**ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**

**Facultad de Ingeniería en Electricidad y Computación**

# IMPLEMENTACIÓN MEDIANTE FPGA, DE UN FILTRO DE KALMAN PARA REDUCIR EL EFECTO DEL RUIDO EN SENSORES DE TEMPERATURA

**PROYECTO DE TITULACIÓN**

Previo la obtención del Título de:

**Magister en Automatización y Control**

Presentado por:

Luis Eduardo Artieda Cruz

GUAYAQUIL - ECUADOR

Año: 2021

# DEDICATORIA

Este proyecto se lo dedico a Dios, a mi mamá y a mi papá ya que sin su amor y esfuerzo no hubiera sido posible lograrlo. A mi tía y a mi abuelita que siempre estuvieron conmigo cuando las necesité y a mi esposa, por su apoyo incondicional y consejos que me permitieron culminar con éxito este proyecto.

# AGRADECIMIENTOS

Mis agradecimientos al PhD Dennys Paillacho por su guía y apoyo a lo largo de la maestría, a la ESPOL y a sus honorables docentes que impartieron sus conocimientos y experiencias adquiridas a lo largo de sus carreras.

# DECLARACIÓN EXPRESA

“Los derechos de titularidad y explotación, me corresponde conforme al reglamento de propiedad intelectual de la institución; *Luis Eduardo Artieda Cruz* y doy mi consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Luis Eduardo Artieda Cruz

# COMITÉ EVALUADOR

Firmado electrónicamente por:
DENNYS FABIAN
PAILLACHO
CHILUIZA

.....

**PhD. Dennys Paillacho Chiluiza**

PROFESOR TUTOR

Firmado electrónicamente por:
VICTOR MANUEL
ASANZA ARMIJOS

.....

**M. Sc. Victor Asanza**

PROFESOR EVALUADOR

# RESUMEN

Los sensores están expuestos a perturbaciones ambientales que conllevan que sus datos crudos presenten errores que requieren ser mitigados a través de algoritmos para luego ser empleados en diferentes aplicaciones. Es por esta razón, que se han desarrollado algoritmos de alta eficiencia como el Filtro de Kalman, el cual mediante ecuaciones que se ejecutan de forma recursiva permiten reducir el error obteniendo datos más confiables.

Por tal razón, en este estudio se implementó el filtro de Kalman a un sensor de temperatura empleando una tarjeta de desarrollo FPGA, para lo cual se requirió en primera instancia duplicar los datos del sensor para luego ser usados en el algoritmo del filtro. En el algoritmo se calculan los diferentes parámetros del filtro de Kalman; estos parámetros fueron transmitidos vía serial para sus respectivos análisis.

Los resultados mostraron que el filtro implementado fue capaz de sintonizarse en 15 segundos aproximadamente y logrando mitigar las fluctuaciones de la señal del sensor, suavizando la señal tanto en escenarios donde existió un incremento o decremento de temperatura. La señal filtrada para una muestra en particular mostró que el error de estimación que se obtuvo fue de ±1.34 °C, a pesar que el sensor empleado no fue uno de precisión, el filtro fue capaz de realizar estimaciones eficientes.

**Palabras claves: sensor, algorithm, Kalman Filter, FPGA.**

# ABSTRACT

*The sensors are exposed to environmental disturbances that lead to their raw data presenting errors that need to be mitigated through algorithms and then used in different applications. For this reason that highly efficient algorithms such as the Kalman Filter have been developed, which through equations that are executed recursively allow to reduce the error obtaining more reliable data.*

*For this reason, in this study the Kalman filter was implemented in a temperature sensor using an FPGA development board, for which it was first required to duplicate the sensor data and then be used in the filter algorithm. In the algorithm the different parameters of the Kalman filter are calculated; these parameters were transmitted via serial for their respective analysis.*

*The results showed that the implemented filter was able to tune in approximately 15 seconds and was able to mitigate the fluctuations of the sensor signal, smoothing the signal both in scenarios where there was an increase or decrease in temperature. The filtered signal for a particular sample showed that the estimation error obtained was ± 1.34 ° C, although the sensor used was not a precision one, the filter was able to make efficient estimates.*



***Keywords: sensor, algorithm, Kalman Filter, FPGA.***

# ÍNDICE GENERAL

# ABREVIATURAS

$ADC$ : Analog Digital Converter

$CLK$ : clock digital

$EKF$ : Filtro de Kalman Extendido

$E\ x(\ )$ : Valor esperado

$E_t$ : Estimación actual

$E_{t\square 1}$: Estimación anterior

$E_{to}$ : Estimación inicial

$FK$ : Filtro de Kalman

$FPGA$: Field Programmable Gate Arrays

$K$ : Ganancia de Kalman

$LCD$: Liquid Cristal Display $Med$

: Mediciones

$Q$ : Varianza del ruido

$UART$ : Universal Asynchronous Receiver-Transmitter

$UKF$ : Filtro sin perfume

$VHDL$: Very High Speed Integrated Circuit Hardware Description Language $\bar{x}$

: Media

$\square$: Media de una distribución Gaussiana

$\square_{est}$ : Error de estimación

$\square_M$ : Error de medición

$\square$: Desviación estándar

$\square^2$: Varianza de una distribución Gaussiana

$\square_{Mo}$ : Error de medición

$\square_{est0}$ : Error de estimación

# SIMBOLOGÍA

mv: milivoltios

V: voltios

°C: grados centigrados

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# CAPÍTULO 1

## 1. INTRODUCCIÓN

A medida que el mundo de la electrónica ha evolucionado durante el transcurso del tiempo esto ha conllevado que las industrias evolucionen para estar a la vanguardia de las nuevas tecnologías. Esto ha generado que cada vez se desarrollen sistemas cada vez más complejos que requieren datos altamente fiables y de alta precisión, como son los datos provenientes de los sensores electrónicos.

Existen sensores que dependiendo de su tecnología son más susceptibles a perturbaciones ambientales como humedad, salinidad, campo electromagnético, etc., como por ejemplo los de temperatura. Como consecuencia, las lecturas crudas de los sensores incluyen un error aleatorio que impiden que sean empleadas directamente en diferentes sistemas donde se requiera una alta precisión, por lo que requieren ser atenuado.

Para lograr aquello, existen métodos de filtrado unos más eficientes que otros, que logran obtener mediciones más parecidas a las mediciones reales. Los filtros más empleados en la electrónica por su facilidad de implementación son los filtros activos (pasa alto, bajo, pasa banda, Notch) [1] pero al momento de filtrar en sensores se corre un alto riesgo en perder información relevante que para sistemas críticos pueden resultar altamente catastróficos.

Otro método empleado es el filtro promedio móvil [2], el cual se basa en tomar un número de muestras (***n***) para generar una media aritmética, si *n* es grande la precisión de este filtro mejora pero su respuesta es lenta, caso contrario, si *n* es pequeña la respuesta es rápida pero la precisión de la predicción es baja.

Por otra parte, uno de los filtros más empleados para el filtrado y estimación es el filtro de Kalman [3], se basa en un algoritmo recursivo y probabilístico que permite

realizar estimaciones basadas en mediciones anteriores con errores o incertidumbres que con el paso de cada iteración proporcionará una medición más precisa y cercana a la medición esperada. Dada a su alta eficiencia, este algoritmo actualmente es empleado en sistemas críticos relacionados a los sistemas de navegación, localización, informáticos, etc. [4] [5].

Por los antes mencionado, en este estudio se plantea filtrar y mejorar las mediciones de un sensor de temperatura aplicando la teoría del Filtro de Kalman de una dimensión, el cual será implementado en un sistema embebido con tecnología FPGA.

Con esta propuesta se pretende demostrar la eficiencia del filtro de Kalman a la hora de filtrar los datos del sensor de temperatura en tiempo real y comprobar su precisión como el tiempo de respuesta del mismo.

## 1.1 Identificación del problema

Dado que gran parte de los sensores que se emplean en distintas aplicaciones son vulnerables a perturbaciones externas [6] [7] que ponen en riesgo la integridad y precisión de la señal generada, esto sumado al ruido que son inducidos por los propios componentes electrónicos de los sensores, hace prácticamente inviable el uso de sus mediciones crudas en aplicaciones que requieren una alta precisión de datos, dado que se obtendrían resultados poco fiables y con una tasa de error muy alta.

## 1.2 Justificación

Dado al avance tecnológico, la tendencia en la actualidad es que la mayor parte de los procesos sean automatizados y con la menor intervención del ser humano. De esta manera se pretende minimizar los errores humanos a consecuencia de la repetitividad de las operaciones dentro de un proceso, como por ejemplo son el cansancio, esfuerzo físico, desconcentración, etc.

Para llevar a cabo lo antes mencionado, las compañías están empleando una gran cantidad de sensores y algoritmos para obtener una mejor producción con base a la precisión de las señales obtenidas. Como es el caso de [8], en el cual emplearon un filtro paso bajo Butterworth en una tarjeta FPGA Spartan 6 para reducir el ruido proveniente de un electrocardiograma. Por otra parte, en [9] realizaron un estudio que consistió en mitigar el ruido de un amplificador de señal mediante un filtro FIR el cual fue sintetizado en una tarjeta FPGA.

Es por esta razón, que para lograr tal precisión se requiere algoritmos de alta eficiencia, como es el filtro de Kalman, que permitan obtener datos confiables y útiles que permitan el desarrollo de tecnologías altamente productivas a partir de datos con alta incertidumbre.

En virtud a lo mencionado, el estudio propuesto pretende enfocarse en las señales provenientes de los sensores, dado a la vulnerabilidad con respecto a las

perturbaciones ambientales se plantea implementar el filtro de Kalman de una dimensión para filtrar las señales y que éstas sean reutilizadas por algoritmos multidisciplinarios.

## 1.3 Solución propuesta

Con el objetivo de mitigar el ruido generado de forma intrínseca por el sensor de temperatura o producido por alguna perturbación externa, se plantea diseñar e implementar un filtro de Kalman de una dimensión en un sistema embebido que permitirá reducir la presencia del ruido en las mediciones de un sensor de temperatura.

## 1.4 Objetivos

### 1.4.1 Objetivo General

- Implementar el Filtro de Kalman para reducir el error en la medición de un sensor de temperatura.

### 1.4.2 Objetivos Específicos

1. Desarrollar el código VHDL del conversor Analógico - Digital (ADC) y calcular la temperatura en grados Celsius.
2. Hallar los parámetros de inicialización del Filtro.
3. Implementar en VHDL de las ecuaciones del Filtro de Kalman.
4. Desarrollar en VHDL el módulo UART (RS-232) que realice la transmisión de la temperatura del sensor, la temperatura filtrada, la Ganancia de Kalman, el Error en la Estimación y el valor del convertidor ADC.
5. Comparar los resultados entre la señal original del sensor y la señal filtrada.

## 1.5 Metodología

En esta sección se detalla el escenario en que se va realizar nuestro estudio. Luego se describirán los materiales empleados para la implementación del filtro

de Kalman. Finalmente se especificará el proceso que se llevará a cabo detallando las diferentes etapas de desarrollo.

### 1.5.1 Descripción del Escenario

En este estudio, el escenario en el cual se va a desplegar nuestra implementación será dentro de una habitación, en la cual se va a realizar distintas perturbaciones empleando un artefacto electrónico que emana calor, tal como se observa en la Figura 1.1. Para simular el aumento de temperatura, el dispositivo electrónico se acercará al sensor para que éste detecte el aumento de temperatura.

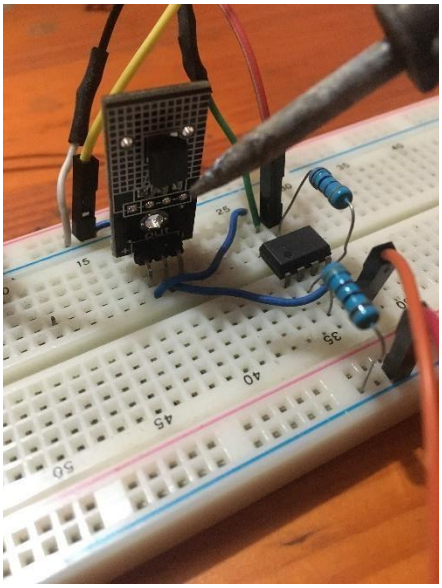

**Figura 1.1 Escenario de pruebas.**

### 1.5.2 Materiales

A continuación, se detallarán los materiales que se emplearán para llevar a cabo el estudio propuesto.

**Tabla 1.1 Lista de materiales usados en la implementación**

| Cantidad | Material |
|---|---|
| 1 | Amplificador operacional ua741 |
| 1 | Sensor LM35 |
| 2 | Resistencias de 1KΩ |
| 1 | Pila de 9 voltios |
| 1 | Fuente de poder de 12 voltios |

| | |
|---|---|
| 1 | Tarjeta Spartan 3e - Xilinx |
| 1 | Cable serial RS-232 |
| 1 | Laptop |

### 1.5.3 Proceso de implementación del Filtro de Kalman

El proceso de implementación utilizado para este estudio es el que se muestra en la Figura 1.2. Este se encuentra dividido en 5 fases, a continuación se detallarán cada una de ellas.

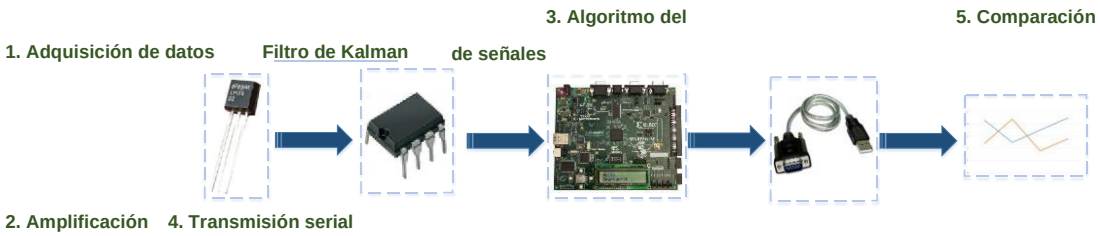


**Figura 1.2 Proceso de implementación del Filtro de Kalman.**

#### 1. Adquisición de datos

En esta fase se empleará el sensor de temperatura LM35, el cual será alimentado por una pila de 9v. Los datos obtenidos del sensor vienen expresados en milivoltios, por lo cual requieren ser convertidos a voltajes mediante la siguiente ecuación:

$$T = 100 * Vout \quad (1)$$

Donde *T* es la temperatura expresada en °C y *Vout* es el voltaje del sensor Lm35.

Para llevar a cabo esto, se empleará el siguiente diagrama de conexión mostrado en la Figura 1.3, donde se requiere energizar el sensor LM35 con una batería de 9v a través de los pines (1) y (3) y por el pin (2) se medirá la salida del sensor en mv.

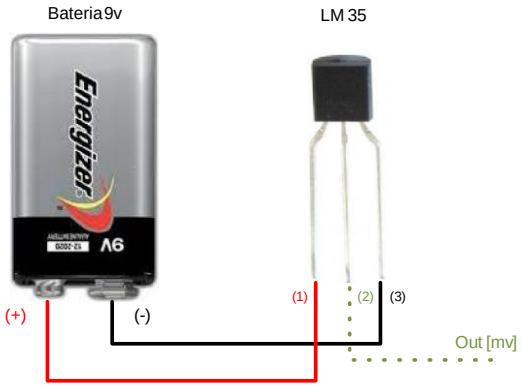


**Figura 1.3 Diagrama de conexión del sensor LM35.**

## 2. Amplificación

En la fase de amplificación se empleará el operacional ua741, el cual se encuentra configurado de tal manera que duplique la señal de entrada. Esto se lo realiza para que el voltaje amplificado se encuentre dentro del rango de operación del convertidor analógico- digital de la tarjeta de desarrollo Spartan 3E, el cual va desde 0.40 v hasta el 2.90 v y así pueda ser configurado dentro del algoritmo.

## 3. Algoritmo del filtro de Kalman

Dentro de esta fase se encuentra dividido por varios módulos, debido a que la programación en el FPGA se encuentra estructurada de esa manera.

Como primer paso se procede a realizar la lectura de las mediciones provenientes del sensor por medio del ADC, para esto se implementó un módulo en VHDL el cual convierte los voltajes en una palabra digital de 14 bits, la lectura de los voltajes se lo realizará cada segundo.

Como segundo paso, la palabra digital pasará por un módulo el cual se encargará de realizar los cálculos matemáticos pertinentes para la conversión a temperatura en grados centígrados.

Como tercer paso, la temperatura calculada pasará por un módulo que realizará el algoritmo del filtro de Kalman, del cual se obtendrá la temperatura estimada o filtrada, la ganancia de Kalman y el error en la estimación.

Adicionalmente, los valores de la temperatura, del ADC y de la temperatura estimada se visualizarán en la pantalla LCD de la tarjeta Spartan 3E.

**4. Transmisión serial**

En esta fase los valores calculados previamente junto con la temperatura del sensor y el valor del convertidor ADC serán trasmitidos vía serial por un módulo UART a un computador donde se registrarán los datos en un archivo .txt.

**5. Comparación de señales**

A partir de los datos obtenidos, se hará el análisis y comparación entre la señal filtrada y la original.

## 1.6 Alcance

Al finalizar la implementación del algoritmo del Filtro de Kalman se espera evidenciar el filtrado de la señal del sensor de temperatura, es decir, obtener una señal más estable donde exista una repetitividad de datos y con una varianza (ruido) menor que la señal original. Además, que el filtro sea capaz de detectar las variaciones rápidas de temperatura y lograr así poder estimar su valor siguiente en el menor tiempo posible.

Tal como se explicó anteriormente, el tema propuesto será implementado en su totalidad en hardware y software, es por esta razón que no se va a presentar alguna simulación previa a la implementación.

# CAPÍTULO 2

# ESTADO DEL ARTE

## 2.1 Introducción

Dentro de la comunidad científica, uno de los desafíos más complejos para llevar a cabo es el modelamiento matemático de los distintos sistemas dinámicos aplicados a las diversas áreas de investigación. La forma más tradicional de llevarlo a cabo es empleando las leyes físicas convencionales que describen el comportamiento dinámico del objeto o sistema en estudio. Pero desafortunadamente, este método no resulta tan efectivo cuando el sistema se vuelve cada vez más complejo, dado que estas leyes no consideran efectos ambientales del entorno o intrínsecas del sistema.

Por estas razones, un modelo matemático suele contener variables, restricciones y fenómenos predominantes del sistema pero a su vez resulta complicado o a veces es imposible modelar señales con presencia de ruido. Por ejemplo, durante el movimiento de un automóvil, es posible modelar la velocidad del auto con ayuda de las ecuaciones del movimiento rectilíneo uniformemente variado pero es imposible determinar o modelar la fuerza del aire que opone resistencia al movimiento del vehículo.

Otro factor a considerar es la forma en cómo se recolectan los datos del sistema, es decir, siguiendo con el ejemplo del automóvil, para medir los diferentes parámetros como velocidad, peso, etc., se emplean sensores que a pesar que sean de gran calidad generan errores en su medición lo que generará que la precisión del modelo matemático se vea afectado.

Con lo descrito anteriormente, los procesos que involucran mediciones aleatorias y que el proceso subsiguiente depende del anterior o que el análisis que se lleva a cabo se lo realice en términos probabilísticos se lo denominan modelos

estocásticos, los cuales requieren fundamentalmente emplear técnicas de filtrado adecuadas que garanticen la optimización del modelo empleado. Estas técnicas son vistas como algoritmos recursivos que permiten estimar variables ocultas del modelo a partir de las variables conocidas que incluyan una alta incertidumbre o ruido blanco Gaussiano, logrando así tener una estimación lo más óptimo posible.

El algoritmo más popular empleado hasta la actualidad y que ha permitido grandes avances en el mundo científico es el Filtro de Kalman [10], fue propuesto por Rudolf Kalman en 1960. El filtro de Kalman permite realizar estimaciones futuras con base a predicciones anteriores logrando así mejorar su precisión en cada iteración.

Hoy en día, el filtro de Kalman dado a su versatilidad y robustez es empleado en numerosas aplicaciones como en la navegación, seguimiento de objetivos con radar [11], en sistemas informáticos y en las comunicaciones inalámbricas. Además es altamente usado en el campo de las imágenes, es decir, en el procesamiento de señales digitales, en reconocimiento de imágenes, identificación de formas, etc [12]. Cabe mencionar que al paso del tiempo, se han desarrollado varios filtros de Kalman, pero para este estudio se centrará en el análisis y desempeño del filtro de Kalman en una dimensión.

