UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERÍA

DESARROLLO DE UNA INTERFAZ DE ADQUISICIÓN PARA UN FOTÓMETRO DOBLE ASTRONÓMICO DE ALTA VELOCIDAD EN PLATAFORMA PC

TESIS

QUE PARA OBTENER EL TITULO DE:

INGENIERO ELÉCTRICO ELECTRÓNICO

PRESENTAN:

HELIOS AARÓN ALVAREZ RIVAS SOLORIO
JORGE VICENTE MORALES VALDEZ

DIRECTOR DE TESIS:

FÍSICO FERNANDO ANGELES URIBE

MÉXICO, D.F. **ABRIL 2006**

Agradecimientos

Gracias Fernando Angeles por ser un maestro y amigo, contigo aprendimos muchísimas cosas que en la escuela no enseñan. Atte. Los Aibos.

Gracias Xochitl, por la paciencia de habernos soportado, no podrás negar que fue muy divertido, por tus correcciones y ayuda para el desarrollo de este trabajo. Muchas Gracias. Xoch se te quiere mil.

Gracias Luis (lamb) por ser un excelente maestro de Linux y programación, porque cuando nos perdimos nos ayudaste un montón. Saludines!!!.

Gracias al M.I. Alberto Fuentes Maya por sus correcciones a esta tesis y así sea un mejor trabajo.

Gracias al Astrónomo Pepe Peña y al equipo del observatorio de Tonantzintla, por su apoyo en las mediciones..

Gracias a la Universidad Nacional Autónoma de México por permitirnos desarrollarnos en sus instalaciones.

Gracias a la Facultad de Ingeniería, por lograr el sueño de dos ingenieros.

Gracias al Instituto de Astronomía por permitirnos el uso de sus instalaciones y su red.

Agradecimientos Helios

Gracias Dios por estar siempre a mi lado, aunque a veces no me daba cuenta de ello, por darme todas las herramientas necesarias para librar los obstáculos y jamas dejarme solo, se que siempre contare contigo. Gracias también por poner a las personas indicadas que han hecho lo que soy, por darme unos padres ejemplares,a mi hermana, así como a mis familiares, amigos y maestros.

Gracias a mis padres por ser un ejemplo a seguir, porque lo que he hecho ha sido gracias a ustedes, por su orientación, por confiar en mi aunque a veces se hayan desesperado. Después de mucho tiempo lo logramos!!!!

Gracias Isis por apoyarme, espero que esto te sirva de ejemplo, recuerda que las cosas no son fáciles, con esfuerzo y dedicación todo se puede.

Gracias a mis abuelitas Alicia y Magdalena, que son los dos pilares de ambas familias, por haber dado todo por nosotros, gracias por su ejemplo.

Gracias maestra Tere† por sus consejos, apoyo y aliento, por que fue una tercera abuela.

Gracias a mis tíos (Mary†, Sol, Moni, Marce, Goya, Irma, Abraham, David†, Victor, Ricardo, Gustavo, Ruben) por sus consejos y ejemplo.

Gracias Yuri, porque también eres una de mis tías.

Gracias a mis amigos casi hermanos Abraham, Gil, Jorge, Samuel, por haberme escuchado y darme apoyo cuando lo necesite, por ser mis amigos inseparables y a lo largo de mi vida demostrarme que cuento con ustedes.

Gracias Jorge Vicente por que solo el que carga el bulto sabe lo que pesa, solo nosotros sabemos el esfuerzo que nos costo realizar este trabajo, gracias por acompañarme a lo largo de la carrera, y fuera de ella, aprendimos mucho, pudimos hacerlo fácil pero no hubiera tenido el mismo sabor. Arriba el Dealer`s Team.

Gracias Rocio por ayudarme a dar el ultimo empujón, e incitar a seguir superándome. Aunque ya lo habían puesto en otra tesis me enseñaste que el verdadero amor si existe y que valió la pena la espera, para poder vivir lo nuestro.

Gracias al grupo del primer semestre, porque junto con ustedes me divertí muchísimo, e hicieron amena la carrera.

