorama general de las situaciones de riesgo del proyecto. Los eventos cuya ocurrencia se encuentre en el rango de los "rojos" tendrán que ser evitados en la medida de lo posible y cada uno de ellos dispondrá de su respectivo Plan de Respuesta los Riesgos. Los eventos que estén catalogados dentro de la zona “naranja” y “amarilla” deberán ser supervisados de manera muy cercana y elaborar sus respectivos Plan de Respuesta. Los eventos que caigan en la Zona “blanca” no requerirán de un Plan de Respuesta sino que serán controlados de manera cercana.

8.3 PLAN DE RESPUESTA

La Tabla 11 ilustra los riesgos involucrados en el proyecto ordenados de manera descendente según su valor de riesgo. Se observa un único riesgo catalogado “Muy grave”, seis “Importantes” y cinco “Apreciables”. Esto mismo, se resume en la tabla 12.

Tabla 13: Matriz de riesgos, datos reordenados según el valor de riesgo

#	RIESGOS	PROBABILIDAD	IMPACTO	VALOR DE RIESGO	NIVEL DE RIESGO
1	Contraer Covid-19	3	5	15	Muy grave
2	Retrasos tareas tercerizadas por Covid-19	4	3	12	Importante
3	Retrasos en envíos o entregas	4	3	12	Importante
4	Cortes de energías	2	5	10	Importante
5	Cierre de importaciones	2	5	10	Importante
6	Interrupciones forzadas del proyecto	2	5	10	Importante
7	Cambios en el presupuesto	5	2	10	Importante
8	Insumos de baja calidad	4	2	8	Apreciable
9	Cambios en el diseño e implementación	2	4	8	Apreciable
10	Daños en equipos e instrumentación	2	3	6	Apreciable
11	Problemas personales	1	5	5	Apreciable
12	Restricciones de circulación por cuarentena	3	1	3	Apreciable

Tabla 14: Matriz de riesgos, tabla resumen

		GRAVEDAD (IMPACTO)				
		MUY BAJO 1	BAJO 2	MEDIO 3	ALTO 4	MUY ALTO 5
APARICIÓN (probabilidad)	MUY ALTA 5		1			
	ALTA 4		1	2		
	MEDIA 3	1				1
	BAJA 2			1	1	3
	MUY BAJA 1					1

NIVEL DE RIESGO	CANTIDAD
Muy Grave	1
Importante	6
Apreciable	5

Es importante crear un plan de respuesta para los siete riesgos implicados, para que el normal desarrollo del proyecto no se vea amenazado. A continuación se trata cada uno de ellos.

A) Muy Grave

1- Contraer Covid-19: Es importante reservar una zona de trabajo segura para la persona que lleva a cabo las tareas del proyecto, asignarle recursos, insumos y componentes previamente sanitizados a fin de minimizar la presencia de virus. Realizar reuniones con los stakeholders de manera virtual e incentivar la comunicación por canales electrónicos. Reducir las actividades cotidianas, a las de primera necesidad, para garantizar un bajo riesgo de contagio, etc.

B) Importante

1- Retrasos tareas tercerizadas por Covid-19: Abordar de manera temprana a todas aquellas tareas que requieran ser tercerizadas a fin que se encuentren listas en el momento requerido.
2- Retrasos en envíos o entregas: Realizar un plan de respuesta similar al del punto anterior.
3- Cortes de energías: Tratar de contar con una zona de trabajo con buena iluminación natural a fin de prever la ausencia de luz artificial.
4- Cierre de importaciones: Gestionar de manera temprana la compra de todos aquellos insumos extranjeros ante posibles cierres de mercado. Es importante contar con un plan de respaldo.
5- Interrupciones forzadas del proyecto y cambios en el presupeustos: mantener reuniones periódicas con los stakeholders.

8.4 RISK PRIORITY NUMBER

La matriz de riesgo nos ayuda a tener una visión general de los riesgos del proyecto pero no nos permite establecer con mayor precisión un orden de prioridades de los mismos. El número de prioridad de riesgo (RPN), se define en base a tres escalas que combinan la probabilidad de ocurrencia, con el impacto y la posibilidad de detección que tiene el mismo. Estas tres escalas se definen de la siguiente manera:

Tabla 15: Determinantes RPN

DETERMINANTES DEL NÚMERO DE PRIORIDAD DE RIESGO		
Probabilidad	Impacto	Detección
1.- No ocurrirá.	1.- No impacta	1.- Evidente
2.- No es probable.	2.- Impacto no significativo.	2.- Fácil de detectar
3.- Podría o no suceder	3.- Impacto que puede ser o no significativo.	3.- Puede o no ser detectado.
4.- Es probable que suceda	4.- Impacto significativo	4.- No es fácil de detectar
5.- Sucederá	5.- Impacto severo	5.- Imposible detección.

El RPN se calcula multiplicando (RPN = Probabilidad x Impacto x Detección), cuanto más alto sea este número mayor será el riesgo de que se produzca el evento y el mismo tenga un impacto importante sobre el proyecto.

Tabla 16: Determinación del RPN del proyecto

#	RIESGOS	PROBABILIDAD	IMPACTO	DETECCION	RPN
1	Contraer Covid-19	3	5	3	45
4	Cortes de energías	2	5	4	40
5	Cierre de importaciones	2	5	3	30
2	Retrasos tareas tercerizadas por Covid-19	4	3	2	24
3	Retrasos en envíos o entregas	4	3	2	24
8	Insumos de baja calidad	4	2	3	24
6	Interrupciones forzadas del proyecto	2	5	2	20
7	Cambios en el presupuesto	5	2	2	20
9	Cambios en el diseño e implementación	2	4	2	16
10	Daños en equipos e instrumentación	2	3	2	12
11	Problemas personales	1	5	2	10
12	Restricciones de circulación por cuarentena	3	1	3	9

9 DESARROLLO DEL PROYECTO

9.1 ESTUDIO DEL SISTEMA DE CONTROL

Para llevar adelante el diseño y la implementación de un sistema de control de posición, es necesario definir antes, algunos conceptos de control automático que ayudarán a la comprensión general del documento.

Un sistema de control es un conjunto de dispositivos encargados de administrar, ordenar, dirigir o regular el comportamiento de otro sistema, con el fin de reducir las probabilidades de fallo y obtener los resultados deseados.

Existen dos clases comunes de sistemas de control, sistemas de lazo abierto y sistemas de lazo cerrado. En los sistemas de control de lazo abierto la salida no interviene en la acción de control; mientras que en los de lazo cerrado si se va a requerir conocer la salida para ejercer el control del sistema. Un sistema de lazo cerrado es llamado también sistema de control retroalimentado.

Katsuhiko Ogata [8], define el control retroalimentado como una operación que, en presencia de perturbaciones, tiende a reducir la diferencia entre la salida de un sistema y alguna entrada de referencia y lo continúa haciendo con base en esta diferencia. El esquema representativo de este modelo se muestra en la Figura 4.

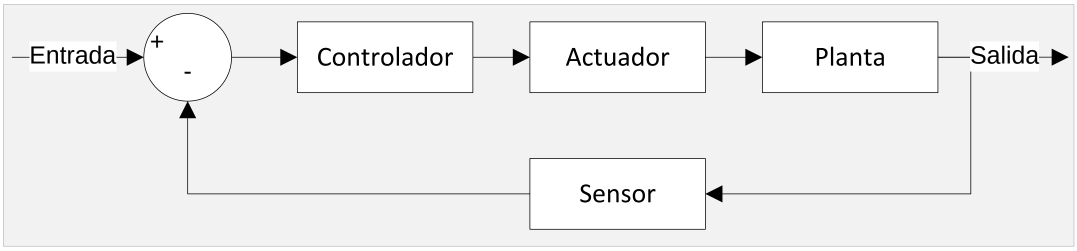


Fig 4 - Diagrama de bloques de un sistema de control retroalimentado.

Donde:

Entradas, son las señales que van a ser controladas o manipuladas de acuerdo a los requerimientos del sistema. Un ejemplo puede ser un sistema de bombeo, en el cual, las entradas pueden ser la presión y el caudal requeridos, los cuales deben controlarse para que estén acordes a las condiciones preestablecidas.

Sumador y Controlador, el sumador es un elemento que recibe las entradas y las compara con las salidas reales del sistema captadas por el sensor. Esta diferencia es conocida como error, y es recibida por el controlador, quien actúa sobre ella dependiendo de sus características de diseño.

Actuador, como su nombre lo indica, es quien desarrolla las acciones ordenadas por el controlador. En un sistema común, estas acciones pueden ser cerrar una válvula, encender un interruptor etc.

Planta/Proceso, la planta es el sistema que se ve sometido a las acciones operativas, es quien provee la señal de salida.

Sensor, el sensor es el elemento que tiene como propósito medir y procesar las salidas de la planta, de tal manera que puedan ser comparadas por parte del sumador y entonces, dependiendo de la diferencia entre lo real y lo esperado, el controlador pueda desarrollar la acción respectiva.

9.1.1 <u>Controlador</u>

Los controladores son elementos que se le agregan al sistema original para mejorar sus características de funcionamiento, con el objetivo de satisfacer las especificaciones de diseño tanto en régimen transitorio como en estado estable. La primera forma para modificar las características de respuesta de los sistemas es el ajuste de ganancia (lo que posteriormente se definirá como control proporcional). Sin embargo, aunque por lo general el incremento en ganancia mejora el funcionamiento en estado estable, se produce una pobre respuesta en régimen transitorio y viceversa. Por tal motivo, es necesario agregar elementos a la simple variación de ganancia, lo cual da lugar a los diversos tipos de controladores:

• Control proporcional (P). • Control integral (I). • Control derivativo (D).

Además, los controladores pueden interactuar entre sí, lo que da por resultado la formación de las siguientes configuraciones:

• Control proporcional-integral (PI). • Control proporcional-derivativo (PD). • Control proporcional-integral-derivativo (PID).

9.1.1.1 Efecto en la adición de polos y ceros

Puesto que los controladores incorporan elementos adicionales al sistema a manera de polo(s) y/o cero(s), es importante establecer cuál es el efecto sobre el sistema a consecuencia de la adición de tales elementos.

9.1.1.2 Adición de polos

El incremento en el número de polos en un sistema ocasiona que el lugar geométrico de raíces se desplace hacia la derecha del eje j, lo que reduce la estabilidad relativa del sistema o, en algunos casos, lo hace inestable. Lo anterior se muestra en la figura 5.

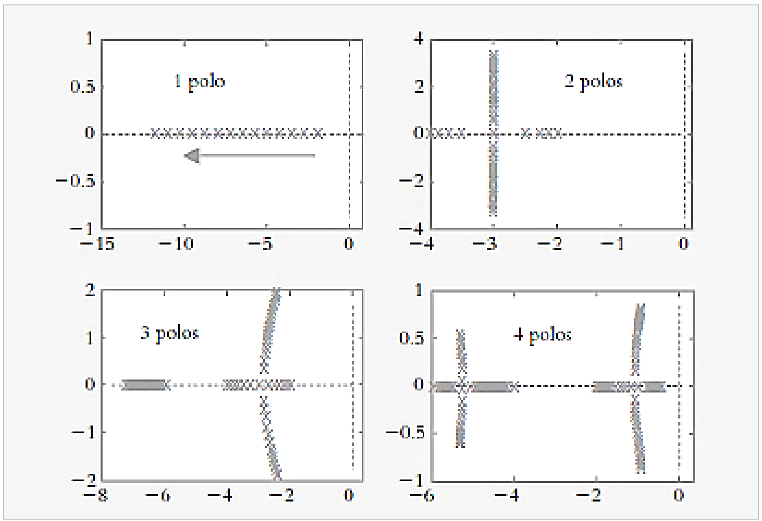


Fig 5 - Tendencia a la reducción de la estabilidad relativa del sistema como consecuencia de la adición de polos.

9.1.1.3 Adición de ceros

Incorporar ceros en un sistema produce que el lugar geométrico de raíces se desplace hacia el semiplano izquierdo, lo que hace estable o más estable al sistema. Lo anterior se muestra en la figura 6.

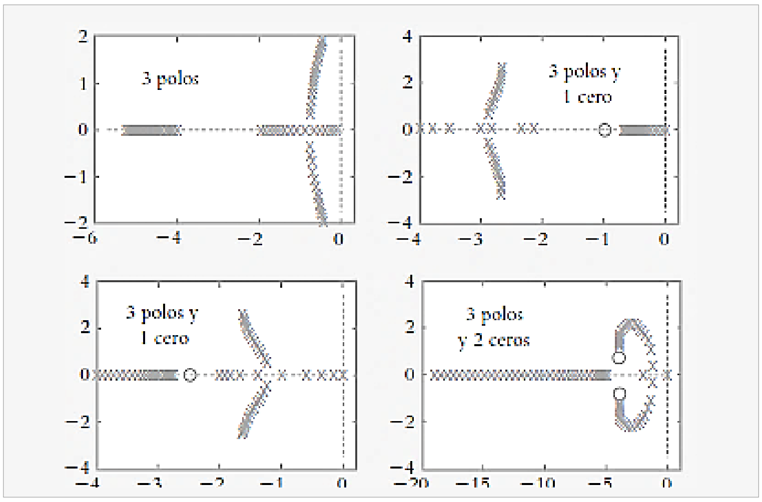


Fig 6 - Tendencia a incrementar la estabilidad relativa del sistema debido a la adición de ceros.

En términos generales, el diseño de los controladores se enfoca en la adición de ceros para mejorar la respuesta transitoria, así como la colocación de un polo en el origen para corregir el comportamiento de estado estable del sistema.

9.1.2 Acciones de control

Sea un sistema de lazo cerrado como el mostrado en la figura 7, donde el error E(s) es igual a la suma algebraica de R(s) − B(s). El diseño del controlador consiste en modificar las características de respuesta de los elementos que se encuentran en la trayectoria directa o en la de retroalimentación, de manera tal que la respuesta de la configuración en lazo cerrado satisfaga los requisitos de funcionamiento.

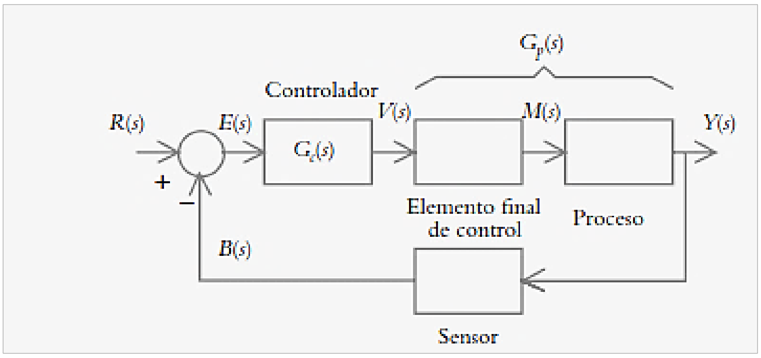


Fig 7 - Sistema de lazo cerrado al que se le agrega un controlador Gc(s) en la trayectoria directa.

9.1.2.1 Control proporcional: P

Se dice que un control es de tipo proporcional cuando la salida del controlador v(t) es proporcional al error e(t):

$$v(t) = K_p e(t) \qquad (8.1)$$

Su equivalente en el dominio s:

$$V(s) = K_p E(s) \therefore G_c(s) = \frac{V(s)}{E(s)} = K_p \qquad (8.2)$$

En general, para pequeñas variaciones de ganancia se logra un comportamiento aceptable en régimen transitorio, pero la respuesta de estado estable lleva implícita una magnitud elevada de error. Al tratar de corregir este problema, los incrementos de ganancia mejorarán las características de respuesta de estado estable en detrimento de la respuesta transitoria. Por lo anterior, aunque el control P es fácil de ajustar e implementar, no suele incorporarse a un sistema de control en forma aislada, sino más bien se acompaña de algún otro elemento, como se verá en la siguiente sección.

9.1.2.2 Control integral: I

Se dice que un control es de tipo integral cuando la salida del controlador v(t) es proporcional a la integral del error e(t):

$$v(t) = K_i \int e(t)dt \qquad (3)$$

Donde K_i es la ganancia del control integral. En cualquier tipo de controlador, la acción proporcional es la más importante, por lo que la constante Ki puede escribirse en términos de K_p:

$$K_i = \frac{K_p}{T_i} \qquad (4)$$

T_i es un factor de proporcionalidad ajustable que indica el tiempo de integración.

El equivalente en el dominio s de la ecuación (8.3) es:

$$V(s) = \frac{K_i}{s}E(s) \therefore G_c(s) = \frac{V(s)}{E(s)} = \frac{K_i}{s} = \frac{K_p}{T_i s} \qquad (5)$$

El control integral tiende a reducir o hacer nulo el error de estado estable, ya que agrega un polo en el origen aumentando el tipo del sistema; sin embargo, dicho comportamiento muestra una tendencia del controlador a sobre-corregir el error. Así, la respuesta del sistema es de forma muy oscilatoria o incluso inestable, debido a la reducción de estabilidad relativa del sistema ocasionada por la adición del polo en el origen por parte del controlador.

9.1.2.3 Control derivativo: D

Se dice que un control es de tipo derivativo cuando la salida del controlador v(t) es proporcional a la derivada del error e(t):

$$v(t) = K_d \frac{de(t)}{dt} \qquad (6)$$

Donde K_d es la ganancia del control derivativo. La constante Kd puede escribirse en términos de K_p:

$$K_d = K_p T_d \qquad (7)$$

T_d es un factor de proporcionalidad ajustable que indica el tiempo de derivación. El equivalente de la ecuación (6) en el dominio s es:

$$V(s) = K_d sE(s) \therefore G_c(s) = \frac{V(s)}{E(s)} = K_d s = K_p T_d s \qquad (8)$$

El significado de la derivada se relaciona con la velocidad de cambio de la variable dependiente, que en el caso del control derivativo indica que éste responde a la rapidez de cambio del error, lo que produce una corrección importante antes de que el error sea elevado. Además, la acción derivativa es anticipativa, esto es, la acción del controlador se adelanta frente a una tendencia de error. Para que el control derivativo llegue a ser de utilidad debe actuar junto con otro tipo de acción de control, ya que aislado, el control derivativo no responde a errores de estado estable.

9.1.3 Combinación de las acciones de control

Las acciones proporcional, integral y derivativa suelen combinarse entre sí para producir los siguientes tipos de controladores.

9.1.3.1 Control proporcional-integral: PI

Se dice que un control es de tipo proporcional-integral cuando la salida del controlador v(t) es proporcional al error e(t), sumado a una cantidad proporcional a la integral del error e(t):

$$v(t) = K_p e(t) + \frac{K_p}{T_i} \int e(t)dt \qquad (9)$$

Al expresar la ecuación anterior en el dominio s, se tiene:

$$V(s) = K_p E(s) + \frac{K_p}{T_i s} E(s) \therefore G_c(s) = \frac{V(s)}{E(s)} = K_p \left[1 + \frac{1}{T_i s}\right] = K_p + \frac{K_p}{T_i s} \qquad (10)$$

$$G_c(s) = \frac{V(s)}{E(s)} = K_p \left[\frac{s+1/T_i}{s}\right] = K_p \left[\frac{s+(K_i/K_p)}{s}\right] \qquad (11)$$

La ecuación (10) corresponde a un factor proporcional K_p que actúa junto con **un cero** ubicado en z = $-1/T_i$ (cuya posición es ajustable sobre el eje real a la izquierda del origen) y **un polo** en el origen. La representación en bloques de la ecuación (8.10) se muestra en la figura 8, mientras la figura 9 es la representación en el plano s de los elementos que forman el control PI.

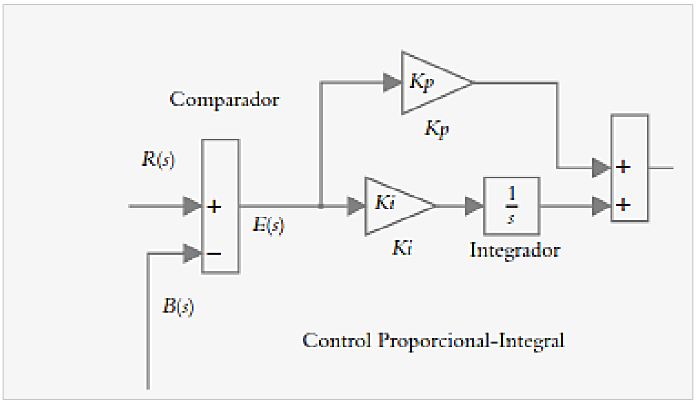


Fig 8 - Representación en bloques del control PI.

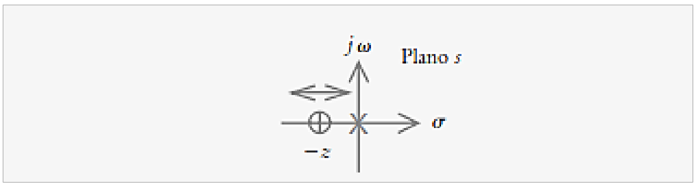


Fig 9 - Representación en el plano s del control PI, donde la posición del cero es ajustable.

9.1.3.2 Control proporcional-derivativo: PD

Se dice que un control es de tipo proporcional-derivativo cuando la salida del controlador v(t) es proporcional al error e(t), sumado a una cantidad proporcional a la derivada del error e(t):

$$v(t) = K_p e(t) + K_p T_d \frac{de(t)}{dt} \qquad (12)$$

Al expresar la ecuación anterior en el dominio s, se obtiene:

$$V(s) = K_p E(s) + K_p T_d s E(s) \therefore G_c(s) = \frac{V(s)}{E(s)} = K_P[1 + T_d s] \qquad (13)$$

$$G_c(s) = \frac{V(s)}{E(s)} = K_p T_d [s + 1/T_d] = K_p T_d [s + (K_p/K_d)] \qquad (14)$$

La ecuación (14) indica un factor proporcional $K_p T_d$, que actúa junto con **un cero** z = $-1/T_d$, cuya posición es ajustable en el eje real. El diagrama de la ecuación (8.14) se muestra en la figura 10, en tanto que la figura 11 es el diagrama de polos y ceros de los elementos que constituyen al control PD.

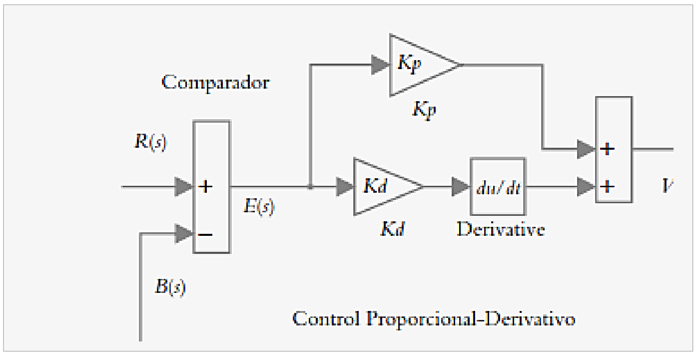


Figura 10 Representación en bloques del control PD.

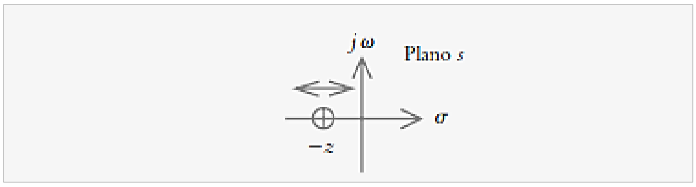


Fig 11 - Representación en el plano s del control PD, la posición del cero es ajustable.

9.1.3.3 Control proporcional-integral-derivativo: PID

Se dice que un control es de tipo proporcional-integral-derivativo cuando la salida del controlador v(t) es proporcional al error e(t), sumado a una cantidad proporcional a la integral del error e(t) más una cantidad proporcional a la derivada del error e(t):

$$v(t) = K_p e(t) + \frac{K_p}{T_i}\int e(t)dt + K_p T_d \frac{de(t)}{dt} \qquad (15)$$

por lo que en el dominio s le corresponde la expresión:

$$V(s) = K_p E(s) + \frac{K_p}{T_i s}E(s) + K_p T_d s E(s) \therefore G_c(s) = \frac{V(s)}{E(s)} = K_p\left[1 + \frac{1}{T_i s} + T_d s\right] \qquad (16)$$

$$G_c(s) = \frac{V(s)}{E(s)} = K_p\left[\frac{s+\frac{1}{T_i}+T_s s}{s}\right] = K_p T_d\left[\frac{s^2+\left(\frac{1}{T_d}\right)s+\frac{1}{T_i T_d}}{s}\right] \qquad (17)$$

$$G_c(s) = K_p + \frac{K_i}{s} + K_d s \qquad (18)$$

La ecuación (17) indica un factor proporcional $K_p T_d$ que actúa junto con un **par de ceros** (distintos, repetidos o complejos, cuya posición es ajustable en el plano s) y **un polo** en el origen. La representación en bloque de la ecuación (8.16) se muestra en la figura 12 y la figura 13 es la representación en el plano s del control PID.

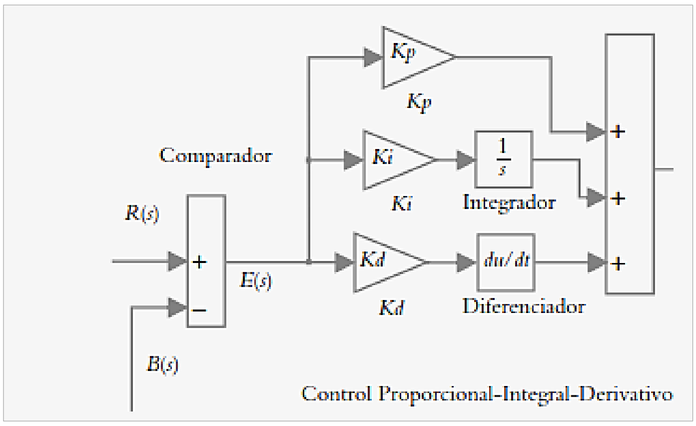


Fig 12 - Representación en bloques del control PID.

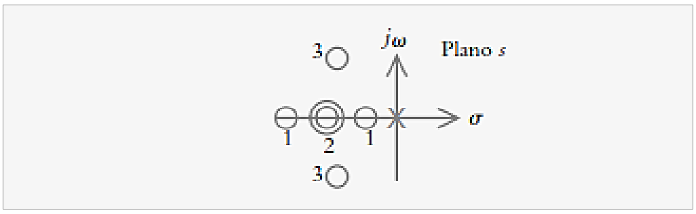


Figura 13 - Representación en el plano s del control PID; hay un polo en el origen. Los ceros pueden ser reales distintos (1), reales repetidos (2) o complejos (3).

9.1.3.4 Características

Como conclusión, se enumeran las principales características de los diferentes tipos de controladores: P, PI, PD y PID.

Control proporcional

- El tiempo de elevación experimenta una pequeña reducción.
- El máximo pico de sobreimpulso se incrementa.
- El amortiguamiento se reduce.
- El tiempo de asentamiento cambia en pequeña proporción.
- El error de estado estable disminuye con incrementos de ganancia.
- El tipo de sistema permanece igual.