## 2.2 Filtro de Kalman

### 2.2.1 Definición

El filtro de Kalman de una dimensión es un algoritmo recursivo que se ejecuta en tiempo real basado en el teorema de Bayes [13], es decir, el filtro para realizar la estimación a futuro $x_k$ requiere conocer la estimación anterior $x_{k-1}$, en términos de probabilidad, la probabilidad de la nueva estimación dependerá de la probabilidad ocurrida anteriormente. El filtro de Kalman de una dimensión se encuentra gobernado por tres ecuaciones matemáticas que permiten realizar la estimación y la corrección de la estimación. Con estas ecuaciones se podrán calcular la ganancia de Kalman ( $K$ ), la estimación ( $E_t$ ) y el error en la estimación ($\square_{est}$ ).

### 2.2.2 Arquitectura

Tal como se mencionó anteriormente, el filtro de Kalman de una dimensión emplea tres ecuaciones [14], las cuales se encuentran divididas en tres fases fundamentales que permite realizar la estimación más óptima por parte del filtro.

La primera fase hace mención a la medición, en ésta se recoge la medición del sistema, la segunda fase representa a la actualización de estado y la tercera se refiere a la predicción que realiza el filtro.

En la segunda fase se encuentran incluidas las tres ecuaciones del filtro. La primera ecuación hace referencia al cálculo de la ganancia de Kalman, la cual está dada por la siguiente ecuación:

$$K = \frac{\sigma_{est}}{\sigma_{est} + \sigma_M} \tag{2.1}$$

Donde $K$ es la ganancia de Kalman; $\sigma_{est}$ es el error en la estimación y $\sigma_M$ es el error en la medición.

A partir del cálculo de la ganancia de Kalman, se procede a usar la segunda ecuación, la cual permite calcular el valor de la nueva estimación.

$$E_t = E_{t-1} + K(Med - E_{t-1}) \tag{2.2}$$

Donde $E_t$ representa el valor de la estimación actual; $E_{t-1}$ es la estimación calculada en un instante previo y $Med$ es el valor de las mediciones.

Partiendo de la estimación calculada se usará la última ecuación del filtro, la cual permite calcular el error de la medición, ésta se encuentra dada por la siguiente ecuación:

$$\square_{est} \square \square(1\ K\ E)_{t\square 1} \qquad (2.3)$$

Donde $\square_{est}$ representa el error en la estimación.

Partiendo de lo mencionado anteriormente, en la Figura 2.1 se muestra un diagrama donde se detallan las fases del algoritmo. En este se aprecia que existe una fase previa antes de la primera fase, la cual se ejecuta una sola vez, esta se la conoce como la inicialización del filtro de Kalman.

Esta fase es muy importante dado que se le indica al filtro los valores de partida tanto para la estimación como el error de la estimación para que de esta manera se pueda realizar la primera iteración. Estos valores se los calculan de forma experimental. La estimación inicial se la genera registrando *n* datos y con ayuda de estadística descriptiva se procede a calcular el valor esperado $E\ x( )$ de dichas mediciones.

Otra técnica rápida de calcular la estimación inicial, es calculando la media $\bar{x}$ de las *n* mediciones. Ésta técnica no es muy recomendada porque no siempre la media de las mediciones coincide con el valor esperado. Por otro lado, para calcular el error en la medición, una de las formas más directas es observar si dicha información se encuentra detallada en los manuales del fabricante, por ejemplo cuando se usa un sensor, el fabricante suele indicarlo como error del sensor.

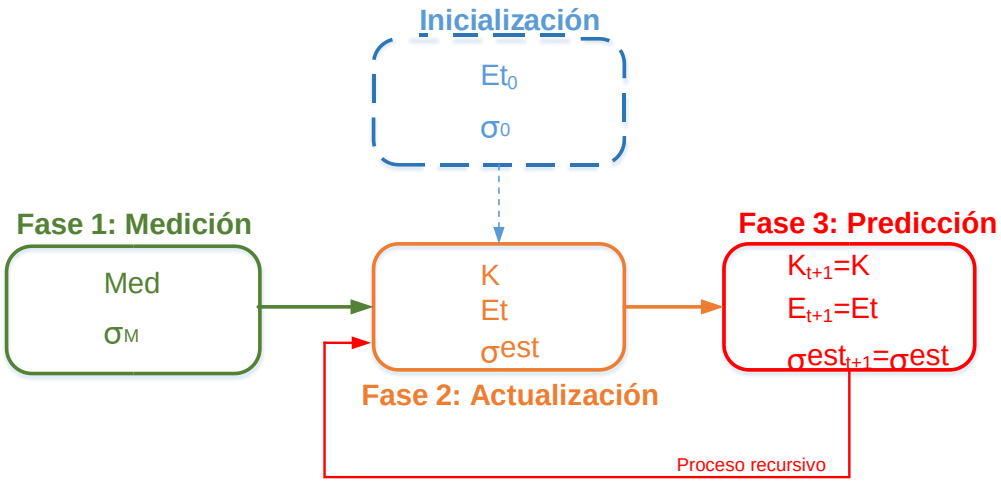


**Figura 2.1 Descripción del algoritmo del filtro de Kalman** [15]**.**

La fase 1 corresponde a las entradas del sistema (Mediciones y el Error de la medición). Cada vez que se genere una iteración del algoritmo, estos valores serán la entrada para la fase 2, junto con los datos de la estimación previa y el error de la medición estimada previa se procederán a calcular los nuevos parámetros del filtro y de esta manera se actualizan los parámetros en mención.

Las salidas de la fase 2 serán la entrada para la fase 3, la de predicción, para este tipo de filtro el modelo se lo considera constante, esto quiere decir, que para la primera iteración los datos de inicialización se los considera como la salida de la fase 3 y para el resto de iteraciones las salidas de la fase serán la ganancia de Kalman, la estimación y el error de la medición estimada calculadas en la fase 2.

### 2.2.3 Parámetros del Filtro de Kalman

Tal como se mencionó en el inciso anterior, el filtro de Kalman de una dimensión se encuentra constituido de tres parámetros que al operar conjuntamente logra que el filtro pueda realizar la estimación más óptima.

#### 1. Ganancia de Kalman

La ganancia de Kalman se la define como un peso que se lo otorga a la estimación como al error en la estimación para poder obtener una nueva estimación. Este parámetro sirve para para corroborar que el filtro de Kalman funciona correctamente, es decir, el valor debe ir disminuyendo en cada iteración hasta que el valor se estabilice. Los valores de la ganancia van desde $0 \leq K \leq 1$.

En la figura 2.2 se puede observar el resultado de obtener una ganancia cercana a 1. Esto ocurre cuando se obtiene un error en la medición baja con respecto al error estimado previamente. Tal como se aprecia, la nueva estimación estará más cerca de la medición, lo cual es lo que no se desea dado que con el filtro se pretende obtener una estimación más cercana al valor real.

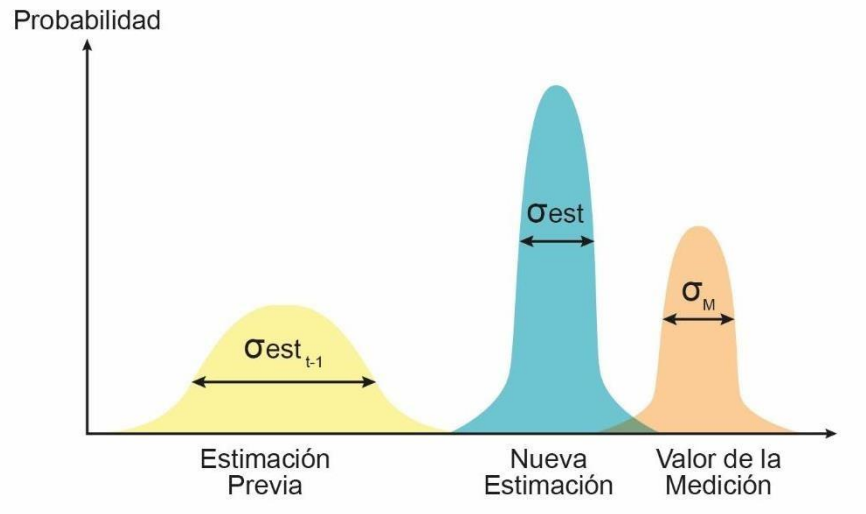


**Figura 2.1 Ganancia de Kalman cuando K≈1.**

Por otra parte, en la figura 2.3 se observa el comportamiento cuando se obtiene una ganancia cercana a 0. A diferencia del caso anterior, este se obtiene cuando se obtiene un error de medición alto con respecto al error estimado previo, por tal razón la nueva estimación estará más cerca de la estimación previa.

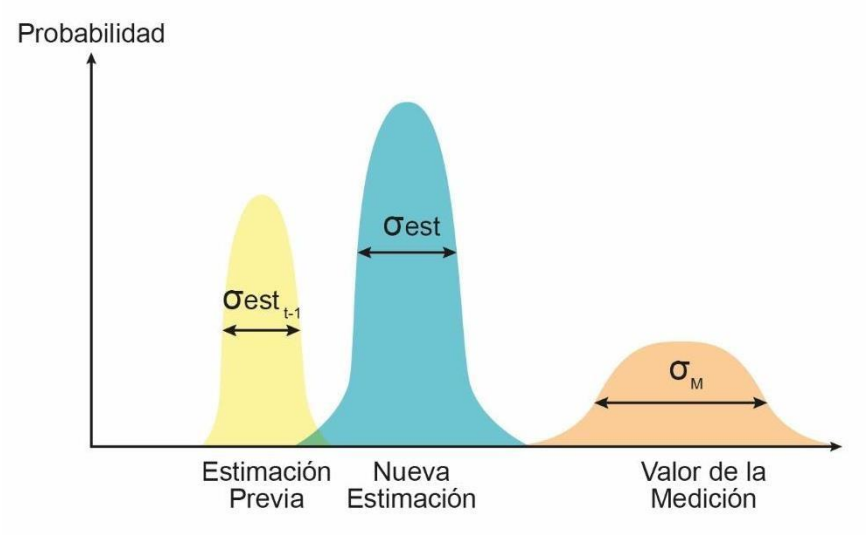


**Figura 2.3 Ganancia de Kalman cuando K□0.**

## 2. Estimación

La estimación es el valor de la variable oculta que calcula el filtro de Kalman. Para esto emplea la medición cruda del sensor, la estimación previa y la ganancia de Kalman.

Al hablar de estimación, se debe también mencionar dos términos relevantes como son exactitud y precisión. La exactitud hace mención a que tan alejados se encuentran las estimaciones con respecto al valor real; mientras que la precisión se refiere a que tan alejados se encuentran una estimación con respecto a otra.

Como se puede apreciar en la figura 2.4 se muestra tres escenarios en la que se logra evidenciar las diferencias entre cada uno de estos conceptos. En el primer escenario se puede observar que las diversas estimaciones se encuentran dispersas entre sí y alejados del valor real, para tal caso se puede aseverar que la estimación tiene baja exactitud y baja precisión. En el segundo escenario, en cambio se puede observar que las estimaciones se encuentren alejados del valor real pero se encuentran cercanas entre sí, por lo cual se dice que la estimación tiene baja exactitud pero alta precisión y en el tercer escenario se aprecia que las estimaciones se encuentran cerca del valor real y cercanas entre sí, con lo que se concluye que la estimación tiene alta exactitud y alta precisión.

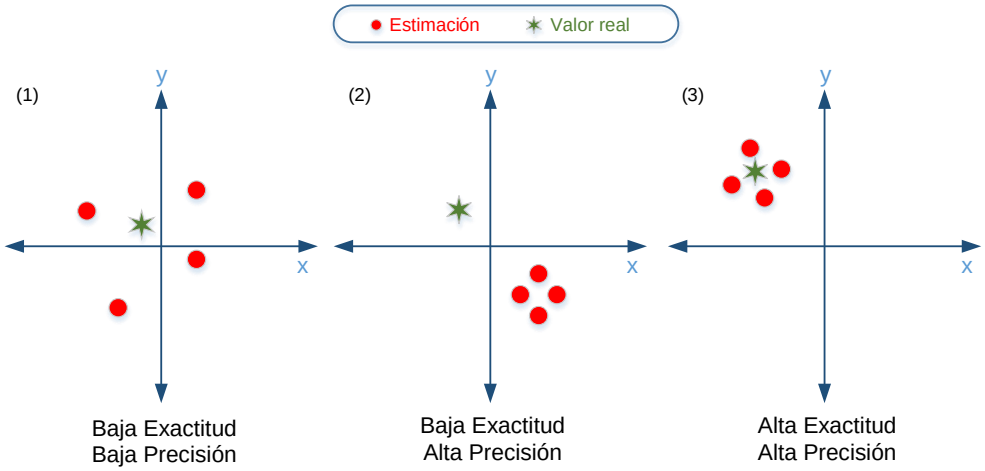


**Figura 2.4 Definición de estimación, exactitud y precisión** [15]**.**

### 3. Error en la estimación

Cada vez que el filtro realice la estimación, éste calcula el error de dicha estimación. Éste parámetro al igual que la ganancia de Kalman permite determinar si el filtro está actuando de forma correcta, es decir, el error debe ir disminuyendo en cada iteración, caso contrario, el algoritmo presenta algún error.

En otras palabras, la relación entre el error y la ganancia de Kalman es inversamente proporcional, es decir, cuando el error es grande la ganancia de Kalman será baja y por ende la velocidad de respuesta del filtro será lenta. Sin embargo, cuando el error es bajo la ganancia de Kalman será alta por lo tanto la velocidad de respuesta será rápida.

## 2.3 Ruido Blanco Gaussiano

Una de las dificultades al emplear distintos sensores, es la presencia del ruido, el cual se lo define como una señal aleatoria que no guarda ninguna relación estadísticas entre ellas, es decir, que no existe una forma de conocer el comportamiento a futuro a pesar de conocer su valor en el tiempo t.

Existe una relación estrecha entre el ruido blanco Gaussiano y la función de densidad de probabilidad (PDF), dado que cualquier valor de la medición sigue la PDF, formando así una campana conocida como distribución Gaussiana con media μ y varianza $\sigma^2$ [16].

Por tal motivo, se asume que la incertidumbre de las mediciones se distribuye siguiendo la PDF, con lo cual el Filtro de Kalman contempla la distribución normal dentro de su algoritmo.

### 2.3.1 Modelamiento del Ruido Gaussiano

El modelamiento para el presente estudio se empleó matemáticas descriptivas para determinar la varianza del ruido del sensor y este valor permitirá actualizar la fase de predicción, es decir, el valor del error estimado se le sumará el valor de la varianza, tal como se muestra en la ecuación 2.4

$$\square_{est} \square est_{t\square 1} \square Q \qquad (2.4)$$

Donde $Q$ representa la varianza del ruido.

## 2.4 Aplicaciones del Filtro de Kalman

Durante mucho tiempo, la comunidad científica ha empleado el filtro de Kalman dado a su alta fiabilidad y resultados exitosos. Esto ha conllevado que el filtro haya sufrido modificaciones sobre el filtro básico, como lo son el Filtro de Kalman Extendido (EKF) y el Filtro sin perfume (UKF).

El filtro de Kalman ha permitido el desarrollo de proyectos que han marcado la historia de la humanidad, como es el caso del proyecto Apollo [14]. En 1960 emplearon el filtro para la estimación de la trayectoria desde la Tierra hacia la luna

y viceversa con naves espaciales tripuladas, lo que representaba que el filtro funcione a la perfección para salvaguardar la vida de los tripulantes.

Por otro lado, el filtro permitió el desarrollo de nuevas tecnologías en plena Guerra fría, específicamente en el ámbito aeroespacial, dado que en el 1957 la Unión Soviética logró realizar su primer lanzamiento de un satélite artificial. Además, el filtro se ha empleado en el ámbito de la medicina, como este estudio publicado en el 2018 [17], en el cual emplearon el filtro de Kalman para determinar si la medicina aumentaba o disminuía el número de convulsiones por día del paciente.

En [18] implementaron un algoritmo de estimación en FPGA para nano partículas levitadas dando como resultado un 20% de reducción de movimiento residual

Tal como se puede apreciar, la versatilidad del filtro y sus bondades permite que sean usadas en áreas muy delicadas y críticas lo que lo hace ser unos de los algoritmos más eficaz y utilizados hasta la actualidad.

### 2.4.1 Estimación de Temperatura

Una de las aplicaciones donde más se emplea el filtro de Kalman es en la estimación de la temperatura ambiente en interiores como en exteriores, de igual manera para estimar la temperatura de las baterías de litio, como en la predicción del clima, etc.

Por ejemplo en este estudio [19] emplearon el filtro de Kalman para mitigar el ruido y las perturbaciones de las mediciones de temperatura proveniente de un transductor electrónico y así aislar las perturbaciones del sistema para cuantificar el ruido.

En otro estudio [20], proponen un método para recolectar y procesar datos meteorológicos complementados con un algoritmo desarrollado en Python

que mejora la precisión de la estimación del pronóstico a corto plazo. A partir de esto, usan el filtro de Kalman para reducir el error de las estimaciones del algoritmo implementado.

En [21] proponen un filtro de Kalman basada en un modelo para estimar la temperatura en las celdas de una batería de Litio.

# CAPÍTULO 3

# IMPLEMENTACIÓN DEL FILTRO DE KALMAN

## 3.1 Descripción del Escenario de Medición

En este trabajo, el escenario de medición se desenvuelve en una habitación acondicionada. Para lograr simular las variaciones de temperatura de una forma más ágil se empleó un dispositivo electrónico que emana calor a medida que la temperatura en su interior incrementa. Con esto, se logra verificar el desempeño del Filtro de Kalman - FK con respecto a las variaciones de temperatura.

El hardware empleado para la implementación del filtro de Kalman es el que se muestra en la Figura 3.1. Este está conformado por una tarjeta de desarrollo Xilinx Spartan 3E el cual cuenta con un FPGA modelo XC3S500E. Además posee un convertidor Analógico - Digital (ADC) modelo LTC6912 de 2 canales con una resolución de 14 bits compatible con el protocolo de comunicación SPI. El ADC se encuentra conectado con un circuito pre-amplificador el cual contiene el integrado UA741. Por otra parte, la tarjeta presenta un puerto serial RS232 (DB9) que permitirá la transmisión de datos hacia el computador para poder procesar los datos y además contiene una pantalla LCD de 2x16 caracteres en el cual se apreciarán los datos de interés.

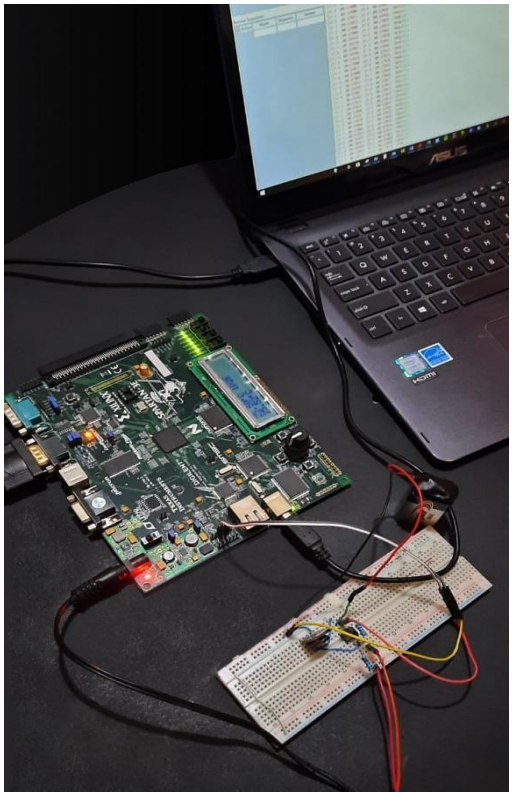

**Figura 3.1 Hardware usado para la implementación del FK.**

## 3.2 Descripción de la Arquitectura de Programación

La implementación del filtro de Kalman fue desarrollado bajo el ambiente de desarrollo ISE Project Navigator de Xilinx. El método de programación se encuentra basado por módulos lo que facilita el desarrollo del mismo. El módulo principal se encuentra definido como “MAIN”, tal como se aprecia en la Figura 3.2, dicho módulo contiene las entradas y salidas de la arquitectura del algoritmo implementado. A continuación se detallan las funciones de cada una de ellas.