A las marías (Teto y Chente) porque con ustedes la carrera fue mas divertida y me ayudaron a ser ingeniero.

Espero este trabajo le pueda servir a otro ingeniero, y si algún día llegan a leer esta tesis, recuerden los futuros ingenieros que no hay que llegar primero si no hay que saber llegar. Esta carrera no es de velocidad sino de resistencia.

Agradecimientos Jorge

Gracias mis padres por su apoyo incondicional, por su amor y cariño, por por ser mis guías y amigos, ustedes son parte de mi éxito. A mi papá en especial, por formar mi carácter y a mi mamá por heredarme su corazón.

Gracias a mi hermana Janik por tu apoyo, por compartir desde siempre muchos de los momentos que nos definieron como las personas que somos ahora, por tu alegría, por tu compañerismo, yo se que vienes a un pasito de mi y eso me hace muy feliz.

Gracias a mis abuelos, Abel y María; Eguiel y María Luisa, por que también son parte de mi formación como persona y por todo su cariño.
Gracias a mi Padrino Carlos, por sus consejos, a mi tío Abel, por su alegría, a mi tío Marcelo por estar siempre al pendiente, a mi tía Lupita, por su porras.
Gracias a mi tío Marce por que indirectamente me ayudo a escoger esta carrera, a mi tío Beto por las horas de charla, a mi tío Javier por pasarme esa forma de ser curioso de mi entorno, a mi tío Tulio por sus chistes, a mi tía Mari por su gran cariño, a mi tío Pepe por ser mi super cuate, a mi tía Rosi por su cariño y amistad.

Hagan por favor extensivo este agradecimiento a sus familias.

A mis primos, Erandy, Moises, Belen, Alfonso, Omar, Abraham, Anahí, Marlen, Karla, Liz, Luis Carlos, Alexa, Diego y los nuevos bebes en camino, espero demostrarles que cuando se quiere se puede y que espero que todos logren sus objetivos como yo estoy cumpliendo uno más de los míos.

Gracias a mi querida Maestra en Astronomía Jill por que siempre estas conmigo demostrándome tu amor y cariño.

Gracias a mis amigos Jose Luis, Carlos, Alejandra, Danitza, Areli, Sandra, Surya, Clementina, Mariela, Juan Luis, Ismael y Helios, por que se que siempre me apoyan como yo a ellos.

Gracias a mis compañeros de carrera, por que sin ellos la vida en la Facultad de Ingeniería no sería tan divertida.

A mi amigo brother Helios, aun que nos llevamos más tiempo del planeado, fue un placer hacer este trabajo contigo.

Gracias a mis profesores por formar en mi una nueva filosofía para enfrentar la vida.

Y principalmente a Dios por ponerme en este camino, del cual he disfrutado mucho.

A los compañeros de carrera les dejo esta cita de Linus Torvalds, yo se que ustedes entenderán el significado.

"Do you pine for the nice days of Minix-1.1, when men were men and wrote their own device drivers?"

Índice de contenido

Introducción....3
Capítulo 1 Fotometría Astronómica....5
1.1 Fotometría:....5
1.1.1 Los fotomultiplicadores....8
1.1.2 Corriente obscura....9
1.1.3 La eficiencia cuántica del cátodo....9
1.1.4 Los pulsos de carga....9
1.1.5 Fotoemisión y emisión secundaria:....9
1.2 Fotoemisión:....11
1.2.1 La fotoemisión....11
1.2.2 Emisión secundaria:....11
1.3 Descripción del fotómetro doble del Instituto de Astronomía....12
1.3.1 Descripción general:....12
1.3.2 Cabezal del fotómetro:....14
1.3.3 Fototubos....16
Capítulo 2 Planteamiento y análisis del problema....17
2.1 Planteamiento general del problema....17
2.2 Características del fotómetro doble del Instituto de Astronomía....18
2.3 Electrónica de Acondicionamiento....19
2.4 Software de Adquisición....19
2.5 Métodos de solución: análisis....20
2.5.1 Primer método de solución diseñando la electrónica y utilizando software libre y de código abierto GNU/Linux....21
2.5.1.1 Ventajas....21
2.5.1.2 Desventajas....22
2.5.2 Segundo método de solución diseñando la electrónica y utilizando software de código cerrado Windows™....22
2.5.2.1 Ventajas....23
2.5.2.2 Desventajas....23
2.5.3 Tercer método de solución, adquirir una solución propietaria de hardware y software, LabVIEW....24
2.5.3.1 Ventajas....25
2.5.3.2 Desventajas....25
2.5.4 Resultado del análisis de los métodos de solución....26
Capítulo 3 Diseño e implementación de la Electrónica de Acondicionamiento y Acoplamiento temporal....27
3.1 Electrónica de Acondicionamiento....28
3.2 Electrónica de Conteo....29
3.3 Interrupciones....33
3.4 Sistema de Interrupciones basados en Controladores de Interrupciones Programables (arquitectura i386)....34
3.5 Interrupciones hardware en Linux....36