Control proporcional-integral

- El amortiguamiento se reduce.
- El máximo pico de sobreimpulso se incrementa.
- Decrece el tiempo de elevación.
- Se mejoran los márgenes de ganancia y fase.
- El tipo de sistema se incrementa en una unidad.
- El error de estado estable mejora por el incremento del tipo de sistema.

Control proporcional-derivativo

- El amortiguamiento se incrementa.
- El máximo pico de sobreimpulso se reduce.
- El tiempo de elevación experimenta pequeños cambios.
- Se mejoran el margen de ganancia y el margen de fase.
- El error de estado estable presenta pequeños cambios.
- El tipo de sistema permanece igual.

Control proporcional-integral-derivativo

- Este tipo de controlador contiene las mejores características del control proporcional-derivativo y del control proporcional-integral. La tabla 17 es una referencia con respecto al tipo de controlador a utilizar en los diversos procesos industriales [7].

Tabla 17: Controladores a utilizar en los procesos industriales.

Tipo de controlador	Proceso por controlar
P	Control de nivel
PID	Control de temperatura
PI	Control de flujo
PI	Control de presión de líquidos

9.1.4 Criterios de sintonización

Una vez que se han definido las acciones de control y sus posibles combinaciones para producir los distintos tipos de controladores, se procede a estudiar algunos de los diferentes criterios para sintonizar.

9.1.4.1 Criterio de Ziegler-Nichols (Ganancia máxima)

Este criterio de ajuste se denomina método de sintonización en lazo cerrado, ya que el controlador permanece en la trayectoria directa como elemento activo, según se muestra en la figura 14.

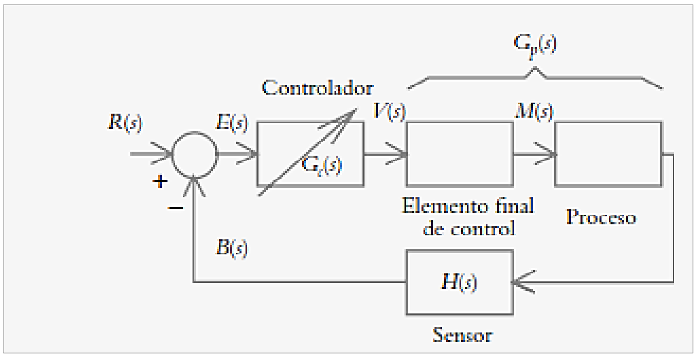


Fig 14 - La ganancia del controlador proporcional K_c se incrementa hasta llevar al sistema a un comportamiento libre oscilatorio.

Primero se incrementa la ganancia del control proporcional K_P hasta que la salida del sistema se comporte como una oscilación sostenida, lo que equivale a un comportamiento marginalmente estable. La forma de onda libre oscilatoria es de interés tanto en la ganancia con la que el sistema presenta dicha oscilación, denominada ganancia máxima K_u, como con el periodo de la oscilación, denominado periodo máximo P_u. En el caso de que el sistema original contenga un controlador con acción integral y derivativa, se procede a cancelar ambas acciones haciendo $T_i = \infty$ y $T_d = 0$. Una vez que se ha calculado T_i y T_d, el controlador queda sintonizado. Si el sistema es incapaz de alcanzar el estado de libre oscilación con incrementos de ganancia, el método de Ziegler-Nichols no se puede aplicar. Al sustituir s por j, en la ecuación característica de la función de transferencia de lazo cerrado T(s), es posible determinar Ku y la frecuencia u en la cual el LGR cruza con el eje j; el periodo Pu se obtiene mediante $P_u = 2/u$. Una vez que se han determinado la ganancia máxima K_u y el periodo máximo P_u, los valores de K_c, T_i y T_d pueden cuantificarse al aplicar la referencia que se muestra en la tabla 8.2 para sintonizar los diferentes tipos de controladores. En este punto, cabe mencionar que con el método de Ziegler-Nichols de la ganancia máxima no es posible ajustar al control proporcional-derivativo.

Tabla 18: Sintonización de controladores mediante el método de Ziegler-Nichols (método de la ganancia maxima).

Tipo de controlador	$G_c(s)$	K_p	T_i	T_d
P	K_p	$0.5\ K_u$		
PI	$K_p\left[1+\frac{1}{T_i\,s}\right]$	$0.45\ K_u$	$\frac{P_u}{1.2}$	
PID	$K_p\left[1+\frac{1}{T_i\,s}+T_d\,s\right]$	$0.6\ K_u$	$\frac{P_u}{2}$	$\frac{P_u}{8}$

9.1.4.2 Criterio de Cohen-Coon

Este criterio de ajuste se denomina método de sintonización en lazo abierto. En él se aplica una entrada escalón R(s) directamente hacia los elementos que forman la trayectoria directa sin incluir al controlador, como se observa en la figura 15.

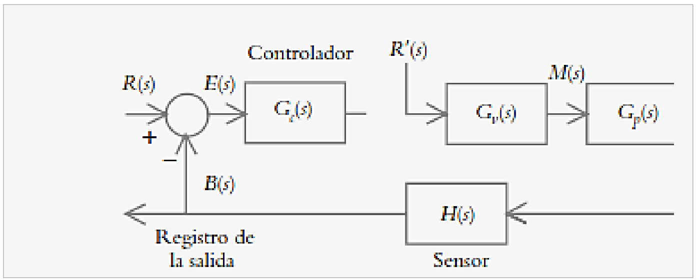
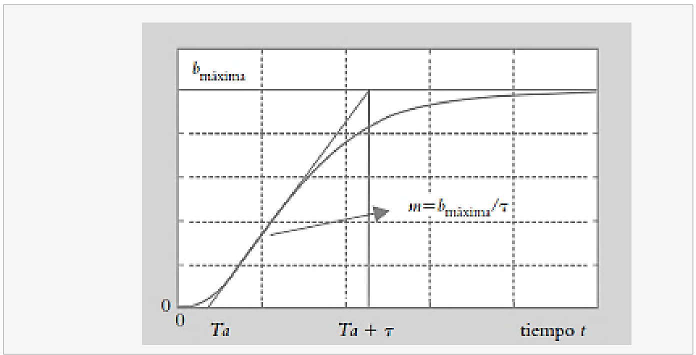


Fig 15 - Procedimiento para registrar la curva de reacción de un determinado sistema en lazo abierto.

La respuesta obtenida a la salida del sensor, denominada curva de reacción del sistema, es el punto de partida para la sintonización de los diversos tipos de controladores.



Una característica típica de la curva de reacción es que presenta una forma de S, debido a la contribución en tiempo compuesta por la suma del atraso de tiempo propio de un sistema de primer grado con constante de tiempo T junto con un atraso de tiempo puro Ta, según se muestra en la figura 16.

Fig 16 - Curva de reacción para determinar la constante de tiempo T del sistema G_p(s) y el atraso del tiempo T_a

$$G_p(s) = \frac{K_e{}^{-T_a s}}{\tau s+1} \quad (8.19) \qquad K = \frac{b_{máxima}}{r\prime(t)} \quad (20)$$

Donde:

K = Ganancia del proceso

τ = Constante de tiempo del sistema

T_d = Atraso de tiempo

A partir de la curva de reacción, se dibuja una recta tangente en el punto de inflexión de la curva, de tal manera que la intersección de la recta tangente con el eje de tiempo representa el atraso de tiempo Ta. La constante de tiempo T en relación con un sistema de primer grado se obtiene de:

$$\tau = \frac{b_{máxima}}{m} \qquad (21)$$

Conociendo los valores de ganancia K del proceso, atraso de tiempo Ta y constante de tiempo T (a partir de la pendiente de la tangente m en el punto de inflexión), la sintonización de los diversos controladores se lleva a cabo a partir de los valores mostrados en la tabla 8.4.

Tabla 19: Sintonización de controladores mediante el método de Cohen-Coo.

Tipo de controlador	Parámetros por sintonizar
P	$K_p = \frac{\tau}{KT_a}\left[1+\frac{T_a}{3\tau}\right]$
PI	$K_p = \frac{\tau}{KT_a}\left[0.9+\frac{T_a}{12\tau}\right]$ $T_i = T_a\frac{30+3T_a/\tau}{9+20T_a/\tau}$
PD	$K_p = \frac{\tau}{KT_a}\left[1.25+\frac{T_a}{6\tau}\right]$ $T_d = T_a\frac{6-2T_a/\tau}{22+3T_a/\tau}$
PID	$K_p = \frac{\tau}{KT_a}\left[1.3333+\frac{T_a}{4\tau}\right]$ $T_i = T_a\frac{32+6T_a/\tau}{13+8T_a/\tau}; \quad T_d = \frac{4T_a}{11+2T_a/\tau}$

9.1.4.3 **Criterio de Ziegler-Nichols** (Curva de reacción)

Como se vio en la sección anterior, el método de sintonización de Cohen-Coon se basa en la curva de reacción, la cual puede utilizarse como punto de partida para definir un segundo procedimiento propuesto por Ziegler-Nichols, denominado también de sintonización en lazo abierto. Este procedimiento se aplica al registro gráfico de la respuesta del proceso para una entrada escalón, donde es necesario determinar tanto el atraso de tiempo Ta como la pendiente m de la tangente en el punto de inflexión. Una vez cuantificados los parámetros mencionados, los coeficientes de los controladores se obtienen a partir de la tabla 20.

Tabla 20: Sintonización de controladores mediante el método de Ziegler-Nichols (Curva de reacción).

Tipo de controlador	K_p	T_i	T_d
P	$\frac{1}{T_a m}$		
PI	$\frac{0.9}{T_a m}$	$3.3\ T_a$	
PID	$\frac{1.2}{T_a m}$	$2\ T_a$	$0.5\ T_a$

9.1.4.4 Conclusión

Como conclusión a los modos de control, así como a los diferentes tipos de sintonización, se puede decir que:

- El modo integral ofrece una corrección que es proporcional a la integral del error, según se indicó por medio de la ecuación 3. Dicha acción tiene la ventaja de asegurar que para un sistema de tipo 0 se aplicará una acción de control suficiente para reducir a cero el error de estado estable; por otro lado, tal acción de control presentará un efecto desestabilizador como consecuencia de la adición de un polo en el origen.
- Con respecto al modo derivativo, se puede decir que ofrece una cierta característica predictiva o anticipativa, como lo muestra la ecuación 6, con lo que se genera una acción de control que es proporcional a la velocidad de cambio del error. Si bien la acción derivativa tiende a mejorar el comportamiento transitorio y le da más estabilidad al sistema, tiene la desventaja de producir elevados valores en la señal de control.
- La característica principal del control PID es que le da al sistema las mejores características, tanto del modo integral como del modo derivativo, según se indicó mediante la ecuación 15, sin presentar las desventajas de los modos individuales. En síntesis, el control PID es simplemente un controlador de segundo grado al que se agrega un integrador.
- Existen diversos criterios empíricos para determinar todos los parámetros de los diferentes tipos de controladores, entre los cuales se analizaron y aplicaron los métodos Ziegler-Nichols, tanto para lazo abierto (curva de reacción) como para lazo cerrado (llevar al sistema a presentar una oscilación libre) y el método de Cohen-Coon, también para lazo abierto analizando la curva de reacción.

Cabe destacar que existen diversos procedimientos adicionales a los presentados, entre los cuales sobresale el método de Chien-Hrones-Reswick. Independientemente del método elegido para la sintonización de los controladores, los parámetros obtenidos en primera instancia deben tomarse como un primer ajuste en el proceso del diseño.

9.2 SÍNTESIS DE HARDWARE

9.2.1 FPGA

Se define como un arreglo de bloques lógicos Configurables (CLB, Figura 14). Cada uno contiene una colección de elementos, incluida una tabla de consulta (LUT), un multiplexor (MUX) y un registro (FLIP FLOP); todos se pueden configurar para que funcionen según sea necesario (Figura 17).

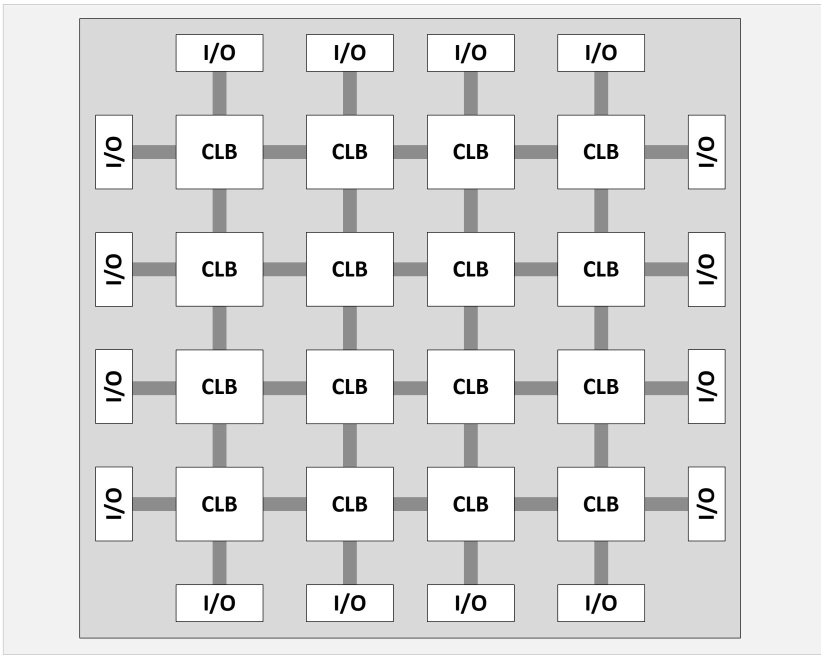


Fig 17 – Estructura interna de una FPGA.

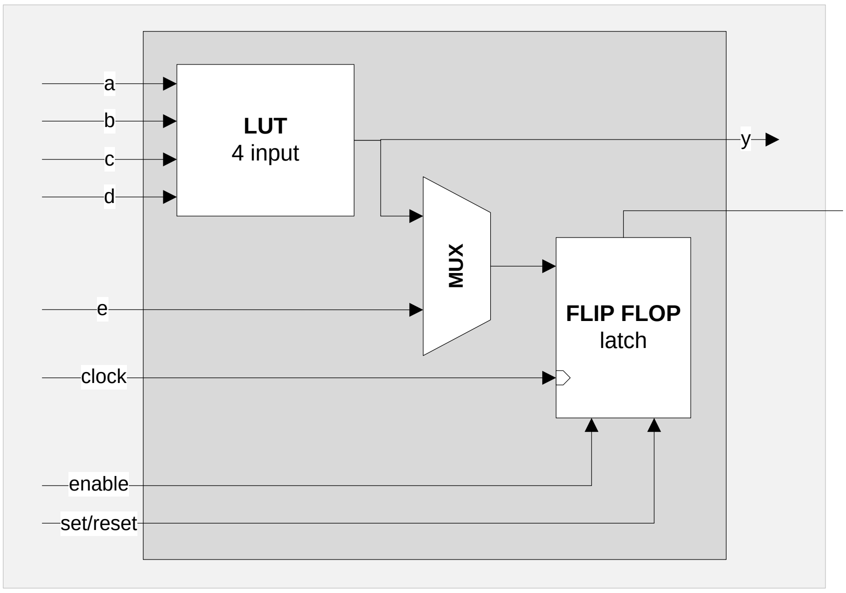


Fig 18 - Cada bloque lógico programable contiene una colección de elementos, que incluyen una tabla de consulta, un multiplexor y un registro; todos se pueden configurar para que funcionen según sea necesario.

Muchos FPGA utilizan LUT de 4 entradas que se pueden configurar para implementar cualquier función lógica de 4 entradas. Con el fin de admitir mejor las amplias rutas de datos utilizadas en algunas aplicaciones, algunos FPGA ofrecen LUT de 6, 7 o incluso 8 entradas. La salida de la LUT está conectada directamente a una de las salidas del bloque lógico y a una de las entradas del multiplexor. La otra entrada al multiplexor está directamente conectada a una entrada de bloque lógico (e). El multiplexor se puede configurar para que seleccione cualquiera de estas entradas.

La salida del multiplexor alimenta la entrada de registro. Cada registro se puede configurar para que funcione como un biestable activado por flanco o un pestillo sensible al nivel (dicho esto, no se recomienda el uso de lógica asincrónica en forma de pestillos dentro de los FPGA). El reloj de cada registro (o la activación) se puede configurar para estar alto activo o bajo activo; de manera similar, el nivel activo de las entradas de set/reset también se puede configurar.

Estos bloques lógicos pueden considerarse como "islas de lógica programable" que flotan en un "mar de interconexión programable". La interconexión se puede configurar para conectar cualquier salida de cualquier bloque lógico a cualquier entrada de otro bloque lógico. De manera similar, las entradas principales al FPGA se pueden conectar a las entradas de cualquier bloque lógico, y las salidas de cualquier bloque lógico se pueden usar para controlar las salidas principales del dispositivo.

Las entradas y salidas principales de uso general (GPIO) se presentan en bancos (grupos), donde cada banco se puede configurar para admitir un estándar de interfaz diferente, como LVCMOS (semiconductor complementario de óxido metálico de bajo voltaje), LVDS (tecnología de señal diferencial de bajo voltaje), LVTTL (lógica de transistor a transistor de bajo voltaje), HSTL (lógica de transceptor de alta velocidad) o SSTL (lógica terminada de la serie stub). Además, la impedancia de las entradas se puede configurar, al igual que la velocidad de respuesta de las salidas.

El siguiente paso en la estructura de los FPGA incluye elementos como bloques de SRAM, llamados RAM de bloque (BRAM), circuitos de sincronización de fase (PLL) y administradores de reloj. También se pueden agregar bloques de procesamiento de señal digital (DSP), también llamados “cortes de DSP” (DSP slices). Estos contienen multiplicadores configurables y un sumador configurable que les permite realizar operaciones de multiplicación-acumulación (MAC).

Otra característica común de los FPGA son los bloques SERDES de alta velocidad, que pueden soportar interfaces gigabit seriales. Es importante tener en cuenta que no todos los FPGA poseen todas las características mencionadas anteriormente. Diferentes FPGA ofrecen diferentes colecciones de características dirigidas a diferentes mercados y aplicaciones.

La estructura programable en un FPGA puede usarse para implementar cualquier función lógica o conjunto de funciones deseadas, hasta el núcleo de un procesador, o incluso múltiples núcleos. Si estos núcleos se implementan en una estructura programable, se denominan "núcleos flexibles". En comparación, algunos FPGA comúnmente denominados FPGA de SoC, contienen uno o más procesadores de "núcleo rígido", que se implementan directamente en el silicio. Estos núcleos rígidos del procesador pueden incluir unidades de coma flotante (FPU) y caché de nivel 1 y 2.

Del mismo modo, las funciones de la interfaz periférica como CAN, I^2C, SPI, UART y USB se pueden implementar como núcleos flexibles en la estructura programable, pero muchos FPGA los incluyen como núcleos rígidos en el silicio. Las comunicaciones entre los núcleos del procesador, las funciones de la interfaz y la estructura programable generalmente se realizan mediante buses de alta velocidad como AMBA y AXI.

Los primeros FPGA, que fueron presentados al mercado por Xilinx en 1985, contenían solo una matriz de bloques lógicos programables de 8 x 8 (sin bloques RAM, bloques DSP, etc.). En comparación, los FPGA de gama alta actuales pueden contener cientos de miles de bloques lógicos, miles de bloques DSP y megabits de RAM. En total, pueden contener miles de millones de transistores que equivalen a decenas de millones de puertas equivalentes.

9.2.1.1 Tecnologías

La forma en que se determinan las funciones de los bloques lógicos y el enrutamiento de la interconexión es mediante celdas de configuración, que pueden visualizarse como interruptores 0/1 (apagado/encendido). Estas celdas también se usan para configurar el estándar de interfaz de GPIO, la impedancia de entrada, la velocidad de respuesta de salida, etc. Dependiendo del FPGA, estas celdas de configuración pueden implementarse usando una de tres tecnologías:

- Antifusible**:** Estas celdas de configuración son programables una sola vez (OTP), lo que significa que una vez que el dispositivo ha sido programado no hay vuelta atrás. Estos dispositivos tienden a estar limitados a aplicaciones espaciales y de alta seguridad. Como se venden en pequeñas cantidades, su precio es alto y son una opción costosa de diseño.
- Flash**:** Al igual que las celdas de configuración basadas en antifusibles, las celdas basadas en flash son no volátiles. A diferencia de las celdas de antifusibles, las celdas de flash pueden reprogramarse según sea necesario. Las celdas de configuración flash son tolerantes a la radiación, lo que hace que estos dispositivos sean adecuados para aplicaciones espaciales (aunque con modificaciones en sus capas y paquetes de metalización superiores).
- SRAM**:** En este caso, los datos de configuración se almacenan en una memoria externa desde donde se cargan cada vez que se activa el FPGA (o según se recomiende en el caso de escenarios de configuración dinámica).

Las ventajas de los FPGA cuyas celdas de configuración están basadas en antifusibles o flash es que "se activan instantáneamente" y consumen poca energía. Una desventaja de estas tecnologías es que requieren pasos de procesamiento adicionales además del proceso de CMOS (semiconductor complementario de óxido metálico) subyacente utilizado para crear el resto del chip.

La ventaja de los FPGA cuyas celdas de configuración se basan en la tecnología SRAM es que se fabrican utilizando el mismo proceso de CMOS que el resto del chip, y ofrecen un mayor rendimiento porque generalmente están una o dos generaciones por delante de las tecnologías antifusible y flash. Las principales desventajas son que las celdas de configuración SRAM consumen más energía que sus contrapartes antifusible y flash (en el mismo nodo tecnológico), y son susceptibles a alteraciones por evento único (SEU) causadas por la radiación.

Durante mucho tiempo, este último punto implicó que los FPGA basados en SRAM se consideraban inadecuados para aplicaciones aeroespaciales y espaciales. Más recientemente, se han empleado estrategias especiales de mitigación, con el resultado de que los FPGA basados en SRAM se encuentran en sistemas como el Mars Curiosity Rover, junto con sus primos basados en flash.

9.2.1.2 Flexibilidad

Los FPGA se utilizan para muchas aplicaciones diversas. Son particularmente útiles para implementar funciones de interfaz inteligente, control de motores, aceleración algorítmica y computación de alto rendimiento (HPC), procesamiento de imágenes y videos, visión artificial, inteligencia artificial (AI), aprendizaje automático (ML), aprendizaje profundo (DL), radares, conformación de haces, estaciones base y comunicaciones.

Como ejemplo de aplicación, considere alguna tarea de computación intensiva, como realizar el procesamiento de señales requerido para implementar un sistema de radar, o la conformación de haces en una estación base de comunicaciones. Los procesadores convencionales con sus arquitecturas von Neumann o Harvard son adecuados para ciertas tareas, pero son menos adecuados para aquellas que requieren que la misma secuencia de operaciones se realice repetidamente. Esto se debe a que un solo núcleo del procesador que usa un solo hilo de ejecución solo puede ejecutar una instrucción a la vez (Figura 19a).

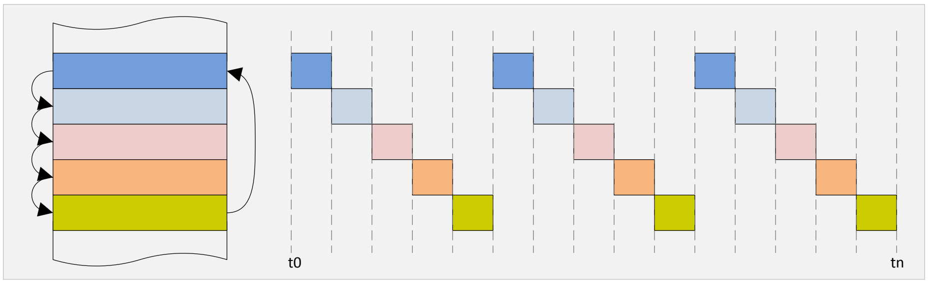


Fig 19a - Solo se ejecuta una instrucción a la vez.

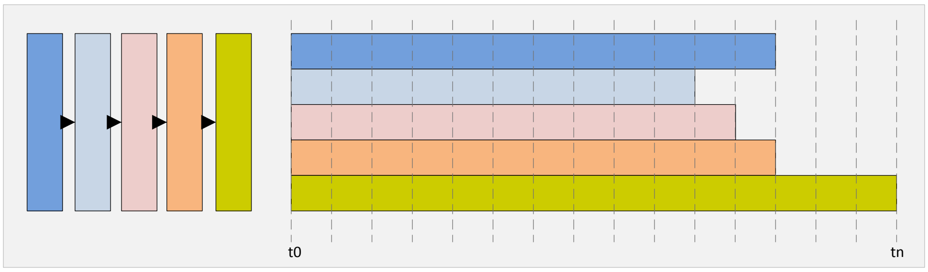


Fig 19b - Multiples procesos pueden ser ejecutadossimultaneamente en una FPGA.

A diferencia de los microprocesadores en los que solo se puede ejecutar una sola instrucción a la vez (secuencialmente), varios bloques funcionales en un FPGA pueden ejecutarse al mismo tiempo (simultáneamente); además, los FPGA pueden implementar algoritmos apropiados de forma paralela y masiva. (Figura 19b).

9.2.1.3 Fabricantes

Los dos principales fabricantes de dispositivos realmente de alta gama con la mayor capacidad y rendimiento son Intel (que adquirió Altera) y Xilinx.

Las ofertas de Intel y Xilinx abarcan los rangos desde FPGA de gama baja hasta FPGA de SoC de gama alta. Otro proveedor que se centra casi exclusivamente en los FPGAs es Lattice Semiconductor, que apunta a aplicaciones de gama baja y media. Por último, pero no menos importante, Microchip Technology (a través de sus adquisiciones de Actel, Atmel, y Microsemi) ahora tiene múltiples familias de FPGA pequeñas y medianas y miembros de gama baja de la categoría de FPGA de SoC.

Elegir el mejor dispositivo para la tarea en cuestión puede ser complicado porque existen muchas familias, y cada una de ellas ofrece diferentes recursos, rendimientos, capacidades y estilos de presentación. Aquí hay algunos ejemplos: Intel, Lattice Semiconductor y Xilinx.

9.2.2 VHDL

VHDL es un lenguaje diseñado para describir sistemas electrónicos digitales. Surgió del programa VHSIC (Very High Speed Integrated Circuits) impulsado por Departamento de Defensa del gobierno de los Estados Unidos de América. Durante el transcurso del programa se hizo notoria la falta de un lenguaje para describir circuitos electrónicos. De esta forma se desarrolló el lenguaje VHDL (VHSIC Hardware Design Language) que fue estandarizado inicialmente en el año 1987 por el Instituto de Ingenieros Eléctricos y Electrónicos (Institute of Electrical and Electronics Engeneers, IEEE) en los Estados Unidos de América.

Dicho estándar se conoce como IEEE Std 1076-1987. Posteriormente en el año 1993 se revisó este estándar originando el estándar conocido como IEEE Std 1076-1993 con algunos cambios respecto del anterior.