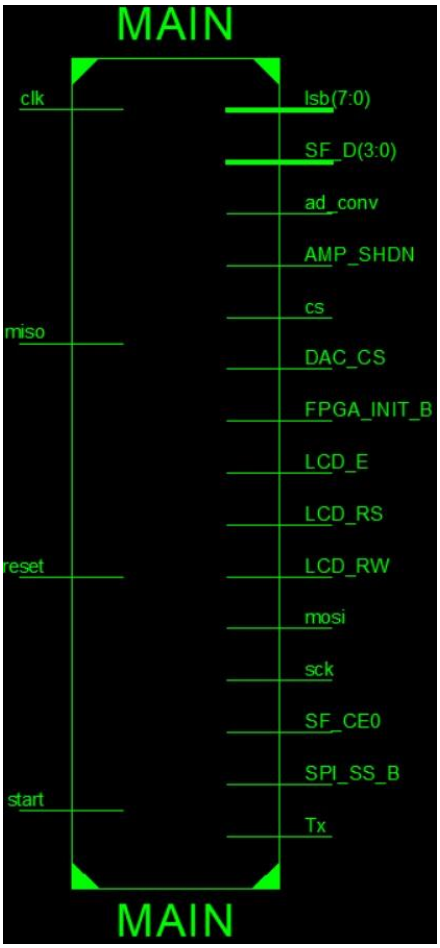


**Figura 3.2 Módulo "MAIN" de la arquitectura.**

**Entradas:**

**Clk**: La frecuencia seleccionada es de 50 Mhz, la cual permitirá sincronizar todos los módulos del algoritmo.

**Start:** Esta señal se encuentra ligada al switch 1 de la tarjeta, al estar en 'ON' dará inicio a la ejecución del algoritmo.

**Reset:** Permitirá realizar un reinicio al algoritmo.

**Miso:** La señal contendrá la conversión por parte del convertidor Analógico Digital.

**Salidas:**

**SPI_SS_B, DAC_CS, SF_CE0, FPGA_INIT_B, AMP_SHDN:** De acuerdo a la hoja técnica de la tarjeta de desarrollo, el bus SPI debe ser desconectada del resto de periféricos, tomando los valores de la Figura 3.3.

| Signal | Disabled Device | Disable Value |
|---|---|---|
| SPI_SS_B | SPI Serial Flash | 1 |
| AMP_CS | Programmable Pre-Amplifier | 1 |
| DAC_CS | DAC | 1 |
| SF_CE0 | StrataFlash Parallel Flash PROM | 1 |
| FPGA_INIT_B | Platform Flash PROM | 0 |

**Figura 3.3 Desactivación del bus SPI** [22]**.**

**cs, mosi, ad_conv, sck:** Representan las señales de control que forman parte del protocolo SPI que requiere el ADC para que pueda funcionar correctamente.

**lsb[7:0]:** Al realizar la conversión por parte del ADC, se podrán visualizar los 8 bit menos significativos en los leds de la tarjeta. Esto es un medio de comprobación del algoritmo del ADC.

**LCD_E, LCD_RS, LCD_RW, SFD[3:0]:** Representan las señales de control que permiten controlar el LCD, donde la señal SFD[3:0] contiene el dato o carácter que se muestra en el LCD.

**Tx:** Es la señal del UART, por el cual se transmitirán los parámetros del filtro de Kalman, la temperatura medida y el valor del ADC.

### 3.2.1 Etapa pre-amplificadora

La tarjeta Spartan 3E incluye un pre amplificador LTC6912-1 y un convertidor ADC LTC1407A-1, el análisis se centrará en el pre amplificador porque se requiere que el voltaje analógico se encuentre en el rango indicado. En la Figura 3.4 se observa una gráfica lineal que va desde los

400 mV hasta los 2.90 V, esto quiere decir, que para valores inferiores de 400 mv no será aceptado por el pre amplificador.

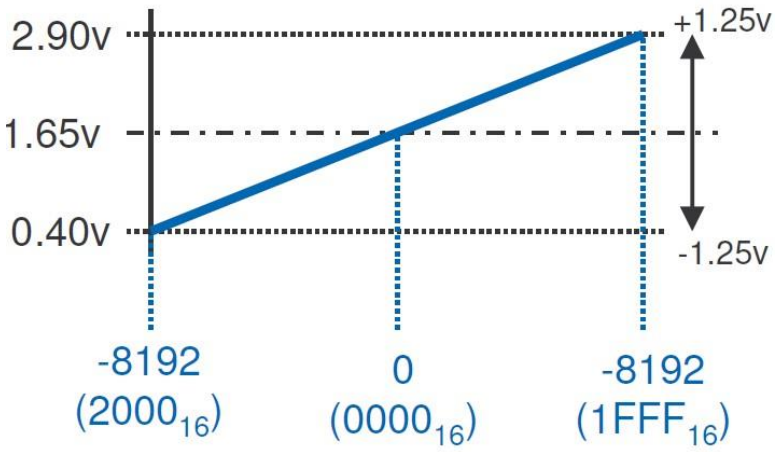


**Figura 3.4 Rango de la entrada analógica** [23]**.**

A partir de esto, el ambiente donde se realizan las pruebas la temperatura ronda entre los 22°C y 30°C por ende el voltaje esperado por el sensor tendría que estar entre los 200 mv y 300 mv. Como éste valor no se encuentra dentro del rango permitido por el pre amplificador, se empleó la etapa pre-amplificadora, la cual contiene implementado el circuito de la Figura 3.5, el cual permitirá duplicar la señal del voltaje del sensor.

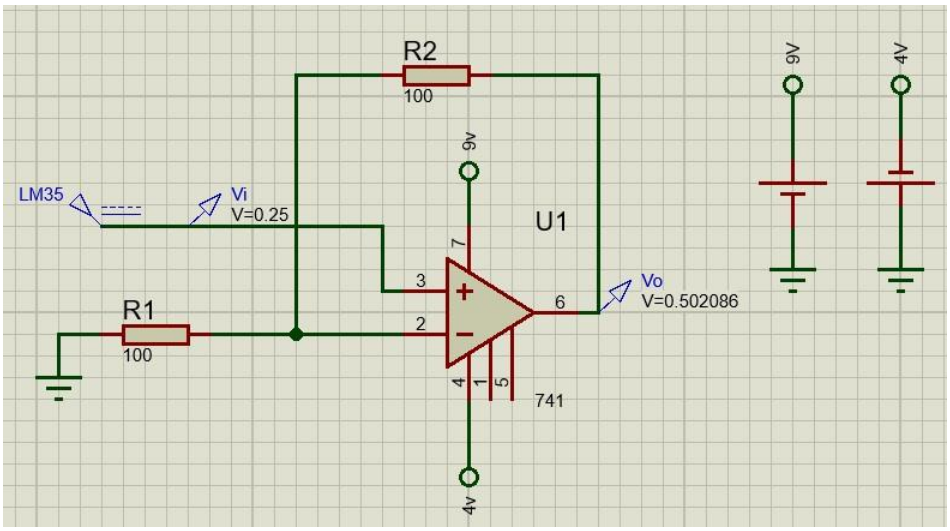


**Figura 3.5 Etapa pre-amplificadora.**

En la figura se puede apreciar la simulación de un escenario, es decir, cuando el sensor LM35 genere un voltaje de 250 mv a través del amplificador ua741 y el juego de resistencias de 100 ohmios el voltaje de salida será de 500 mv. Con esta configuración se garantiza que el voltaje

generado por el sensor se encuentre dentro del rango del pre amplificador. Cabe mencionar que una vez que se realice todo el proceso de conversión por el ADC a nivel software se realizará la división para 2 para obtener el voltaje inicial y por ende la temperatura real.

### 3.2.2 Etapa de conversión Analógica - Digital (ADC)

El módulo CONTROLADOR_ADC será el encargado de realizar las mediciones del sensor LM35 cada segundo a través de la señal de entrada En_Delay. Como se puede apreciar en la Figura 3.6 el resto de señales de salidas y entradas son las mismas del módulo “MAIN”.

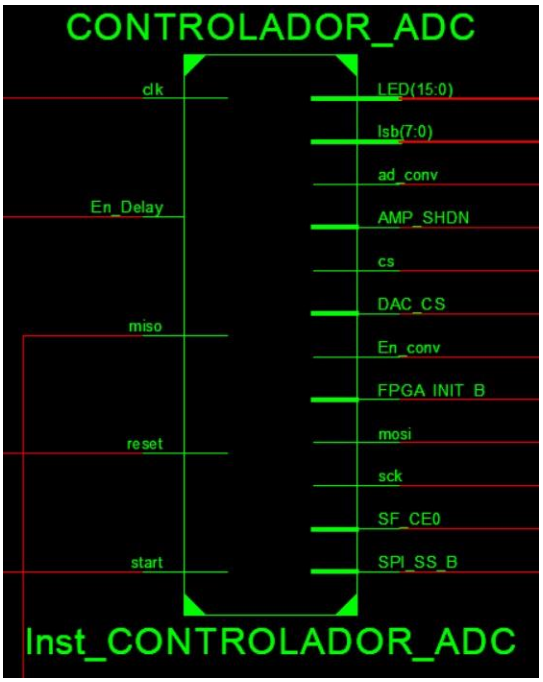


**Figura 3.6 Módulo “CONTROLADOR_ADC”.**

El controlador para el ADC fue desarrollado con base en la hoja técnica de la tarjeta Spartan 3E, en el cual se puede apreciar el protocolo de comunicación SPI del pre amplificador y del convertidor ADC. Lo primero que se debe programar es el pre amplificador cumpliendo el protocolo detallado en la Figura 3.7. El protocolo empieza cuando AMP_CS pasa de alto a bajo y cuando esto ocurra debe generarse 8 ciclos de reloj con una frecuencia de 50 Mhz a través de la señal SPI_SCK. Paralelamente a través de la señal SPI_MOSI se deberá enviar un comando de 8 bits en donde se especifica la ganancias de los 2 canales del pre amplificador, el comando

enviado es “10001000”, una vez enviado el comando la señal SPI_SCK volverá a alto, dando por terminado la configuración del pre amplificador.

Una vez enviado el comando, se deberá programar el convertidor ADC siguiendo el diagrama de la Figura 3.8. Para esto la señal AD_CONV deberá pasar de alto a bajo, a continuación se deben generar 34 ciclos de reloj, a partir del 3er ciclo de reloj el ADC empezará a realizar las conversiones, una vez ocurrido esto la señal AD_CONV volverá a alto dando por terminado la etapa de conversión. Este algoritmo se volverá a ejecutar cuando la señal Delay tome el valor de ‘1’, caso contrario el algoritmo quedará en estado de espera.

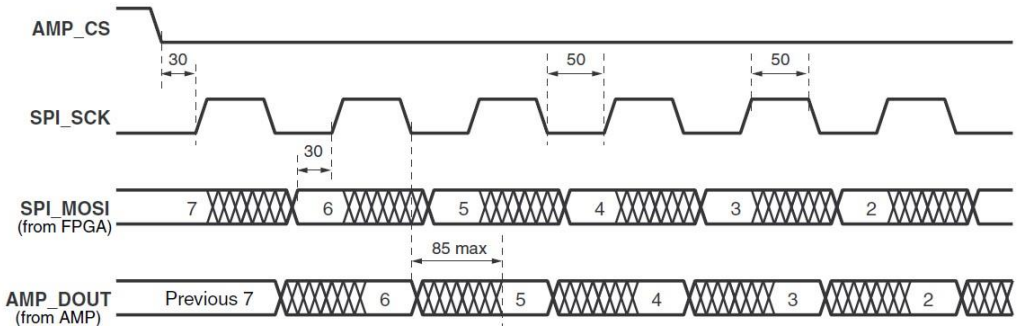


**Figura 3.7 Protocolo SPI del pre amplificador programable** [22]**.**

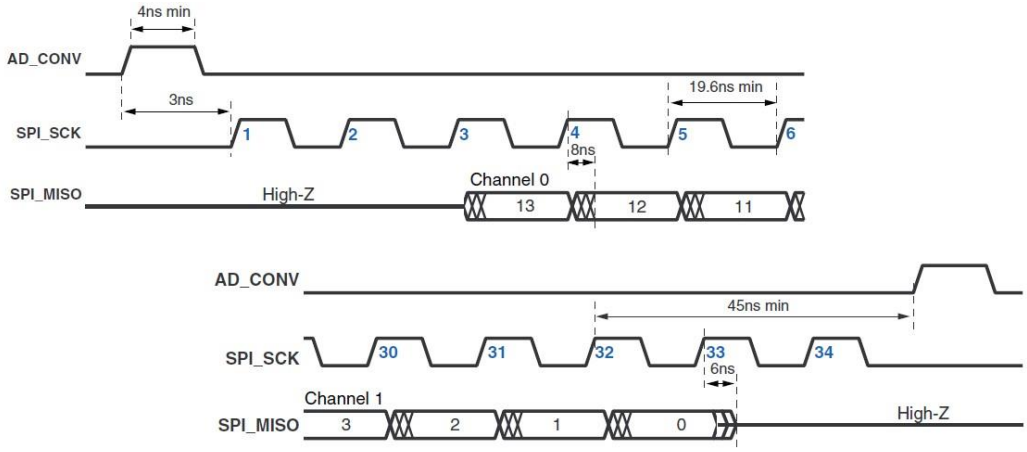


**Figura 3.8 Protocolo SPI del convertidor ADC** [22]**.**

### 3.2.3 Etapa de medición

En esta etapa se detallarán los módulos empleados y las técnicas para realizar los cálculos para determinar la temperatura del sensor y los parámetros del filtro de Kalman.

### 3.2.3.1 Cálculo de la temperatura del sensor

Para realizar el cálculo de la temperatura se debe tomar en cuenta la fórmula matemática del ADC detallado en la hoja técnica, como se muestra en la ecuación 3.1.

$$Salida\ Digital = \frac{(V_{in} - 1.65) * Gain}{1.25} * 8192 \quad (3.1)$$

En la fórmula la variable desconocida es $V_{in}$, mientras $Gain$ y la $Salida\ Digital$ son conocidas. Como se lo explicó en la sección 3.2.2 el comando enviado hace referencia a la ganancia del pre amplificador, basado en la hoja técnica el valor asignado para el comando seleccionado es de -1, como se muestra en la figura 3.9.

| $A_3$ | $A_2$ | $A_1$ | $A_0$ | |
|---|---|---|---|---|
| $B_3$ | $B_2$ | $B_1$ | $B_0$ | Gain |
| 0 | 0 | 0 | 1 | -1 |
| 0 | 0 | 1 | 0 | -2 |
| 0 | 0 | 1 | 1 | -5 |
| 0 | 1 | 0 | 0 | -10 |
| 0 | 1 | 0 | 1 | -20 |
| 0 | 1 | 1 | 0 | -50 |
| 0 | 1 | 1 | 1 | -100 |

**Figura 3.9 Valores de ganancias de acuerdo al comando seleccionado** [23]**.**

A partir de lo mencionado es importante indicar que el valor de la $Salida\ Digital$ es una palabra digital de 14 bits del ADC. Para

convertirlo a un valor entero se desarrolló en VHDL un módulo que contiene el algoritmo de 'binario a BCD' como se muestra en la Figura 3.10. La señal Dato(15:0) contiene la palabra binaria del ADC y el algoritmo se encarga de generar 4 señales de 8 bits que representan cada dígito del valor entero.

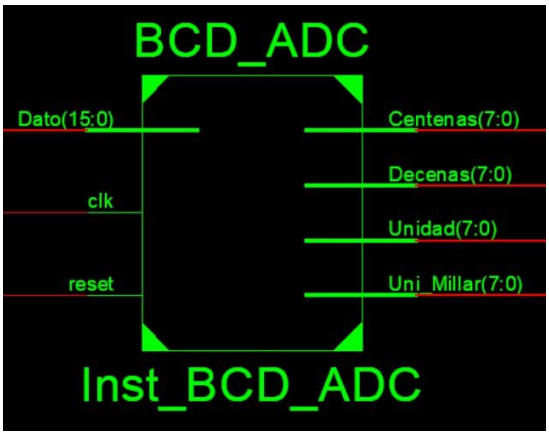


**Figura 3.10 Módulo BCD_ADC.**

Una vez obtenido el valor entero se procede a emplear el IP Core Floating-Point, éste permite seleccionar la opción *Fixed to Floating,* es decir, punto fijo a punto flotante, con lo cual se obtiene un valor equivalente en hexadecimal con 64 bits de precisión, como se muestra en la Figura 3.11, donde a(15:0) es la palabra digital, ce es el habilitador y result(63:0) es el resultado obtenido del IP CORE.

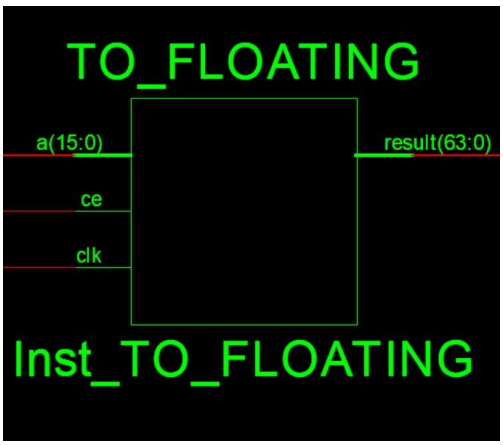


**Figura 3.11 IP Core Fixed to Floating.**

Una vez obtenido el valor en hexadecimal, se procede a realizar los cálculos respectivos para calcular el $V_{in}$ a partir de la fórmula (3.1). Para esto se emplearán los IP CORES de suma, multiplicación, una memoria RAM, el CONTROLADOR Vin, los TRI STATE 1, 3, 5, 7, 9, 11, 15, 16, 19, 20, CORE TO_FIXED y BCD_Vin.

Los TRI STATE son módulos que a la salida fijan una alta impedancia 'Z' cuando el habilitador o Enable está desactivado, caso contrario, fijan el valor de la entrada. Estos módulos se emplean para evitar colisiones como lo muestra la Figura 3.12, las salidas de los dos TRI STATE 3 y 4 se juntan y se conectan al SUMADOR, al deshabilitar el TRI STATE 3 y activando el TRI STATE 4, la salida del TRI STATE 3 se coloca en alta impedancia o haciendo una analogía con un componente electrónico este se coloca en circuito abierto, permitiendo que la salida del TRI STATE 4 sea el único dato que se conecte al SUMADOR.

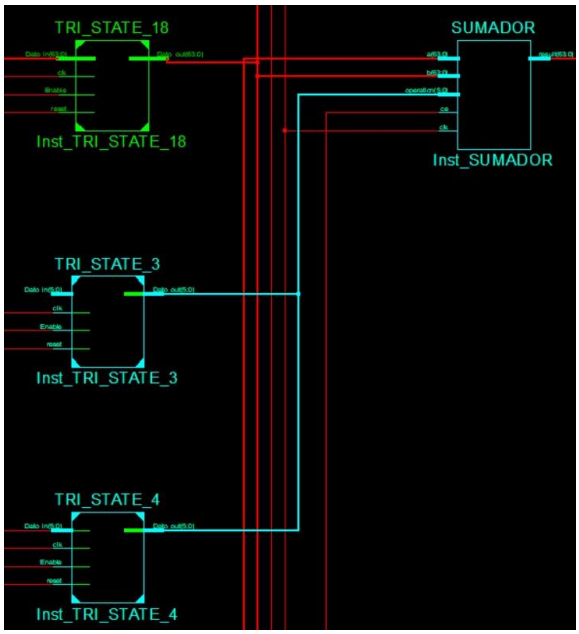


**Figura 3.12 Función de los TRI STATE.**

Siguiendo con el procedimiento para calcular la temperatura se empleará el CONTROLADOR Vin, ver Figura 3.13, el cual contiene un algoritmo que activa de forma sincronizada los IP

CORE suma y multiplicación. Con respecto al CORE suma, para habilitar el core se lo hará a través del TRI STATE 1, mediante las señales En_Suma y H_Sum, las cuales contienen el dato y el habilitador para el TRI STATE 1. El core puede operar como sumador o restador, siendo el comando “000000” para el sumador y el “000001” para el restador. Esto se lo hará mediante las señales Add_Sub(5:0) y H_Add_Sub que se conectarán a través del TRI STATE 3 y la salida del mismo se conectará a la señal OPERATION del sumador.

De igual manera se realizará el mismo procedimiento para controlar el CORE de multiplicación, para habilitar se lo hará mediante el TRI STATE 5 con las señales En_Mult y H_Mult enviadas desde el CONTROLADOR Vin.

Dado que los CORE sumador y el multiplicador serán empleados posteriormente por otro controlador, se implementó el TRI STATE 9 para el CORE de suma, éste permitirá activar el TRI STATE 19 mediante las señales En_Tri_19 y H_Tri_19 enviadas desde el CONTROLADOR Vin, EL TRI STATE 19 recibirá cada suma o resta proveniente del sumador y se activará mediante la salida del TRI STATE 9. La salida proveniente del TRI STATE 19 se retroalimentará al CONTROLADOR Vin por medio de la señal definida como Dato_R(63:0). El mismo principio se aplica para el CORE de multiplicación, para el cual se usará el TRI STATE 11, que a su vez activará el TRI STATE 20, mediante las señales En_Tri_20 y H_Tri_20. La salida del TRI STATE 20 se conectará a la salida del TRI STATE 19, es decir, a la señal Dato_R(63:0), esto se realiza porque esta señal contendrá todas las operaciones de la ecuación (3.1) y a su vez el algoritmo del CONTROLADOR Vin usará estas operaciones para realizar el resto de operaciones.