Capítulo 4 Diseño e implementación del Software de Adquisición (GUI).....37
4.1 El Sistema Operativo Unix.....37
4.2 El nacimiento del Sistema Operativo GNU/Linux.....38
4.3 El Kernel (Núcleo).....39
4.4 Controlador de dispositivo ó módulo.....41
4.5 Drivers o módulos en Linux.....41
4.5.1 Dispositivos de carácter ó tipo char.....42
4.5.2 Dispositivos de Bloque.....43
4.5.3 Dispositivos de Red (Network).....43
4.5.4 Espacio de usuario (“user space”) y espacio kernel (“kernel space”).....43
4.5.5 Operaciones de Archivo.....45
4.5.6 La estructura de archivo (file).....46
4.6 Código Fuente del driver Fotómetro.....46
4.6.1 Declaración de las librerías de encabezado para la creación del módulo Fotómetro. 46
4.6.2 Declaración de los métodos básicos del fotómetro.....48
4.6.3 Init_module, y cleanup_module.....49
4.6.4 Lectura y escritura.....53
4.6.5 Manejo de interrupciones para el Módulo.....55
4.7 Compilación e instalación del Módulo.....57
4.8 Diseño de la interfaz de Usuario.....59
4.8.1 Diagrama de Flujo para el diseño de la Interfaz de Usuario.....59
5.1 Integración del Sistema.....63
5.2 Pruebas y calibración en laboratorio.....66
5.2.1 Interfaz de Usuario.....73
5.3 Pruebas en Sitio Astronómico.....81
5.3.1 Instalación en el Telescopio.....81
5.3.2 Proceso de encendido del Fotómetro.....84
5.3.3 Proceso de apagado del fotómetro.....86
Capítulo 6 Resultados y Conclusiones.....88
Anexos.....90
Hojas de especificaciones.....90
Circuito Impreso.....99
Código Fuente.....104

Introducción.

En cuanto uno contempla el cielo en una noche despejada es evidente que los objetos celestes a la vista cuentan con brillos distintos. El más luminoso de todos, el sol, es tan intenso que su brillo opaca casi cualquier otro objeto que se encuentre en el cielo en ese momento, por esa razón la mayoría de las estrellas solo son visibles durante la noche. El ojo humano es un detector excelente, por lo que podemos apreciar brillos tan diminutos como 10^{33} veces menores que el del sol. No obstante, eso no es indicativo de que no haya objetos más débiles, simplemente no los podemos observar a simple vista. Para ello, se debe recurrir a los telescopios e instrumentos de detección muy sofisticados que permitan extender nuestros sentidos, así es como se ha descubierto que el universo es vasto y está repleto de toda una variedad de objetos.

La gama es tal que se han desarrollado diferentes disciplinas dentro de la astronomía que se dedican al estudio de cada familia de cuerpos celestes, por ejemplo: estrellas, galaxias, pulsares, etcétera. Una de estas disciplinas, la fotometría, estudia particularmente las variaciones de brillo de las estrellas.