El lenguaje VHDL está diseñado para cubrir varias necesidades que surgen durante el proceso de diseño. Permite realizar una descripción funcional o de comportamiento del circuito, utilizando técnicas procedurales y familiares de programación. Permite describir la estructura del diseño y declarar las entidades y subentidades que lo forman especificando una jerarquía entre las mismas y como son sus interconexiones. Por último permite simular el diseño y sintetizarlo con herramientas de síntesis especiales, permitiendo su manufacturado y encapsulado como un microcircuito. Un aspecto que resulta importante destacar es que gracias a este tipo de lenguajes, se elimina la fase de prototipación de componentes lo cual reduce los costos de diseño y producción.

9.2.2.1 Comportamientos

Cuando comienza la fase de diseño de un sistema electrónico digital, resulta útil realizar una descripción funcional o de comportamiento del mismo. Una descripción VHDL usualmente está compuesta por un conjunto de entidades y subentidades, algunas de las cuales se pueden adquirir manufacturadas. De esta forma se elimina la necesidad de diseñarlas por completo. Sin embargo su comportamiento resulta necesario durante la fase de simulación del sistema. En este punto es conveniente realizar una descripción funcional del componente, que podrá ser utilizada como versión previa para simular el sistema final. La descripción funcional no hace referencia a la estructura interna del componente, que es visto como una caja negra, sino sólo se refiere a su funcionamiento.

Muchas veces la descripción funcional se divide a su vez en dos, dependiendo del nivel de abstracción y del modo en que se ejecutan las instrucciones. Estas dos formas se denominan *algorítmica* y de *flujo de datos*.

9.2.2.2 Estructuras

Un sistema electrónico digital puede ser descripto como un módulo con puertos de entrada y de salida como muestra la figura 20. Los valores de los puertos de salida son una función de los valores de los puertos de entrada y del estado del sistema.

Fig 20 - Módulo con puertos de entrada y salida.

Una manera de describir un componente o sistema digital es a través de los componentes que lo forman. Cada componente es la instancia de alguna entidad. Sus respectivos puertos están interconectados por señales (figura 21). Este tipo de descripción se denomina estructural o de estructura.

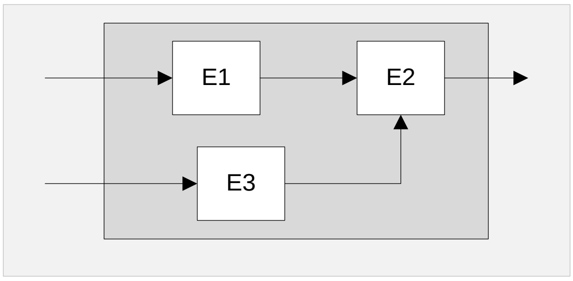


Fig 21 - Ejemplo de descripción Estructural.

9.2.2.3 Ejemplo

El lenguaje VHDL presenta tres estilos de descripción que dependen del nivel de abstracción. El menos abstracto es el nivel estructural mientras que el más abstracto y lejana a una posible implementación física es el algorítmico. A modo de ejemplo, para ilustrar cada uno de los estilos, se describirá un multiplexor de dos bits. Un posible esquema del multiplexor se muestra en la figura 22.

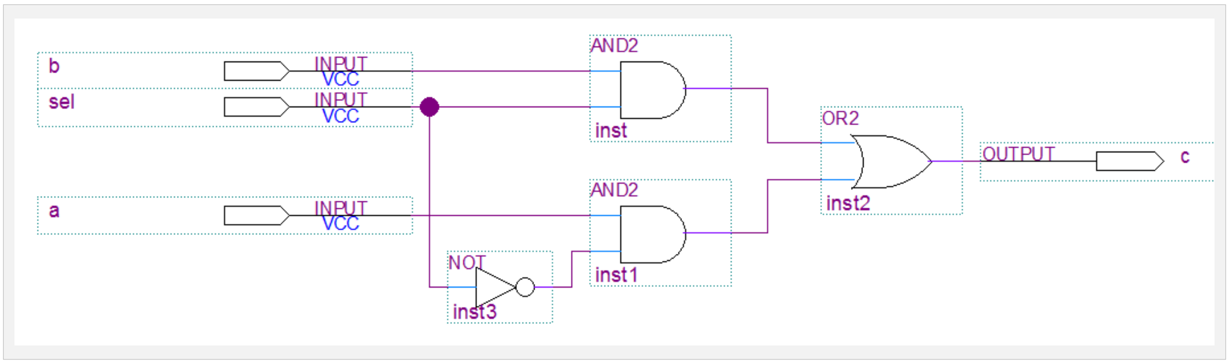


Fig 22 – Comportamiento estructural para el Multiplexor.

La descripción de un circuito en VHDL comienza con la definición de la entidad. Lo primero que se definen son sus entradas y salidas. Se denomina entidad porque en VHDL la palabra clave que se utiliza para su definición es ENTITY.

```
entity Mux is
port
(
  a, b, sel      :       in  bit;
  c              :       out bit
);
end Mux;
```

La entidad definida con el nombre Mux tiene tres entradas y una salida. Las entradas y la salida son de tipo bit. El tipo bit es un tipo predefinido del lenguaje que contiene los valores ‘0’ y ‘1’.

Para este ejemplo la entidad en VHDL es única, ya que se describe el mismo esquema del multiplexor, con la finalidad de ilustrar los distintos estilos, sin embargo esto no es necesario.

La entidad anteriormente declarada tiene asociados tres cuerpos descriptos con diferentes estilos. Estos cuerpos se denominan arquitecturas y se definen en VHDL mediante la palabra clave ARCHITECTURE. A continuación se muestran éstas arquitecturas. Cada arquitectura está descripta con un estilo distinto, comenzando por la descripción algorítmica, pasando por la descripción de flujo de datos para culminar con la descripción estructural.

En la descripción algorítmica no se hace referencia a los operadores lógicos presentes en el esquema del multiplexor, sino sólo al funcionamiento del mismo. Con una secuencia sencilla de instrucciones se puede describir el circuito. La forma en que se ejecutan las descripciones algorítmicas es secuencial, como se verá más adelante, por eso se le da la denominación de algorítmica para diferenciarla de las descripciones concurrentes. La ejecución secuencial se hace posible mediante la inclusión de la palabra clave PROCESS. Sin embargo ésta no es la única construcción secuencial que posee VHDL, otras construcciones de naturaleza secuencial son las funciones y los procedimientos, estructuras de alto nivel que también pueden emplearse para realizar descripciones funcionales.

```
architecture Algoritmica OF Mux is
begin
 process(a, b, sel)
 begin
        if(sel = '1') then
                c <= b;
        else
                c <= a;
        end if;
```

Una descripción algorítmica de más alto nivel puede realizarse utilizando un procedimiento. Este tipo de descripción es la más lejana a una implementación física. Una descripción usando procesos puede ser de la siguiente forma:

```
procedure Mux(a, b, sel : in bit; c : out bit) is
begin
 if(sel = '1') then
        c := b;
 else
        c := a;
 end if;
```

A veces resulta interesante describir el circuito de forma que esté más cercano a una posible implementación física. El lenguaje VHDL posee una forma de describir circuitos que permite la paralelización de instrucciones, y a su vez es más parecida a una descripción estructural pero que se clasifica como de comportamiento. Todas las sentencias que se encuentran entre el BEGIN ... END del cuerpo de la arquitectura se ejecutan en forma paralela y mientras dure la simulación. Esta es una característica importante frente a los lenguajes de programación convencionales, en los que las sentencias a iterar deben encerrarse dentro de un bucle. VHDL soporta la ejecución concurrente por omisión en el cuerpo de una arquitectura. El estilo que se presenta a continuación se denomina de flujo de datos.

```
architecture FlujoDeDatos OF Mux is
begin
 C <= (a and (not sel)) or (b and sel);
```

Una descripción estructural es la más cercana a la implementación física de un circuito y hace referencia a los componentes que intervienen y a sus interconexiones. Los componentes que se utilizan son declarados en la parte declarativa de la arquitectura e instanciados en el cuerpo de la misma las veces que sea necesario.

```
architecture Estructural OF Mux is

component not is
port(
  a : in bit;
  b : out bit
);
end component;

component and is
port(
 a : in bit;
 b : in bit;
 c : out bit
);
end component;

component or is
port(
 a : in bit;
 b : in bit;
 c : out bit
);
end component;

signal x, y, z : bit;
begin
```

Una descripción en VHDL puede ser mixta, es decir puede comprender varios estilos. Por ejemplo, en una descripción estructural, los elementos más bajos de la jerarquía de diseño pueden definirse en forma de flujo de datos, debido a que el lenguaje incluye un conjunto de operadores predefinidos, como es el caso de los operadores lógicos, que pueden emplearse en la descripción [9].

9.3 DISEÑO E IMPLEMENTACIÓN

9.3.1 Planta

9.3.1.1 Robot Manipulador

Para el caso de estudio se toma como planta a las articulaciones del robot manipulador. Recordemos que el mismo posee 5 grados de libertad, los cuales están denotados por sus similitudes al ser humano, estos son: cintura, hombro, codo, muñeca elevación y muñeca giro. La figura 23, muestra una vista general del manipulador junto a una imagen real del mismo.

El robot presenta una estructura mecánica antropomórfica o articulada, de configuraciones cinemáticas del tipo RRR; cintura Rotacional, hombro Rotacional y codo Rotacional. Posee como actuadores, motores de corriente continua acoplados a cajas reductoras. Sensores internos del tipo potenciómetros resistivos, forman su sistema sensorial y como elemento terminal incorpora una pinza mecánica, sin herramienta, para la manipulación de pequeños objetos.

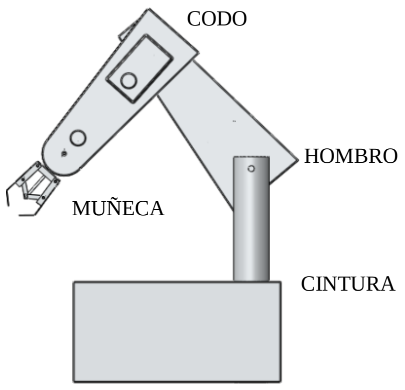


Fig. 23 Vista general de manipulador.

Por cada articulación o grados de libertad, se realiza el modelado matemático individual, tomando como planta al conjunto Motor-Caja reductora-Potenciómetro. La figura 24 ilustra lo mencionado.

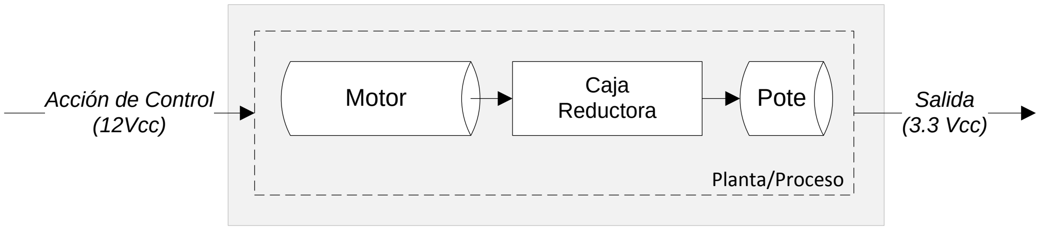


Fig 24 - Lazo de control PID.

Gracias al potenciómetro, se logra traducir de manera indirecta, un desplazamiento rotacional en un nivel de voltaje continuo. Más adelante éste último, pasará a llamarse señal de realimentación.

9.3.1.2 Modelado de Matemático

Para el modelado matemático se emplea el método empírico de la función de transferencia FT, en donde se aplica una señal escalón al sistema “Motor-Caja reductora-Potenciómetro” y se estudia el comportamiento de la respuesta. Se plantean las ecuaciones de cada variable en el dominio del tiempo y luego se migra al dominio de Laplace, a fin de obtener la función de transferencia de la planta.

La señal de entrada, corresponde a un escalón de amplitud igual al 63% de la tensión nominal del motor, cuya transformada de Laplace es:

$$U(s) = \frac{V}{s} = \frac{7{,}56V}{s} \quad ; \quad u(t) = V = 7{,}56V \qquad (22)$$

En la salida se obtiene una recta de pendiente m, tal como se aprecia en la Figura 25.

$$m = \frac{\Delta y}{\Delta x} = \frac{6{,}2 - 4}{15 - 10} = 0{,}44$$

Cuya transformada de Laplace es:

$$Y(s) = \frac{m}{s^2} = \frac{0{,}44}{s^2} \quad ; \quad y = mt = 0{,}44t \qquad (23)$$

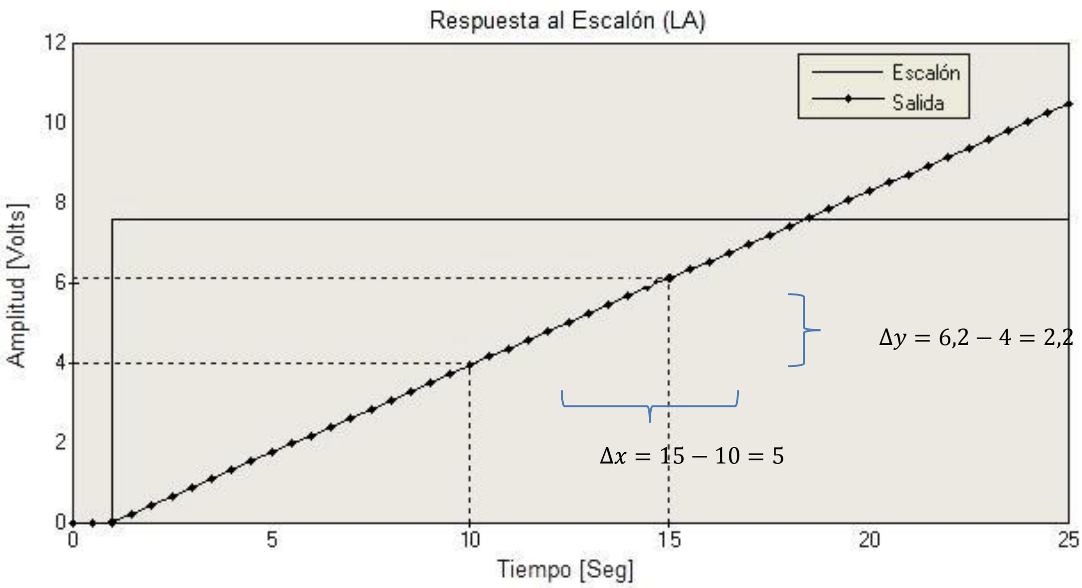


Fig 25 - Respuesta al escalón.

Por lo tanto, la FT de la planta está dada por la siguiente expresión:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{m/V}{s} = \frac{0{,}0582}{s} \qquad (24)$$

La ecuación 24, describe el modelo dinámico de la planta, que nos permite conocer de manera aproximada la naturaleza de la misma. Si bien este modelo está alejado del modelo físico real, su sencillez lo hace útil como punto de partida para el análisis.

Vemos que la planta a lazo abierto LA, presenta un integrador, es decir, un polo en el origen, que lo hace un sistema de tipo 1. Este sistema ante una entrada parabólica, tiene un error en estado estacionario e_{ss} igual a infinito, es decir, que este tipo de sistemas es incapaz de seguir una excitación de tales características. Para una entrada rampa, el error se vuelve finito, donde la velocidad de salida es igual a la velocidad de entrada, con un error de posición. Este error es proporcional a la velocidad de entrada y es inversamente proporcional a la ganancia K. Por último, si al sistema se lo excita con una señal escalón, el error tiende a cero.

Corroboramos lo enunciado, evaluado la planta ante una excitación tipo rampa: $R(s) = \frac{1}{s^2}$

$$e_{ss} = \lim_{t\to\infty} e(t) = \lim_{s\to 0} s.E(s) \qquad (25)$$

El error en lazo cerrado, está dado por $E(s) = \frac{R(s)}{1+G(s)}$, por lo tanto:

$$e_{ss} = \lim_{s\to 0} s.\frac{R(s)}{1+G(s)} = \lim_{s\to 0} \frac{s}{1+G(s)}.\frac{1}{s^2} = \lim_{s\to 0} \frac{1}{s+sG(s)} = \frac{1}{K_v} \qquad (26)$$

Siendo $K_v = \lim_{s\to 0} sG(s) = \lim_{s\to 0}\left(s.\frac{0{,}0{,}0582}{s}\right) = 0{,}0582$ Constante de error de velocidad.

Tabla 21: Error en estado estacionario en función de la ganancia K.

Sistema tipo	**Excitación escalón** (entrada de posición) $r(t) = 1$	**Excitación rampa** (entrada de velocidad) $r(t) = t$	**Excitación parabólica** (entrada de aceleración) $r(t) = \frac{1}{2}t^2$
0	$\frac{1}{1+K}$	∞	∞
1	0	$\frac{1}{K}$	∞
2	0	0	$\frac{1}{K}$

Se observa, que en principio, el error en estado estacionario se puede reducir simplemente aumentando la ganancia, siempre y cuando de no afectar las especificaciones de la respuesta transitoria.

La siguiente tabla, reúne las características de los sistemas tipo 0, 1 y 2 ante excitaciones del tipo escalón, rampa y parábola.

Podemos mejorar el comportamiento en estado estacionario, si se aumenta el tipo del sistema (de tipo 1 a tipo 2) agregando un integrador en la trayectoria directa. De este modo, logramos que el sistema sea capaz de seguir los tres tipos de excitación mencionados. Sin embargo, esto introduce un problema adicional de estabilidad [1].

9.3.2 Controlador

9.3.2.1 PID Digital

Como se mencionó anteriormente, un controlador PID compara el valor real de la salida de planta con una entrada de referencia, determina la desviación y produce una señal de control que reducirá la desviación a cero o a un valor pequeño. En la figura 26 se representa esquemáticamente un lazo de control PID.

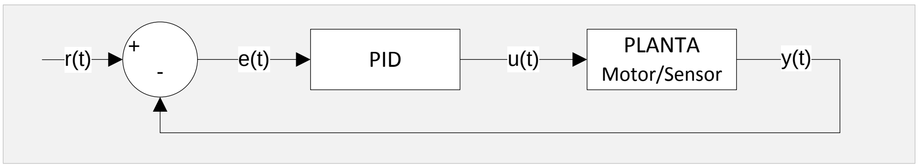


Fig 26 - Lazo de control PID.

r(t) señal de referencia, y(t) señal de salida, e(t) señal de error y u(t) señal de control.

La ecuación diferencial que rige el comportamiento dinámico entrada-salida de un PID continuo, es:

$$u(t) = K_p e(t) + \frac{K_p}{T_i}\int_0^t e(\tau)d\tau + K_p T_d \frac{de(t)}{dt} \quad (27)$$

Donde: K_p es la ganancia proporcional, T_i es la constante de tiempo integral y T_d es la constante de tiempo derivativa. Estos tres parámetros interactúan uno con el otro y su ajuste para obtener el mejor control, puede ser complicado.

Existen distintas posibilidades a la hora de realizar de manera práctica un controlador PID. En este trabajo se utiliza un diseño paralelo, debido a la facilidad que presenta a la hora de describir el algoritmo, la figura 27 muestra un diagrama del mismo.

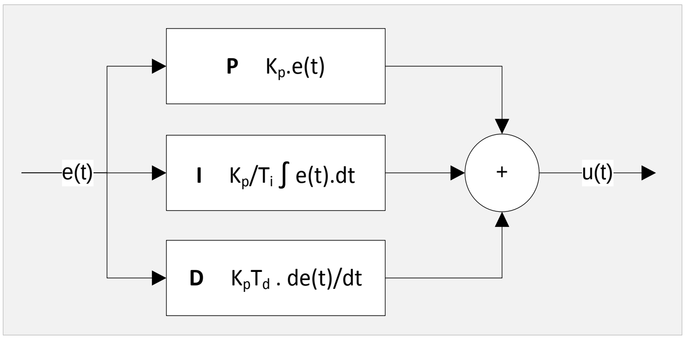


Fig 27 - Diseño tipo paralelo del controlador PID.

Para control digital, las señales continuas deben discretizarse. Si el tiempo de muestreo T es suficientemente pequeño, se puede hacer una discretización directa de la ecuación (27) para obtener la ecuación en diferencias. De este modo la derivada primera se transforma en diferencia de primer orden y la integral en sumatoria (Kuchen y Carelli, 1996). La ecuación (28) responde a un comportamiento discreto aproximado para el controlador PID.

$$u[n] = K_p e[n] + T\frac{K_p}{T_i}\sum_{n=0}^{N} e[n-1] + \frac{K_p T_d}{T}(e[n] - e[n-1]) \qquad (28)$$

Siendo K_p ganancia proporcional, $T\frac{K_p}{T_i} = K_i$ ganancia integral y $\frac{K_p T_d}{T} = K_d$ ganancia derivativa.

La figura siguiente ilustra la representación de la Ec. 28 en pseudocódigo, que simboliza un primer borrador del algoritmo de control, antes de su implementación en VHDL.

```
eT = rT - yT;               // Cálculo error
iT = Ki*eT + iT0;           // Cálculo integral
dT = Kd*(eT - eT0);         // Cálculo derivativo
uT = iT + Kp*eT + dT;       // Cálculo uT

iT0 = iT;                   // Traspaso de valores
eT0 = eT;
```

Se observa que las operaciones aritméticas involucradas son tres: suma, resta y multiplicación. De aquí en adelante el documento centra su atención en la implementación del algoritmo, utilizando diferentes representaciones numéricas, a fin de determinar dos aspectos de interés:

- Por un lado, el impacto que tienen las diferentes representaciones numéricas en el cálculo del algoritmo PID y por ende, en la respuesta en el dominio del tiempo, analizando su comportamiento transitorio y permanente.
- Por otro lado, la cantidad de recursos de hardware (LUT) necesarios para su implementación en FPGA, a fin de encontrar la representación numérica que arroje resultados que se ubiquen dentro de las especificaciones de diseño y que además, consuman un bajo número de recursos, permitiendo así, utilizar FPGA económicas.

9.3.2.2 Representaciones Numéricas

Las representaciones numéricas a utilizar pueden ser: enteros o reales, y los últimos en una de sus dos representaciones computacionales: punto fijo o punto flotante. Si se utiliza VHDL para el diseño [1], las operaciones con enteros se realizan de forma sencilla, gracias a librerías existentes. En cambio sí se opera con reales la complejidad del diseño crece. Las herramientas de diseño como ISE de Xilinx o Quartus II de Altera presentan módulos configurables, IP-Cores, que permiten operar en punto flotante, tanto de simple (32 bits) como de doble precisión (64 bits) de acuerdo al estándar IEEE-754 [2]. Sin embargo, no presentan esta opción para punto fijo, ni tampoco hay una norma que regule su formato.

Para aquellas aplicaciones en donde no se requiere de un rango dinámico amplio (en tal caso es más adecuado emplear punto flotante), se puede optar por trabajar con punto fijo. Para lo cual, existen varias opciones, entre ellas: hacer uso de los tipos de datos ufixed y sfixed de la versión 2008 de VHDL [3], o usar HDL-Coder de Matlab [6].

9.3.2.3 Números Enteros

Una palabra de N bits puede representar un total de 2^N valores distintos, cubriendo un rango de $0\ to\ 2^{N-1} - 1$, con una precisión de uno en uno.

Tabla 22: Enteros sin signo

Rango	$0\ to\ 2^{N-1} - 1$
Precisión	1

Si se desea trabajar con números enteros con signo es necesario agregar un bit extra, a la izquierda de la palabra binaria, para representar con un 0 cuando es positivo y con un 1 cuando es negativo. Hay diferentes representaciones como Signo Mantisa, Complemento a 1 (C1) y Complemento a

2 (C2). De estas opciones, la más utilizada es C2, ya que presenta como ventaja una única representación para el cero y mayor simplicidad en la sustracción.

El complemento a 2 de un número binario se obtiene tomando el complemento a 1, y sumándole 1 al bit menos significativo.

Tabla 23: Enteros con signo (Complemento a 2)

Rango	$-2^{N-1}\ to\ 2^{N-1}-1$
Precisión	1

9.3.2.3.1 PID usando Enteros (integer)

A la hora de operar con números enteros en VHDL, los sintetizadores cuentan con paquetes que permiten trabajar de manera rápida y sencilla como es el caso de ***std_logic_1164*** y ***numeric_std,*** ambos pertenecientes a la librería ***ieee***.

Para operar con números enteros y realizar las operaciones aritméticas básicas, se requiere incluir la librería ieee y sus paquetes dentro del proyecto, según a como se observa a continuación.

```
library ieee;
use ieee.std_logic_1164.all;
```

En la ***entidad,*** se declaran cuatro puertos de entrada; clk, reset, rt, yT para las señales de reloj, reset, referencia y realimentación y un puerto de salida; uT como señal actuante.

```
entity PIDint is
port
(
 clk, reset     :       in  std_logic;
 rT, yT         :       in  integer range 0 to 255;
 pid_out        :       out integer range 0 to 255
);
```

En la ***arquitectura*** se detalla las instrucciones que describen al algoritmo de control por medio de un proceso de comportamiento secuencial y de estilo algorítmico. La imagen siguiente muestra la descripción en VHDL del algoritmo usando enteros.

```
architecture behav1 of PIDint is
        constant Kp      : integer := 5;
        constant Ki      : integer := 1;
        constant Kd      : integer := 1;
begin
calculo : process(clk, reset)
  variable eT, iT, dT, iT0      : integer;
  variable eT0, uT              : integer;
begin
  if reset = '1' then
         iT0 := 0;
         eT0 := 0;
  elsif rising_edge(clk)then
         eT := rT - yT;                      -- Cálculo del error
         iT := Ki*eT + iT0;                  -- Cálculo integral
         dT := Kd*(eT - eT0);                -- Cálculo derivativo
         uT := iT + Kp*eT + dT;              -- Cálculo uT
         pid_out <= uT;
         iT0 := iT;                          -- Traspaso de valores
         eT0 := eT;
  end if;
end process calculo;
end behav1;
```

9.3.2.4 Números Reales

Existen limitaciones cuando se desea representar números reales en una arquitectura binaria, ya que en principio, se requieren infinitos bits para representar los infinitos valores que pertenecen al conjunto de los reales. Como los recursos son limitados y escasos, el problema se traduce en seleccionar un número finito de valores para su representación computacional.

Existen dos representaciones numéricas, bien conocidas, que permiten trabajar con números enteros y decimales; una es Punto Fijo, donde se emplean tres campos para la representación: signo, parte entera y parte decimal. Otra forma es en notación científica o punto flotante, donde un número se expresa también con tres campos: signo, mantisa y exponente. Generalmente esta última notación se emplea cuando el número a representar es muy grande o muy pequeño, por lo que se requeriría de muchos dígitos si se representa en punto fijo, lo cual puede resultar a errores de cálculo, de representación, etc. Más adelante se presentan las ventajas y desventajas de cada una de ellas.

9.3.2.5 Punto Fijo

En esencia, un número en punto fijo es un entero que se encuentra escalado por un cierto factor. Por ejemplo, el numero 2,125 puede ser representado como 2125/100, donde el factor de escalamiento es 100. La representación en Punto fijo es de gran utilidad para representar números fraccionales en base 2 dentro de los sistemas digitales, especialmente cuando el procesador en uso no cuenta con una FPU (Unidad de Punto Flotante) o cuando el uso de este proporciona mejor rendimiento o precisión para alguna aplicación dada. Gran parte de los microprocesadores embebidos de bajo costo, no cuentan con una FPU, tal es el caso de los sistemas desarrollados dentro de los FPGA en los que generalmente no se cuenta con dichas estructuras. La característica principal de los números en punto fijo es que cuentan con una cantidad fija de bits tanto a la izquierda como a la derecha del punto fraccional, de ahí se deriva su nombre.