Luego se implementaron los TRI STATE 15 y 16, estos se estarán conectados a las salidas de la memoria RAM, memA y memB. Ambos TRI STATE comparten la misma señal para su

habilitación, la señal En_Vin será enviado desde el CONTROLADOR Vin. La salida del TRI STATE 15 se conectará al operando 'a' de los CORE de suma, multiplicación y división, mientras que la salida del CORE 16 se conectara al operando 'b' de los CORE mencionados. Los operandos son los valores con los que los CORE podrán realizar las operaciones matemáticas.

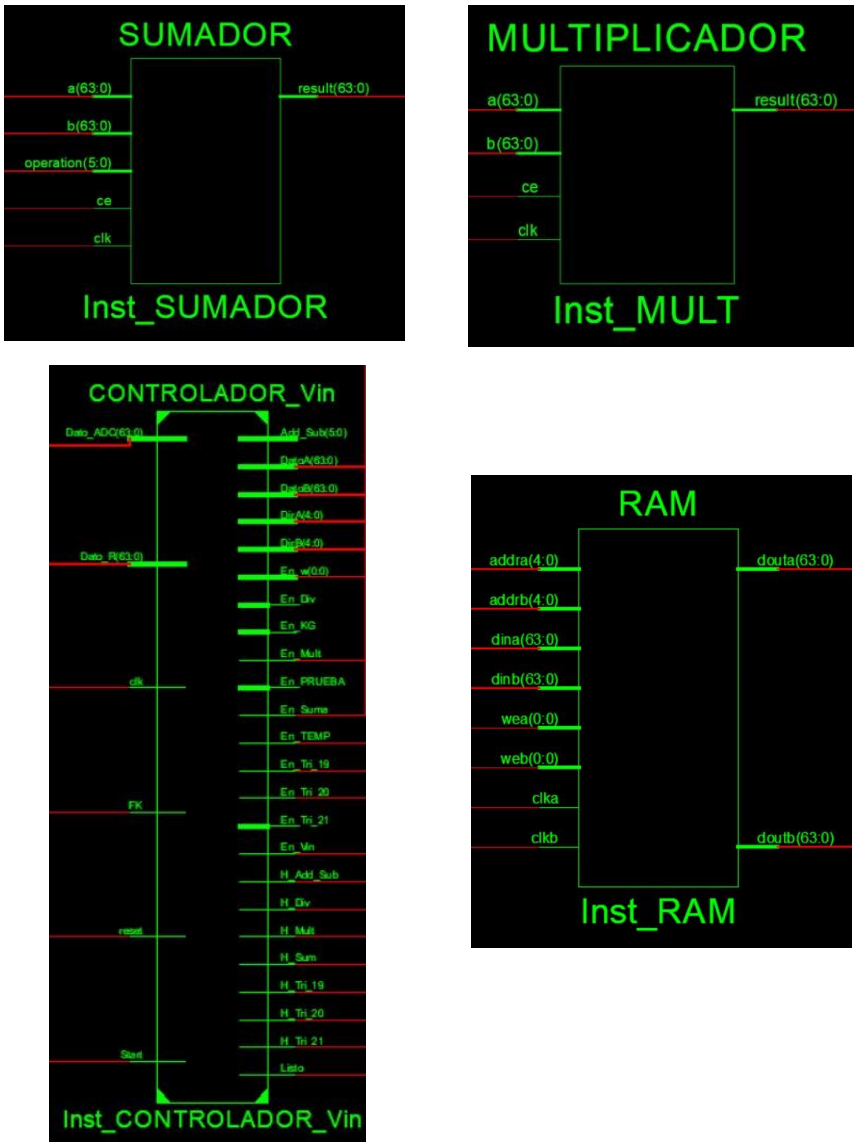


**Figura 3.13 IP Core Sumador, Multiplicador y módulo Controlador_Vin.**

Para obtener el valor de $V_{in}$ en formato decimal, requiere que sea convertido a un valor entero por el CORE TO_FIXED y posteriormente se le aplicará el algoritmo de 'binario a BCD' implementado en el módulo BCD_Vin para obtener el valor de la temperatura. Para esto se emplearán los módulos de la Figura

3.14.

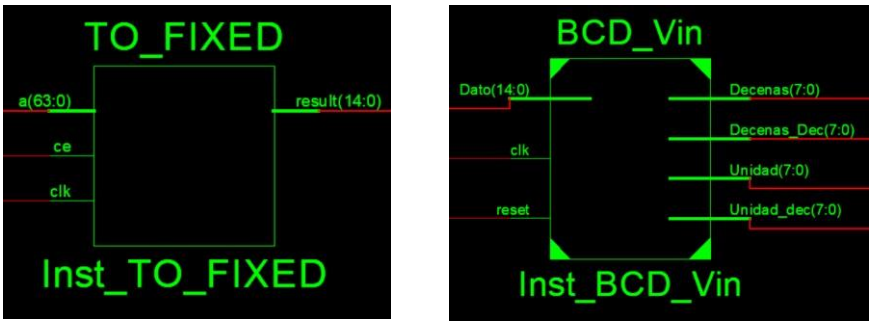


**Figura 3.14 Core TO_FIXED y módulo BCD_Vin.**

### 3.2.3.2 Cálculo de la ganancia de Kalman, la estimación y el error en la estimación

Para calcular los parámetros del filtro han empleado las ecuaciones (2.1), (2.2) y (2.3). Para esto se usará el módulo CONTROLADOR_FK, memoria RAM_FK, IP CORE de suma, multiplicación y división, los TRI STATE 2, 4, 6, 8, 10, 12, 14, 17, 18, 19, 20, 21, CORE KG, CORE ESTIMACIÓN, CORE Eest y BCD_KG, BCD_EST y BCD_Eest, como se observa en la Figura 3.15.

De igual manera que para el cálculo de la temperatura, se aplicará el mismo mecanismo con respecto al uso de los TRI STATE y el resto de módulos. Dado que el CORE suma es empleado por el CONTROLADOR Vin requiere que se adicione el TRI STATE 2 el cual será controlado por el CONTROLADOR FK mediante las señales En_Suma y H_Suma, La salida del TRI STATE 2 activará al CORE de suma. De esta manera tanto la salida del TRI STATE 1 como del TRI STATE 2 se encuentran conectados al habilitador del sumador, para estos casos se aplicará lo explicado en la Figura 3.12 con el objetivo de evitar las colisiones.

Así mismo, se requiere cambiar el modo de operación del sumador, por lo cual se empleará el TRI STATE 4 que serán controladas por la señales H_Add_Sub y Add_Sub(5:0) por el CONTROLADOR_FK. De igual manera la salida del TRI STATE 4 compartirá la salida con el TRI STATE 3. El mismo principio se seguirá para la activación del CORE de multiplicación, en el cual se emplearán las señales En_Mult y H_Mult a través del TRI STATE 6, con lo cual los TRI STATE 5 Y 6 compartirán el habilitador para el CORE del multiplicación. Para habilitar el CORE de división se emplearán las señales En_Div y H_Div por medio del TRI STATE 8, con lo cual el habilitador lo compartirá con el TRI STATE 7.

Para activar el TRI STATE 19 desde el CONTROLADOR_FK se implementó el TRI STATE 10 que mediante las señales En_Tri_19 y H_Tri_19 activará el TRI STATE 19 cuya salida definido como DATO_R(63:0) se retroalimentará al CONTROLADOR_FK. De igual manera para activar el TRI STATE 20 se implementó el TRI STATE 12, que será controlado por las señales En_Tri_20 y H_Tri_20, donde la salida del TRI STATE 20 compartirá con la salida del TRI STATE 19. Para activar el TRI STATE 21 se implementó el TRI STATE 14, el cual será controlado por las señales En_Tri_21 y H_Tri_21 a través del TRI STATE 21. La salida de este último se conectará a la salida del TRI STATE 20.

Los TRI STATE 17 y 18 estarán conectadas a las salidas de la memoria RAM_FK. Estos TRI STATE se los habilitará con la señal En_FK, la salida del TRI STATE 17 compartirá con la salida del TRI STATE 15 y estará comunicado con el operando ‘a’ de los CORE aritméticos. Así mismo, se conectará al CORE ESTIMACIÓN, este realiza la conversión de punto flotante a punto fijo de 15 bits. Este se conectará al módulo BCD_EST, el cual generará 4 señales de 8 bits que representan la estimación del Filtro de Kalman.

También se encuentra conectada al CORE KG, el cual convierte el valor flotante a un valor fijo de 25 bits. Este pasará por el módulo BCD_KG el cual generará 7 salidas Unidad_dec, Decenas_Dec,

Centenas_Dec, Uni_mill_dec, Dec_mill_dec, Decenas y Unidad. Estas señales contendrán el valor de la ganancia de Kalman.

Así mismo el CORE Eest se encuentra conectado de igual manera a salida del TRI STATE 17, el cual genera un valor fijo de 25 bits. Este pasara por el BCD_Eest que genera 7 salidas de 8 bits, las cuales son Unidad_dec, Decenas_Dec, Centenas_Dec,Uni_mill_dec,Dec_mill_dec.


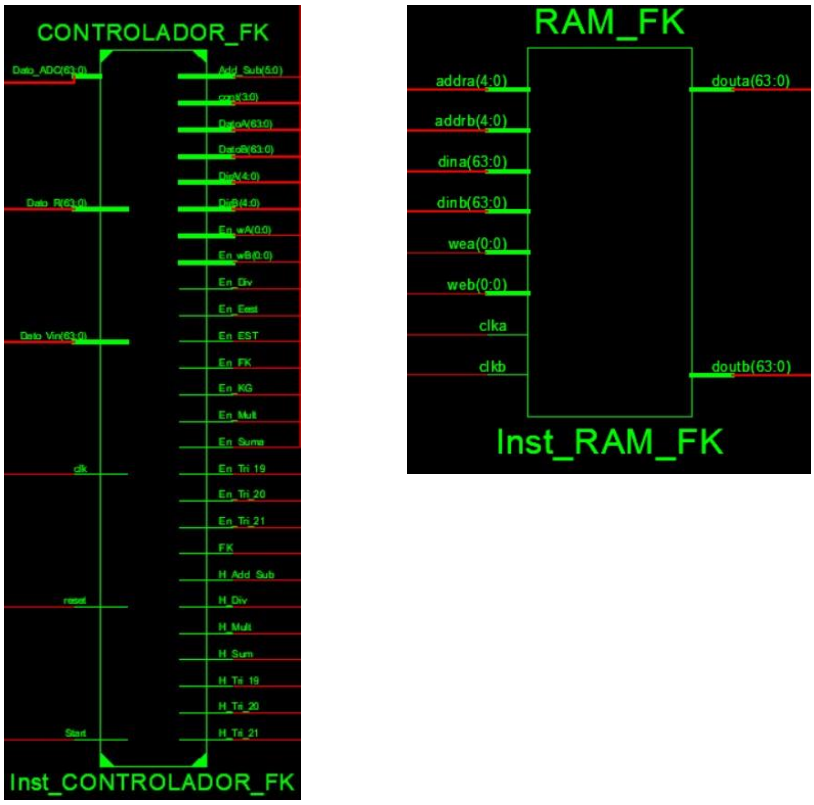


**Figura 3.15 Módulo CONTROLADOR_FK, Memoria RAM_FK, CORE Eest, KG, ESTIMACIÓN y módulos BCD_EST, BCD_Eest y BCD_KG.**

### 3.2.4 Etapa de visualización de datos

Se desarrolló el módulo LCD es que será el encargado de mostrar la temperatura, el valor del ADC y el parámetro de estimación del filtro de Kalman. Como se observa en la figura 3.16 se aprecia el módulo implementado y la visualización de los valores.

Con respecto al módulo, se observa que las entradas son las respectivas señales provenientes de los módulos BCD de la temperatura, la ganancia de Kalman, Estimación, error de la estimación y del ADC.

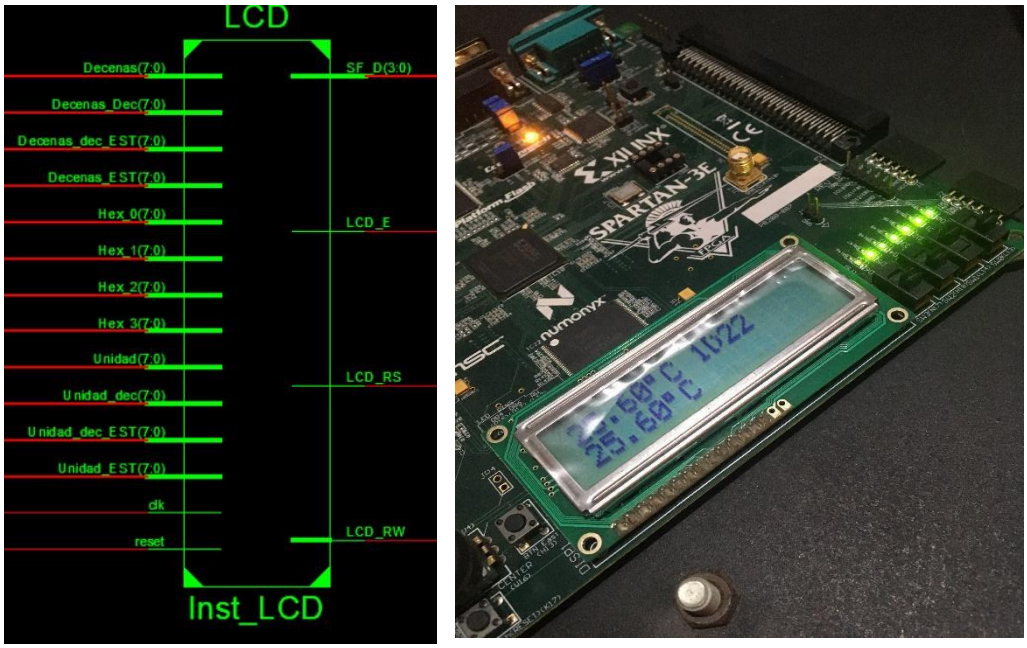


**Figura 3.16 Módulo de la LCD.**

### 3.2.5 Etapa de transmisión de datos

Para la trasmisión de datos, se desarrolló el módulo UART mostrado en la Figura 3.17, el cual está configurado con una tasa de 9600 baudios. El módulo recibe como entradas los valores de temperatura, ADC y los parámetros del filtro de Kalman. El módulo tiene un habilitador definido como ’Start’ porque solo se requiere que el UART transmita solo cuando todos los valores estén calculados, dichos valores se podrán visualizar en un software serial como en la Figura 3.18.

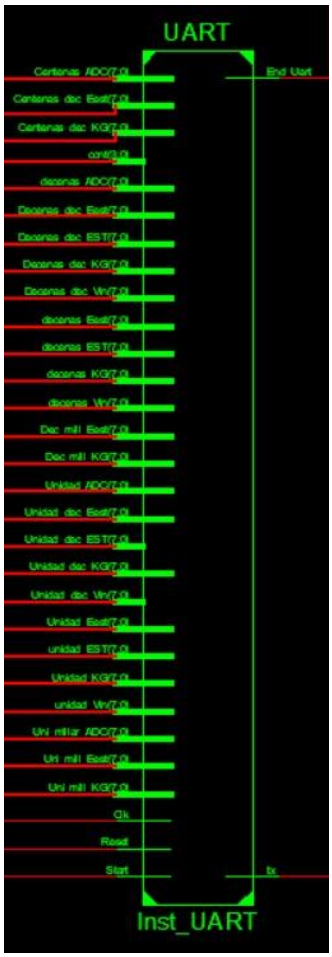


**Figura 3.17 Módulo UART.**

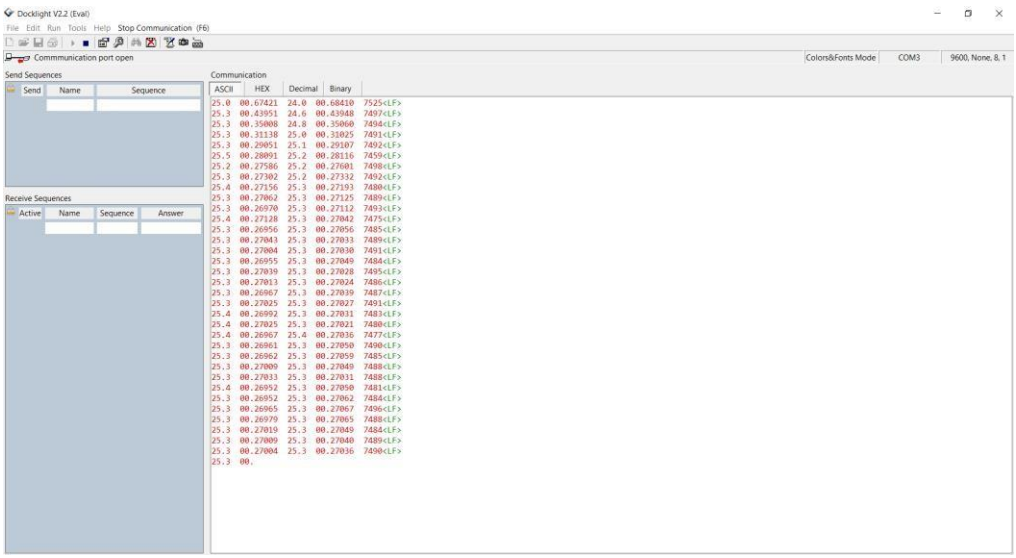


**Figura 3.18 Visualización de los datos seriales.**

En virtud de lo mencionado anteriormente, el algoritmo implementado o el esquemático se puede visualizar a través de una opción dentro del ISE DESIGN de Xilinx como se aprecia en la Figura 3.19. En él se puede constatar cada uno de los módulos descritos y a su vez los IP CORE implementados.

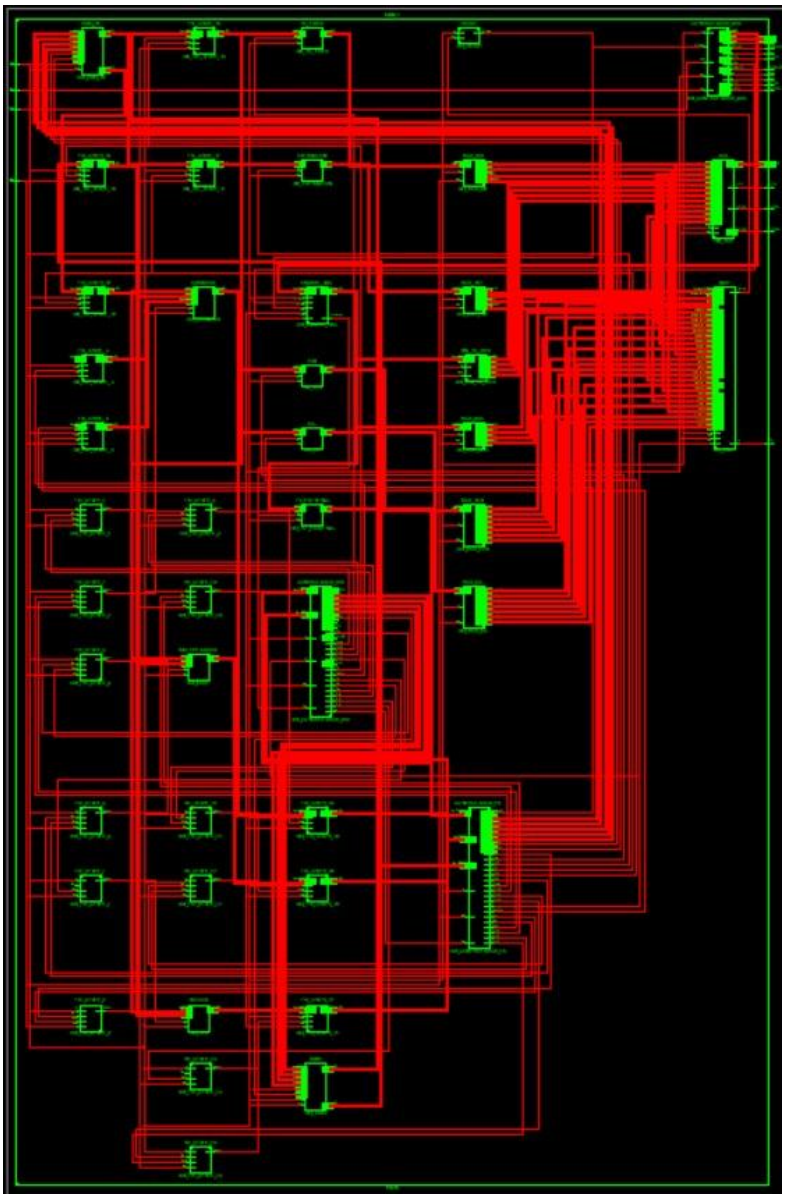

**Figura 3.19 Esquemático del Filtro de Kalman.**

Además es importante mencionar que este algoritmo fue optimizado para la tarjeta Spartan 3E, tal como lo muestra en la Figura 3.20. Se observa que la cantidad de slices utilizados fueron de 4654 slices lo que equivale un 99% de los recursos del PFGA. La mayor parte de sus recursos son usados por los CORE aritméticos, entre los 3 consumen 2545 slices de los 4656 disponibles lo que equivale al 54.66%. Es por esta razón, el uso de los TRI STATE los cuales permiten reusar estos CORES para diferentes operaciones matemáticas y así ahorrar recursos computacionales del FPGA.

| Device Utilization Summary | | | | [-] |
|---|---|---|---|---|
| **Logic Utilization** | **Used** | **Available** | **Utilization** | **Note(s)** |
| Number of Slice Flip Flops | 1,763 | 9,312 | 18% | |
| Number of 4 input LUTs | 8,299 | 9,312 | 89% | |
| Number of occupied Slices | 4,654 | 4,656 | 99% | |
| Number of Slices containing only related logic | 4,654 | 4,654 | 100% | |
| Number of Slices containing unrelated logic | 0 | 4,654 | 0% | |
| Total Number of 4 input LUTs | 8,729 | 9,312 | 93% | |
| Number used as logic | 8,299 | | | |
| Number used as a route-thru | 430 | | | |
| Number of bonded IOBs | 29 | 232 | 12% | |
| Number of RAMB16s | 4 | 20 | 20% | |
| Number of BUFGMUXs | 1 | 24 | 4% | |
| Number of MULT18X18SIOs | 16 | 20 | 80% | |
| Average Fanout of Non-Clock Nets | 2.83 | | | |

**Figura 3.20 Recurso computacional usado por el FPGA.**

En la Figura 3.21 se observa el diagrama de flujo del algoritmo empleado donde se aprecia la iteración de cada módulo descrito anteriormente.