El instrumento utilizado para este fin recibe el nombre de fotómetro. Uno de estos instrumentos, el fotómetro doble rápido, fue adquirido a la universidad de Texas en Austin por el Instituto de Astronomía de la UNAM en 1974. Con esto, se abrieron campos importantes en el estudio de fotometría estelar, glóbulos estelares, así como extinción y evolución de galaxias. De igual forma, pueden obtenerse los colores de las estrellas de diversos tipos espectrales, midiendo sus intensidades en varias longitudes de onda.

Su operación requiere del manejo de una computadora personal, con procesador x86. En una versión anterior se utilizó el procesador intel 80286, en el interior de la computadora, se contaba con tres tarjetas electrónicas, una diseñada en el Instituto de Astronomía la cual realizaba el conteo de pulsos y otro par de tarjetas comerciales para la adquisición y almacenamiento de datos, dicho sistema era controlado por un programa diseñado en turbo Pascal versión 6 para el despliegue y posterior procesamiento de los datos.

Recientemente, el instrumento falló y fue imposible repararlo debido a la antigüedad del instrumento, debido a esto y a las actuales necesidades de los astrónomos así como las nuevas tendencias en instrumentación que exigen alta compatibilidad con los sistemas existentes y operatividad desde sitios remotos para hacer más versátil su uso, se decidió renovar el sistema de adquisición por completo.
Para este fin se debe replantear la electrónica de adquisición, los programas de control del instrumento y las rutinas primarias de preproceso de datos, para ofrecer una herramienta de observación confiable y completa.

Por lo tanto el objetivo de esta tesis es: Reactivar el fotómetro doble rápido del Instituto de Astronomía de la UNAM. Realizando los siguientes pasos.

Realizar un informe del estado del fotómetro doble rápido y restaurarlo a un estado funcional.
Actualizar la electrónica de adquisición obsoleta que gobernaba al instrumento.

Diseñar un controlador que obtenga la información de la electrónica de adquisición y la inserte en el Sistema Operativo para su posterior procesamiento.
Diseñar una interfaz gráfica de usuario amigable para la visualización y almacenamiento de los datos.

El presente trabajo de tesis se desglosa de la siguiente manera:

Capítulo 1.

Es una introducción al estudio de la fotometría y una breve descripción del fotómetro.

Capítulo 2.
Se discute a cerca de los distintos planteamientos y soluciones que encontramos para reactivar las interfases que usa el fotómetro.

Capítulo 3.

Describe el diseño e implementación de la electrónica de acuerdo a la solución propuesta par su implementación.

Capitulo 4.

En este capítulo se desarrolla el diseño del controlador y de la interfaz de usuario.

Capitulo 5.

En este capítulo da la integración del sistema y las pruebas realizadas al mismo, tanto en el laboratorio como en el sitio astronómico.

Capitulo 6.

Finalmente se dan resultados así como las conclusiones.

Capítulo 1 Fotometría Astronómica

En este capítulo se exponen conceptos necesarios para el desarrollo de la tesis, los cuales introducen al lector en temas como: la historia de la fotometría y el desarrollo de la misma, la evolución de los instrumentos y de las técnicas utilizadas en la fotometría. Las características principales del fotómetro así como de sus componentes.

1.1 Fotometría:

La palabra fotometría se deriva de las raíces griegas phos = luz y metrón = medida. La fotometría es una rama de la ciencia que mide la intensidad de la luz.

En los siglos XVI y XVII, el estudio de la luz se llevo acabo en el campo de la óptica geométrica, la cual estudia la interacción de los rayos de luz con espejos, lentes y prismas pero sin relevancia en cuanto a mediciones cuantitativas.

Muchas disciplinas científicas y de ingeniería son dependientes de la fotometría, especialmente cuando no se limita el rango de longitudes de onda al espectro visual. El desarrollo de dispositivos como pantallas de vídeo, proyectores, equipo fotográfico y plantas solares, requieren estudios de fotometría tanto como la exploración satelital y la observación meteorológica, en el campo de la astronomía se refiere a:

- La medición de la distribución espacial de la luz emitida por un objeto celeste en diferentes regiones del espectro.
- El registro en una región espectral específica de las variaciones de brillo de dichos objetos.
- En la interpretación del significado astrofísico de dichas mediciones.