En esta representación se utilizan m bits para representar en C2 la parte entera de un número y n bits para representar en C2 la parte fraccionaria. Son necesarios m + n + 1 bits para almacenar un número en punto fijo. El bit extra es usado para indicar, en la posición más significativa, el signo del número. El rango representable es de $-2^m\ to\ 2^m - 2^{-n}$ con una precisión de 2^{-n}.

Tabla 24: Reales (Punto Fijo)

Rango	$-2^m \ to \ 2^m - 2^{-n}$
Precisión	2^{-n}

Si n = 0, se observa que los números enteros son un caso particular de la representación en punto fijo debido a que la precisión es igual a 1, esto es: $precisión = 2^{-n} = 2^0 = 1$.

Por lo general, cuando se trabaja en punto fijo se suele indicar el número en formato Qm.n. Por ejemplo, el número 5.6875 en formato Q3.2 es:

Q3.4 = 0|101|1011 ($2^2 + 2^0 + 2^{-1} + 2^{-3}2^{-4} = 5.6875$)

9.3.2.5.1

9.3.2.5.2 David Bishop Library

VHDL 2008 define un nuevo paquete genérico que permite la aritmética en punto fijo desarrollado por el ingeniero especialista en ASIC y FPGA David Bishop. El paquete se compone de 2 elementos:

1. **FIXED_FLOAT_TYPES**: define los parámetros de configuración para operar con el paquete de punto fijo.
   - ***fixed_overflow_style_type***: indica qué hacer cuando el resultado de una operación es de masiado grande para ser representado.
     - fixed_saturate: devuelve el máximo valor representable.
     - fixed_wrap: desecha los bits más significativos sobrantes (puede haber cambio de signo).
   - ***fixed_round_style_type***: indica qué hacer cuando el resultado de una operación requiere más bits de los disponibles para representarlo.
     - fixed_truncate: desecha los bits menos significativos sobrantes
     - fixed_round: aplica un algoritmo de redondeo.
   - ***round_type***: indica el algoritmo de redondeo a aplicar cuando sea necesario.
     - round_nearest: redondea al número representable más cercano.
     - round_inf: redondea hacia el número representable más cercano al infinito positivo.
     - round_neginf: redondea hacia el número representable más cercano al infinito negativo.
     - round_zero: redondea el número representable más cercano al cero.
2. **FIXED_PKG**: define los tipos de datos y las operaciones aritméticas para trabajar en punto fijo.

<u>Características, tipos de datos y ejemplos</u>:

El paquete define 2 tipos de datos vectoriales que, basados en std_logic, son interpretables como reales codificados en punto fijo.

- Asume que el punto se halla siempre entre la posición 0 y -1 del vector.
- El subrango positivo del vector codifica los bits enteros del número.
- El subrango negativo del vector codifica los bits decimales del número.
- El tipo ufixed especifica un número real sin signo representado en binario puro.

```
type ufixed is array (natural range <>) of std logic;
```

El tipo sfixed especifica un número real con signo representado en C2

```
type sfixed is array (natural range <>) of std_logic;
```

Ejemplo:

```
ufixed (3 downto -3) => 1100|101 = +12.625
sfixed (3 downto -3) => 1100|101 = -3.375
```

A diferencia de los operadores de numeric_std, este paquete está diseñado para que nunca haya posibilidad de **overflow**. El rango del resultado queda definido según el rango de los operandos.

Tabla 25: Rangos de las operaciones

Operación	Rango del resultado
A + B, A – B	Max(A'left, B'left) + 1 **downto** Min(A'right, B'right)
A * B	A'left + B'left + 1 **downto** A'right + B'right
A rem B, A mod B (con signo)	Min(A'left, B'left) **downto** Min(A'right, B'right)
A / B (con signo)	A'left - B'right + 1 **downto** A'right - B'left
reciprocal(A) (con signo)	-A'right **downto** -A'left - 1
abs(A), -A	A'left + 1 **downto** A’right
A / B (sin signo)	A'left - B'right **downto** A'right - B'left – 1
A mod B (sin signo)	B'left **downto** Min(A'right, B’right)

Las funciones: **ufixed_high** / **ufixed_low** / **sfixed_high** / **sfixed_low** permiten definir el rango del resultado sin memorizar las reglas de dimensionado. Ejemplo:

```
signal a : ufixed (5 downto -3);
signal b : ufixed (7 downto -9);
signal c : ufixed(ufixed_high(5,-3,'/',7,-9) downto ufixed_low(5,-3,'/',7,-9));
signal d : ufixed(ufixed_high(a,'*',b)       downto ufixed_low(a,'*',b );
...
c <= a / b;
d <= a * b;
```

La función **resize** se usa cuando se desea que el resultado tenga un rango determinado. Ejemplo:

```
signal a    :   ufixed (5 downto -3);
signal b    :   ufixed (7 downto -9);
signal c, d :   ufixed (7 downto -3);
...
c <= resize(a + b, 7, -3 ); -- o bien c <= resize(a + b, c'high, c'low);
d <= resize(a - b, d );
```

Para pasar de un tipo a otro se usan funciones de **conversión** tales como:

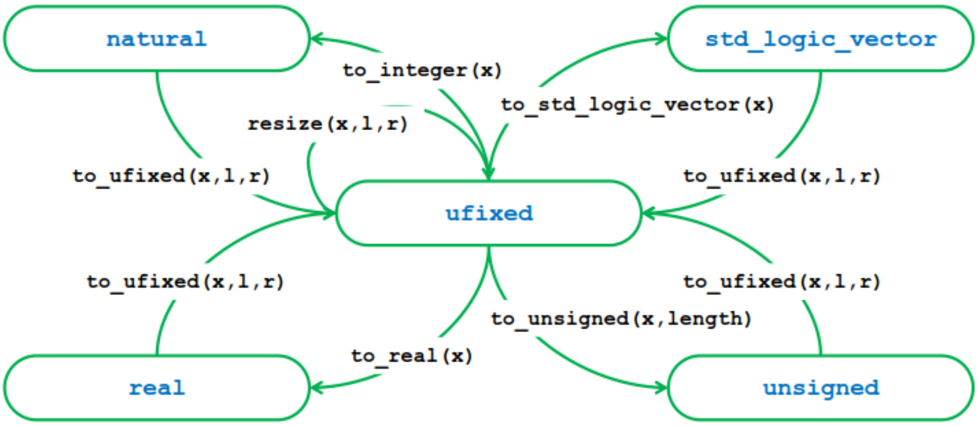


Fig 28 – Conversión entre tipos de datos.

9.3.2.5.2.1 **PID usando Punto Fijo** (sfixed)

Para operar en punto fijo usando el paquete sfixed, es necesario incluir las bibliotecas ieee y ieee_proposed junto a sus paquetes. Esto se muestra a continuación:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;  -- necesaria para usar conv_std_logic_vector
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

library ieee_proposed;
use ieee_proposed.fixed_pkg.all;
use ieee_proposed.fixed_float_types.all;
```

Al igual que para el caso de la implementación del controlador usando enteros, la ***entidad*** está declarada por cuatro puertos de entrada; clk, reset, rt, yT para las señales de reloj, reset, referencia y realimentación y un puerto de salida; pid_out (uT para el caso anterior).

```
entity PIDsfixed is
port
(
 clk, reset     :      in  std_logic;
 rT, yT         :      in  std_logic_vector (7 downto 0);
 pid_out        :      out std_logic_vector (7 downto 0)
);
```

En la ***arquitectura*** se detalla las instrucciones que describen al algoritmo de control por medio de un proceso de comportamiento secuencial y de estilo algorítmico. Esto se observa en el siguiente código.

```
architecture behav1 of PIDint is
       signal int : integer range 0 to 4095 := 0;
begin

calculo : process(clk, reset)
 variable eT, iT, dT, iT0       : sfixed(16 downto -16);
 variable eT0, w                : sfixed(16 downto -16);
 variable uT, Kp, Ki, Kd        : sfixed(16 downto -16);

begin
 if reset = '1' then

        iT0 := 0;
        eT0 := 0;

 elsif rising_edge(clk)then

        Kp := to_sfixed (0.7, w);
        Ki := to_sfixed (0.005, w);
        Kd := to_sfixed (0.01, w);

        eT := resize(to_sfixed(rT, w) -
               to_sfixed(yT, w),  w);                -- Cálculo error

        iT := resize(Ki*eT + iT0, w);                -- Cálculo integral
        dT := resize(Kd*(eT - eT0), w);              -- Cálculo derivativo
        uT := resize(iT + Kp*eT + dT, w);            -- Cálculo uT

        int     <= to_integer(uT);                   -- De sfixed a integer
        pid_out <= conv_std_logic_vector(int, 8);    -- De integer a slv

        iT0 := iT;                                   -- Traspaso de valores
        eT0 := eT;

 end if;
end process calculo;

end behav1;
```

9.3.2.5.3 HDL-Coder

HDL Coder, es un toolbox de Matlab que genera código VHDL y Verilog sintetizable. La herramienta convierte funciones de Matlab, modelos de Simulink y gráficos de Stateflow en HDL equivalentes. Además convierte automáticamente operaciones basadas en punto flotante en operaciones en punto fijo. En el presente, HDL Coder soporta más de 300 bloques de Simulink. Haciendo uso de los bloques, es posible diseñar sistemas complejos en un corto periodo de tiempo.

En el contexto del proyecto actual, se utilizan tres herramientas (*PID Controller, Fixed Point tool* y *HDL Coder*) para obtener código VHDL de un controlador PID. A continuación se explica su uso.

9.3.2.5.3.1 **PID usando Punto Fijo** (HDL-Coder)

Como punto de partida, se construye en Simulink un lazo de control a partir de los siguientes bloques.

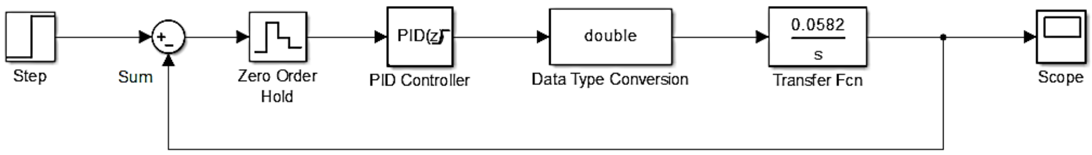


Fig 29 – Diagrama en bloques en Simulink.

Donde:

Step genera un escalón entre dos niveles definidos en un espacio de tiempo específico.

Sum es la implementación del bloque suma. Este bloque realiza las operaciones de adición o sustracción de sus entradas, pudiendo sumar o sustraer entradas escalares, vectoriales o matriciales.

Zero Order Hold retiene su entrada durante un período de tiempo específico.

Data Type Conversion Convierte la señal de entrada al tipo de datos especificado.

Transfer Fcn implementa una función de transferencia con la entrada U(s) y la salida Y(s). Recordar que para el caso de estudio, la función de transferencia está dada por la ecuación (29):

$$G(s) = \frac{Y(s)}{U(s)} = \frac{m/V}{s} = \frac{0{,}0582}{s} \qquad (29)$$

Scope representa gráficamente la entrada conectada a este bloque con respecto al tiempo de simulación.

PID Controller implementa un controlador PID cuya salida es una suma ponderada, integrada y derivada de la señal de entrada. Los pesos son los parámetros de ganancia proporcional, integral y derivativa. Algunas de las configuraciones presentes del bloque son:

- Tipo de controlador (PID, PI, PD, solo P o solo I).
- Forma de controlador (paralela o ideal).
- Dominio del tiempo (continuo o discreto).
- Límites de saturación de salida y mecanismo anti-windup integrado.
- Tipo de datos a operar.

En la pestaña *Data Types* se especifica el tipo de datos con el que se va a trabajar. Al igual que con punto fijo usando la librería de David Bishop, la palabra queda determinada por 32 bits, siendo 1 bit de signo, 15 bits para la parte entera y 16 bits para la parte decimal. Esto se define como fixdt(1,32,16). Ver imagen 30.

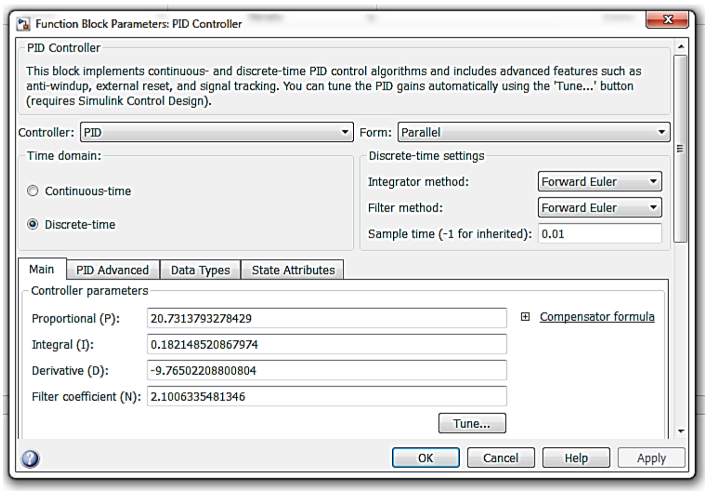


Fig 30 – Configuración de PID Controller.

Además, el bloque incorpora un módulo de sintonización automático (*Tune*) que permite sintonizar al controlador a partir de la información de todo el lazo de control y también del tipo de dato especificado.

Una vez configurado el bloque PID Controller y realizada la sintonización se procede a utilizar la herramienta *HDL Coder* para generar el código VHDL. Para lo cual, solo se ocupa el bloque del controlador y se le adiciona un puerto de entrada y otro de salida, ambos de 32 bits. La imagen siguiente se ilustra la cadena de bloques (Figura 32).

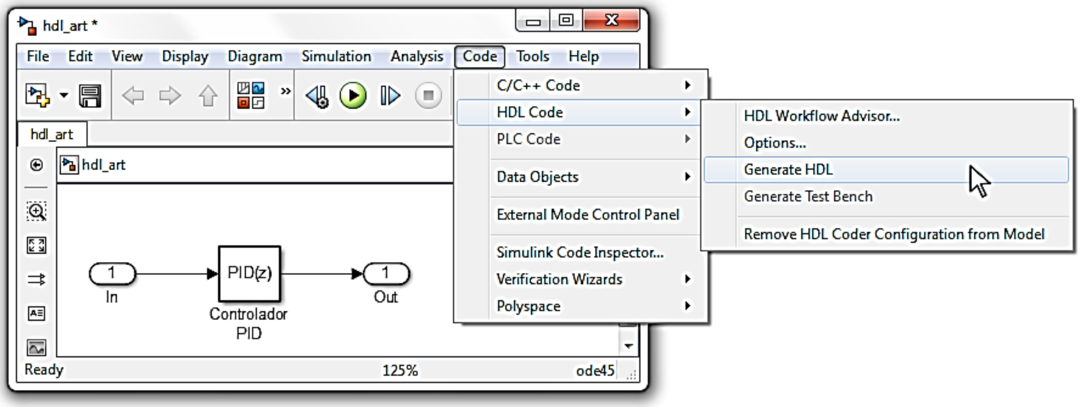


Fig 31 – Generación de código VHDL con HDL-Coder.

El código generado por HDL-Coder se adjunta a continuación.

```
library ieee;
use ieee.std_logic_1164.ALL;
use ieee.numeric std.all;
```

```
entity PIDhdlcoder is
port
(
       Clk, reset, enb  :  in  std_logic;
       u                :  in  std_logic_vector(31 downto 0);  -- sfix18_En6
       y                :  out std_logic_vector(31 downto 0)   -- sfix18_En6
);
end PIDhdlcoder;
```

```
architecture behav1 of PIDhdl-coder is
  signal u_signed                      : signed(31 downto 0);  -- sfix32_En16
  signal Proportional_Gain_mul_temp    : signed(63 downto 0);  -- sfix64_En32
  signal Proportional_Gain_out1        : signed(31 downto 0);  -- sfix32_En16
  signal Integral_Gain_mul_temp        : signed(63 downto 0);  -- sfix64_En32
  signal Integral_Gain_out1            : signed(31 downto 0);  -- sfix32_En16
  signal Integrator_indtc              : signed(31 downto 0);  -- sfix32
  signal gain_mul_temp                 : signed(63 downto 0);  -- sfix64_En37
  signal Integrator_u_gain             : signed(31 downto 0);  -- sfix32
  signal Integrator_u_dtc              : signed(31 downto 0);  -- sfix32_En16
  signal Integrator_x_reg              : signed(31 downto 0);  -- sfix32_En16
  signal Integrator_u_add              : signed(31 downto 0);  -- sfix32_En16
  signal Derivative_Gain_mul_temp      : signed(63 downto 0);  -- sfix64_En32
  signal Derivative_Gain_out1          : signed(31 downto 0);  -- sfix32_En16
  signal Filter_Coefficient_out1       : signed(31 downto 0);  -- sfix32_En16
  signal Filter_indtc                  : signed(31 downto 0);  -- sfix32
  signal gain_mul_temp_1               : signed(63 downto 0);  -- sfix64_En37
  signal Filter_u_gain                 : signed(31 downto 0);  -- sfix32
  signal Filter_u_dtc                  : signed(31 downto 0);  -- sfix32_En16
  signal Filter_x_reg                  : signed(31 downto 0);  -- sfix32_En16
  signal Filter_u_add                  : signed(31 downto 0);  -- sfix32_En16
  signal SumD_out1                     : signed(31 downto 0);  -- sfix32_En16
  signal Filter_Coefficient_mul_temp   : signed(63 downto 0);  -- sfix64_En32
  signal Sum_add_temp                  : signed(31 downto 0);  -- sfix32_En16
  signal Sum_out1                      : signed(31 downto 0);  -- sfix32_En16

 begin
  u_signed <= signed(u);

  Proportional_Gain_mul_temp <= 12366435 * u_signed;
  Proportional_Gain_out1 <= Proportional_Gain_mul_temp(47 downto 16) + ('0' & (
 Proportional_Gain_mul_temp(15)      OR      Proportional_Gain_mul_temp(14)      OR
 Proportional_Gain_mul_temp(13) OR Proportional_Gain_mul_temp(12) OR
 Proportional_Gain_mul_temp(11)      OR      Proportional_Gain_mul_temp(10)      OR
 Proportional_Gain_mul_temp(9)        OR    Proportional_Gain_mul_temp(8)        OR
 Proportional_Gain_mul_temp(7)        OR    Proportional_Gain_mul_temp(6)        OR
 Proportional_Gain_mul_temp(5)        OR    Proportional_Gain_mul_temp(4)        OR
 Proportional_Gain_mul_temp(3)        OR    Proportional_Gain_mul_temp(2)        OR
 Proportional_Gain_mul_temp(1)   OR Proportional_Gain_mul_temp(0)));

  Integral_Gain_mul_temp <= 904688 * u_signed;
  Integral_Gain_out1 <= Integral_Gain_mul_temp(47 downto 16) + ('0' & (
 Integral_Gain_mul_temp(15)         OR         Integral_Gain_mul_temp(14)         OR
 Integral_Gain_mul_temp(13)         OR         Integral_Gain_mul_temp(12)         OR
 Integral_Gain_mul_temp(11)         OR         Integral_Gain_mul_temp(10)         OR
 Integral_Gain_mul_temp(9)           OR      Integral_Gain_mul_temp(8)           OR
 Integral_Gain_mul_temp(7)           OR      Integral_Gain_mul_temp(6)           OR
 Integral_Gain_mul_temp(5)           OR      Integral_Gain_mul_temp(4)           OR
 Integral_Gain_mul_temp(3)           OR      Integral_Gain_mul_temp(2)           OR
 Integral_Gain_mul_temp(1)   OR Integral_Gain_mul_temp(0)));

  Integrator_indtc <= Integral_Gain_out1;
```

```
    gain_mul_temp <= 1374389535 * Integrator_indtc;
    Integrator_u_gain <= (resize(gain_mul_temp(63 downto 37), 32)) + ('0' & (
         gain_mul_temp(36) OR gain_mul_temp(35) OR gain_mul_temp(34) OR
         gain_mul_temp(33) OR gain_mul_temp(32) OR gain_mul_temp(31) OR
         gain_mul_temp(30) OR gain_mul_temp(29) OR gain_mul_temp(28) OR
         gain_mul_temp(27) OR gain_mul_temp(26) OR gain_mul_temp(25) OR
         gain_mul_temp(24) OR gain_mul_temp(23) OR gain_mul_temp(22) OR
         gain_mul_temp(21) OR gain_mul_temp(20) OR gain_mul_temp(19) OR
         gain_mul_temp(18) OR gain_mul_temp(17) OR gain_mul_temp(16) OR
         gain_mul_temp(15) OR gain_mul_temp(14) OR gain_mul_temp(13) OR
         gain_mul_temp(12) OR gain_mul_temp(11) OR gain_mul_temp(10) OR
         gain_mul_temp(9)  OR gain_mul_temp(8)  OR gain_mul_temp(7)  OR
         gain_mul_temp(6)  OR gain_mul_temp(5)  OR gain_mul_temp(4)  OR
         gain_mul_temp(3)  OR gain_mul_temp(2)  OR gain_mul_temp(1)  OR
         gain_mul_temp(0)));

    Integrator_u_dtc <= Integrator_u_gain;
    Integrator_u_add <= Integrator_x_reg + Integrator_u_dtc;

  integrator_reg_process : process (clk, reset)
    begin
      if reset = '1' then
        integrator_x_reg <= to_signed(0, 32);
      elsif clk'event AND clk = '1' then
        if enb = '1' then
          integrator_x_reg <= integrator_u_add;
        end if;
      end if;
    end process integrator_reg_process;



    Derivative_Gain_mul_temp <= (-834708) * u_signed;
    Derivative_Gain_out1 <= Derivative_Gain_mul_temp(47 downto 16) + ('0' & (
   Derivative_Gain_mul_temp(15)       OR       Derivative_Gain_mul_temp(14)       OR
   Derivative_Gain_mul_temp(13)       OR       Derivative_Gain_mul_temp(12)       OR
   Derivative_Gain_mul_temp(11)       OR       Derivative_Gain_mul_temp(10)       OR
   Derivative_Gain_mul_temp(9)         OR     Derivative_Gain_mul_temp(8)         OR
   Derivative_Gain_mul_temp(7)         OR     Derivative_Gain_mul_temp(6)         OR
   Derivative_Gain_mul_temp(5)         OR     Derivative_Gain_mul_temp(4)         OR
   Derivative_Gain_mul_temp(3)         OR     Derivative_Gain_mul_temp(2)         OR
   Derivative_Gain_mul_temp(1)  OR Derivative_Gain_mul_temp(0)));

    Filter_indtc <= Filter_Coefficient_out1;

    gain_mul_temp_1 <= 1374389535 * Filter_indtc;
    Filter_u_gain <= (resize(gain_mul_temp_1(63 downto 37), 32)) + ('0' & (
         gain_mul_temp_1(36) OR gain_mul_temp_1(35) OR gain_mul_temp_1(34) OR
   gain_mul_temp_1(33)    OR    gain_mul_temp_1(32)    OR    gain_mul_temp_1(31)    OR
   gain_mul_temp_1(30) OR gain_mul_temp_1(29) OR gain_mul_temp_1(28) OR
         gain_mul_temp_1(27) OR gain_mul_temp_1(26) OR gain_mul_temp_1(25) OR
         gain_mul_temp_1(24) OR gain_mul_temp_1(23) OR gain_mul_temp_1(22) OR
         gain_mul_temp_1(21) OR gain_mul_temp_1(20) OR gain_mul_temp_1(19) OR
         gain_mul_temp_1(18) OR gain_mul_temp_1(17) OR gain_mul_temp_1(16) OR
         gain_mul_temp_1(15) OR gain_mul_temp_1(14) OR gain_mul_temp_1(13) OR
         gain_mul_temp_1(12) OR gain_mul_temp_1(11) OR gain_mul_temp_1(10) OR
         gain_mul_temp_1(9)  OR gain_mul_temp_1(8)  OR gain_mul_temp_1(7)  OR
         gain_mul_temp_1(6)  OR gain_mul_temp_1(5)  OR gain_mul_temp_1(4)  OR
         gain_mul_temp_1(3)  OR gain_mul_temp_1(2)  OR gain_mul_temp_1(1)  OR
         gain_mul_temp_1(0)));
```

```
Filter_u_dtc <= Filter_u_gain;
Filter_u_add <= Filter_x_reg + Filter_u_dtc;

filter_reg_process : process (clk, reset)
begin
  if reset = '1' then
    filter_x_reg <= to_signed(0, 32);
  elsif clk'event AND clk = '1' then
    if enb = '1' then
      filter_x_reg <= filter_u_add;
    end if;
  end if;
end process filter_reg_process;

SumD_out1 <= Derivative_Gain_out1 - Filter_x_reg;