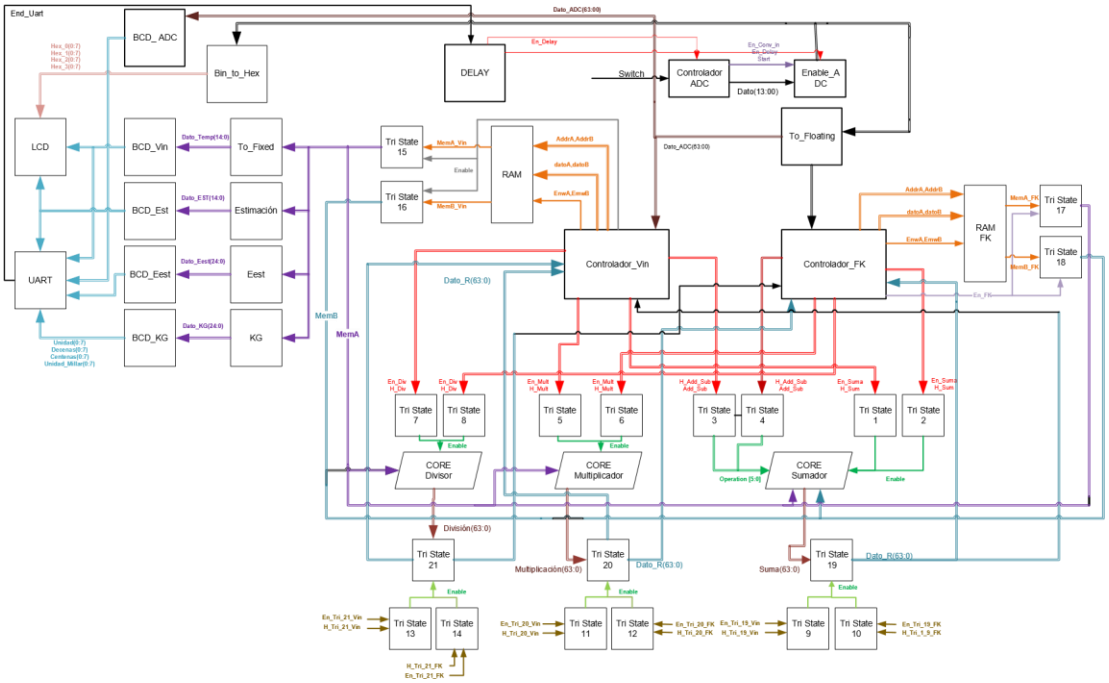


**Figura 3.21 Diagrama de flujo del algoritmo del FK.**

# CAPÍTULO 4

## RESULTADOS

En este capítulo, se detallarán los valores de los parámetros de inicialización del filtro. Posteriormente se analizarán las gráficas obtenidas de la ganancia de Kalman y del error de estimación y finalmente se realizará el análisis de la señal filtrada y observar como el filtro logra mitigar el ruido del sensor obteniendo datos más estables.

## 4.1 Parámetros para el análisis de desempeño del Filtro

El filtro de Kalman requiere ser inicializado para obtener el mejor desempeño posible, como se detalla en la tabla 4.1 se puede constatar los valores de cada uno de ellos. Se definió un valor crítico de 22°C como estimación inicial con la finalidad de observar la rapidez con la que el filtro es capaz de sintonizarse a las mediciones del sensor.

**Tabla 4.1 Parámetros de Inicialización del Filtro de Kalman.**

| | |
|---|---|
| Estimación inicial $E_{to}$ | 22°C |
| Error de medición $\square_{Mo}$ | 1.1°C |
| Error de estimación $\square_{est0}$ | 2°C |

Con respecto al error de la medición, tal como se indicó en la sección 2.2.2 en la hoja técnica menciona que la precisión del sensor se la define como el error entre el voltaje de salida y 10 mV/ °C multiplicado por la temperatura de la carcasa del dispositivo, en condiciones como la temperatura. En la Figura 4.1 se muestra como el error va variando a medida que la temperatura aumenta o disminuye, dado que el filtro de una dimensión asume que el error es constante, se estimó que el valor para el error más acorde para el ambiente de las pruebas fuera de 1.1°C. Es decir, como la temperatura ambiente ronda entre los 26°C y 32°C se trazó una línea y al intersectar la curva nos da el valor mencionado.

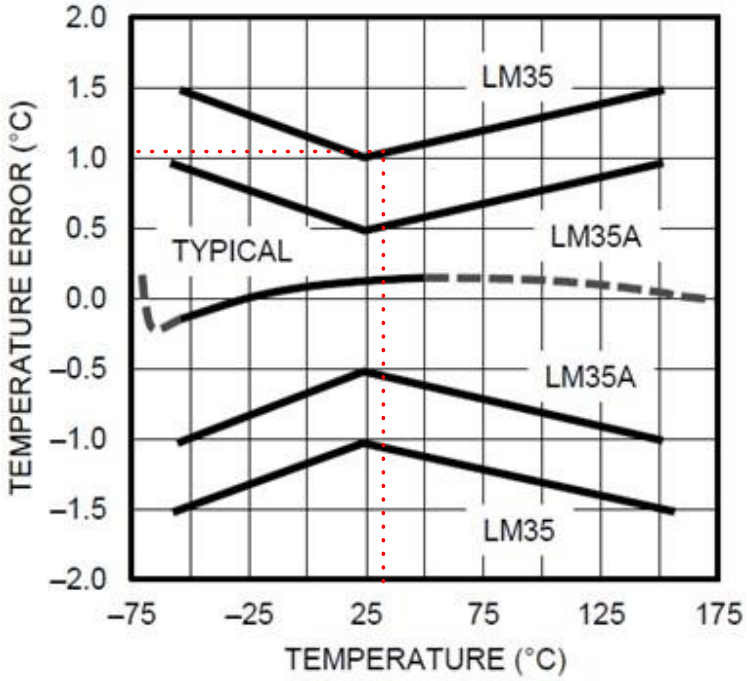


**Figura 4.1 Error de medición del sensor LM35.**

El error de estimación hace referencia a la diferencia de temperatura entre la sensación térmica que uno percibe y la temperatura real. Indistintamente el valor que se le asigne, el filtro debe ser capaz de sintonizarse, caso contrario, el filtro debe estar mal inicializado o el algoritmo implementado no es el correcto.

## 4.2 Análisis del comportamiento de la ganancia de Kalman y del error en la Estimación

Luego de la aplicación del filtro se obtuvieron los resultados mostrados en las Figuras 4.2 y 4.3. En la Figura 4.2 muestra el comportamiento de la ganancia de Kalman, el primer valor calculado por el filtro es de 0.4387 y este va disminuyendo hasta estabilizarse en un valor de 0.35 aproximadamente. Si se reescribe la ecuación (2.2) llegamos a la siguiente ecuación:

$$E_t = (1 - K)E_{Estimación} + (K)Med_{Medición} \quad (4.1)$$

El primer término indica que la ganancia de Kalman ejerce un peso a la estimación, mientras que el segundo término ejerce un peso a la medición, con base en los

resultados obtenidos, en cada iteración este peso va disminuyendo logrando así que el valor de la estimación actual converja a la estimación anterior.

Los valores de $K$ obtenidos son considerados bajos debido a que el peso de la mediciones resultan ser un poco más altas que el peso sobre la estimación, por tal motivo los valores de $K$ se encuentran en un valor medio, es decir, ni muy cerca de 1 ni muy cerca de 0.

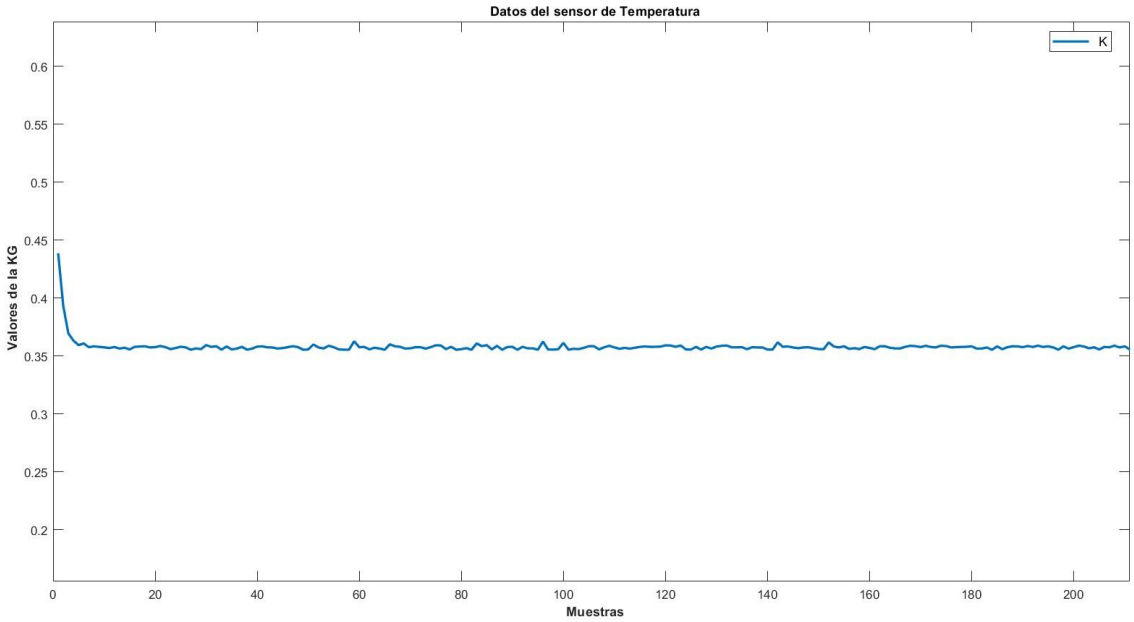


**Figura 4.2 Valores de la ganancia de Kalman obtenidos desde el FPGA.**

En la Figura 4.3 se aprecia en cambio el comportamiento del error en la estimación, de igual manera que en la ganancia de Kalman, en cada iteración el error debe disminuir hasta estabilizarse. Para la interpretación de estos datos, se usará como ejemplo los valores de la estimación y del error de estimación de la última iteración. El valor de la última estimación fue de 34.5°C mientras que el error fue de 1.8. Usando estadística descriptiva, se define que la varianza corresponde al error.

$$\sigma^2 = 1.8 ; \quad \sigma = 1.34 \tag{4.2}$$

Con lo que se obtiene, que la temperatura estimada es de 34.5 ± 1.34 °C.

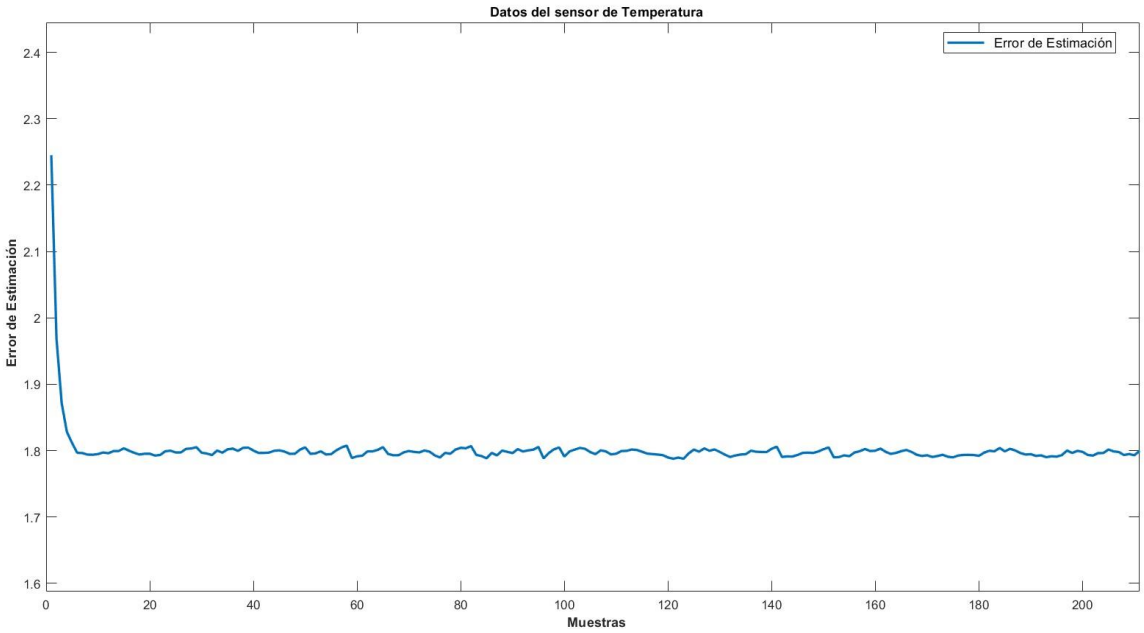


**Figura 4.3 Valores del error de estimación obtenidos desde el FPGA.**

## 4.3 Análisis de la señal filtrada

En la Figura 4.4 se muestra los datos en bruto del sensor LM35, se puede observar que los datos fluctúan constantemente a pesar cuando no existe un cambio importante de temperatura, como se aprecia desde la muestra 14 hasta la 46. Cuando se aplica el filtro de Kalman se aprecia que los valores de la señal filtrada empiezan muy por debajo de los 31.5°C. Esto es de esperarse, dado que el valor de la estimación inicial fue de 22°C y el filtro siempre logrará ajustarse en función de las estimaciones previas. Por esta razón el filtro tarda alrededor de 15 muestras o segundos en sintonizarse y a partir de la muestra 16 el filtro es capaz de alcanzar estimaciones más precisas y con menor presencia de fluctuaciones. Realizando el análisis entre las muestras 16 y 46 se logra evidenciar el poder del filtro de Kalman a la hora de realizar las estimaciones, dado que es capaz de reducir considerablemente las fluctuaciones, logrando suavizar la señal como se evidencia en la muestra 39, la temperatura del sensor llega hasta los 32.1°C mientras que la señal filtrada estima una valor de 32.4°C.

A partir de la muestra 52 hasta la 112 se perturba el ambiente con la finalidad de aumentar la temperatura, se observa como el filtro es capaz de seguir ese cambio de temperatura y se evidencia como se eliminan considerablemente las fluctuaciones a pesar de los cambios bruscos en las mediciones del sensor. Se puede apreciar un pequeño retraso porque la señal filtrada depende de las

estimaciones anteriores y más no de los datos del sensor. Desde la muestra 113 hasta la 163 se produce un decremento de temperatura, y se observa picos más pronunciados y con cierta periodicidad, en especial en el muestra 163 cuyo valor medido pasa de 33.9°C a 33.2°C generando una fluctuación importante mientras que el filtro pasa de un valor estimado de 34.0°C a 33.8°C reduciendo 0.4°C en dicha muestra.

Desde la muestra 164 hasta la 192, se produce un incremento de temperatura volviendo a evidenciarse las fluctuaciones de parte del sensor, mientras que el filtro logra seguir los datos del sensor eliminando casi en su totalidad las fluctuaciones.

Para el resto de muestras se produce un decremento leve de temperatura, sin embargo genera fluctuaciones críticas como en la muestra 207, pero a pesar de eso el filtro es capaz de reducirlo y estimar un valor óptimo logrando tener datos más confiables y estables.

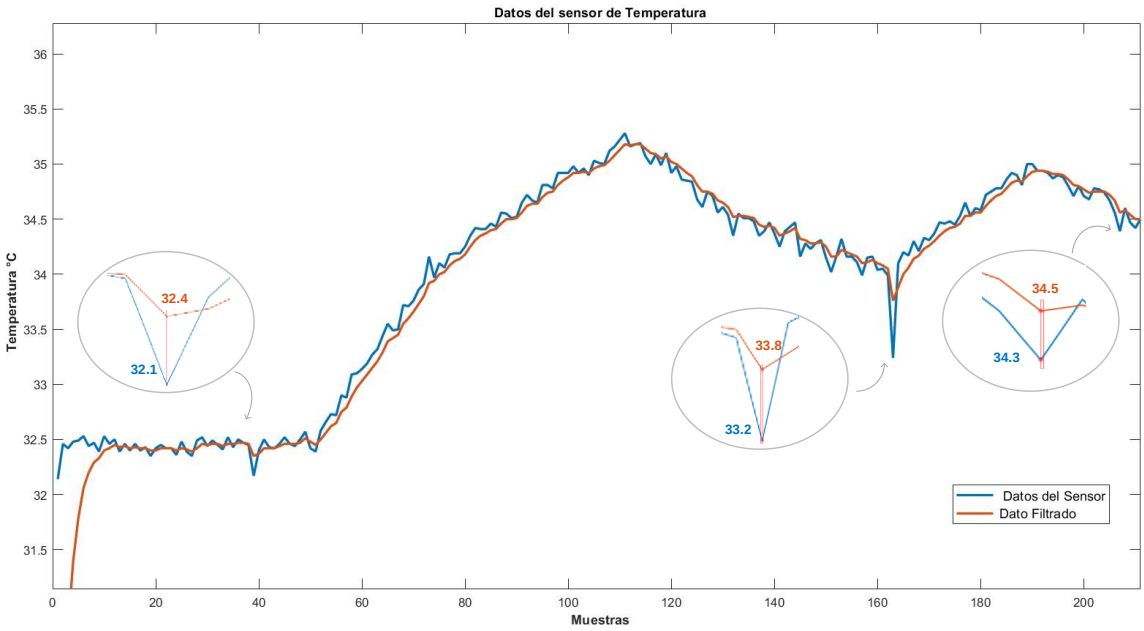


**Figura 4.4 Datos del sensor vs. Señal filtrada obtenida desde el FPGA.**

# CAPITULO 5

## CONCLUSIONES

1. A la señal filtrada le toma alrededor de 15 segundos en sintonizarse logrando obtener estimaciones más óptimas con un error de estimación de 1.34°C. Esto refleja que el tiempo de respuesta del filtro es útil para escenarios críticos que requieren acciones inmediatas.

2. Con los datos obtenidos, se determina que cuando los valores de $K$ son pequeños implica que va a existir un delay, debido que el filtro realiza las estimaciones con base en las estimaciones anteriores y no con los datos del sensor. Por otra parte, cuando los valores son grandes, las estimaciones se verán afectada por el ruido del sensor. Dado que esto coincide con la sintonización del filtro, no se logra apreciar como el ruido incide considerablemente sobre las estimaciones.