La fotometría de las estrellas es en general, para los astrónomos, una medida directa de la energía emitida por las estrellas a varias longitudes de onda.[12]

El color de las estrellas se determina por mediciones en dos diferentes regiones del espectro usando filtros, da información de la temperatura de las estrellas, algunas veces ambas mediciones son utilizadas como prueba de la existencia de polvo interestelar. La fotometría es frecuentemente usada para establecer la distancia y tamaño de las estrellas.[12]

La fotometría se aplica a algunas estrellas que presentan una variación en la luz que emiten debido a cambios internos, por un eclipse ocasional o por su pareja binaria, en ambos casos, las curvas de luz obtenidas por fotometría conducen a información importante a cerca de la estructura y características

de las estrellas. La fotometría de las estrellas, especialmente a varias longitudes de onda es una de las técnicas observacionales más importantes en la astronomía.

Una persona no necesita su propio telescopio o fotómetro para saber que las estrellas difieren enormemente en apariencia y brillo. En consecuencia es de esperarse que la primera manera de categorizar estrellas fue basado solamente en el ojo humano. Hace más de 2000 años el astrónomo griego Hiparco de Samos dividió a simple vista las estrellas en 6 clases de brillo. Aproximadamente en el 180 antes de cristo Claudio Ptolomeo extendió el trabajo de Hiparco y desde entonces el sistema de magnitudes se volvió parte de la tradición astronómica. El sistema de magnitudes esta basado en el ojo humano, el cual tiene una respuesta logarítmica a la luz. El ojo humano esta diseñado para suprimir diferencias en brillo.

La magnitud aparente en la banda x se puede definir como:

$$M_X = -2.5\log_{10}(F_X) + C$$

donde F_X es el flujo observado en la banda x, y C es una constante que depende de las unidades de flujo y de la banda.

Ejemplos de escala de magnitudes aparentes

Mag. Aparente	Objeto celestial
-26,8	Sol
-12,6	Luna llena
-4,4	Brillo máximo de Venus
-2,8	Brillo máximo de Marte
-1,5	Estrella más brillante: Sirius
-0,7	Segunda estrella más brillante: Canopus
+3,0	Estrellas débiles que son visibles en una vecindad urbana
+6,0	Estrellas débiles visibles al ojo humano
+12,6	Quasar más brillante
+30	Objetos observables más débiles con el Telescopio Espacial Hubble

Es esta característica del ojo la cual permite ir de un cuarto obscuro a la luz del día sin daño alguno. Un tubo fotomultiplicador o una cámara de televisión los cuales responden linealmente no pueden manejar este cambio sin tomar las debidas precauciones. Es esta misma característica la cual hace del ojo un

discriminador muy pobre de pequeñas diferencias de brillo y al tubo fotomultiplicador uno muy bueno. El ojo humano generalmente puede interpolar el brillo de una estrella relativamente cercana con un error de alrededor de 0.2 de magnitud. Esto es un error aceptable para ciertos programas como el sensado en un largo periodo a las estrellas variables de gran amplitud. Debido a la velocidad de las mediciones, la fotometría visual puede ser desarrollada en condiciones del cielo inapropiadas para otro tipo de mediciones. Sin embargo, existen algunos problemas con la fotometría visual, no haciendo menos errores sistemáticos como las diferencias en sensibilidad de color entre los observadores, dificultad de extrapolación de estrellas débiles y falta de precisión.

La fotografía fue rápidamente aplicada a la fotometría por Bond[5] y otros en Harvard en los años 1850. La densidad y el tamaño de la imagen parecen estar directamente relacionada al brillo de la estrella. Sin embargo, las magnitudes determinadas por la placa fotográfica no son, en general las mismas que las determinadas por el ojo.