Filter_Coefficient_mul_temp <= 732380 * SumD_out1;
Filter_Coefficient_out1 <= Filter_Coefficient_mul_temp(47 downto 16) + ('0' & (
Filter_Coefficient_mul_temp(15)     OR     Filter_Coefficient_mul_temp(14)     OR
Filter_Coefficient_mul_temp(13)     OR     Filter_Coefficient_mul_temp(12)     OR
Filter_Coefficient_mul_temp(11)     OR     Filter_Coefficient_mul_temp(10)     OR
Filter_Coefficient_mul_temp(9)      OR   Filter_Coefficient_mul_temp(8)        OR
Filter_Coefficient_mul_temp(7)      OR   Filter_Coefficient_mul_temp(6)        OR
Filter_Coefficient_mul_temp(5)      OR   Filter_Coefficient_mul_temp(4)        OR
Filter_Coefficient_mul_temp(3)      OR   Filter_Coefficient_mul_temp(2)        OR
Filter_Coefficient_mul_temp(1)  OR Filter_Coefficient_mul_temp(0)));
```

Se observa que el código generado automáticamente por la herramienta HDL-Coder es poco amigable y de difícil seguimiento. Si bien al usar la herramienta se gana tiempo en diseño e implementación pero requerirá de un mayor esfuerzo si se desea modificar alguna parte del mismo.

9.3.2.6 Punto Flotante

La representación en punto flotante es un sistema de representación numérica en la que una cadena de dígitos representa un número racional. El termino flotante se refiere al hecho de que el punto decimal puede ser colocado en cualquier posición dentro de los dígitos significativos en el número. Esta posición es indicada de manera separa internamente en la representación, por lo que este tipo de esquema puede ser concebida como la versión digital de la notación científica. A través de los años se han usado diferentes formatos para este sistema de representación, sin embargo, hace algunos años se adoptó la representación especificada en el estándar IEEE 754.

La ventaja de la representación en punto flotante sobre punto fijo es que soporta un rango mucho más grande de valores. Por ejemplo, un número en punto fijo con siete dígitos decimales y el punto decimal posicionado después del cuarto digito, podría representar los números:

123,4567 9,8765 21,345

y así sucesivamente, mientras que usando la representación en punto flotante con los mismos siete dígitos podría representar adicionalmente los números:

1,234567 765432,1 12345670000

y así sucesivamente. La desventaja es que esta representación requiere un poco más de espacio de almacenamiento para indicar la posición del punto fraccional, por lo que al ser almacenados en el mismo espacio, los números en punto flotante adquieren su máximo rango al costo de un poco menos de precisión.

En 1985, el IEEE (Institute of Electrical and Electronics Engineers) publicó la norma IEEE 754. Una especificación relativa a la precisión y formato de los números de punto flotante. Incluye una lista de las operaciones que pueden realizarse con dichos números, entre las que se encuentran las cuatro básicas: suma, resta, multiplicación, división. Así como el resto, la raíz cuadrada y diversas conversiones. También incluye el tratamiento de circunstancias excepcionales, como manejo de números infinitos y valores no numéricos. El estándar se desarrolló para facilitar la portabilidad de los programas de un procesador a otro y para alentar el desarrollo de programas numéricos sofisticados. Una versión actual es el estándar IEEE 754- 2008 que fue publicado en Agosto del 2008.

El estándar define lo siguiente:

- Formatos Aritméticos: conjunto de datos binarios o decimales en punto flotante, los cuales consisten de números finitos (ceros y números subnormales), infinitos y valores especiales de NaNs (Not a Number).
- Formatos intercambiables: los números punto flotante pueden ser intercambiados y compactados en formato binario y decimal de diferentes tamaños de bits.
- Algoritmos de redondeo: varios métodos pueden ser usados por los números durante las operaciones.
- Operaciones: permiten operaciones con formatos aritméticos.
- Excepciones: indica si hay condiciones de excepción como overflow, underflow, división por cero, etc.

El nuevo estándar IEEE 754-2008 específica formatos para representar números binarios en punto flotante (BFP) y decimal en punto flotante (DFP). La diferencia entre ambas proviene no solo de la base utilizada sino también de la normalización de las mantisas, el BFP se normaliza con el punto decimal a la derecha del bit más significativo (MSB) mientras el DFP solo se representa por decimales. El DFP es representado en formatos de 32, 64 y 128 bits. Un número DFP es representado por un signo (S), un exponente (E) que es normalizada a un exponente desplazado (biased exponent) q, y una mantisa (M) con p dígitos de precisión. El valor de un número DFP D es representado como:

$$D = -1^S . M . 10^q \qquad q = E - bias$$

El número D es expresado en formato DFP como se muestra en el esquema:

SIGNO	EXPONENTE	MANTISA
1 bit	8 bit	23 bit

Simple Precisión 32 bits

SIGNO	EXPONENTE	MANTISA
1 bit	11 bit	52 bit

Doble Precisión 64 bits

Figura 32 - Representación de un número en el estándar IEEE 754-2008-

9.3.2.6.1 David Bishop Library

VHDL 2008 define un nuevo paquete genérico que permite la aritmética en punto flotante desarrollado por el ingeniero especialista en ASIC y FPGA David Bishop. El paquete se compone de 2 elementos:

9.3.2.6.1.1 PID usando Punto Flotante (float)

```
library ieee;
use ieee.std_logic_1164.all;

library ieee_proposed;
use ieee_proposed.fixed_float_types.all;
use ieee_proposed.float_pkg.all;
```

```
entity PIDsfixed is
port
(
 clk, reset     :      in  std_logic;
 rT, yT         :      in  std_logic_vector (7 downto 0);
 pid_out        :      out std_logic_vector (7 downto 0)
);
```

```
architecture behav1 of PIDint is
       signal int : integer range 0 to 4095 := 0;
begin

calculo : process(clk, reset)
 variable eT, iT, dT, iT0       : float(8 downto -23);
 variable eT0, w                : float(8 downto -23);
 variable uT, Kp, Ki, Kd        : float(8 downto -23);

begin
 if reset = '1' then
        iT0 := 0;
        eT0 := 0;
 elsif rising_edge(clk)then

        Kp := to_float (0.7, w);
        Ki := to_float (0.005, w);
        Kd := to_float (0.01, w);

        eT := to_float(rT, w) -
              to_float(yT, w);                     -- Cálculo error

        iT := Ki*eT + iT0;                         -- Cálculo integral
        dT := Kd*(eT - eT0);                       -- Cálculo derivativo
        uT := iT + Kp*eT + dT;                     -- Cálculo uT

        int     <= to_integer(uT);                 -- De sfixed a integer
        pid_out <= conv_std_logic_vector(int, 8);  -- De integer a slv

        iT0 := iT;                                 -- Traspaso de valores
        eT0 := eT;

 end if;
end process calculo;

end behav1;
```

9.3.2.6.2 Altera IP-Cores

La instalación del software Altera Quartus II (actualmente Intel Quartus Prime) incluye una biblioteca de núcleos de propiedad intelectual (IP-Cores) con funciones específicas que permiten acortar los tiempos de diseño y maximizar el rendimiento de los proyectos. La biblioteca IP incluye las siguientes categorías:

Funciones Básicas	Protocolos de interfaz
Puentes y adaptadores	Funciones de bajo consume
Funciones DSP	Interfaces de memoria y controladores
Interconexión Intel FPGA	Procesadores y periféricos

Aquí nos concentraremos en la categoría “Funciones Básicas” en donde podemos encontrar IP-Cores de aritmética de punto Flotante para simple y doble precisión. A continuación se listan todos los núcleos disponibles.

Tabla 26: Rangos de las operaciones

LIST OF FLOATING-POINT IP CORES	
IP CORE NAME	**FUNCTION OVERVIEW**
Operator Functions	
ALTFP_ADD_SUB	Adder/Subtractor
ALTFP_DIV	Divider
ALTFP_MULT	Multiplier
ALTFP_SQRT	Square Root
Algebraic and Transcendental Functions	
ALTFP_EXP	Exponential
ALTFP_INV	Inverse
ALTFP_INV_SQRT	Inverse Square Root
ALTFP_LOG	Natural Logarithm
Trigonometric Functions	
ALTFP_ATAN	Arctangent
ALTFP_SINCOS	Trigonometric Sine/Cosine
Other Functions	
ALTFP_ABS	Absolute value
ALTFP_COMPARE	Comparator
ALTFP_CONVERT	Converter
FP_ACC_CUSTOM Intel FPGA IP	An application specific accumulator
FP_FUNCTIONS Intel FPGA IP	A collection of floating-point functions.

Los núcleos ofrecen las siguientes características:

- Compatibilidad con formatos de punto flotante.

- Soporte de entrada para números que no son números (NaN), infinito, cero y números normales.
- Puertos de entrada asíncrona opcionales que incluyen borrado asíncrono (aclr) y habilitación de reloj (clk_en).
- Compatibilidad con el modo de redondeo de redondeo al par más cercano.
- Calcule los resultados de cualquier operación matemática de acuerdo con el cumplimiento del estándar IEEE-754 con un máximo de 1 unidad en el último lugar (u.l.p.) error. Esta suposición se aplica a todas las direcciones IP de punto flotante.

9.3.2.6.2.1 **PID usando Punto Flotante** (IP-Core)

Para implementar el algoritmo PID se utilizan dos núcleos; uno para sumar y restar (ALTFP_ADD_SUB) y otro para multiplicar (ALTFP_MULT), ambos de aritmética de punto flotante de 32 bits (|S: 1| |E: 8| |M: 23| bits).

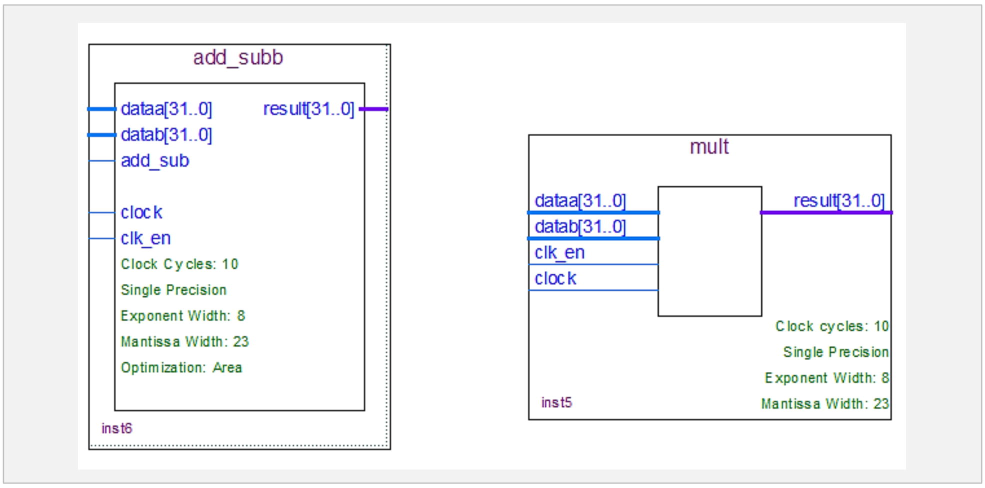


Fig 33 – IP-Cores para suma/sustracción y de multiplicación.

Adicionalmente se implementa una máquina de estados finitos (FSM) para parametrizar y controlar los núcleos según el algoritmo a implementar. Hay momentos en donde ambos núcleos pueden trabajar en simultáneo, mientras que en otro, uno de ellos debe esperar el resultado del otro. La imagen siguiente grafica a la izquierda el código del algoritmo PID básico y a la derecha las operaciones que debe resolver cada núcleo.

```
eT = rT - yT;

iT = Ki*eT + iT0;

dT = Kd*(eT - eT0);
uT = iT + Kp*eT + dT;
```

Fig 34a – Algoritmo a implementar.

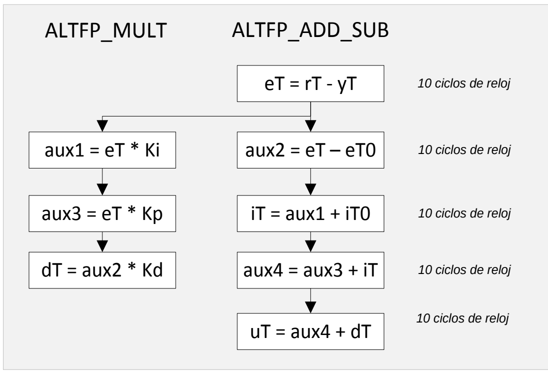


Fig 34b – Secuencia.

Se observa que el núcleo de multiplicación **ALTFP_MULT** debe realizar tres operaciones, mientras que el de suma/sustracción **ALTFP_ADD_SUB** debe realizar cinco operaciones. La máquina de estados finitos es quien coordina y encausa los operados a cada core a fin de obtener los resultados de las operaciones. La imagen a continuación ilustra la FSM formada por diez estados.

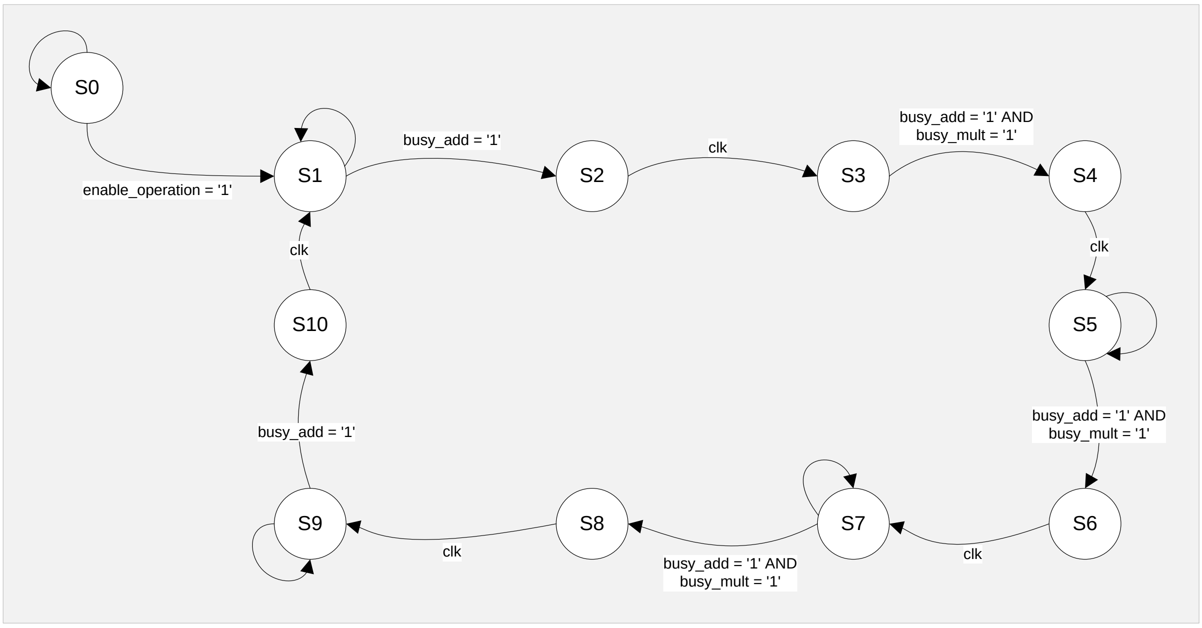


Fig 35 - Maquina de estados finitos para la implementación del controlador PID usando IP Cores

Cada estado en particular realiza una acción en particular, ya sea resolver parte de la ecuación PID o simplemente ubicar los operandos a la entrada de cada núcleo para luego tomar los resultados a sus salidas. Esto se muestra en detalle cuando se analice el código VHDL de la máquina de estados.

Las transiciones entre los estados están regidas por las siguientes señales:

- ***enable_operation:*** Habilita el normal funcionamiento de la máquina de estados.
- ***busy_add:*** Indica si el núcleo de adición se encuentra disponible (busy_add = 1).
- ***busy_mult:*** Indica si el núcleo de multiplicación se encuentra disponible (busy_mult = 1).
- ***clk:*** Indica que la transición al siguiente estado se debe producir en el próximo ciclo de reloj.

```
library ieee;
use ieee.std_logic_1164.all;
```

```
entity state_machine_op is
port(
        clk, reset, enable_operation     : in  std_logic;
        rT, yT, result_add, result_mult  : in  std_logic_vector(31 downto 0);
        busy_add, busy_mult              : in  std_logic;
        enable_add, enable_mult, add_sub : out std_logic;

        operando_add1, operando_add2     : out std_logic_vector(31 downto 0);
        operando_mult1, operando_mult2   : out std_logic_vector(31 downto 0);
        uT                               : out std_logic_vector(31 downto 0);
        busy_float                       : out std_logic

);
```

```
architecture rtl of state_machine_op is
 constant Kp           : std_logic_vector(31 downto 0):= X"3FC00000"; -- 1.5
 constant Ki           : std_logic_vector(31 downto 0):= X"3FC00000"; -- 1.5
 constant Kd           : std_logic_vector(31 downto 0):= X"3FC00000"; -- 1.5

       -- Build an enumerated type for the state machine
 type state_type is (s0, s1, s2, s3, s4, s5, S6, S7, S8, S9, S10);

       -- Register to hold the current state
 signal state   : state_type;
begin
 -- Logic to advance to the next state
 process (clk, reset)
 begin
        if reset = '1' then
               state <= s0;
        elsif (rising_edge(clk)) then
               case state is
                      when s0=>
                             if enable_operation = '1' then
                                    state <= s1;
                                    else
                                    state <= s0;
                             end if;
                      when s1=>
                             if busy_add = '1' then
                                    state <= s2;
                             else
                                    state <= s1;
                             end if;
                      when s2=>
                             state <= s3;
                      when s3=>
                             if (busy_add = '1' AND busy_mult = '1') then
                                    state <= s4;
                             else
                                    state <= s3;
                             end if;
                      when s4 =>
                             state <= s5;
                      when s5=>
                             if (busy_add = '1' AND busy_mult = '1') then
                                    state <= s6;
                             else
                                    state <= s5;
                             end if;
                      when s6=>
                             state <= s7;
                      when s7=>
                             if (busy_add = '1' AND busy_mult = '1') then
                                    state <= s8;
                             else
                                    state <= s7;
                             end if;
```

```
-- Output depends solely on the current state
process (state)
variable rT_v, yT_v, eT, aux1, aux2, aux3, aux4 : std_logic_vector(31 downto 0);
variable iT, dT   : std_logic_vector(31 downto 0);
variable iT0, eT0 : std_logic_vector(31 downto 0):= X"00000000";
begin
        case state is
                when s0 =>
                 enable_add  <= '0';
                 enable_mult <= '0';
                when s1 =>
                 rT_v := rT;
                 yT_v := yT;
                 busy_float  <= '0';
                 operando_add1 <= rT_v;
                 operando_add2 <= yT_v;
                 add_sub       <= '1';
                 enable_add    <= '1';        -- ADD 1 --
                when s2 =>
                 eT := result_add;
                when s3 =>
                 operando_mult1 <= eT;
                 operando_mult2 <= Ki;
                 enable_mult <= '1';          -- MULT 1 --
                 operando_add1 <= eT;
                 operando_add2 <= eT0;
                 add_sub       <= '1';
                 enable_add    <= '1';        -- ADD 2 --
                when s4 =>
                 aux1 := result_mult;
                 aux2 := result_add;
                when s5 =>
                 operando_mult1 <= eT;
                 operando_mult2 <= Kp;
                 enable_mult <= '1';          -- MULT 2 --
                 operando_add1 <= aux1;
                 operando_add2 <= iT0;
                 add_sub       <= '0';
                 enable_add <= '1';           -- ADD 3 --
                when s6 =>
                 aux3 := result_mult;
                 iT   := result_add;
                when s7 =>
                 operando_mult1 <= aux2;
                 operando_mult2 <= Kd;
                 enable_mult <= '1';          -- MULT 3 --
                 operando_add1 <= aux3;
                 operando_add2 <= iT;
                 add_sub       <= '0';
                 enable_add    <= '1';        -- ADD 4 --
                when s8 =>
                 dT   := result_mult;
                 aux4 := result_add;
```

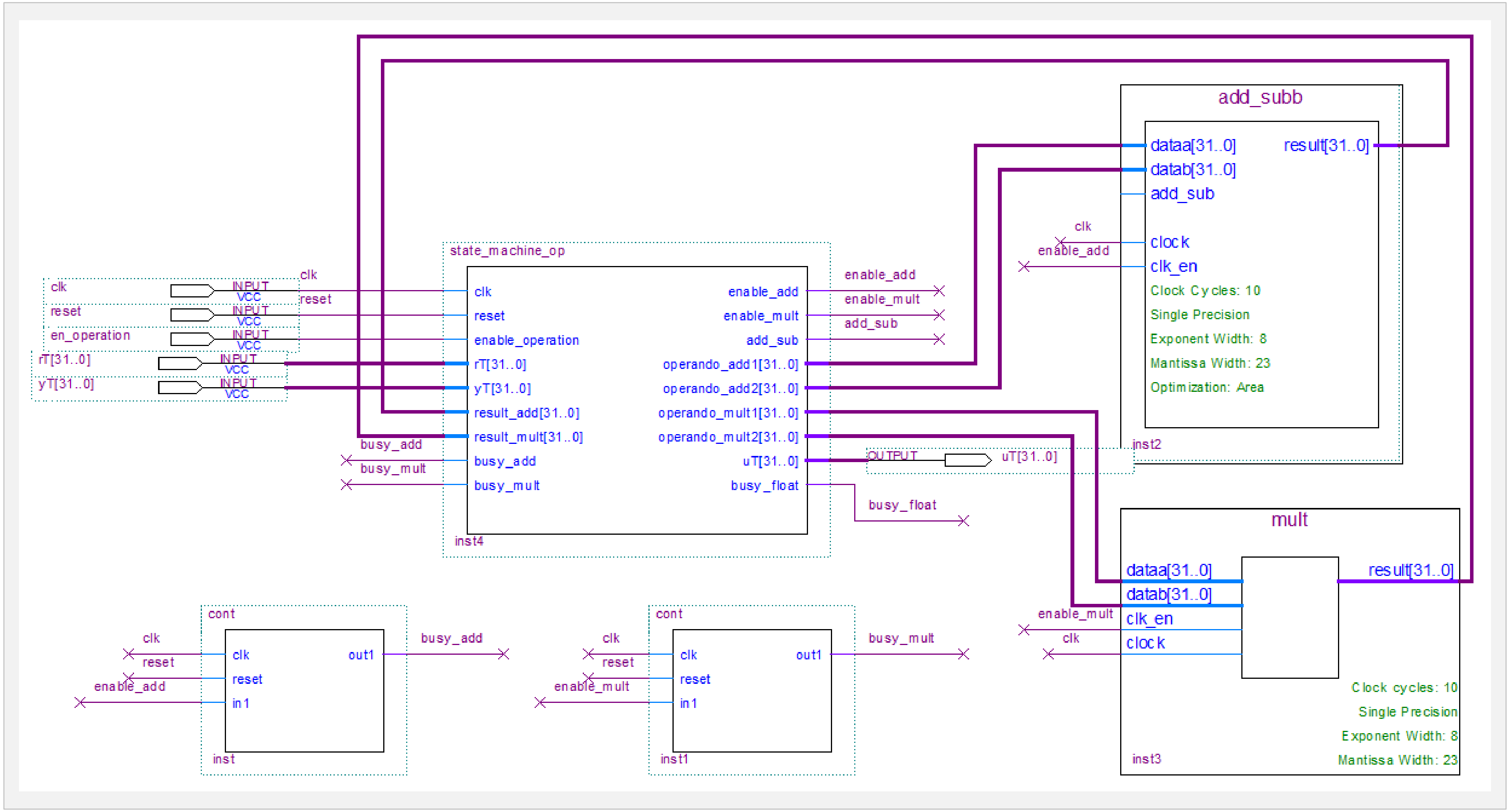


Fig 36 - Descripción Estructural de la Maquina de estados finitos para la implementación del controlador PID usando IP Cores.

9.3.3 PWM

La modulación por ancho de pulsos (también conocida como PWM, por sus siglas en inglés de *pulse-width modulation*) de una señal o fuente de energía es una técnica en la que se modifica el ciclo de trabajo de una señal periódica (senoidal o cuadrada), ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga.

El ciclo de trabajo de una señal periódica es el ancho relativo de su parte positiva en relación con el período. Expresado matemáticamente:

$$D = \frac{\tau}{T} \qquad (30)$$

D: Ciclo de trabajo

τ: Tiempo en que la función es positiva (ancho del pulso)

T: Período de la función

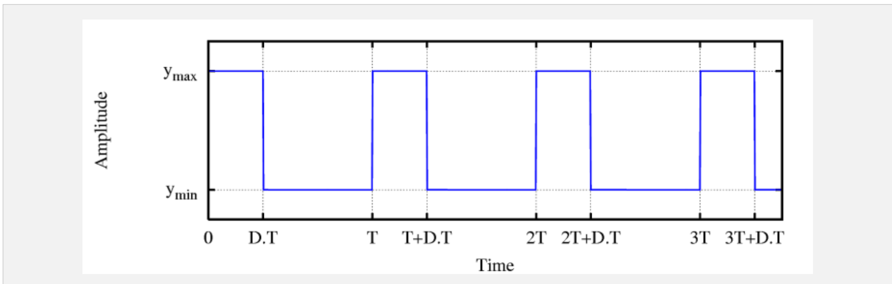


Fig 37 - Señal de onda cuadrada de amplitud acotada (ymin, ymax) mostrando el ciclo de trabajo D.

Típicamente suele ser utilizada para regular la velocidad de giro de los motores eléctricos de inducción o asíncronos. Mantiene el par motor constante y no supone un desaprovechamiento de la energía eléctrica. Se utiliza tanto en corriente continua como en alterna, como su nombre lo indica, al controlar: un momento alto (encendido o alimentado) y un momento bajo (apagado o desconectado), controlado normalmente por relès (baja frecuencia) o MOSFET o tiristores (alta frecuencia).

9.3.3.1 Implementación

Su implementación en VHDL es muy sencilla; primero se construye un contador de 256 estados (para obtener una resolución de 8 bits). Segundo se compara el ciclo de trabajo deseado con el valor actual del contador. Si la cuenta es menor al ciclo de trabajo, entonces la señal de pwm_out se pone en alto, de lo contrario permanecerá en estado bajo. Esto se detalla a continuación:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity pwm is
    port(
        clk     : in  std_logic;
        reset   : in  std_logic;
        pwm_in    : in  std_logic_vector(7 downto 0);
        pwm_out : out std_logic
    );
end pwm;
```

```
architecture behavioral of pwm is
    signal cnt : unsigned(7 downto 0);
begin
    contador: process (clk, reset, pwm_in) begin
        if reset = '0' then
            cnt <= (others => '0');
        elsif rising_edge(clk) then
            if cnt = 255 then
                cnt <= (others => '0');
            else
                cnt <= cnt + 1;
            end if;
        end if;
    end process;
    pwm_out <= '1' when (cnt < UNSIGNED(pwm_in)) else '0';
end Behavioral;
```

Por otro lado, es importante colocar a la entrada del PWM un divisor de frecuencia para poder trabajar a una frecuencia más baja, ya que los osciladores principales de las FPGA suelen ser de 50Mhz o mayor.

Para el caso de estudio se elige una frecuencia de 20Khz:

$$relacion = \frac{f_{osc}}{f_{pwm}} = \frac{50Mhz}{20Khz} = 2500 \qquad Pulsos\ a\ contar = \frac{relacion}{2} - 1 = \frac{2500}{2} - 1 = 1249$$

Por lo tanto, para obtener una frecuencia de 20Khz será necesario contar 1249 pulsos de la señal de reloj.

```
library ieee;
use ieee.std_logic_1164.all;

entity pwm_div is
    port (
        pwm_in   : in  std_logic;
        reset    : in  std_logic;
        pwm_out  : out std_logic
    );
end;
```

```
architecture behavioral of pwm_div is
    signal temporal : std_logic;
    signal contador : integer range 0 to 1249 := 0; -- 20 khz
begin
    pwm_divisor_frecuencia: process (reset, pwm_in) begin
        if (reset = '0') then
            temporal <= '0';
            contador <= 0;
        elsif rising_edge(pwm_in) then
            if (contador = 1249) then           -- 20 khz
                temporal <= not(temporal);
                contador <= 0;
            else
                contador <= contador+1;
            end if;
        end if;
    end process;
    pwm_out <= temporal;
end behavioral;
```

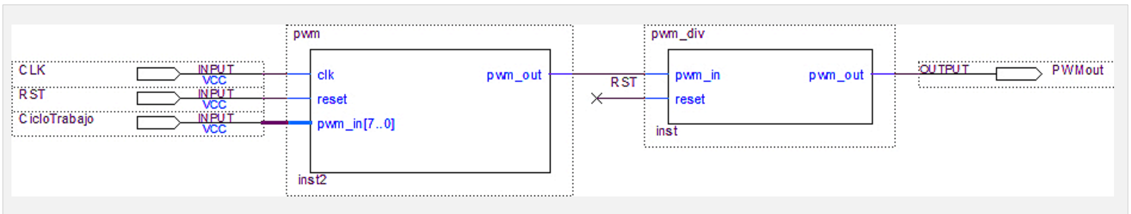


Fig 38 - Señal de onda cuadrada de amplitud acotada (ymin, ymax) mostrando el ciclo de trabajo D.