3. Las mediciones obtenidas del filtro muestran tener un margen de error de 1.34°C, el cual es más que aceptable dado que el sensor LM35 no es considerado como uno de alta precisión.

4. El filtro implementado resultó efectivo a la hora de eliminar las fluctuaciones del sensor. En especial cuando se presenta un decremento de temperatura, la presencia de los picos son más pronunciados, pero a pesar de eso, el filtro fue capaz de estimar el valor más óptimo mejorando la calidad de la señal.

5. Con la correcta inicialización del filtro se logró evidenciar que se obtuvieron los resultados esperados, dado que al proporcionarle los parámetros correctos se ayudó al filtro a que pueda sintonizarse rápidamente y realizar las estimaciones eficientes sobre todo cuando hubo decremento de la temperatura.

## RECOMENDACIONES

1. A la hora de implementar el algoritmo de FK, es aconsejable emplear hardware de alto rendimiento como un FPGA, dado que garantiza obtener resultados más confiables y precisos sobre todo al implementar las ecuaciones matemáticas.

2. Para ahorrar recursos computacionales del FPGA, es recomendable implementar los TRI STATE necesarios con la finalidad de emplear los IP CORE matemáticos las veces que sean requeridas.

3. Al momento de seleccionar el hardware es imprescindible usar ADC con la mayor resolución posible para reducir el error en los resultados del filtro. Así mismo, seleccionar un FPGA que cuente con los recursos computaciones suficientes para implementar el filtro.

4. Dado que el filtro requiere datos altamente confiables, se sugiere emplear aritmética de doble precisión (64 bits de precisión).

5. Dada la versatilidad del filtro de Kalman, se puede emplear el algoritmo de estudio con otros sensores, garantizando una mejora en la calidad de la señal.

6. Para mejorar los resultados obtenidos, se propone considerar un modelo matemático acorde a los requerimientos para luego emplear un Filtro de Kalman con varias variables de estados con el objetivo de inyectarle una gran cantidad de información que le permitan al filtro ser más robusto y estimar con una mayor precisión.

# BIBLIOGRAFÍA

[1] H. Ge, Y. Zhen, Y. Wang, and D. Wang, “Research on LCL filter active damping strategy in active power filter system,” *Proc. 2017 9th Int. Conf. Model. Identif. Control. ICMIC 2017*, vol. 2018-March, no. Icmic, pp. 476–481, 2018, doi: 10.1109/ICMIC.2017.8321691.

[2] Y. Gu, S. Liu, D. Wang, and L. Zhang, “A Generalized Moving Average Filter for Active Power Filter Applications,” *IEEE Int. Symp. Ind. Electron.*, vol. 2019-June, pp. 428–433, 2019, doi: 10.1109/ISIE.2019.8781279.

[3] S. Engelberg and B. Milgrom, “Tracking using state estimation: A brief introduction,” *IEEE Instrum. Meas. Mag.*, vol. 22, no. 3, pp. 36–42, 2019, doi: 10.1109/MIM.2019.8716274.

[4] N. A. Kumar, G. S. Rao, and N. Arasavali, “Development of advanced extended kalman filter for precise estimation of GPS receiver position,” *2019 Int. Conf. Wirel. Commun. Signal Process. Networking, WiSPNET 2019*, no. 1, pp. 213–216, 2019, doi: 10.1109/WiSPNET45539.2019.9032769.

[5] Y. Luo, G. Ye, Y. Wu, J. Guo, J. Liang, and Y. Yang, “An adaptive kalman filter for UAV attitude estimation,” *2019 2nd Int. Conf. Electron. Technol. ICET 2019*, pp. 258–262, 2019, doi: 10.1109/ELTECH.2019.8839496.

[6] W. Zhang and B. An, “Study on Influence of Electromagnetic Disturbance on Wireless Sensor Unit in Substation,” *2018 IEEE Symp. Electromagn. Compat. Signal Integr. Power Integrity, EMC, SI PI 2018*, pp. 654–657, 2018, doi: 10.1109/EMCSI.2018.8495164.

[7] Y. Wang, Q. Li, and F. Zhou, “Transient power quality disturbance denoising and detection based on improved iterative adaptive kernel regression,” *J. Mod. Power Syst. Clean Energy*, vol. 7, no. 3, pp. 644–657, 2019, doi: 10.1007/s40565-0180467-4.

[8] P. V. Dutande, S. L. Nalbalwar, and S. V. Khobragade, “FPGA Implementation of Filters for Removing Muscle Artefacts from EEG Signals,” *Proc. 2nd Int. Conf. Intell.*

*Comput. Control Syst. ICICCS 2018*, no. Iciccs, pp. 728–732, 2019, doi: 10.1109/ICCONS.2018.8662998.

[9] R. R. Sudharsan, “Synthesis of FIR Filter using ADC-DAC: A FPGA Implementation,” *2019 Int. Conf. Clean Energy Energy Effic. Electron. Circuit Sustain. Dev. INCCES 2019*, pp. 57–59, 2019, doi: 10.1109/INCCES47820.2019.9167696.

[10] H. Liu, F. Hu, J. Su, X. Wei, and R. Qin, “Comparisons on Kalman-Filter-Based Dynamic State Estimation Algorithms of Power Systems,” *IEEE Access*, vol. 8, pp. 51035–51043, 2020, doi: 10.1109/ACCESS.2020.2979735.

[11] W. T. Chen, Y. S. Chiou, F. J. Wen, S. L. Chen, T. L. Lin, and Y. K. Lin, “FPGAbased implementation of reduced-complexity filtering algorithm for real-time location tracking,” *Proc. - IEEE 17th Int. Conf. Dependable, Auton. Secur. Comput. IEEE 17th Int. Conf. Pervasive Intell. Comput. IEEE 5th Int. Conf. Cloud Big Data Comput. 4th Cyber Sci.*, pp. 721–726, 2019, doi: 10.1109/DASC/PiCom/CBDCom/CyberSciTech.2019.00136.

[12] O. Iqbal *et al.*, “Design and FPGA Implementation of an Adaptive video Subsampling Algorithm for Energy-Efficient Single Object Tracking,” *Proc. - Int. Conf. Image Process. ICIP*, vol. 2020-Octob, pp. 3065–3069, 2020, doi: 10.1109/ICIP40778.2020.9191146.

[13] N. S. B. Sembiring, E. Ginting, M. Fauzi, Yudi, F. Tambunan, and E. V. Haryanto, “An Expert System to Diagnose Herpes Zoster Disease Using Bayes Theorem,” *2019 7th Int. Conf. Cyber IT Serv. Manag. CITSM 2019*, pp. 6–8, 2019, doi: 10.1109/CITSM47753.2019.8965381.

[14] Z. Khan, H. Bugti, and A. S. Bugti, “Single dimensional generalized Kalman filter,” *2018 Int. Conf. Comput. Electron. Electr. Eng. ICE Cube 2018*, pp. 1–5, 2019, doi: 10.1109/ICECUBE.2018.8610960.

[15] A. Becker, “Filtro Kalman en una dimensión,” 2018. https://www.kalmanfilter.net/ES/kalman1d_es.html (accessed May 24, 2021).

[16] D. Chowdhury, S. K. Das, S. Nandy, A. Chakraborty, R. Goswami, and A. Chakraborty, “An atomic technique for removal of gaussian noise from a noisy gray

scale image using lowpass-convoluted gaussian filter," *2019 Int. Conf. Opto-Electronics Appl. Opt. Optronix 2019*, pp. 1–6, 2019, doi: 10.1109/OPTRONIX.2019.8862330.

[17] S. Moontaha, A. Galka, T. Meurer, and M. Siniatchkin, "Analysis of the effects of medication for the treatment of epilepsy by ensemble Iterative Extended Kalman filtering," *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. EMBS*, vol. 2018-July, pp. 187–190, 2018, doi: 10.1109/EMBC.2018.8512179.

[18] J. Liao *et al.*, "FPGA Implementation of a Kalman-Based Motion Estimator for Levitated Nanoparticles," *IEEE Trans. Instrum. Meas.*, vol. 68, no. 7, pp. 2374–2386, 2019, doi: 10.1109/TIM.2018.2879146.

[19] M. Xu, Z. Li, S. Lu, and J. Fan, "Online Ambient Temperature Monitoring of Electronic Transducer Based on Thermocouple," *2019 3rd IEEE Conf. Energy Internet Energy Syst. Integr. Ubiquitous Energy Netw. Connect. Everything, EI2 2019*, pp. 2295–2298, 2019, doi: 10.1109/EI247390.2019.9062173.

[20] N. Bogdanovs, V. Bistrovs, E. Petersons, A. Ipatovs, and R. Belinskis, "Weather Prediction Algorithm Based on Historical Data Using Kalman Filter," *Proc. - 2018 Adv. Wirel. Opt. Commun. RTUWO 2018*, pp. 94–99, 2018, doi: 10.1109/RTUWO.2018.8587795.

[21] S. Marelli and M. Corno, "Model-Based Estimation of Lithium Concentrations and Temperature in Batteries Using Soft-Constrained Dual Unscented Kalman Filtering," *IEEE Trans. Control Syst. Technol.*, vol. 29, no. 2, pp. 926–933, 2021, doi: 10.1109/TCST.2020.2974176.

[22] "Spartan-3E FPGA Starter Kit Board," vol. 230, 2011.

[23] S. S. Kit, "Amplifier and A / D Converter Control," no. February, 2006.

# ANEXOS

## “Controlador_ADC”

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.ALL;
use ieee.numeric_std.all;

entity CONTROLADOR_ADC is
Port (
clk, reset, miso, start,En_Delay : in std_logic;
cs,mosi,ad_conv,En_conv, sck: out std_logic;
SPI_SS_B,DAC_CS,SF_CE0,FPGA_INIT_B,AMP_SHDN : out  STD_LOGIC;
LED: out std_logic_vector(15 downto 0);
lsb: out std_logic_vector(7 downto 0)
);

end CONTROLADOR_ADC;

architecture Behavioral of CONTROLADOR_ADC is

signal counter: std_logic_vector (6 downto 0):="0000000";
signal num_bit: std_logic_vector (6 downto 0):="0000000";
signal sclk,AD: std_logic:='0';
signal dato: std_logic_Vector(31 downto 0):=x"00000000";
signal gan: std_logic_vector(7 downto 0):="10001000";
signal count: integer range 0 to 3:=0;

type y is( Idle,Inicio,Configuracion,Conversion,Resultado,Enable,Espera,Espera1);
signal state: y;

begin

SPI_SS_B<='1';
DAC_CS<='1';
SF_CE0<='1';
FPGA_INIT_B<='1';
AMP_SHDN<= '0';

process (clk, Start)
begin

if reset = '1' then
state<=Idle;
elsif clk'event and clk = '1' then

case state is

when Idle =>
if Start = '1' then
state<=Inicio;
else
state<=Idle;
dato<=x"00000000"; LED<="0000000000000000";
end if;

when Inicio =>
counter <= "0000000";
num_bit <= "0000000";
cs <= '1';
sclk <= '0';
state<= configuracion;

when Configuracion =>
cs<='0';
counter<=counter+'1';
```

```
if num_bit<=7 then
if counter <38 then

mosi <= gan(to_integer(unsigned((num_bit))));
if counter>3 and counter <21 then
sclk<='1';
else
sclk<='0';
state<=Configuracion;
end if;
else
counter<="0000000";
num_bit<=num_bit+'1';
end if;
else
state<=conversion;
num_bit<="0100001";
counter<="0000000";
mosi<='1';
ad_conv<='1';

end if;

when conversion =>
ad<='0';

if counter=2 then
cs<='1';
end if;

counter<=counter+'1';

if num_bit<35 then
if counter <38 then
if counter>3 and counter <21 then
sclk<='1';
else
sclk<='0';
state<=conversion;
end if;
else
counter<="0000000";
num_bit<=num_bit-'1';
end if;

else
state<=Resultado;
num_bit<="0100001";
ad<='1';

end if;


if ((num_bit<=31 and num_bit>=18)) then
if counter=21 then
dato(to_integer(unsigned((num_bit))))<=miso;
end if;
end if;

when Resultado =>

if count=2 then
LED<="00"&dato(31 downto 18);
lsb<=dato(25 downto 18);
count<=2;
state<=Enable;
else
state<=conversion;
count<=count+1;
end if;

when Enable =>

En_conv<='1';
State<=Espera;
when Espera =>
State<=Espera1;
when Espera1 =>
En_conv<='0';

state<=Conversion;

when others =>
state <= idle;

end case;

end if;

sck<=sclk;
ad_conv<=ad;

end process;
end Behavioral;
```

## “Enable_ADC”

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

Entity ENABLE_ADC is port(

clk,reset,Start, En_Conv_in, En_Delay: in std_logic;
Dato_in: in  std_logic_Vector(15 downto 0);
En_Conv_out: out std_logic;
Dato_out: out std_logic_Vector(15 downto 0)
);

End ENABLE_ADC;

architecture Behavioral of ENABLE_ADC is

type y is( Idle,s0,s1,s2);
signal state: y;

begin

process (clk) begin

If clk'event and clk='1' then


Case state is

when Idle =>
If Start='1' then
state<=s0;
Else
state<=Idle;
End if;

when s0 =>
Dato_out<=Dato_in;
En_Conv_out<=En_Conv_in;
state<=s1;

when s1=>
state<=s2;

when s2 =>
If En_Delay='1' then
state<=Idle;
Else
En_Conv_out<='0';
state<=s2;
End if;
End case;
End if;
End process;

End Behavioral;
```

## “BIN_TO_HEX”

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity BIN_TO_HEX is
port(
Clk,Reset: in std_logic;
Dato_in:  in std_logic_vector(15 downto 0);
Hex_0,Hex_1,Hex_2,Hex_3: out std_logic_vector(7 downto 0)
);

end BIN_TO_HEX;

architecture Behavioral of BIN_TO_HEX is
signal H_0,H_1,H_2,H_3: std_logic_vector(3 downto 0);

begin

process(Clk,Reset) begin

if Reset='1' then
H_0<="0000"; H_1<="0000"; H_2<="0000"; H_3<="0000";
elsif clk'event and clk='1' then
H_0<=Dato_in(3 downto 0);
H_1<=Dato_in(7 downto 4);
H_2<=Dato_in(11 downto 8);
H_3<=Dato_in(15 downto 12);
end if;

end process;

with H_0 select

Hex_0<=
"00110000" when "0000",  --0
"00110001" when "0001",  --1
"00110010" when "0010",  --2
"00110011" when "0011",  --3
"00110100" when "0100",  --4
"00110101" when "0101",  --5
"00110110" when "0110",  --6
"00110111" when "0111",  --7
"00111000" when "1000",  --8
"00111001" when "1001",  --9
"01000001" when "1010",  --A
"01000010" when "1011",  --B
"01000011" when "1100",  --C
"01000100" when "1101",  --D
"01000101" when "1110",  --F
"01000110" when others;

with H_1 select

Hex_1<=
"00110000" when "0000",  --0
"00110001" when "0001",  --1
"00110010" when "0010",  --2
"00110011" when "0011",  --3
"00110100" when "0100",  --4
"00110101" when "0101",  --5
"00110110" when "0110",  --6
"00110111" when "0111",  --7
"00111000" when "1000",  --8
"00111001" when "1001",  --9
"01000001" when "1010",  --A
"01000010" when "1011",  --B
"01000011" when "1100",  --C
"01000100" when "1101",  --D
"01000101" when "1110",  --F
"01000110" when others;

with H_2 select

Hex_2<=
"00110000" when "0000",  --0
"00110001" when "0001",  --1
"00110010" when "0010",  --2
"00110011" when "0011",  --3
"00110100" when "0100",  --4
"00110101" when "0101",  --5
"00110110" when "0110",  --6
"00110111" when "0111",  --7
"00111000" when "1000",  --8
"00111001" when "1001",  --9
"01000001" when "1010",  --A
"01000010" when "1011",  --B
"01000011" when "1100",  --C
"01000100" when "1101",  --D
"01000101" when "1110",  --F
"01000110" when others;

with H_3 select

Hex_3<=
"00110000" when "0000",  --0
"00110001" when "0001",  --1
"00110010" when "0010",  --2
"00110011" when "0011",  --3
"00110100" when "0100",  --4
"00110101" when "0101",  --5
"00110110" when "0110",  --6
"00110111" when "0111",  --7
"00111000" when "1000",  --8
"00111001" when "1001",  --9
"01000001" when "1010",  --A
"01000010" when "1011",  --B
"01000011" when "1100",  --C
"01000100" when "1101",  --D
"01000101" when "1110",  --F
"01000110" when others;

end Behavioral;
```

## “BCD_ADC”

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.MATH_REAL.ALL;

entity BCD_ADC is

port(
clk,reset: in std_logic;
Dato: in std_logic_vector(15 downto 0);
Uni_Millar,Centenas, Decenas,Unidad: out std_logic_vector(7 downto 0)
);

end BCD_ADC;

architecture Behavioral of BCD_ADC is


begin

process(Dato)
variable temp1 : std_logic_vector (14 downto 0);
variable bcd1 : unsigned (15 downto 0) := (others => '0');

begin
bcd1 := (others => '0');
temp1(14 downto 0) := Dato(14 downto 0);

for i in 0 to 14 loop  --Números de desplazamientos
if bcd1(3 downto 0) > 4 then
bcd1(3 downto 0) := bcd1(3 downto 0) + 3;
end if;

if bcd1(7 downto 4) > 4 then
bcd1(7 downto 4) := bcd1(7 downto 4) + 3;
end if;

if bcd1(11 downto 8) > 4 then
bcd1(11 downto 8) := bcd1(11 downto 8) + 3;
end if;

if bcd1(15 downto 12) > 4 then
bcd1(15 downto 12) := bcd1(15 downto 12) + 3;
end if;

bcd1 := bcd1(14 downto 0) & temp1(14);
temp1 := temp1(13 downto 0) & '0';
end loop;

Unidad <= "0011"&(STD_LOGIC_VECTOR(bcd1(3 downto 0)));
Decenas <= "0011"&STD_LOGIC_VECTOR(bcd1(7 downto 4));
Centenas <= "0011"&STD_LOGIC_VECTOR(bcd1(11 downto 8));
Uni_millar<= "0011"&STD_LOGIC_VECTOR(bcd1(15 downto 12));

end process;
end behavioral;
```