9.3.4 Comunicación I2C

Circuito inter-integrado (I2C por sus siglas en ingles de *Inter-Integrated Circuit*) es un bus serie de datos desarrollado en 1982 por Philips Semiconductors (hoy NXP Semiconductors). Se utiliza principalmente para la comunicación entre diferentes partes de un circuito, por ejemplo, entre un controlador y circuitos periféricos integrados.

El sistema original fue desarrollado por Philips a principios de 1980 con el fin de controlar varios chips en televisores de manera sencilla. Desde mediados de 1990 el I2C también es utilizado por algunos competidores para designar los sistemas compatibles I2C Philips, incluyendo Siemens AG (posteriormente Infineon Technologies AG), NEC, STMicroelectronics, Motorola (Freescale más adelante), Intersil, etc.

El I2C está diseñado como un bus maestro-esclavo. La transferencia de datos es siempre inicializada por un maestro; el esclavo reacciona. Es posible tener varios maestros mediante un modo multimaestro, en el que se pueden comunicar dos maestros entre sí, de modo que uno de ellos trabaja como esclavo. El arbitraje (control de acceso en el bus) se rige por las especificaciones, de este modo los maestros pueden ir turnándose.

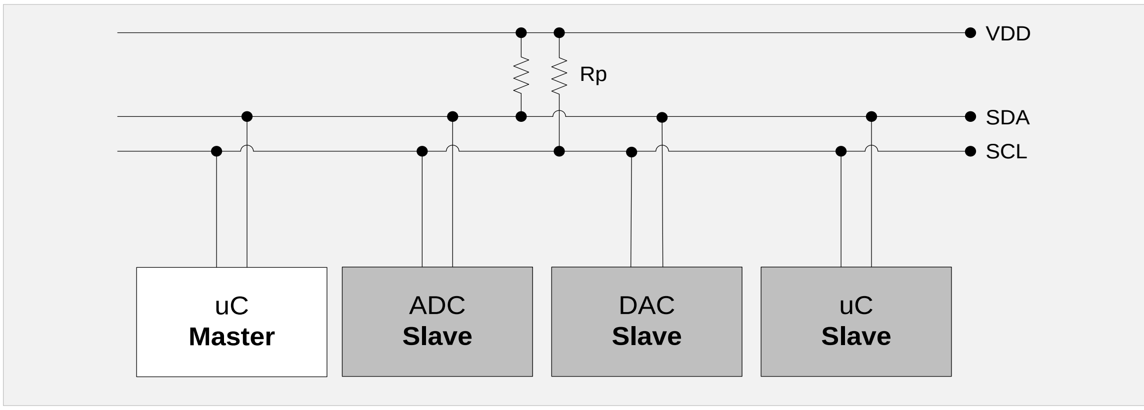


Fig 39 - Dispositivos conectados por I2C.

En el diagrama se encuentran representados tres dispositivos esclavos y un maestro. El bus precisa de dos líneas de señal: reloj (SCL, Serial Clock) y la línea de datos (SDA, Serial Data). Ambas líneas precisan resistencias de pull-up hacia VDD. Cualquier dispositivo conectado a estas líneas es de drenador o colector abierto (Open Collector), lo cual en combinación con las resistencias pull-up, crea un circuito Wired-AND.

El nivel alto debe ser de al menos 0,7 x VDD y el nivel bajo no debe ser más de 0,3 x VDD. Las resistencias en serie Rs (no representadas en el diagrama), en la entrada de los dispositivos, son opcionales y se usan como resistencias de protección. El Bus I2C trabaja con lógica positiva, esto quiere decir que un nivel alto en la línea de datos corresponde a un 1 lógico, el nivel bajo a un 0.

9.3.4.1 Direccionamiento

La dirección de I2C estándar es el primer byte enviado por el maestro, aunque los primeros 7 bits representan la dirección y el octavo bit (R/W-Bit) es el que comunica al esclavo si debe recibir datos del maestro (low/bajo) o enviar datos al maestro (high/alto). Por lo tanto, I2C utiliza un espacio de direccionamiento de 7 bits, lo cual permite hasta 112 nodos en un bus (16 de las 128 direcciones posibles están reservadas para fines especiales).

Cada uno de los circuitos integrados con capacidad de soportar un I2C tiene una dirección predeterminada por el fabricante, de la cual los últimos tres bits (subdirección) pueden ser fijados por tres pines de control. En este caso, pueden funcionar en un I2C hasta 8 circuitos integrados. Si no es así, los circuitos integrales (que precisan ser idénticos) deben ser controlados por varios buses I2C separados.

Debido a la escasez de direcciones, se introdujo más tarde un direccionamiento de 10 bits. Es compatible con el estándar de 7 bits mediante el uso de 4 de las 16 direcciones reservadas. Ambos modos de direccionamiento pueden utilizarse simultáneamente, lo que permite hasta 1136 nodos en un único bus.

9.3.4.2 Protocolo de transferencia

El inicio de una transmisión es indicado por la señal de inicio del maestro, seguido de la dirección. Ésta es confirmada por el ACK-Bit del esclavo correspondiente. En función del R/W-Bit se escriben bytes de datos (datos al esclavo) o se leen (datos al maestro). El ACK es enviado desde el esclavo al escribir, y desde el maestro al leer. El último byte "leído" es reconocido por el maestro como un NACK (not acknowledge), para indicar el final de una transmisión. Una transmisión es finalizada por la señal de parada. Como alternativa, puede ser enviada una señal de reset al arranque de una nueva transmisión, sin necesidad de parar la transmisión anterior con una señal de parada.

Todos los bytes son transferidos de esta manera como "Most Significant Bit First" (bit más significativo primero).

Para el modo de alta velocidad (High-Speed-Mode) es enviado un código del maestro a través del modo estándar o rápido, antes de cambiar al aumento de frecuencia.

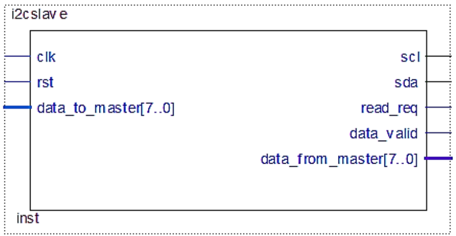
9.3.4.3 Implementación

Se puede implementar el bus serial por medio de una máquina de estados finitos siguiendo las especificaciones del protocolo o bien ganar tiempo de diseño y verificación, haciendo uso nuevamente de los núcleos de propiedad intelectual (IP-Cores). Estos pueden ser propios del sintetizador o de un tercero ya sea libre y gratuito o pago.

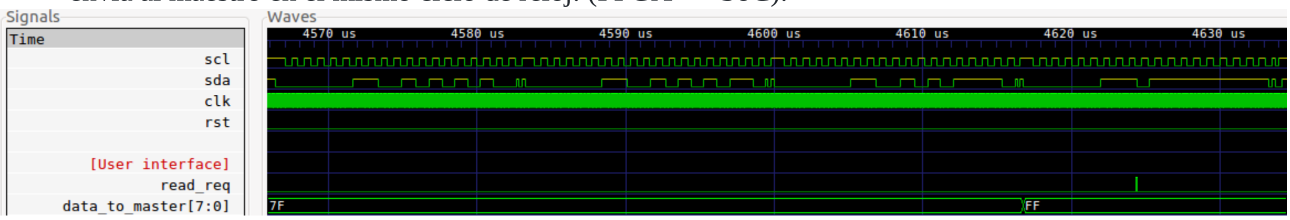
OpenCores es una comunidad de hardware de código abierto más grande del mundo que desarrolla hardware libre digital a través de la automatización de diseño electrónico, con un ethos similar al del movimiento del software libre. Por medio de esta comunidad se hace uso del core I2C_minion desarrollado por Peter Samarin peter.samarin@gmail.com.

El módulo se puede utilizar a través de la siguiente interfaz:

```
read_req         : out std_logic;
data_to_master   : in  std_logic_vector(7 downto 0);
data_valid       : out std_logic;
data_from_master : out std_logic_vector(7 downto 0));
```



Cuando ***read_req*** pasa a nivel alto, el esclavo toma los datos disponibles en ***data_to_master*** y los envía al maestro en el mismo ciclo de reloj. (FPGA => SoC).



Cuando ***data_valid*** pasa a nivel alto, el esclavo toma los datos disponibles en ***data_from_master*** que llegaron del maestro en el mismo ciclo de reloj. (FPGA <= SoC).

El esclavo ignora todos los comandos del maestro dirigidos a direcciones que no coinciden con SLAVE_ADDR. Los 7 bits de SLAVE_ADDR generalmente se dan en el primer formato de bits más significativos (MSB) en el lado maestro. Por ejemplo, una dirección de 7 bits "0000011" corresponde al decimal 3 en el esclavo, en realidad es un 6 en el maestro porque nunca se envía el LSB.

A continuación se adjunta el código para el control del modulo de comunicación I2C.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
```

```
entity i2c_32 is
  port (
    scl              : inout std_logic;
    sda              : inout std_logic;
    clk              : in    std_logic;
    rst              : in    std_logic;
    loByte           : out   std_logic_vector(7 downto 0);
    hiByte           : out   std_logic_vector(7 downto 0)
  );
end entity;
```

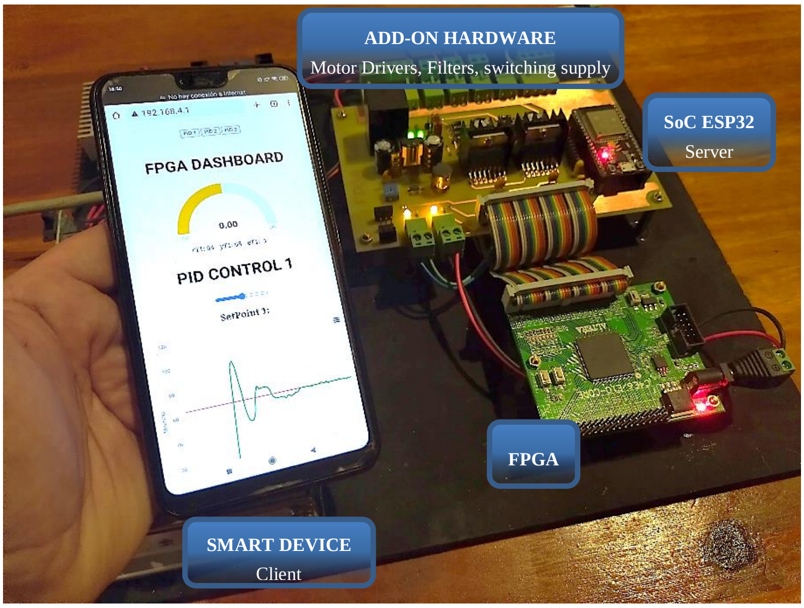
```
architecture RTL of i2c_32 is

signal read_req         : std_logic;
signal data_to_master   : std_logic_vector(7 downto 0);
signal data_valid       : std_logic;
signal data_from_master : std_logic_vector(7 downto 0);

signal data_reg1 : std_logic_vector(7 downto 0) := "00000000";
signal data_reg2 : std_logic_vector(7 downto 0) := "00000000";

type state_type is (s0, s1, s2);      -- Estados de la maquina de estados
signal state   : state_type := s0;    -- Registro del estado actual
type state_type2 is (r0, r1);         -- Estados de la maquina de estados
signal state2   : state_type2 := r0;  -- Registro del estado actual
signal i : integer range 0 to 1 := 0; -- 1 seg

begin
i2c_slave0  :  entity  work.i2cslave(arch)  port  map(scl,sda,clk,rst,  read_req,
data_to_master,data_valid,data_from_master);
```

```
  process (read_req)
  begin
        if (rising_edge(read_req)) then
              case state is
                    when s0=>
                                state <= s1;
                    when s1=>
                                state <= s2;
                    when s2=>
                                state <= s0;
              end case;
        end if;
  end process;

  process (state)
  begin
        case state is
              when s0 =>
                    data_to_master <= "00000001";
              when s1 =>
                    data_to_master <= "00000010";
              when s2 =>
                    data_to_master <= "00000011";
              end case;
  end process;

  process (data_valid)
  begin
        if (falling_edge (data_valid)) then
              case state2 is
                    when r0=>
                          state2 <= r1;
                          loByte <= not data_from_master;
                    when r1=>
                          state2 <= r0;
                          hiByte <= not data_from_master;
              end case;
        end if;
  end process;

end architecture;
```

9.4 HARDWARE COMPLEMENTARIO

9.4.1 Dashboard Control

De acuerdo a lo mencionado anteriormente, el uso de la FPGA permite realizar, en tiempo real, los cálculos de los algoritmos PID para cada articulación del robot, todo esto gracias a su gran capacidad de cómputo que permite realizar múltiples tareas de manera concurrente. Adicionalmente, toda la información procesada por la FPGA es visualizada en una página web, por medio de la implementación de un WebServer en el SoC ESP32. La imagen siguiente muestra en detalle los dispositivos involucrados.



Cada articulación del robot es gobernada por un motor de corriente continua, cuyo accionado se lleva a cabo, a través de los drivers de potencia L298 situados en la placa de hardware complementario (Add-on Hardware). La placa también cuenta con filtros EMI para la protección contra señales espurias y una fuente switching step down para estabilizar la tensión en 3.3V.

El SoC ESP32 emite una red Wifi de acuerdo al estándar 802.11 que al conectarse un cliente, por medio de un dispositivo Smart (Celular, Tablet, PC, etc.) se accede a la página web que contiene toda la información de los los PID como; señal de referencia, de realimentación y error para cada uno. La información se presenta en un widgets y se visualiza temporalmente en un gráfico cartesiano. De este modo se logra una interacción más amigable entre la máquina y el hombre.

9.4.2 SOC ESP32

ESP32 es la denominación de una familia de chips SoC de bajo costo y consumo de energía, con tecnología Wi-Fi y Bluetooth de modo dual integrada. El ESP32 emplea un microprocesador Tensilica Xtensa LX6 en sus variantes de simple y doble núcleo e incluye interruptores de antena, balun de radiofrecuencia, amplificador de potencia, amplificador receptor de bajo ruido, filtros, y módulos de administración de energía. El ESP32 fue creado y desarrollado por Espressif Systems y es fabricado por TSMC utilizando su proceso de 40 nm. Es un sucesor de otro SoC, el ESP8266.

En la siguiente imagen, tomada de la hoja de datos, se muestran todos los bloques funcionales que conforman un SoC ESP32.

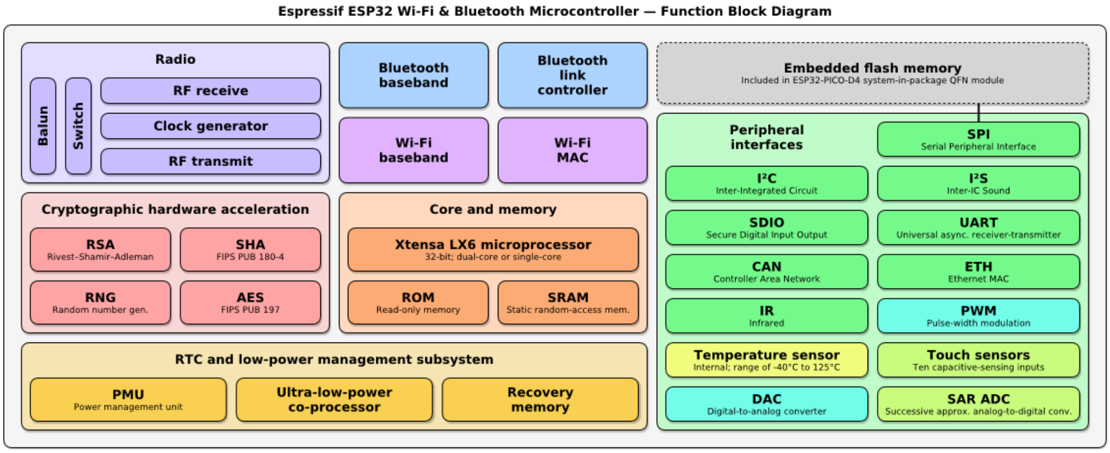


Fig 40 – Bloques funcionales ESP32.

9.4.2.1 Comunicación I2C

La comunicación con la FPGA se realiza por medio de la interfaz serial I2C, actuando como maestro el SoC y como esclavo la FPGA. A continuación se muestran los códigos para la recepción y transmisión de datos.

Recepción de datos (SoC <= FPGA)

```
  Wire.requestFrom(3,6);                      //       Wire.requestFrom(address,
quantity)
  while (Wire.available()) {
    out = Wire.read();
    Serial.print(out, DEC); Serial.print(" ");}
```

En la función `Wire.requestFrom(address, quantity)` se especifica en el primer argumento la dirección del esclavo (0x03) y en el segundo la cantidad de bytes a recibir (6 bytes).

Transmisión de datos (SoC => FPGA)

```
  if(!Wire.available()){
    Wire.beginTransmission(0x3);
    Wire.write(b1); Wire.write(b2); Wire.write(b3);
    Wire.write(b4); Wire.write(b5); Wire.write(b6);
    Wire.endTransmission();}
```

Si el canal de comunicaciones se encuentra disponible entonces se procede a establecer comunicación (con `Wire.beginTransmission`), para luego enviar los bytes a transmitir. Finalmente se finaliza la comunicación con `Wire.endTransmission()`.

9.4.2.2 Conversión AD y Filtrado Digital

El ESP32 integra dos ADC SAR (registro de aproximación sucesiva) de 12 bits que soporta un total de 18 canales de medición. La API del controlador ADC admite para el ADC1 (6 canales, conectados a GPIO 32-36, 39) y para ADC2 (10 canales, conectados a GPIO 0, 2, 4, 12-15 y 25-27). Sin embargo, el uso de ADC2 tiene algunas restricciones para la aplicación:

- ADC2 es utilizado por el controlador de Wifi. Por lo tanto, la aplicación solo puede usar ADC2 cuando el controlador de Wifi no se ha iniciado.
- Algunos de los pines ADC2 se utilizan como pines de sujeción ("strapping pins") (GPIO 0, 2, 15), por lo que no se pueden utilizar libremente.

El voltaje máximo admisible para los conversores es de 3.3VDC siendo su resolución en 12 bits de:

$$Resolución = \frac{3.3V}{2^{12}} = \frac{3.3V}{4096} = 0.8mV$$

Para el caso de estudio se trabaja con 9 bits para contar con una resolución de 6.5mV, por debajo de este valor no se producen cambios perceptibles en los motores utilizados. La modificación se especifica mediante la función analogReadResolution(9) en el setup de configuración. Posteriormente, la conversión análogo-digital se inicia con la llamada a la función analogRead(ADCpin). A continuación se muestra el código ejemplo con el agregado de un filtro de media móvil exponencial (EMA) para mejorar la respuesta del mismo.

```
#define alpha 0.25
int a, filter;

void setup () {
  analogReadResolution(9);}
```

```
void loop(){
  a = analogRead(A0);
  filter = (alpha*a) + ((1-alpha)*filter);}
```

El código del loop se repite seis veces para muestrear las señales de los potenciómetros de cintura, codo, hombro, muñeca giro, muñeca elevación y pinza.

La imagen siguiente contrasta la señal del adc en color azul versus la señal filtrada representada en trazo rojo para un alpha de 0.25.

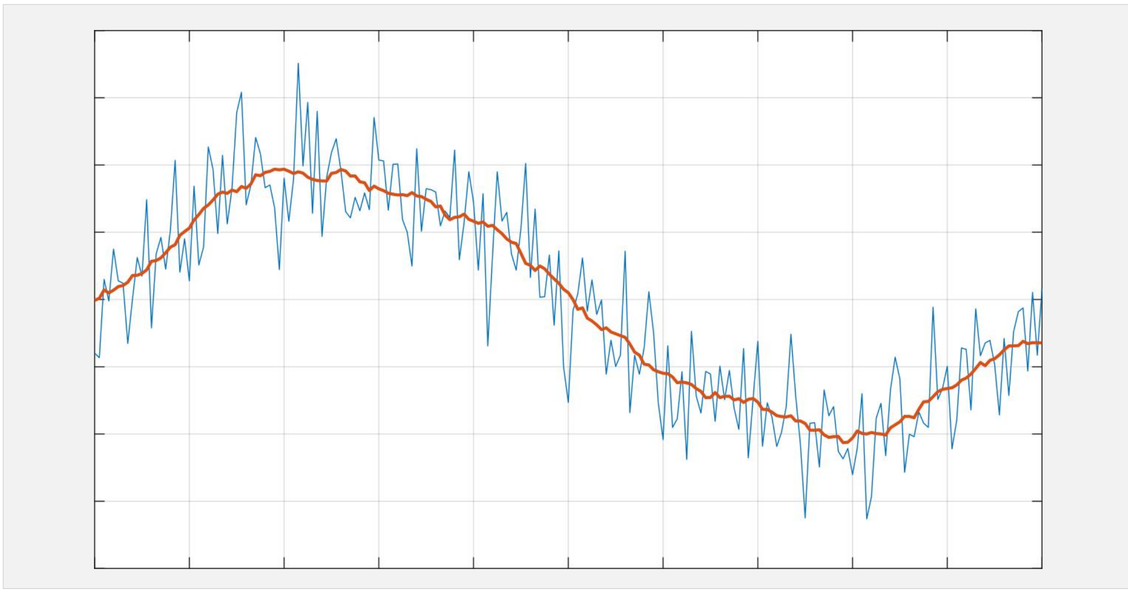


Fig 41 - Señal sin filtrar vs señal filtrada.

Recordemos que el filtro EMA consiste en obtener un valor filtrado a partir de la aplicación de la siguiente expresión:

$$A_n = \alpha . M + (1 - \alpha). A_{n-1}$$

Siendo An el valor filtrado, An-1 el valor filtrado anterior, M es el valor muestreado de la señal a filtrar, y alpha es un factor entre 0 y 1.

Por tanto, el filtro EMA presenta un aporte de información "nueva" a través de la medición M, y un efecto de suavizado basado en la memoria que aporta el valor filtrado anterior An-1. **El resultado de un filtro exponencial EMA es una señal suavizada** donde la cantidad de suavizado depende del factor alpha.

Las ventajas en cuanto a **sencillez y eficiencia computacional** son evidentes. El cálculo requiere una única instrucción sencilla. En cuanto a requisitos de memoria necesitamos almacenar únicamente el valor filtrado anterior. Esto supone una gran ventaja computacional frente a otros filtros que requieren guardar N valores y ejecutar cálculos sobre todos ellos.

Influencia del factor alpha

El factor Alpha condiciona el comportamiento del filtro exponencial y está relacionado con la frecuencia de corte del filtro. Sin embargo una relación sencilla no es siempre posible ya que depende del tiempo de muestreo de nuestro sistema que, en principio, es desconocido y posiblemente variable entre ciclos.

De forma cuantitativa:

- Un valor **Alpha = 1 proporciona la señal sin filtrar**, ya que prescinde del efecto filtrado que proporciona la medición anterior.
- Un valor de **Alpha = 0 provoca que el valor filtrado siempre sea 0**, ya que prescinde la información nueva que aporta la medición al sistema.

Disminuir el factor Alpha aumenta el suavizado de la señal, pero a costa de introducir consecuencias también negativas. Por un lado, podemos eliminar componentes frecuenciales que realmente nos fueran de interés, clasificando como ruido algo que realmente era una variación real de la señal. Por otro lado, **disminuir el factor Alpha aumenta el tiempo de respuesta del sistema**, es decir, el tiempo que tarda el sistema en estabilizarse ante una entrada constante. Esto se traduce en la introducción de un retraso entre la señal original y la señal filtrada. Lógicamente **el valor de Alpha adecuado dependerá de las características de nuestro sistema**, de la señal muestreada, y el ruido que queramos eliminar. En principio, deberemos ajustar el valor para que resulte adecuado a nuestro montaje, siendo valores habituales 0.2-0.6.

9.4.2.3 Punto de Acceso

El objetivo de este punto es montar una red de área local inalámbrica del tipo WLAN por sus siglas en ingles de **Wireless Local Area Network**, por medio de la implementación de un *punto de acceso* por software (**Software Enabled Access Point o simplemente SoftAP***)*. Esta es una de las tres formas de conexión WiFi que dispone el procesador ESP32 instalado en la placa.

Modo	Descripción
WIFI_STA	Estación
WIFI_AP	Punto de Acceso
WIFI_AP_STA	Estación y Punto de Acceso

Para montar el Access Point se recurre a la función WiFi.softAP(ssid, password, channel, hidden).

Los cuatro parámetros que la componen se definen como:

- ssdi: es un parámetro obligatorio, el nombre de red, SSID (Service Set Identifier). Debe tener entre 8 y 63 caracteres.
- password: es un parámetro opcional, la clave. Para una conexión WPA2-PSK debe tener al menos 8 caracteres. Si no se especifica, el Access Point estará abierto a los clientes.
- channel: es un parámetro opcional, para establecer el canal WiFi de comunicación. Los canales válidos son del 1 al 13, por defecto se utiliza el canal 1.
- hidden: es un parámetro opcional, para permitir ver u ocultar el SSID. 1 permite ver la red y 0 la oculta.

Para finalizar la conexión de los dispositivos conectados al Access point se invoca a la función WiFi.softAPdisconnect().

Con WiFi.softAPConfig(local_ip, gateway, subnet) se especifica una dirección IP determinada. Los tres parámetros que la componen se definen a continuación:

- local_ip: dirección IP del Access Point.
- gateway: es la dirección IP de la pasarela o puerta de enlace (puede coincidir con la local_ip).
- subnet: dirección IP de la subred.

Otra función de interés es WiFi.softAPIP() que devuelve la dirección IP con la que se ha creado la conexión (AccessPoint). Por defecto la dirección IP será: 192.168.4.1.

Con WiFi.softAPmacAddress(macAddr) permite conocer la dirección MAC del AccessPoint. El parámetro macAddr es voluntario, y es un array de seis elementos que permite almacenar cada uno de los elementos de la dirección MAC.

A continuación se muestra el código ejemplo para montar una WLAN desde el ESP32 utilizando las funciones descriptas anteriormente.

```
#include <WiFi.h>

const char *ssid = "ssid";
const char *password = "password";

void setup() {
 Serial.begin(115200); delay(10);
 WiFi.mode(WIFI_AP);

 while(!WiFi.softAP(ssid, password)) {
  Serial.println("."); delay(100); }

 Serial.print("Iniciado AP "); Serial.println(ssid);
 Serial.print("IP address: "); Serial.println(WiFi.softAPIP()); }

void loop() { }
```

9.4.2.4 Cliente-Servidor

La arquitectura cliente-servidor es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta. Esta idea también se puede aplicar a programas que se ejecutan sobre una sola computadora, aunque es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras.

Algunos ejemplos de aplicaciones computacionales que usan el modelo cliente-servidor son el Correo electrónico, los Servidores de impresión y la World Wide Web.

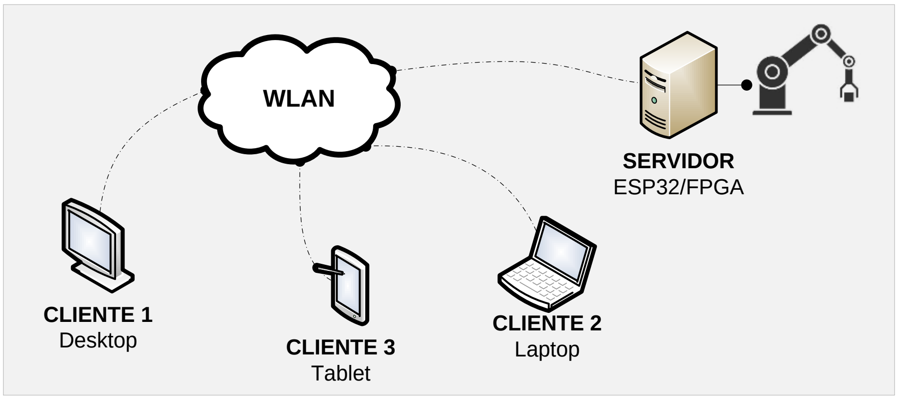


Fig 42 - Modelo Cliente Servidor.

Siguiendo con este modelo, se implementa en un ESP32 un ***Servidor Web*** para visualizar los parámetros de interés relacionados al robot manipulador en forma de dashboard. Con esto se logra proporcionar una interfaz visual a los clientes del webserver y de este modo, controlar y monitorear las acciones del robot. A continuación se detalla su implementación.

9.4.2.5 Webserver y websocket

Para montar un servidor web, se emplean tres bibliotecas. La primera es ESPAsyncWebServer que permite configurar un servidor HTTP asíncrono y Websocket, lo que significa que puede manejar más de una conexión al mismo tiempo. La segunda biblioteca necesaria es AsyncTCP, que es una dependencia de la anterior. Por lo tanto, no se interactuará directamente con ella en el código. La tercera biblioteca es WiFi que se encarga de gestionar y establecer la conexión a la red.

El uso de un servidor web asíncrono ofrece muchos beneficios, tales como:

- Permite manejar más de una conexión al mismo tiempo.
- Cuando envía la respuesta, inmediatamente está listo para manejar otras conexiones mientras el servidor se encarga de enviar la respuesta en segundo plano.
- Presenta una velocidad de respuesta muy rápida.
- API fácil de usar, HTTP Basic y Digest MD5 Authentication (predeterminado), ChunkedResponse.
- Fácilmente extensible para manejar cualquier tipo de contenido.
- Complemento Async WebSocket que ofrece diferentes ubicaciones sin servidores ni puertos adicionales.
- Complemento Async EventSource (Server-Sent Events) para enviar eventos al navegador.

- Complemento ServeStatic que admite caché, última modificación, índice predeterminado y más.

A continuación se detalla un código ejemplo para montar el webserver asíncrono.

1er Paso:
Incluir las librerías necesarias dentro del entorno de trabajo.

```
#include <WiFi.h>
#include <ESPAsyncWebServer.h>
#include <WebSocketsServer.h>
```

2do Paso:
Crear los objetos AsyncWebServer y WebSocketsServer asociados a los puertos 80 y 81 respectivamente.

```
AsyncWebServer server(80);
WebSocketsServer websockets(81);
```

3er Paso:
Establecer un manejador de eventos para las conexiones del WebSocket. Siendo:

- `WStype_CONNECTED` Cuando un cliente ha iniciado sesión.
- `WStype_DISCONNECTED` Cuando un cliente ha cerrado sesión.
- `WStype_TEXT` Cuando se recibe un paquete de datos del cliente.

```
void webSocketEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t length) {
  switch (type)
  {
    case WStype_DISCONNECTED:
      Serial.printf("[%u] ¡Desconectado!\n", num);
      break;

    case WStype_CONNECTED: {
       IPAddress ip = websockets.remoteIP(num);
       Serial.printf("[%u] Conectado en %d.%d.%d.%d url: %s\n",
         num, ip[0], ip[1], ip[2], ip[3], payload); }
      break;

    case WStype_TEXT:
      Serial.printf("[%u] Texto: %s\n", num, payload);
      String mensaje = String((char*)( payload));
      Serial.println(mensaje);
      break;
  }
}
```

4to Paso:
Configurar dentro del Setup los siguientes puntos:

- Velocidad de la comunicación serial.
- Setear el ESP32 en modo Access Point, especificando nombre y password de la red.

- Establecer la acción cuando un cliente se conecte (mostrar el mensaje de Hola SHUMA).
- Iniciar funciones (server, websockets y websockets.onEvent).

```
void setup () {
  Serial.begin (115200);

  WiFi.softAP("SHUMA47", "shuma2020");
  Serial.print("\nAccess Point: ");
  Serial.println(WiFi.softAPIP());

  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
        request->send(200, "text/plain", "Hola SHUMA");
    });

  server.onNotFound(notFound);
```

5to Paso:

Correr dentro del loop el websockets.

```
void loop(){
  websockets.loop(); }
```

9.4.2.6 **SPIFFS** (SPI Flash File System)

SPIFFS (SPI Flash File System) es un sistema de archivos diseñado para funcionar en memorias flash conectadas por SPI en dispositivos embebidos y con escasa cantidad de RAM, como es el caso del ESP8266 y del ESP32.

Para el caso de estudio, el dashboard se diseña utilizando HTML, CSS y funciones en JavaScript que permiten la actualización de la información en tiempo real. Todos los archivos son cargados en la memoria RAM del ESP32. De tal modo que cuando un cliente inicie sección y realice una petición HTTP, el servidor responderá con la información del dashboard levantada directamente desde su RAM.

Para trabajar con ficheros SPIFFS el ESP32 dispone de la librería SPIFFS que se debe incluir dentro del proyecto. Algunas de las funciones que incorpora la librería son las siguientes:

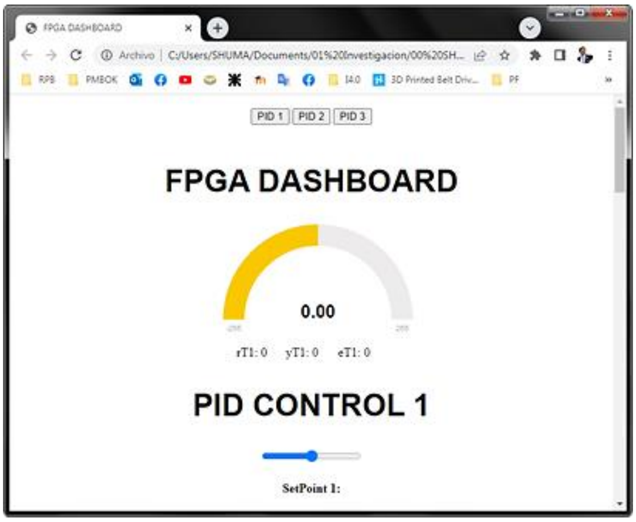


Fig 43 – Dashboard.

`SPIFFS.begin()` Este método monta el sistema de ficheros SPIFFS. Debe ser llamado antes de usar cualquier otro API del sistema de ficheros. Devuelve true si el sistema de archivos se ha montado satisfactoriamente, false en caso contrario.

`SPIFFS.end()` Este método desmonta el sistema de ficheros SPIFFS. Utiliza este método antes de realizar una actualización OTA del SPIFFS.

`SPIFFS.format()` Formatea el sistema de ficheros. Se puede llamar antes o después de llamar begin. Devuelve verdadero si el formateo tuvo éxito.

`SPIFFS.open(path, mode)` Abre un fichero. [path] debe ser un camino absoluto comenzando con un slash (p.ej. /dir/filename.txt). [mode] es una palabra que especifica el modo de acceso. Puede ser una de las siguientes: «r», «w», «a», «r+», «w+», «a+». El significado de estos modos es el mismo que para la función fopen en C.

A continuación se detalla el ejemplo anterior con la incorporación de SPIFFS.

```
#include <WiFi.h>
#include <ESPAsyncWebServer.h>
#include <WebSocketsServer.h>
#include <SPIFFS.h>

AsyncWebServer server(80);
WebSocketsServer websockets(81);

void webSocketEvent(uint8_t num, WStype_t type, uint8_t * payload, size_t length)
{
  switch (type)
  {
    case WStype_DISCONNECTED:
      Serial.printf("[%u] ¡Desconectado!\n", num);
      break;

    case WStype_CONNECTED: {
      IPAddress ip = websockets.remoteIP(num);
      Serial.printf("[%u] Conectado en %d.%d.%d.%d url: %s\n",
                num, ip[0], ip[1], ip[2], ip[3], payload); }
      break;

    case WStype_TEXT:
      //Serial.printf("[%u] Texto: %s\n", num, payload);
      String mensaje = String((char*)( payload));
      //Serial.println(mensaje);
      break;
  }
}
```

```
void setup () {
  Serial.begin (115200);

  if (! SPIFFS.begin ()) {
    Serial.println ("An Error has occurred while mounting SPIFFS");
    return; }

  WiFi.softAP("SHUMA47", "shuma2020");
  Serial.print("\nAccess Point: ");
  Serial.println(WiFi.softAPIP());

  server.on ("/", HTTP_GET, [] (AsyncWebServerRequest * request) {
    request-> send (SPIFFS, "/index.html"); });