## “TRI STATE 1,2,3,..21”

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity TRI_STATE_1 is
port(
clk,reset,Enable: in std_logic;
Dato_in: in  std_logic;
Dato_out: out std_logic
);
end TRI_STATE_1;

architecture Behavioral of TRI_STATE_1 is

begin

process(clk,reset) begin

if reset='1' then
Dato_out<='Z';

elsif clk'event and clk='1' then

If Enable='1' then
Dato_out<=Dato_in;

else

Dato_out<='Z';
end if;

end if;

end process;
end Behavioral;
```

## “CONTROLADOR_VIN”

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity CONTROLADOR_Vin is

Port(
clk, reset, Start, FK: in std_logic;
Dato_ADC,Dato_R: in std_logic_Vector(63 downto 0);
En_TEMP,En_PRUEBA, En_KG, Listo: out std_logic;

En_Suma, En_Mult, En_Div, En_Tri_19, En_Tri_20, En_Tri_21, En_Vin: out std_logic;
Add_Sub: out std_logic_Vector(5 downto 0);

H_Sum,H_Add_Sub,H_Mult,H_Div,H_Tri_19,H_Tri_20,H_Tri_21: out std_logic;

En_w: out std_logic_Vector(0 downto 0);
DirA,DirB: out std_logic_Vector(4 downto 0);
DatoA, DatoB: out std_logic_vector(63 downto 0)

);

end CONTROLADOR_Vin;

architecture Behavioral of CONTROLADOR_Vin is

signal count: integer range 0 to 10:=0;

type y is(Idle,Inicio,s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15,s16,s17,s18,s19,s20,s21,s22,Esperar);

signal State: y;

begin

process(Clk,Reset) begin

If reset='1' then

State<= Idle;

Elsif clk'event and clk = '1' then

Case State is

when Idle =>

If Start='1' then
If count=1 then
count<=0;
En_Vin<='1';
State<=Inicio;
Else
count<=count+1;
State<=idle;
End if;
Else
En_Vin<='0';
H_Sum<='1'; H_Add_Sub<='1'; H_Mult<='1'; H_Div<='1'; En_Suma<='0'; Add_Sub<="000000"; En_Mult<='0';
En_Div<='0';
H_Tri_19<='1'; H_Tri_20<='1'; H_Tri_21<='1'; En_Tri_19<='0'; En_Tri_20<='0'; En_Tri_21<='0';
En_w<="0"; DirA<="00001"; DirB<="00001";

State<=Idle;

end if;
```

```
when Inicio=>

H_Sum<='1'; H_Add_Sub<='1'; H_Mult<='1'; H_Div<='1'; En_Suma<='0'; Add_Sub<="000000"; En_Mult<='0';
En_Div<='0';
H_Tri_19<='1'; H_Tri_20<='1'; H_Tri_21<='1'; En_Tri_19<='0'; En_Tri_20<='0'; En_Tri_21<='0';

En_w<="0";  DirA<="00001"; DirB<="00001";
state<=s0;

when s0 =>

En_w<="1";
DatoA<= Dato_ADC;
DatoB<=x"BF23FFFFFFF1ED1F";
DirA<="00001";
DirB<="00010";

State<=s1;

when s1 =>

En_Mult<='1';
State<=s2;

when s2=>

State<=s3;

when s3=>

En_Mult<='0';
En_Tri_20<='1';

State<=s4;

when s4=>

En_Tri_20<='0';
State<=s5;

when s5 =>

state<=s6;

when s6 =>

En_w<="1";
DatoA<=Dato_R;
DatoB<=x"3FFA666666666666";
DirA<="00011";
DirB<="00100";
Dirección 4
State<=s7;

when s7=>

En_Suma<='1';
Add_Sub<="000000";
State<=s8;

when s8=>
State<=s9;

when s9=>
En_Suma<='0';
En_Tri_19<='1';
```

```
State<=s10;

when s10=>
En_Tri_19<='0';
State<=s11;

when s11=>
State<=s12;

when s12=>
En_w<="1";
DatoA<=Dato_R;
DatoB<=x"4049000000000000";
DirA<="00101";
DirB<="00110";
State<=s13;

when s13=>
En_Mult<='1';
State<=s14;

when s14=>
State<=s15;

when s15=>
En_Mult<='0';
En_Tri_20<='1';
State<=s16;

when s16=>
En_Tri_20<='0';
State<=s17;

when s17=>
State<=s18;

when s18=>
En_w<="1";
DatoA<=Dato_R;
DirA<="00111";

State<=s19;

when s19=>
State<=s20;

when s20=>
En_TEMP<='1';
Listo<='1';
State<=s21;

when s21=>
En_TEMP<='0';

State<=s22;

when s22=>
Listo<='0';
En_Vin<='0';
State<=Esperar;

when Esperar=>
If FK='1' then
State<=Idle;
Else
H_Sum<='0'; H_Add_Sub<='0'; H_Mult<='0'; H_Div<='0';
H_Tri_19<='0'; H_Tri_20<='0'; H_Tri_21<='0';
State<=Esperar;
End if;

end case;
end if;
end process;
end Behavioral;
```

**“BCD_Vin”**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.MATH_REAL.ALL;

entity BCD_Vin is

port(
clk,reset: in std_logic;
Dato: in std_logic_vector(14 downto 0);
Unidad_dec, Decenas_Dec, Decenas, Unidad: out std_logic_vector(7 downto 0)
);

end BCD_Vin;
architecture Behavioral of BCD_Vin is
signal dec: std_logic_vector(9 downto 0);
begin

process(Dato)

variable temp1 : std_logic_vector (6 downto 0);
variable bcd1 : unsigned (8 downto 0) := (others => '0');

begin
bcd1 := (others => '0');

temp1(6 downto 0) := Dato(13 downto 7);
for i in 0 to 6 loop  --Números de desplazamientos
if bcd1(3 downto 0) > 4 then
bcd1(3 downto 0) := bcd1(3 downto 0) + 3;
end if;
if bcd1(7 downto 4) > 4 then
bcd1(7 downto 4) := bcd1(7 downto 4) + 3;
end if;

bcd1 := bcd1(7 downto 0) & temp1(6);
temp1 := temp1(5 downto 0) & '0';

end loop;

Unidad <= "0011"&(STD_LOGIC_VECTOR(bcd1(3 downto 0)));
Decenas <= "0011"&STD_LOGIC_VECTOR(bcd1(7 downto 4));

end process;

process(dato)

variable  x0,x1,x2,x3,x4,x5,x6: integer range 0 to 500;
variable decimal: integer range 0 to 1000;

begin

if Dato(6)='1' then
x0:=500;
else
x0:=0;
end if;

if Dato(5)='1' then
x1:=250;
else
x1:=0;
end if;
```

```
if Dato(4)='1' then
x2:=125;
else
x2:=0;
end if;

if Dato(3)='1' then
x3:=62;
else
x3:=0;
end if;

if Dato(2)='1' then
x4:=31;
else
x4:=0;
end if;

if Dato(1)='1' then
x5:=16;
else
x5:=0;
end if;

if Dato(0)='1' then
x6:=7;
else
x6:=0;
end if;

Decimal:= x0+x1+x2+x3+x4+x5+x6;
dec<=std_logic_vector(to_unsigned(decimal, dec'length));

end process;

process(dec)
variable temp : std_logic_vector (9 downto 0);
variable bcd : unsigned (11 downto 0) := (others => '0');

begin
bcd := (others => '0');

temp(9 downto 0) := dec(9 downto 0);
for i in 0 to 9 loop  --Números de desplazamientos

if bcd(3 downto 0) > 4 then
bcd(3 downto 0) := bcd(3 downto 0) + 3;
end if;

if bcd(7 downto 4) > 4 then
bcd(7 downto 4) := bcd(7 downto 4) + 3;
end if;

if bcd(11 downto 8) > 4 then
bcd(11 downto 8) := bcd(11 downto 8) + 3;
end if;

bcd := bcd(10 downto 0) & temp(9);
temp := temp(8 downto 0) & '0';
end loop;

Unidad_dec <= "0011"&(STD_LOGIC_VECTOR(bcd(7 downto 4)));
Decenas_dec <= "0011"&STD_LOGIC_VECTOR(bcd(11 downto 8));

end process;

end behavioral;
```

## “CONTROLADOR_FK”

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CONTROLADOR_FK is

Port(
clk, reset, Start: in std_logic;
Dato_Vin, Dato_ADC, Dato_R: in std_logic_Vector(63 downto 0);
En_EST, En_KG, En_Eest, En_FK, FK: out std_logic;
cont: out std_logic_vector(3 downto 0);
En_Suma, En_Mult, En_Div, En_Tri_19, En_Tri_20, En_Tri_21: out std_logic;
Add_Sub: out std_logic_Vector(5 downto 0);
H_Sum,H_Add_Sub,H_Mult,H_Div,H_Tri_19,H_Tri_20,H_Tri_21: out std_logic;
En_wA, En_wB: out std_logic_Vector(0 downto 0);
DirA,DirB: out std_logic_Vector(4 downto 0);
DatoA, DatoB: out std_logic_vector(63 downto 0)
);

End CONTROLADOR_FK;

architecture Behavioral of CONTROLADOR_FK is

signal count: integer range 0 to 10:=0;
signal conta: std_logic_vector(3 downto 0):="0000";
type y is(Idle,Inicio,s0,cond,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14, s15,s16,s17,
s18,s19,s20,s21,s22,s23,s24,s25,s26,s27,s28,s29,s30,
s31,s32,s33,s34,s35,s36,s37,s38,s39,s40,s41,s42,s43,s44,s45,s46,s47,s48,s49,s50,s51,s52,s53,s54,s55,
Esperar,Esperar1,Esperar2);
signal State: y;

begin

process(Clk,Reset) begin

If reset='1' then
State<= Idle;
Elsif clk'event and clk = '1' then

Case State is

when Idle =>
If Start='1' then
If count=1 then
count<=0;
En_FK<='1';

State<=Inicio;
Else
count<=count+1;
State<=idle;
End if;
Else
En_FK<='0';

H_Sum<='0'; H_Add_Sub<='0'; H_Mult<='0'; H_Div<='0'; En_Suma<='0'; Add_Sub<="000000"; En_Mult<='0';
En_Div<='0';
H_Tri_19<='0'; H_Tri_20<='0'; H_Tri_21<='0'; En_Tri_19<='0'; En_Tri_20<='0'; En_Tri_21<='0';
En_wA<="0"; En_wB<="0"; DirA<="00000"; DirB<="00000";
State<=Idle;

End if;
```

```
when Inicio=>

H_Sum<='1'; H_Add_Sub<='1'; H_Mult<='1'; H_Div<='1'; En_Suma<='0'; Add_Sub<="000000"; En_Mult<='0';
En_Div<='0';
H_Tri_19<='1'; H_Tri_20<='1'; H_Tri_21<='1'; En_Tri_19<='0'; En_Tri_20<='0'; En_Tri_21<='0';
En_wA<="0"; En_wB<="0"; DirA<="00000"; DirB<="00000";

state<=s0;

when s0 =>

En_wA<="1"; En_wB<="1";
DatoA<=x"4036000000000000";
DatoB<=x"3F847AE147AE147B";
DirA<="00001";
DirB<="00010";

State<=s1;

when s1 =>

En_wA<="1"; En_wB<="1";
DatoA<=x"4000CCCCCCCCCCD";
DatoB<=x"3FF0000000000000";
DirA<="00011";
DirB<="00100";

State<=s2;
when cond =>

State<=s2;
when s2=>

En_wA<="0";
DirA<="00011";
En_wB<="0";
DirB<="00100";
State<=s3;

when s3=>
En_Suma<='1';
Add_Sub<="000000";
State<=s4;

when s4=>
State<=s5;

when s5=>
En_Suma<='0';
En_Tri_19<='1';
State<=s6;

when s6=>
En_Tri_19<='0';
State<=s7;

when s7=>
State<=s8;

when s8=>
En_wA<="0"; En_wB<="1";
DatoB<=Dato_R;
DirA<="00011";
DirB<="00101";
State<=s9;
```

```
when s9=>
En_Div<='1';
State<=s10;

when s10=>
State<=s11;

when s11=>
En_Div<='0';
En_Tri_21<='1';
State<=s12;

when s12=>
En_Tri_21<='0';
State<=s13;

when s13=>
State<=s14;

when s14=>
En_wA<="1"; En_wB<="1";
DirA<="00110";
DirB<="00111";
DatoA<=Dato_R;
DatoB<=Dato_Vin;
state<=s15;

when s15=>
En_wA<="0"; En_wB<="0";
DirA<="00111";
DirB<="00001";
State<=s16;

when s16=>
En_Suma<='1';
Add_Sub<="000001";
State<=s17;
En_KG<='1';

when s17=>
En_KG<='0';
State<=s18;

when s18=>
En_Suma<='0';
En_Tri_19<='1';
State<=s19;

when s19=>
En_Tri_19<='0';
State<=s20;

when s20=>
State<=s21;

when s21=>
En_wA<="0"; En_wB<="1";
DatoB<=Dato_R;
DirA<="00110";
DirB<="01000";
State<=s22;

when s22=>
En_Mult<='1';
State<=s23;

when s23=>
State<=s24;

when s24=>
En_Mult<='0';
En_Tri_20<='1';
State<=s25;

when s25=>
En_Tri_20<='0';
State<=s26;

when s26=>
state<=s27;

when s27=>
En_wA<="0"; En_wB<="1";
DatoB<=Dato_R;
DirA<="00001";
DirB<="01010";
State<=s28;

when s28=>
En_Suma<='1';
Add_Sub<="000000";
State<=s29;

when s29=>
State<=s30;

when s30=>
En_Suma<='0';
En_Tri_19<='1';
State<=s31;

when s31=>
En_Tri_19<='0';
State<=s32;

when s32=>
State<=s33;

when s33=>
En_wA<="1";
DatoA<=Dato_R;
DirA<="00001";
State<=s34;

when s34=>
En_wA<="1"; En_wB<="0";
DatoA<=x"3FF0000000000000";
DirA<="01011";
DirB<="00110";
state<=s35;

when s35=>
En_Suma<='1';
Add_Sub<="000001";
State<=s36;
En_EST<='1';

when s36=>
En_EST<='0';
State<=s37;

when s37=>
En_Suma<='0';
```

```
En_Tri_19<='1';
State<=s38;

when s38=>
En_Tri_19<='0';
State<=s39;

when s39=>
State<=s40;

when s40=>
En_wA<="1"; En_wB<="0";
DatoA<=Dato_R;
DirA<="01100";
DirB<="00011";
state<=s41;

when s41=>
En_Mult<='1';
State<=s42;

when s42=>
State<=s43;

when s43=>
En_Mult<='0';
En_Tri_20<='1';
State<=s44;

when s44=>
En_Tri_20<='0';
State<=s45;

when s45=>
state<=s46;

when s46=>
En_wA<="1"; En_wB<="0";
DatoA<=Dato_R;
DirA<="01101";
DirB<="00010";
state<=s47;

when s47=>
En_Suma<='1';
Add_Sub<="000000";
State<=s48;

when s48=>
En_Eest<='1';
State<=s49;

when s49=>
En_Suma<='0';
En_Tri_19<='1';
State<=s50;
En_Eest<='0';

when s50=>
En_Tri_19<='0';
State<=s51;

when s51=>
State<=s52;

when s52=>
En_wA<="1";
```

```
DatoA<=Dato_R;
DirA<="00011";
state<=s53;

when s53 =>
FK<='1';
State<=s54;

when s54=>
State<=s55;

when s55=>
FK<='0';
En_FK<='0';
H_Sum<='0'; H_Add_Sub<='0'; H_Mult<='0'; H_Div<='0';
H_Tri_19<='0'; H_Tri_20<='0'; H_Tri_21<='0';
State<=Esperar;

when Esperar =>
If Start='1' then
State<=Esperar1;
End if;

when Esperar1=>
En_FK<='1';
state<=Esperar2;

when Esperar2 =>
H_Sum<='1'; H_Add_Sub<='1'; H_Mult<='1'; H_Div<='1'; En_Suma<='0'; Add_Sub<="000000"; En_Mult<='0';
En_Div<='0';
H_Tri_19<='1'; H_Tri_20<='1'; H_Tri_21<='1'; En_Tri_19<='0'; En_Tri_20<='0'; En_Tri_21<='0';
En_wA<="0";  En_wB<="0"; DirA<="00001"; DirB<="00110";

state<=cond;
end case;
end If;
end process;
end Behavioral;
```

## “BCD_KG”

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.MATH_REAL.ALL;

entity BCD_KG is

port(
clk,reset: in std_logic;
Dato: in std_logic_vector(24 downto 0);
Unidad_dec, Decenas_Dec, Centenas_Dec,Uni_mill_dec,Dec_mill_dec, Decenas, Unidad: out std_logic_vector(7
downto 0)
);

end BCD_KG;

architecture Behavioral of BCD_KG is
signal dec: std_logic_vector(16 downto 0);

begin

process(Dato)

variable temp1 : std_logic_vector (6 downto 0);
variable bcd1 : unsigned (7 downto 0) := (others => '0');


begin
bcd1 := (others => '0');
temp1(6 downto 0) := Dato(23 downto 17);

for i in 0 to 6 loop
if bcd1(3 downto 0) > 4 then
bcd1(3 downto 0) := bcd1(3 downto 0) + 3;
end if;

if bcd1(7 downto 4) > 4 then
bcd1(7 downto 4) := bcd1(7 downto 4) + 3;
end if;

bcd1 := bcd1(6 downto 0) & temp1(6);
temp1 := temp1(5 downto 0) & '0';

end loop;

Unidad <= "0011"&(STD_LOGIC_VECTOR(bcd1(3 downto 0)));
Decenas <= "0011"&STD_LOGIC_VECTOR(bcd1(7 downto 4));

end process;

process(dato)
variable  x0,x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16: integer range 0 to 50000;
variable decimal: integer range 0 to 99999;

begin

if Dato(16)='1' then
x0:=50000;
else
x0:=0;
end if;

if Dato(15)='1' then
```

```
x1:=25000;
else
x1:=0;
end if;

if Dato(14)='1' then
x2:=12500;
else
x2:=0;
end if;

if Dato(13)='1' then
x3:=6250;
else
x3:=0;
end if;

if Dato(12)='1' then
x4:=3125;
else
x4:=0;
end if;

if Dato(11)='1' then
x5:=1562;
else
x5:=0;
end if;

if Dato(10)='1' then
x6:=781;
else
x6:=0;
end if;

if Dato(9)='1' then
x7:=390;
else
x7:=0;
end if;

if Dato(8)='1' then
x8:=195;
else
x8:=0;
end if;

if Dato(7)='1' then
x9:=97;
else
x9:=0;
end if;

if Dato(6)='1' then
x10:=48;
else
x10:=0;
end if;

if Dato(5)='1' then
x11:=24;
else
x11:=0;
end if;

if Dato(4)='1' then
x12:=12;
else
x12:=0;
end if;

if Dato(3)='1' then
x13:=6;
else
x13:=0;
end if;

if Dato(2)='1' then
x14:=3;
else
x14:=0;
end if;

if Dato(1)='1' then
x15:=1;
else
x15:=0;
end if;

if Dato(1)='1' then
x16:=0;
else
x16:=0;
end if;

Decimal:= x0+x1+x2+x3+x4+x5+x6+x7+x8+x9+x10+x11+x12+x13+x14+x15+x16;
dec<=std_logic_vector(to_unsigned(Decimal, dec'length));

end process;

process(dec)

variable temp : std_logic_vector (16 downto 0);
variable bcd : unsigned (19 downto 0) := (others => '0');

begin
bcd := (others => '0');

temp(16 downto 0) := dec(16 downto 0);

for i in 0 to 16 loop

if bcd(3 downto 0) > 4 then
bcd(3 downto 0) := bcd(3 downto 0) + 3;
end if;

if bcd(7 downto 4) > 4 then
bcd(7 downto 4) := bcd(7 downto 4) + 3;
end if;

if bcd(11 downto 8) > 4 then
bcd(11 downto 8) := bcd(11 downto 8) + 3;
end if;

if bcd(15 downto 12) > 4 then
bcd(15 downto 12) := bcd(15 downto 12) + 3;
end if;

if bcd(19 downto 16) > 4 then
bcd(19 downto 16) := bcd(19 downto 16) + 3;
end if;

bcd := bcd(18 downto 0) & temp(16);
```

```
temp := temp(15 downto 0) & '0';

end loop;

Unidad_dec <= "0011"&(std_logic_vector(bcd(3 downto 0)));
Decenas_dec <= "0011"&(std_logic_vector(bcd(7 downto 4)));
Centenas_dec <= "0011"&(std_logic_vector(bcd(11 downto 8)));
Uni_mill_dec <= "0011"&(std_logic_vector(bcd(15 downto 12)));
Dec_mill_dec <= "0011"&(std_logic_vector(bcd(19 downto 16)));

end process;


end behavioral;
```

## “BCD_Eest”

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.MATH_REAL.ALL;

entity BCD_KG is

port(
clk,reset: in std_logic;
Dato: in std_logic_vector(24 downto 0);
Unidad_dec, Decenas_Dec, Centenas_Dec,Uni_mill_dec,Dec_mill_dec, Decenas, Unidad: out std_logic_vector(7
downto 0)
);

end BCD_KG;

architecture Behavioral of BCD_KG is
signal dec: std_logic_vector(16 downto 0);

begin

process(Dato)
variable temp1 : std_logic_vector (6 downto 0);
variable bcd1 : unsigned (7 downto 0) := (others => '0');

begin
bcd1 := (others => '0');

temp1(6 downto 0) := Dato(23 downto 17);

for i in 0 to 6 loop

if bcd1(3 downto 0) > 4 then
bcd1(3 downto 0) := bcd1(3 downto 0) + 3;
end if;

if bcd1(7 downto 4) > 4 then
bcd1(7 downto 4) := bcd1(7 downto 4) + 3;
end if;

bcd1 := bcd1(6 downto 0) & temp1(6);
temp1 := temp1(5 downto 0) & '0';

end loop;

Unidad <= "0011"&(STD_LOGIC_VECTOR(bcd1(3 downto 0)));
Decenas <= "0011"&STD_LOGIC_VECTOR(bcd1(7 downto 4));

end process;

process(dato)

variable  x0,x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16: integer range 0 to 50000;
variable decimal: integer range 0 to 99999;

begin

if Dato(16)='1' then
x0:=50000;
else
x0:=0;
end if;
```

```
if Dato(15)='1' then
x1:=25000;
else
x1:=0;
end if;

if Dato(14)='1' then
x2:=12500;
else
x2:=0;
end if;

if Dato(13)='1' then
x3:=6250;
else
x3:=0;
end if;

if Dato(12)='1' then
x4:=3125;
else
x4:=0;
end if;

if Dato(11)='1' then
x5:=1562;
else
x5:=0;
end if;

if Dato(10)='1' then
x6:=781;
else
x6:=0;
end if;

if Dato(9)='1' then
x7:=390;
else
x7:=0;
end if;

if Dato(8)='1' then
x8:=195;
else
x8:=0;
end if;

if Dato(7)='1' then
x9:=97;
else
x9:=0;
end if;

if Dato(6)='1' then
x10:=48;
else
x10:=0;
end if;

if Dato(5)='1' then
x11:=24;
else
x11:=0;
end if;

if Dato(4)='1' then
x12:=12;
else
x12:=0;
end if;

if Dato(3)='1' then
x13:=6;
else
x13:=0;
end if;

if Dato(2)='1' then
x14:=3;
else
x14:=0;
end if;

if Dato(1)='1' then
x15:=1;
else
x15:=0;
end if;

if Dato(1)='1' then
x16:=0;
else
x16:=0;
end if;

Decimal:= x0+x1+x2+x3+x4+x5+x6+x7+x8+x9+x10+x11+x12+x13+x14+x15+x16;
dec<=std_logic_vector(to_unsigned(Decimal, dec'length));
end process;

process(dec)

variable temp : std_logic_vector (16 downto 0);
variable bcd : unsigned (19 downto 0) := (others => '0');

begin
bcd := (others => '0');

temp(16 downto 0) := dec(16 downto 0);

for i in 0 to 16 loop

if bcd(3 downto 0) > 4 then
bcd(3 downto 0) := bcd(3 downto 0) + 3;
end if;

if bcd(7 downto 4) > 4 then
bcd(7 downto 4) := bcd(7 downto 4) + 3;
end if;

if bcd(11 downto 8) > 4 then
bcd(11 downto 8) := bcd(11 downto 8) + 3;
end if;

if bcd(15 downto 12) > 4 then
bcd(15 downto 12) := bcd(15 downto 12) + 3;
end if;

if bcd(19 downto 16) > 4 then
bcd(19 downto 16) := bcd(19 downto 16) + 3;
end if;

bcd := bcd(18 downto 0) & temp(16);
```

```
temp := temp(15 downto 0) & '0';

end loop;

Unidad_dec <= "0011"&(std_logic_vector(bcd(3 downto 0)));
Decenas_dec <= "0011"&(std_logic_vector(bcd(7 downto 4)));
Centenas_dec <= "0011"&(std_logic_vector(bcd(11 downto 8)));
Uni_mill_dec <= "0011"&(std_logic_vector(bcd(15 downto 12)));
Dec_mill_dec <= "0011"&(std_logic_vector(bcd(19 downto 16)));

end process;

end behavioral;
```