  server.begin();
  websockets.begin();
  websockets.onEvent(webSocketEvent);
}

void loop(){
  websockets.loop(); }
```

9.4.3 Etapa de Potencia

9.4.3.1 MainBoard

MainBoard o también llamado hardware complementario es una placa PCB diseñada para albergar a todos los dispositivos involucrados del proyecto y es donde convergen todas las señales eléctricas provenientes de la FPGA y del ESP32. Cuenta con un filtro EMI para la protección contra señales espurias y una fuente conmutada Step Down para estabilizar y controlar la tensión en 3.3V.

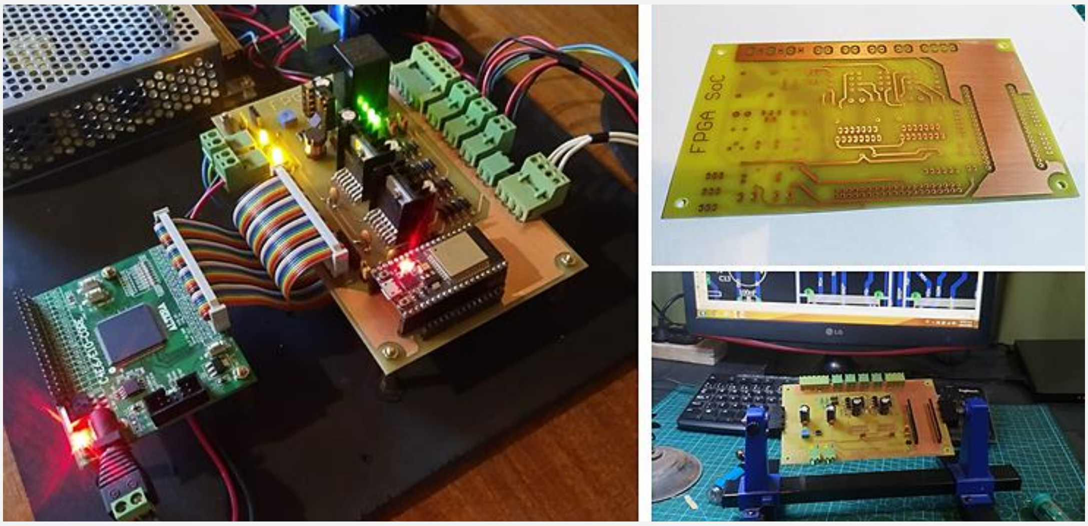


Fig 44 – Imágenes de la PCB de la implementación.

Para el control de motores se incluye dos drivers L298N que permite controlar tanto la velocidad como sentido de giro. La corriente máxima que el driver puede suministrar a los motores es, en teoría, 2A por salida (hasta 3A de pico) y una tensión de alimentación de 3V a 35V. Sin embargo, el L298N tiene una eficiencia baja, lo que implica una caída de tensión de 3V, es decir, la tensión que recibe el motor es unos 3V inferior a la tensión de alimentación. Estas pérdidas se disipan en forma de calor lo que se traduce en que, a efectos prácticos, una dificultad para obtener más de 0.8-1A por canal sin exceder el rango de temperatura de funcionamiento.

El driver dispone de protecciones contra sobre intensidad, sobre temperatura, y diodos de protección contra corrientes inducidas (flyback). La elección del chip en el proyecto, se debe fundamentalmente por su sencillez de uso, bajo coste, y buena calidad precio. A continuación se adjunta un diagrama en bloques que ilustra la interconexión de cada dispositivo sobre la placa.

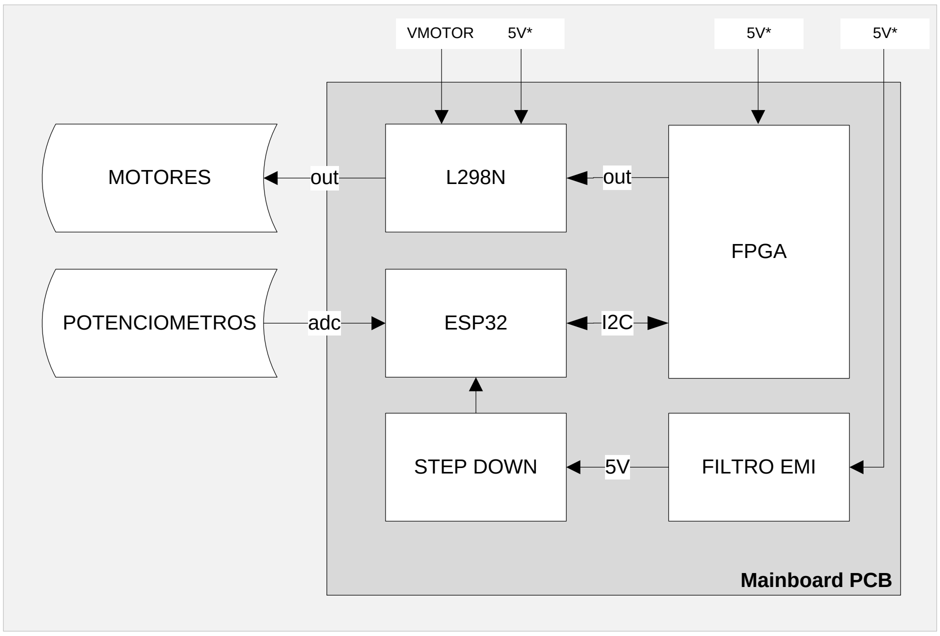


Fig 45 – Interconexión de los dispositivos.

Para el diagrama electrónico y ruteo de la placa PCB se utiliza el software de diseño electrónico Eagle, por sus siglas en ingles de Easily Applicable Graphical Layout Editor. A continuación se muestra el diseño final de la placa junto a su esquema eléctrico.

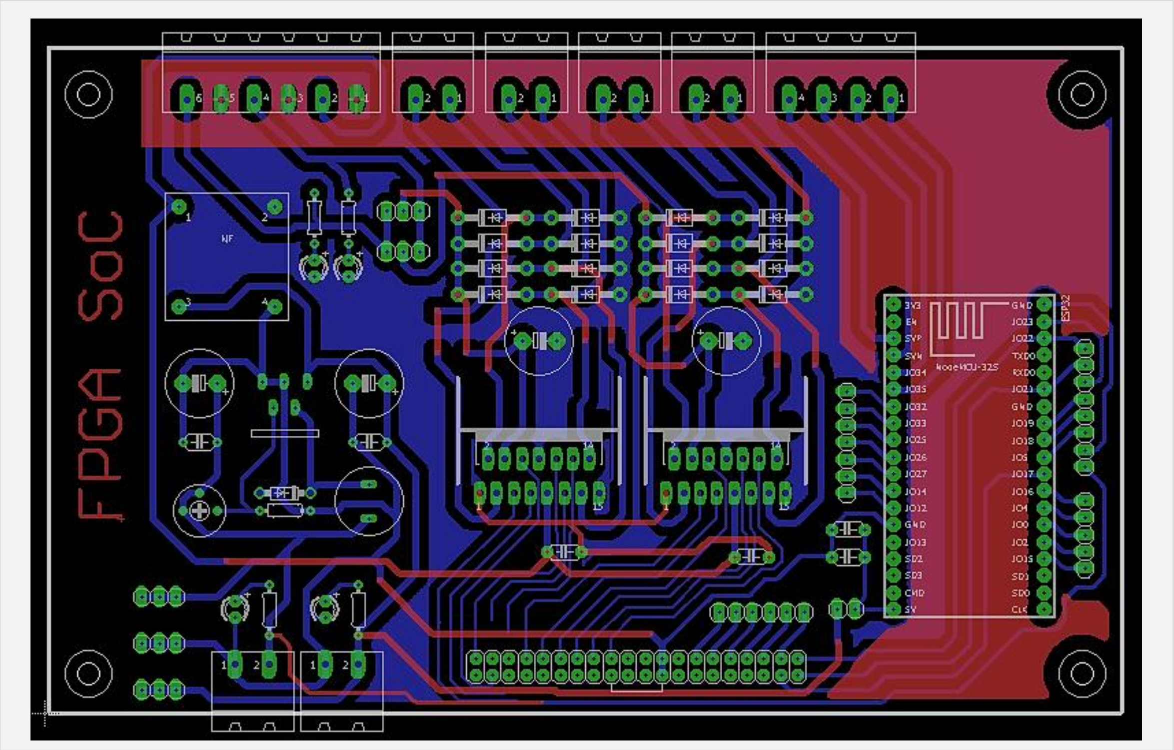


Fig 46 – Diseño de PCB usando Eagle.

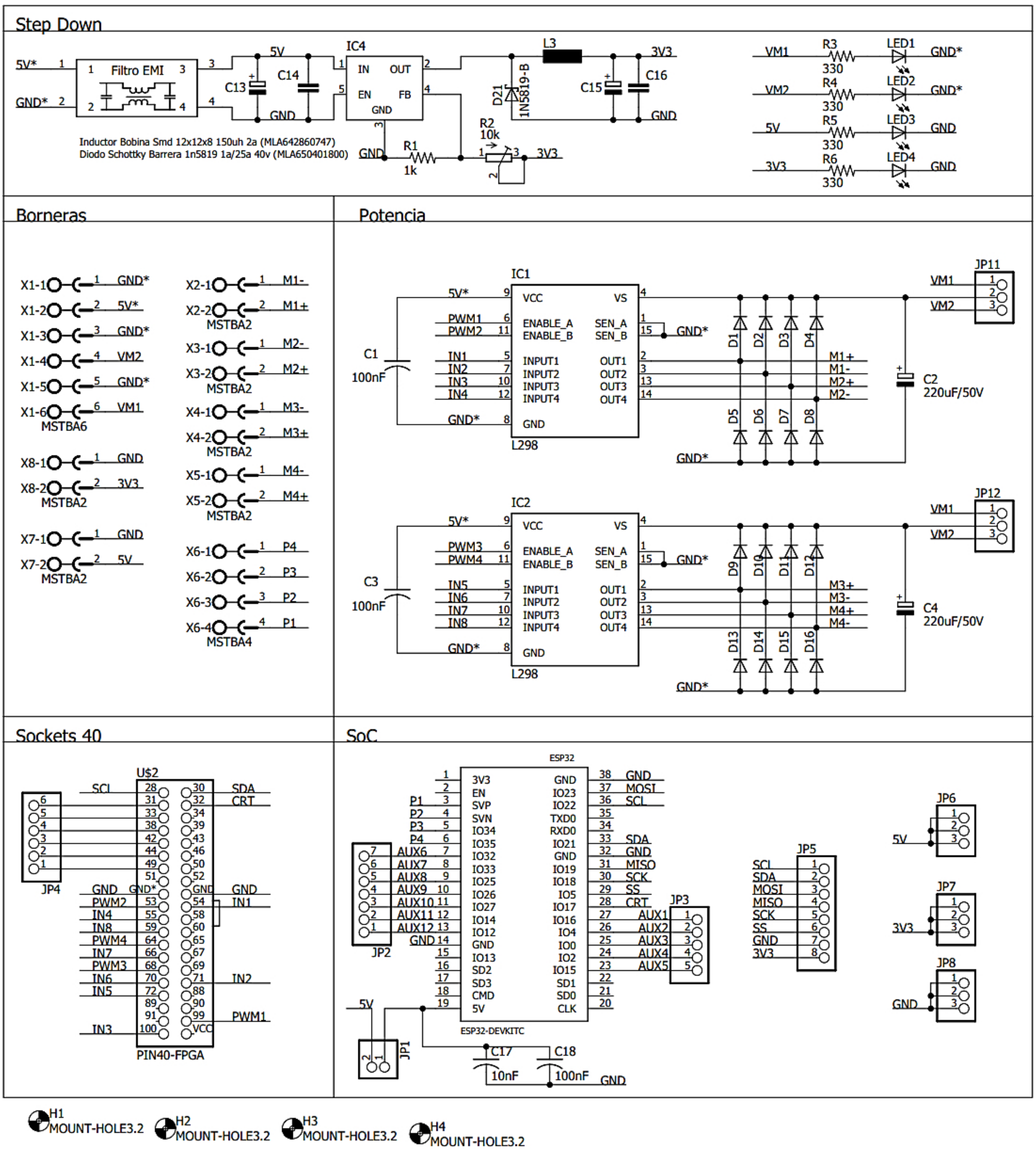


Fig 47 – Esquemas Electrónicos.

9.5 ANÁLISIS

El presente proyecto centra su atención en la implementación de cinco algoritmos de control PID sobre una FPGA Cyclone IV, para controlar la posición de cintura, hombro, codo, muñeca elevación y muñeca giro de un robot manipulador. Todo esto con el objetivo de analizar la relación costo beneficio entre el comportamiento de la respuesta en el tiempo y la cantidad de recursos ocupados o consumidos sobre la FPGA.

Este análisis permite encontrar una representación numérica que entregue resultados dentro de las especificaciones de diseño y que además, consuman un bajo número de recursos de hardware, a fin de trabajar con FPGA pequeñas y económicas.

9.5.1 Resultados

A continuación se presentan los resultados obtenidos luego de la implementación del algoritmo de control, empleando diferentes representaciones numéricas. Se resume en una primera tabla los valores asociados al consumo de recursos de área, tales como en número total de elementos combinaciones y secuenciales, el número de multiplicadores de 9 bits utilizados y la frecuencia máxima de operación a la cual el núcleo puede operar. En una segunda tabla se reúnen los valores asociados a la respuesta en el tiempo cuando el sistema es excitado con una señal escalón. Los parámetros a estudiar son el tiempo de subida, el tiempo de establecimiento, el porcentaje de sobre pico, entre otros.

9.5.1.1 Enteros

Las tablas 27 y 28 reúnen los resultados luego de la implementación operando con enteros de 32 bits.

Tabla 27: Recursos consumidos

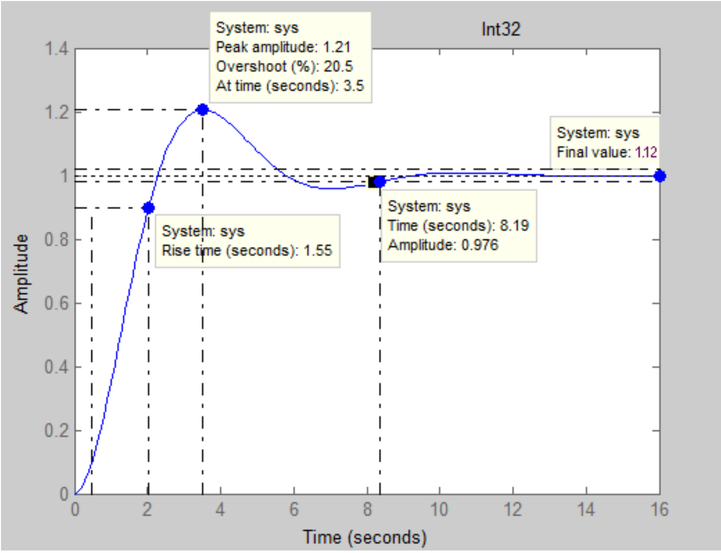
Device	Data Type	Total combinational functions	Dedicated logic registers	Embedded Multiplier 9-bit elements	Maximum operating frequency
Cyclone IV E EP4CE10E22C8	Int32	132 / 10,320 (< 1 %)	22 / 10,320 (< 1 %)	0 / 46 (0 %)	176,82 MHz

Tabla 28: Respuesta en el tiempo. Implementación usando enteros.

Data Type	Rise Time	Peak Response			Settling Time	Steady State
Int32	1,55s	1,21	20,5%	3,5s	8,34s	1,12%

Se observa que los recursos utilizados (LUTs) caen por debajo del 1%, tomando como referencia una FPGA de Altera Cyclone IV (EP4CE10E22C8) de US$ 3,5 aproximadamente.

Si bien se destaca el bajo uso de recursos y principalmente su sencillez de implementación, no corre con la misma suerte la respuesta al escalón. En tal caso, se observa un error en estado permanente mayor al 12%. Esta situación no es tolerable por lo que se explora el uso de números reales para la implementación.



9.5.1.2 **Punto fijo** (sfixed)

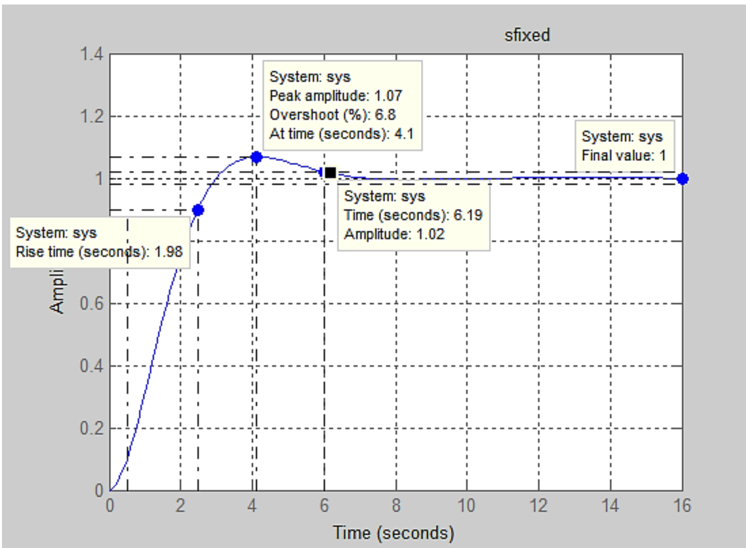
Las tablas 3 y 4 reúnen los resultados obtenidos luego de la implementación de un algoritmo PID usando sfixed como tipo de dato.

Tabla 3: Recursos consumidos

Device	Data Type	Total combinational functions	Dedicated logic registers	Embedded Multiplier 9-bit elements	Maximum operating frequency
Cyclone IV E EP4CE10E22C8	sfixed	1.412 / 10,320 (14 %)	40 / 10,320 (< 1 %)	12 / 46 (26 %)	108,54 MHz

Tabla 4: Respuesta en el tiempo. Implementación usando sfixed.

Data Type	Rise Time	Peak Response			Settling Time	Steady State
sfixed	1,98s	1.07	6,85%	4,1s	6,19s	1,0%



La implementación del algoritmo PID en punto fijo usando la librería sfixed permite obtener una respuesta en el tiempo optimizada de acuerdo a lo observado en la figura X y en la tabla S. Esta mejora trae aparejado un aumento del consumo de recursos de hardware de casi diez veces más que para la implementación con enteros. Es decir, se mejora el comportamiento del sistema a costa de requerir más recursos de hardware. También se ve fuertemente afectada la frecuencia máxima de operación que cae de 187,32 a 32,56 MHz.

Con sfixed se obtiene mejores resultados en la respuesta en el tiempo, a la vez de un uso moderado de los recursos de la FPGA. El dato de la frecuencia máxima de operación es importante conocerlo sobre todo cuando se desea reutilar varias veces un núcleo dentro de un proyecto.

9.5.1.3 **Punto fijo** (HDL-Coder)

Al igual que en los casos anteriores se muestra a continuación los resultados de síntesis y postfiter para la implementación utilizando HDL-Coder.

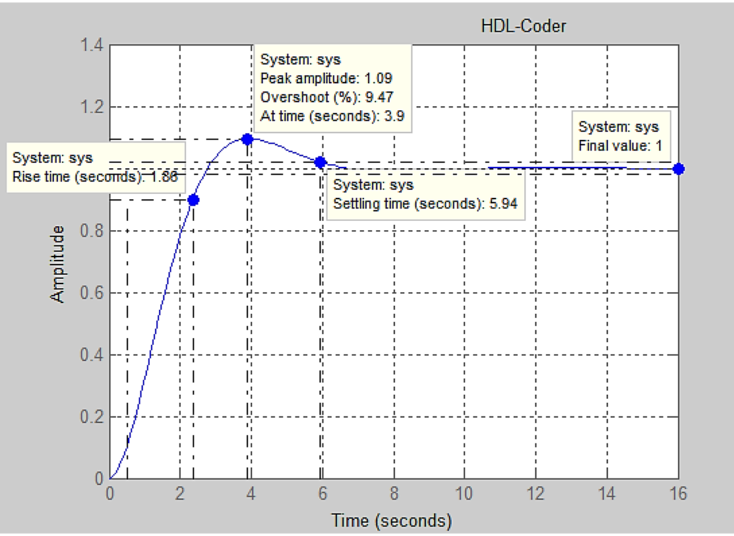
Tabla 3: Recursos consumidos

Device	Data Type	Total combinational functions	Dedicated logic registers	Embedded Multiplier 9-bit elements	Maximum operating frequency
Cyclone IV E EP4CE10E22C8	HDL Coder	858 / 10,320 (8 %)	64 / 10,320 (< 1 %)	46 / 46 (100 %)	81,51 MHz

Tabla 4: Respuesta en el tiempo. Implementación usando HDL-Coder.

Data Type	Rise Time	Peak Response			Settling Time	Steady State
HDL-Coder	1,98s	1.07	6,85%	4,1s	6,19s	1,0%

Con una respuesta en el tiempo aceptable sumado a su sencilla implementación hace que el uso de HDL-Coder sea una herramienta sumamente interesante a la hora de crear diseños en VHDL.



Como comentario desfavorable se cita al hecho de que su código se vuelve complejo y de difícil entendimiento. Además desde el punto vista de área consumida, se observa que en un único PID se consumen los 46 multiplicadores de 9 bits, esto podría ser un grave problema a la hora de tener que sintetizar más bloques en el proyecto.

9.5.1.4 **Punto flotante** (float)

La respuesta al escalón y los recursos de hardware ocupados para la implementación de un algoritmo PID utilizando punto flotante como tipo de datos son los siguientes:

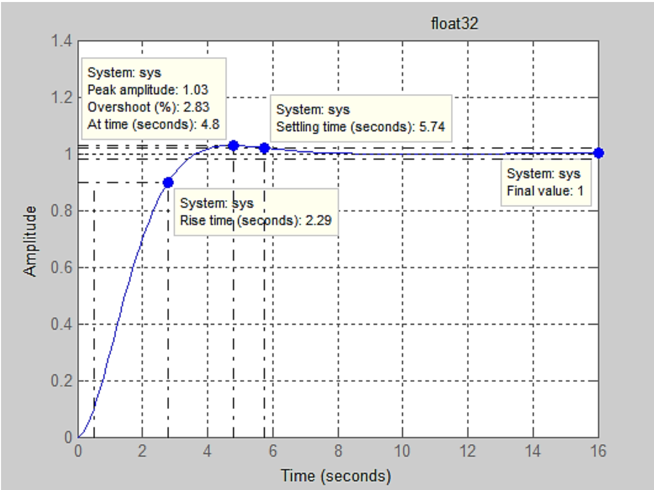
Tabla 3: Recursos consumidos

Device	Data Type	Total combinational functions	Dedicated logic registers	Embedded Multiplier 9-bit elements	Maximum operating frequency
Cyclone IV E EP4CE10E22C8	Float32	4.236 / 10,320 (41 %)	72 / 10,320 (< 1 %)	19 / 46 (41 %)	43,54 MHz

Tabla 4: Respuesta en el tiempo. Implementación usando HDL-Coder.

Data Type	Rise Time	Peak Response			Settling Time	Steady State
Float32	2,29s	1.03	2,83%	4,8s	5,74s	1,0%

Existe una mejora sustancial de la respuesta en el dominio del tiempo a costa de un consumo elevado de recursos de hardware. Esto es un problema ya que se tendrá que saltar a otro dispositivo con mayores recursos (y por ende mas costoso) a medida que se agregan más bloques al proyecto.



9.5.1.5 **Punto flotante** (IP-Cores)

Finalmente se muestran en las siguientes tablas los resultados de síntesis y postfiter para la implementación en punto flotante usando IP-Cores.

Tabla 3: Recursos consumidos

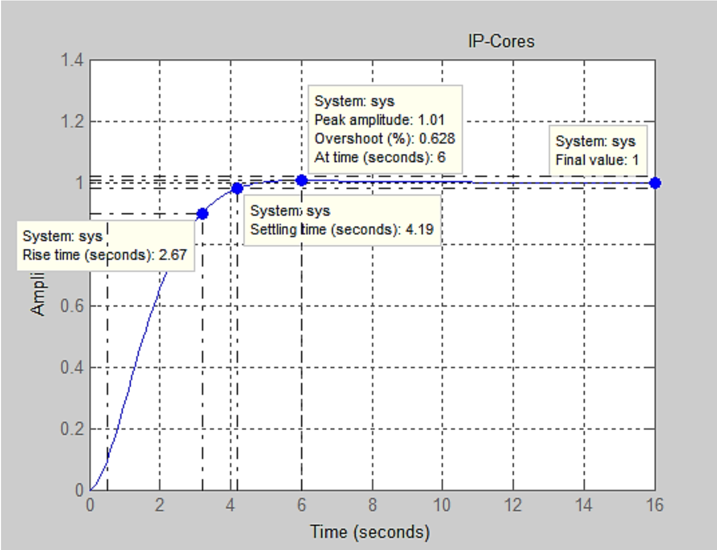
Device	Data Type	Total combinational functions	Dedicated logic registers	Embedded Multiplier 9-bit elements	Maximum operating frequency
Cyclone IV E EP4CE10E22C8	Float32	1.732 / 10,320 (17 %)	932 / 10,320 (9 %)	7 / 46 (15 %)	115,56 MHz

Tabla 4: Respuesta en el tiempo. Implementación usando IP-Cores.

Data Type	Rise Time	Peak Response			Settling Time	Steady State
Float32	2,67s	1.01	0,63%	6,0s	4,19s	1,0%

Sin lugar a dudas el uso de IP-Cores aporta muy buenos resultados de síntesis junto a un uso moderado de recursos de hardware.

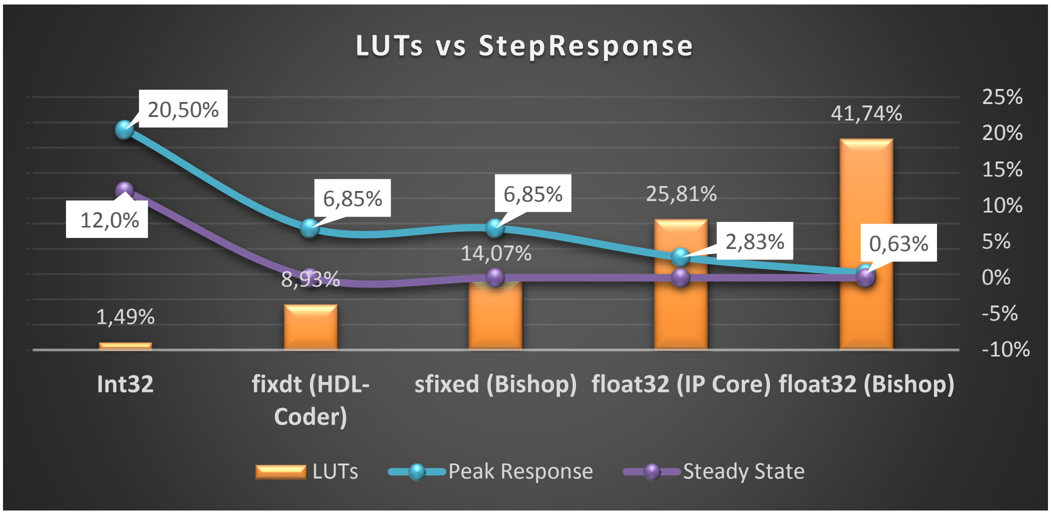
La ventaja de trabajar con IP-Cores propios del sintetizador es que cuentan con un nivel de optimización muy elevado. Esto permite un bajo consumo de recursos acompañado de una alta frecuencia de operación.



9.5.2 Comparativa

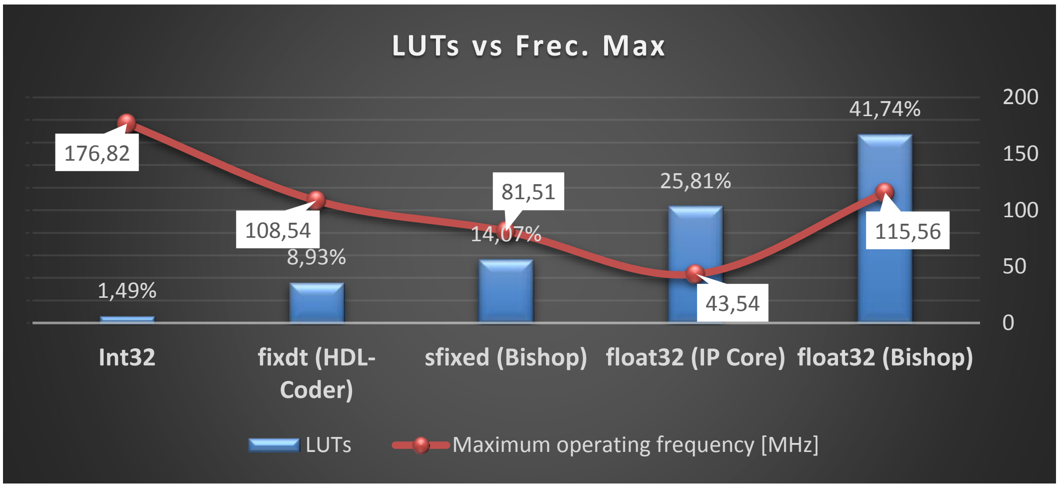
En este apartado se realiza una comparativa entre los diferentes tipos de datos con el fin de hallar aquella representación numérica que permita cumplir con las especificaciones de diseño y que además, presente un bajo consumo de recursos de hardware.

Cuando se trabaja con números enteros la implementación es rápida y sencilla gracias al uso de librerías propias del sintetizador. El consumo de LUTs es muy bajo, por debajo del 2% pero presenta el inconveniente que la respuesta transitoria y permanente del sistema no es la adecuada. En la imagen XX se observa que el error en estado transitorio ronda un 20,5% mientras que el error en estado permanente es del 12%. Esta situación es no tolerable por lo que se descarta su implementación.



Para el caso de la implementación en punto fijo la respuesta en el tiempo mejora notablemente, ya que el sobre pico se reduce de 20,5% a 12% y el error en estado estacionario tiende a cero. Desde el punto de vista de hardware, la implementación en punto fijo usando HDL-Coder presenta un menor consumo frente a la implementación de la librería de David Bishop, pero recordar que una vez generado el código VHDL, se vuelve muy difícil su comprensión. Esto puede representar un problema si se trabaja con múltiples recursos en un mismo proyecto.

La implementación en punto flotante trae aparejada varias mejoras respecto a las anteriores, como por ejemplo, se cuenta con un amplio rango dinámico, esto puede ser determinante para ciertas aplicaciones. Además, el error transitorio se reduce de 6,85% a 2,83% para el caso de IP-Cores y a 0,63% para el caso de la libreria de Bishop. Como aspecto negativo se cita al hecho de que el consumo de LUTs es mayor, sobre todo cuando se utiliza Bishop. Se puede observar que la implementación de un único controlador PID consume casi el 42% de los recursos de la FPGA (EP4CE10E22C8). Esto es un problema ya que para el control de posición del robot manipulador, se requiere de cinco controladores. Como solución se puede optar por seleccionar otra FPGA de mayores prestaciones (no deseable) o bien implementar un único controlador y ocuparlo cinco veces. Pero realizar esto último, implica perder la característica de concurrencia. Frente a esto, se puede afirmar que si la aplicación requiere contar de un amplio rango dinámico, lo conveniente será trabajar con los módulos IP-Cores, propios del sintetizador.

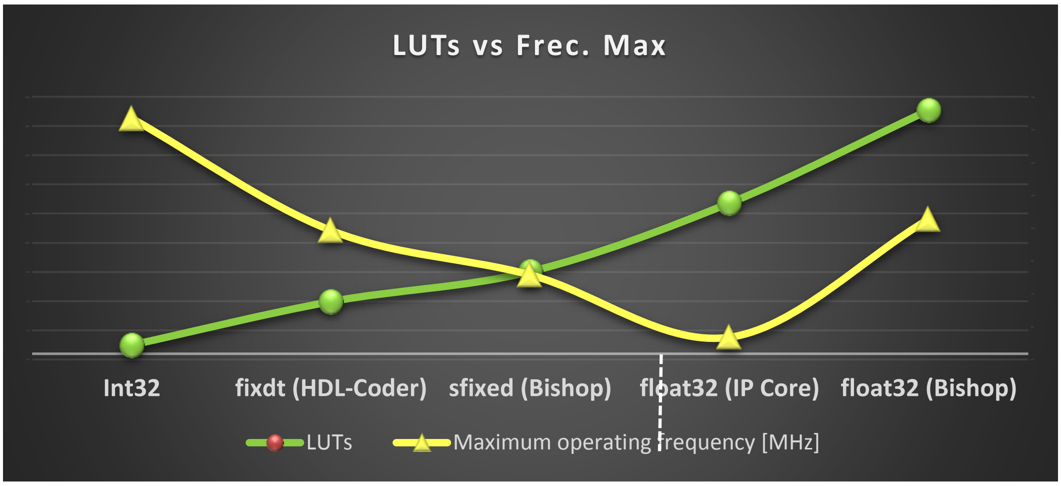


De acuerdo a lo mencionado anteriormente, otro parámetro a tener en cuenta es la frecuencia máxima de operación a la cual puede trabajar un núcleo. Esto es de gran interés cuando se requiere obtener gran capacidad de cómputo. En la gráfica anterior se observa que la implementación usando Bishop para punto flotante de simple precisión, es la que más recursos de hardware consume, pero presenta la mayor frecuencia de operación. Esto permite, de ser necesario, la reutilización de recursos.

Expuesto los resultados, el paso siguiente es elegir una de las representaciones numéricas evaluadas a fin de implementar los controladores PID, como así también el resto de bloques necesarios para el proyecto, tales como:

- Bloques para modulación de ancho de pulso PWM.
- Bloque de comunicación serial I2C.
- Bloque de coordinación y control FSM.
- Entre otros.

Si tomamos las series numéricas del grafico anterior y graficamos ambas como curvas, vemos que el cruce entre ellas se produce en las cercanías de la implementación de Bishop para punto fijo. Esto es un buen indicio de que el uso de recursos es moderado, acompañado de una muy buena frecuencia máxima de operación.

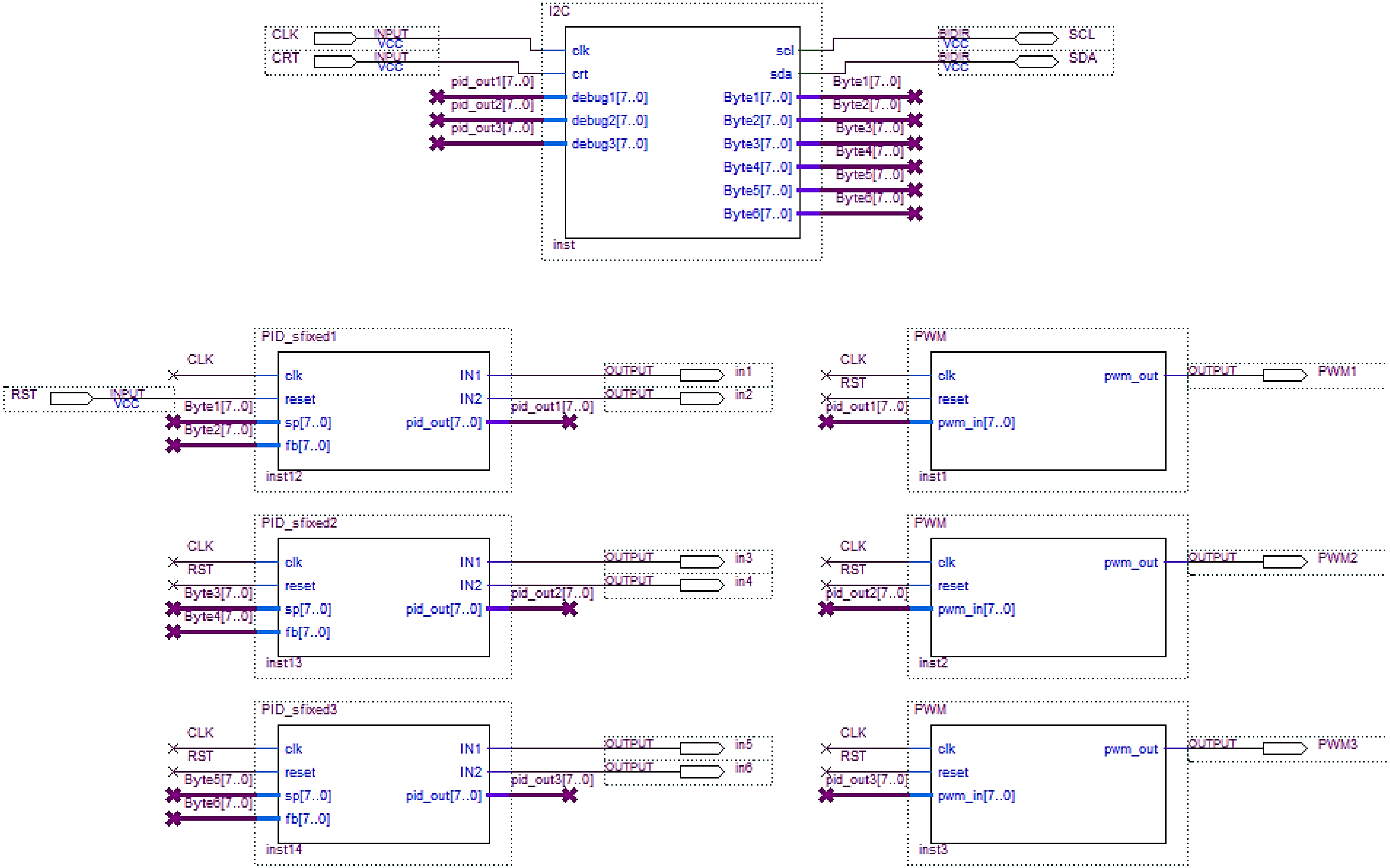


El bajo consumo de hardware, la aceptable frecuencia máxima de operación, la sencillez de implementación y los buenos resultados de síntesis, hacen que el uso de la librería de David Bishop para punto fijo sea la representación numérica más adecuada para el caso de estudio.

A continuación se muestra la descripción estructural para la implementación de tres controladores PID junto a sus respectivos bloques PWM y un core I2C para la comunicación serial entre la FPGA y el ESP32. La aritmética se resuelve utilizando punto fijo por medio de la librería sfixed, cuya palabra binaria está formada por 32 bits:

```
sfixed(16 downto -16)
```

1 bit de signo. 15 bits para la parte entera. 16 bits para la parte decimal.


I2C
CLK
CRT
INPUT
VCC
clk
crt
pid_out1[7..0]
pid_out2[7..0]
pid_out3[7..0]
debug1[7..0]
debug2[7..0]
debug3[7..0]
scl
sda
Byte1[7..0]
Byte2[7..0]
Byte3[7..0]
Byte4[7..0]
Byte5[7..0]
Byte6[7..0]
BIDIR
SCL
SDA
inst
PID_sfixed1
PID_sfixed2
PID_sfixed3
RST
reset
sp[7..0]
fb[7..0]
IN1
IN2
pid_out[7..0]
OUTPUT
in1
in2
in3
in4
in5
in6
inst12
inst13
inst14
PWM
pwm_out
pwm_in[7..0]
PWM1
PWM2
PWM3
inst1
inst2
inst3

10 CONCLUSIONES

Se presentó el diseño, la implementación y el análisis de sistemas de control sobre una FPGA Cyclone IV de Altera, empleando como caso de estudio un robot manipulador de 5 grados de libertad. Para el trabajo se sintetizo en hardware tres controladores PID, evaluando previamente el impacto sobre la respuesta en el dominio del tiempo y en el uso de recursos de hardware, para diferentes representaciones numéricas: enteros, punto fijo y punto flotante. Todo esto con el fin de obtener resultados aceptables sin caer al empleo de FPGAs costosas.

Gracias a la gran capacidad de cómputo que tienen las FPGA fue posible realizar en tiempo real, los cálculos de control de posición. Por otro lado, toda la información procesada fue visualizada en una página web, por medio de la implementación de un WebServer en un SoC ESP32.

El ESP32 emite una red Wifi de acuerdo al estándar 802.11 que al conectarse un cliente, por medio de un dispositivo Smart (Celular, Tablet, PC, etc.) se accede a la página web que contiene toda la información de los los PID como; señal de referencia, de realimentación y error para cada uno. La información se presenta en un widgets y se visualiza temporalmente en un gráfico cartesiano. De este modo se logra una interacción más amigable entre la máquina y el hombre.

A continuación se listan los objetivos cumplidos:

Objetivo General

- Diseñar un sistema de control de posición para un robot manipulador.
- Evaluar diferentes representaciones numéricas.
- Implementar los diseños sobre una FPGA usando VHDL.
- Crear un HMI.
- Generar documentación detallada para próximos proyectos.

Objetivo Particular

- Aplicar métodos de identificación de sistemas para modelado de plantas.
- Usar procedimientos para el ajuste de parámetros de PIDs.
- Simular lazos de control para análisis en el dominio tiempo y de la frecuencia.
- Poner a prueba las herramientas existentes para el uso de operaciones aritméticas con enteros y reales para HDL.
- Describir IP Cores (Intellectual Property Core) para I2C (Inter Integrated Circuit) y Floating Point necesarios para comunicación y cálculo de algoritmos. Además núcleos para modulación PWM (Pulse Width Modulation), Control PID y LCD (Liquid Crystal Display), entre otros.
- Empleo de los siguientes software para objetivos específicos:
  - MATLAB & Simulink - MathWorks.
  - SolidWorks CAD (Computer Aided Design) - Dassault Systemes
  - Altera Quartus II & modelSim
- Estudio de las siguientes herramientas de software
  - System Identification Toolbox - MATLAB - MathWorks
  - PID Controller for Simulink - MATLAB - MathWorks
  - Control System Toolbox Root Locus Design GUI - MathWorks
  - Generate Verilog and VHDL code for FPGA and ASIC designs using HDL-Coder – MathWorks
- Realizar PCB necesarias.

10.1 REFLEXIÓN

Existe una amplia gama de aplicaciones electrónicas, cada una de las cuales puede funcionar mejor con un enfoque diferente, que incluye microprocesadores, microcontroladores, unidades de procesamiento de gráficos, FPGA y dispositivos system on chip. Decidir cuál usar requiere examinar los requisitos y consideraciones de la aplicación bajo estudio.

Por ejemplo, cuando se trabaja con tecnología de vanguardia, los diseñadores deben tener en cuenta que los estándares y protocolos subyacentes todavía están evolucionando. Esto significa que los diseñadores deben poder responder rápida y eficientemente ante cualquier cambio en las especificaciones que estén fuera de su control.

Asimismo, necesitan la flexibilidad para responder ante los cambios futuros en los estándares y protocolos que ocurren después de que los sistemas se hayan implementado en el campo. Del mismo modo, también es necesario poder responder a errores inesperados en la funcionalidad del sistema o fallas en la seguridad del sistema, modificar la funcionalidad existente o agregar una nueva funcionalidad para extender la vida útil del sistema.

Si bien los SoC proporcionan el más alto rendimiento, esta ruta es costosa y requiere de mucho tiempo. Además, cualquier algoritmo implementado en la estructura del chip está esencialmente "congelado en silicio". Esta inflexibilidad inherente se convierte en un problema dadas las consideraciones indicadas anteriormente. Para encontrar el punto ideal entre rendimiento y flexibilidad óptimos, se requiere una ruta alternativa. Esa ruta a menudo es proporcionada por las FPGA, combinaciones de microprocesadores/microcontroladores y FPGA, o por FPGA que cuentan con núcleos rígidos de procesadores como parte de su estructura.

Es por ello que el presente trabajo intenta mostrar la gran flexibilidad que otorga el uso de las FPGA y que va mucho más allá del caso de estudio. Es solo un pretexto que pretende sembrar la curiosidad en los lectores (colegas) para animarlos a involucrarse en la implementación de estos dispositivos en sus proyectos. Son herramientas poderosas y en constante evolución que permiten al diseñador “fabricar sus chips en casa”, siendo esto una inmejorable herramienta en tiempos de carencia en cuanto a chips electrónicos.

11 REFERENCIAS

[1] Berardo, G. M. & Salvadeo P. A., “Controlador de Posición basado en FPGA para Robot Manipulador,” 24º Congreso Argentino de Control Automático: Área Estudiantil, Buenos Aires, Argentina, Octubre, 2014, pp. 1-6.

[2] Berardo, G. M. & Salvadeo P. A., “Consideraciones numéricas al implementar un sistema de control sobre FPGA,” XVI Reunión de Trabajo en Procesamiento de la Información y Control: Área Estudiantil, Córdoba, Argentina, Octubre de 2015, pp. 1-6.

[3] Bronw S. & Vranesic Z. Fundamentos de Lógica Digital con diseño en VHDL, GrawHill, 2008. XVI Reunión de Trabajo en Procesamiento de la Información y Control, 6 al 9 de octubre de 2015

[4] IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std. 754-2008).

[5] Ashenden P. J. & Lewis J., VHDL-2008: Just the New Stuff. Ch. 8: Standard Packages. Elsevier/Morgan Kaufmann, 2008.

[6] Simulink HDL Coder. [Online].

[7] Introducción a los Sistemas de Control - Ing. Ricardo Hernández Gaviño.

[8] Katsuhiko Ogata Sistemas de control Moderno.

[9] Descripción en VHDL de arquitecturas para implementar el algoritmo CORDIC.