**“BCD_FK”**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.MATH_REAL.ALL;

entity BCD_EST is

port(
clk,reset: in std_logic;
Dato: in std_logic_vector(14 downto 0);
Unidad_dec, Decenas_Dec, Decenas, Unidad: out std_logic_vector(7 downto 0)
);

end BCD_EST;

architecture Behavioral of BCD_EST is
signal dec: std_logic_vector(9 downto 0);

begin


process(Dato)

variable temp1 : std_logic_vector (6 downto 0);
variable bcd1 : unsigned (8 downto 0) := (others => '0');


begin
bcd1 := (others => '0');
temp1(6 downto 0) := Dato(13 downto 7);

for i in 0 to 6 loop

if bcd1(3 downto 0) > 4 then
bcd1(3 downto 0) := bcd1(3 downto 0) + 3;
end if;

if bcd1(7 downto 4) > 4 then
bcd1(7 downto 4) := bcd1(7 downto 4) + 3;
end if;

bcd1 := bcd1(7 downto 0) & temp1(6);
temp1 := temp1(5 downto 0) & '0';

end loop;

Unidad <= "0011"&(STD_LOGIC_VECTOR(bcd1(3 downto 0)));
Decenas <= "0011"&STD_LOGIC_VECTOR(bcd1(7 downto 4));
end process;


process(dato)

variable  x0,x1,x2,x3,x4,x5,x6: integer range 0 to 500;
variable decimal: integer range 0 to 1000;

begin

if Dato(6)='1' then
x0:=500;
else
x0:=0;
end if;
```

```
if Dato(5)='1' then
x1:=250;
else
x1:=0;
end if;

if Dato(4)='1' then
x2:=125;
else
x2:=0;
end if;

if Dato(3)='1' then
x3:=62;
else
x3:=0;
end if;

if Dato(2)='1' then
x4:=31;
else
x4:=0;
end if;

if Dato(1)='1' then
x5:=16;
else
x5:=0;
end if;

if Dato(0)='1' then
x6:=7;
else
x6:=0;
end if;

Decimal:= x0+x1+x2+x3+x4+x5+x6;
dec<=std_logic_vector(to_unsigned(decimal, dec'length));

end process;

process(dec)

variable temp : std_logic_vector (9 downto 0);
variable bcd : unsigned (11 downto 0) := (others => '0');

begin
bcd := (others => '0');

temp(9 downto 0) := dec(9 downto 0);

for i in 0 to 9 loop

if bcd(3 downto 0) > 4 then
bcd(3 downto 0) := bcd(3 downto 0) + 3;
end if;

if bcd(7 downto 4) > 4 then
bcd(7 downto 4) := bcd(7 downto 4) + 3;
end if;

if bcd(11 downto 8) > 4 then
bcd(11 downto 8) := bcd(11 downto 8) + 3;
end if;

bcd := bcd(10 downto 0) & temp(9);

temp := temp(8 downto 0) & '0';

end loop;

Unidad_dec <= "0011"&(STD_LOGIC_VECTOR(bcd(7 downto 4)));
Decenas_dec <= "0011"&STD_LOGIC_VECTOR(bcd(11 downto 8));

end process;

end behavioral;
```

## “UART”

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity UART is
port ( Clk,Reset,Start: in std_logic;
cont: in std_logic_vector(3 downto 0);

unidad_Vin,decenas_Vin,Unidad_dec_Vin,Decenas_dec_Vin,
unidad_EST,decenas_EST,Unidad_dec_EST,Decenas_dec_EST,
Unidad_KG,decenas_KG,Unidad_dec_KG,Decenas_dec_KG,Centenas_dec_KG,Uni_mill_KG,Dec_mill_KG,
Unidad_Eest,decenas_Eest,Unidad_dec_Eest,Decenas_dec_Eest,Centenas_dec_Eest,Uni_mill_Eest,Dec_mill_Eest
, Unidad_ADC,decenas_ADC,Centenas_ADC,Uni_millar_ADC: in std_logic_vector(7 downto 0);
End_Uart: out std_logic;
tx: out std_logic
);

end UART;

architecture Behavioral of UART is

signal regtx: std_logic_vector(7 downto 0):="00000000";
signal cnttx: std_logic_vector(15 downto 0):="0000000000000000";
signal ttx: std_logic_vector(3 downto 0):="0000";
signal caracter: integer range 0 to 39:=0;
signal baudtx: std_logic_vector (15 downto 0):= "0001010001011000";
signal End_t: std_logic:='0';
signal count: integer range 0 to 2:=0;


type y is( Idle, Inicio,s0,s1,s2);
signal state: y;


begin
process (clk, Reset)  begin

if clk'event and clk = '1' then
case state is

when idle =>
if Reset='1' then
cnttx<= baudtx-1;
ttx <= "0000";
caracter <= 0;
state <=Idle;
else
state <=Inicio;
end if;

when Inicio =>

if Start='1' then
If count=1 then
count<=0;
state<=s0;
else
count<=count+1;
state<=Inicio;
end if;

end if;
when s0 =>
```

```
cnttx<=cnttx+1;
if (cnttx=baudtx) then
cnttx<=x"0000";
ttx<=ttx+1;

if ttx="1010" then
Caracter<=Caracter+1;
end if;
end if;

if caracter >36 then
caracter<= 0;
state<=s1;

end if;

when s1=>
End_t<='1';
state<=s2;

when s2=>
End_t<='0';
state<=Inicio;

end case;
end if;

End_Uart<=End_t;

end process;

with caracter select

regtx <=

Decenas_Vin      when 0,
Unidad_Vin       when 1,
x"2E"            when 2,
Decenas_dec_Vin  when 3,
"00100000"       when 4,
"00100000"       when 5,
Decenas_KG       when 6,
Unidad_KG        when 7,
x"2E"            when 8,
Dec_mill_KG      when  9,
Uni_mill_KG      when  10,
Centenas_dec_KG  when  11,
Decenas_dec_KG   when  12,
Unidad_dec_KG    when 13,
"00100000"       when 14,
"00100000"       when 15,
Decenas_EST      when 16,
Unidad_EST       when 17,
x"2E"            when 18,
Decenas_dec_EST  when 19,
"00100000"       when  20,
"00100000"       when  21,
Decenas_Eest     when  22,
Unidad_Eest      when  23,
x"2E"            when  24,
Dec_mill_Eest    when  25,
Uni_mill_Eest    when  26,
Centenas_dec_Eest  when  27,
Decenas_dec_Eest   when 28,
Unidad_dec_Eest    when 29,
"00100000"       when 30,
"00100000"       when  31,
Uni_millar_ADC   when  32,
Centenas_ADC     when  33,
Decenas_ADC      when  34,
Unidad_ADC       when  35,
"00001010"       when  36,
x"00"            when others;

with ttx select

tx <=
'1'  when "0000",
'0'  when "0001",
regtx(0)  when "0010",
regtx(1)  when "0011",
regtx(2)  when "0100",
regtx(3)  when "0101",
regtx(4)  when "0110",
regtx(5)  when "0111",
regtx(6)  when "1000",
regtx(7)  when "1001",
'1'  when "1010",
'1'  when others;

end Behavioral;
```

**“DELAY”**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity DELAY is
port(

clk, Start: in std_logic;
En_Delay: out std_logic);
end DELAY;

architecture Behavioral of DELAY is
signal count: integer range 0 to 49999999:=0;
type y is( Idle,s0);
signal state: y;

begin

process (clk)  begin
if clk'event and clk = '1' then
case state is
when Idle =>
if Start='1' then
state<=s0;
else
En_Delay<='0';
end if;

when s0 =>
if count=49999999 then
En_Delay<='1';
count<=0;
state<=Idle;

else

count<=count+1;
state<=s0;

end if;
end case;
end if;
end process;
end Behavioral;
```

**“LCD”**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity LCD is
 port(
 clk, reset:  in std_logic;
 Hex_0,Hex_1,Hex_2,Hex_3,Unidad_dec, Decenas_Dec, Decenas, Unidad, Decenas_dec_EST,Unidad_dec_EST,
Decenas_EST, Unidad_EST: in std_logic_vector(7 downto 0); --,Decenas_dec_EST,Unidad_dec_EST,
Decenas_EST, Unidad_EST
 SF_D : out std_logic_vector(3 downto 0);
 LCD_E, LCD_RS, LCD_RW: out std_logic
);
end LCD;
architecture behavior of lcd is
type display_state is (

init, function_set, entry_set, set_display, clr_display, pause,set_Addr,set_Addr1,
S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,
S13,S14,S15,S16,S17,S18,S19,done
);

signal cur_state : display_state := init;
signal SF_D0, SF_D1 : std_logic_vector(3 downto 0);
signal LCD_E0, LCD_E1 : std_logic;
signal mux : std_logic;
type tx_sequence is (high_setup, high_hold, oneus, low_setup, low_hold, fortyus, done);
signal tx_state : tx_sequence := done;
signal tx_byte : std_logic_vector(7 downto 0);
signal tx_init : std_logic := '0';
type init_sequence is (idle, fifteenms,one, two, three, four, five, six, seven, eight, done);
signal init_state : init_sequence := idle;
signal init_init, init_done : std_logic := '0';
signal i : integer range 0 to 750000 := 0;
signal i2 : integer range 0 to 2000 := 0;
signal i3 : integer range 0 to 82000 := 0;
signal i4: integer range 0 to 500000000 :=0;
signal num : std_logic_vector(3 downto 0);

 begin

 LCD_RW <= '0';
 with cur_state select
 tx_init <= '1' when

 function_set | entry_set | set_display | clr_display| set_Addr| done| set_Addr1|
 S1|S2|S3|S4|S5|S6|S7|S8|S9|S10|S11|S12|S13|S14|S15|S16|S17|S18|S19,

 '0' when others;

 with cur_state select
 mux <= '1' when init,
 '0' when others;

 with cur_state select
 init_init <= '1' when init,
 '0' when others;

 with cur_state select
 LCD_RS <= '0' when  function_set|entry_set|set_display|clr_display|set_Addr|done|set_Addr1,

 '1' when others;
 with cur_state select
```

```
 tx_byte <=

"00101000" when function_set,
"00000110" when entry_set,
"00001100" when set_display,
"00000001" when clr_display,
"10000000" when set_addr,
Decenas  when S1,
Unidad  when S2,
"00101110" when S3,
Decenas_dec when S4,
Unidad_dec when S5,
"11011111" when S6,
"01000011" when S7,
"00100000" when S8,
Hex_3    when S9,
Hex_2    when S10,
Hex_1    when S11,
Hex_0    when S12,
"11000000" when set_addr1,
Decenas_EST when S13,
Unidad_EST    when S14,
"00101110"     when S15,
Decenas_dec_EST when S16,
Unidad_dec_EST   when S17,
"11011111"  when S18,
"01000011" when S19,


"00000000" when others;

 display: process(clk, reset)
begin

if(reset='1') then
cur_state <= function_set;

        elsif(clk='1' and clk'event) then

        case cur_state is

        when init =>

                if(init_done = '1') then
                cur_state <= function_set;
                else
                cur_state <= init;
                end if;

        when function_set =>

                if(i2 = 2000) then
                cur_state <= entry_set;
                else
                cur_state <= function_set;
                end if;

        when entry_set =>

                if(i2 = 2000) then
                cur_state <= set_display;
                else
                cur_state <= entry_set;
                end if;

        when set_display =>

                if(i2 = 2000) then
                cur_state <= clr_display;
                else
                cur_state <= set_display;
                end if;

when clr_display =>
                i3 <= 0;
                if(i2 = 2000) then
                cur_state <= pause;
                else
                cur_state <= clr_display;
                end if;

when pause =>
                if(i4 = 82000) then
                cur_state <= set_addr;
                i4<=0;
                else
                cur_state <= pause;
                i4 <= i4 + 1;
                end if;

when set_addr =>
                if(i2 = 2000) then
                cur_state <= S1;
                else
                cur_state <= set_addr;
end if;

when S1 =>
                if(i2 = 2000) then
                cur_state <= S2;
                else
                cur_state <= S1;
end if;

when S2 =>
                if(i2 = 2000) then
                cur_state <= S3;
                else
                cur_state <= S2;
end if;

when S3 =>
                if(i2 = 2000) then
                cur_state <= S4;
                else
                cur_state <= S3;
end if;

when S4 =>
                if(i2 = 2000) then
                cur_state <= S5;
                else
                cur_state <= S4;
end if;

when S5 =>
                if(i2 = 2000) then
                cur_state <= S6;
                else
                cur_state <= S5;
end if;

when S6 =>
```

```
                if(i2 = 2000) then
                cur_state <= S7;
                else
                cur_state <= S6;
        end if;

        when S7 =>
                if(i2 = 2000) then
                cur_state <= S8;
                else
                cur_state <= S7;
        end if;

        when S8 =>
                if(i2 = 2000) then
                cur_state <= S9;
                else
                cur_state <=S8;
        end if;

when S9 =>
                if(i2 = 2000) then
                cur_state <= S10;
                else
                cur_state <=S9;
        end if;

        when S10 =>
                if(i2 = 2000) then
                cur_state <= S11;
                else
                cur_state <=S10;
        end if;

        when S11 =>
                if(i2 = 2000) then
                cur_state <= S12;
                else
                cur_state <=S11;
        end if;

        when S12 =>
                if(i2 = 2000) then
                cur_state <= set_addr1;
                else
                cur_state <=S12;
        end if;

        when set_addr1 =>
                if(i2 = 2000) then
                cur_state <= S13;
                else
                cur_state <=set_addr1;
        end if;

        when S13 =>
                if(i2 = 2000) then
                cur_state <= S14;
                else
                cur_state <=S13;
        end if;

        when S14 =>
                if(i2 = 2000) then
                cur_state <= S15;
                else
                cur_state <=S14;
        end if;

        when S15 =>
                if(i2 = 2000) then
                cur_state <= S16;
                else
                cur_state <=S15;
        end if;

        when S16 =>
                if(i2 = 2000) then
                cur_state <= S17;
                else
                cur_state <=S16;
        end if;

        when S17 =>
                if(i2 = 2000) then
                cur_state <= S18;
                else
                cur_state <=S17;
        end if;

        when S18 =>
                if(i2 = 2000) then
                cur_state <= S19;
                else
                cur_state <=S18;
        end if;

        when S19 =>
                if(i2 = 2000) then
                cur_state <= set_addr;
                else
                cur_state <=S19;
        end if;

        when done =>
          cur_state <= done;

end case;
end if;
end process display;

with mux select
 SF_D <= SF_D0 when '0',
 SF_D1 when others;
 with mux select
 LCD_E <= LCD_E0 when '0',
 LCD_E1 when others;

  with tx_state select
 LCD_E0 <= '0' when high_setup | oneus | low_setup | fortyus | done,
 '1' when high_hold | low_hold;

 with tx_state select
 SF_D0 <= tx_byte(7 downto 4) when high_setup | high_hold | oneus,
 tx_byte(3 downto 0) when low_setup | low_hold | fortyus | done;

 transmit : process(clk, reset, tx_init)
 begin
 if(reset='1') then
 tx_state <= done;
 elsif(clk='1' and clk'event) then
 case tx_state is
 when high_setup =>
 if(i2 = 2) then
```

```
tx_state <= high_hold;
i2 <= 0;
else
tx_state <= high_setup;
i2 <= i2 + 1;
end if;
when high_hold =>
if(i2 = 12) then
tx_state <= oneus;
i2 <= 0;
else
tx_state <= high_hold;
i2 <= i2 + 1;
end if;
when oneus =>
if(i2 = 50) then
tx_state <= low_setup;
i2 <= 0;
else
tx_state <= oneus;
i2 <= i2 + 1;
end if;
when low_setup =>
if(i2 = 2) then
tx_state <= low_hold;
i2 <= 0;
else
tx_state <= low_setup;
i2 <= i2 + 1;
end if;
when low_hold =>
if(i2 = 12) then
tx_state <= fortyus;
i2 <= 0;
else
tx_state <= low_hold;
i2 <= i2 + 1;
end if;
when fortyus =>
if(i2 = 2000) then
tx_state <= done;
i2 <= 0;
else
tx_state <= fortyus;
i2 <= i2 + 1;
end if;
when done =>
if(tx_init = '1') then
tx_state <= high_setup;
i2 <= 0;
else
tx_state <= done;
i2 <= 0;
end if;
end case;
end if;
end process transmit;


with init_state select
init_done <= '1' when done,
'0' when others;

with init_state select
SF_D1 <= "0011" when ONE | T
"0010" when others;

with init_state select
LCD_E1 <= '1' when ONE | THREE | FIVE | SEVEN,
'0' when others;

power_on_initialize: process(clk, reset, init_init)
begin
if(reset='1') then
init_state <= idle;
elsif(clk='1' and clk'event) then
case init_state is
when idle =>
if(init_init = '1') then
init_state <= fifteenms;
i <= 0;
else
init_state <= idle;
i <= i + 1;
end if;

when fifteenms =>
if(i = 750000) then
init_state <= ONE;
i <= 0;
else
init_state <= fifteenms;
i <= i + 1;
end if;


when ONE =>
if(i = 11) then
init_state<=TWO;
i <= 0;
else
init_state<=ONE;
i <= i + 1;
end if;

when TWO =>
if(i = 205000) then
init_state<=THREE;
i <= 0;
else
init_state<=TWO;
i <= i + 1;
end if;

when THREE =>
if(i = 11) then
init_state<=FOUR;
i <= 0;
else
init_state<=THREE;
i <= i + 1;
end if;

when FOUR =>
if(i = 5000) then
init_state<=FIVE;
i <= 0;
else
init_state<=FOUR;
i <= i + 1;
end if;

when FIVE =>
if(i = 11) then
```

```
 init_state<=SIX;
 i <= 0;
 else
 init_state<=FIVE;
 i <= i + 1;
 end if;

 when SIX =>
 if(i = 2000) then
 init_state<=SEVEN;
 i <= 0;
 else
 init_state<=SIX;
 i <= i + 1;
 end if;

 when SEVEN =>
 if(i = 11) then
 init_state<=EIGHT;
 i <= 0;
 else
 init_state<=SEVEN;
 i <= i + 1;
 end if;

 when EIGHT =>
 if(i = 2000) then
 init_state<=done;
 i <= 0;
 else
 init_state<=EIGHT;
 i <= i + 1;
 end if;

 when done =>
 init_state <= done;
 end case;
 end if;
 end process power_on_initialize;
end behavior;
```