La fotometría fotográfica sigue en uso, pero principalmente como método de interpolación entre la comparación de estrellas cercanas, dan un error de aproximadamente 0.02 de magnitud. La fotografía ofrece un registro permanente de almacenado de miles de imágenes de una sola vez.

Debido a los errores inherentes en los métodos visuales y fotográficos, la aplicación de métodos fotoeléctricos para medir la luz de las estrellas comenzó a finales de 1800 conduciendo hacia una nueva era en la astronomía. Minchin[5] fue uno de los primeros en usar celdas de selenio fotoconductiva, las cuales cambian su resistencia una vez expuestas a la luz. Estas celdas son similares a las fotoceldas usadas en algunas de las cámaras más modernas. Un voltaje constante es aplicado a la celda y la variación de corriente resultante se mide con un amperímetro.
Una de las mayores desventajas de las celdas de selenio fue su baja sensibilidad (solo las estrellas brillantes y la luna fueron medidas) limitando la respuesta del espectro y careciendo de disponibilidad comercial. Cada celda tiene que construirse individualmente y normalmente requiere de decenas de intentos producir una celda sensitiva.

En 1911 con el descubrimiento de la celda fotoeléctrica prometió mediciones más sensitivas. Estas celdas fueron similares a un tubo tipo diodo usando sodio, potasio u otro electrodo alcalino. Al aplicarles un voltaje de aproximadamente 300 volts y al exponer las celdas a la luz, los electrones liberados por el proceso fotoeléctrico crearon una corriente pequeña. La respuesta obtenida es lineal. Schultz y Stebbins[5] usaron la celda fotoeléctrica para detectar la luz proveniente de Arcturus y Capella. Sistemas similares que se desarrollaron en Europa por Guthnick y Rosenberg[5]. Las celdas fotoeléctricas no estuvieron disponibles de manera comercial hasta 1930.

El amplificador electrónico fue introducido a la astronomía por Whitford[5], incrementando la débil foto-corriente hasta el punto donde se puede hacer uso de medidores más baratos e incluso más importante y utilizar graficadoras. Sin embargo el ruido termiónico y la inestabilidad del amplificador fueron el componente limitante del sistema fotoeléctrico. Entre 1920s y 1930s se vio la llegada de nuevos filtros de banda ancha y una creciente adopción del fotómetro fotoeléctrico.

La invención del tubo electrónico multiplicador o fotomultiplicador a finales de 1930s fue un

importante avance en la astronomía. Este tubo es en esencia un arreglo de fotoceldas, en varias etapas a manera de cascada, las cuales permiten la amplificación de la corriente fotoeléctrica sin ruido. Kron fue el primero en usar el tubo fotomultiplicador para propósitos astronómicos. Con los tubos prototipo y el galvanómetro, tomaron medidas a estrellas de onceava magnitud en un telescopio refractor de 36 pulgadas.
En los años recientes se han visto mejoras en los sistemas fotoeléctricos existentes, pero no cambios mayores. Varias combinaciones de filtros y nuevos materiales de fotocátodo han extendido las mediciones desde el ultravioleta hasta el infrarrojo. Amplificadores con bajo ruido y técnicas de conteo de pulsos han sido desarrolladas para recuperar el débil pulso de corriente. Innovaciones y nuevos diseños en estado de prototipo prometen un brillante futuro para las mediciones fotoeléctricas de la luz de las estrellas.

Las medidas directas del brillo que son obtenidas por cualquier estudio fotométrico son funciones no sólo de los parámetros antes mencionados, sino también de otros más relacionados con el equipo y con la atmósfera. Aún el espacio interestelar por el que viaja puede afectar apreciablemente las mediciones de brillo. En muchos casos es posible obtener estudios fotométricos útiles sin la necesidad de tomar en cuenta cada uno de los factores que gobiernan las mediciones observadas de brillo. Este es el caso cuando sólo se realizan mediciones relativas entre estrellas, como en la fotometría de banda media. En contraste con esto existe la fotometría de banda angosta o espectrofotometría y, en el otro extremo, el trabajo en la banda ancha o radiometría. En este último caso diversos factores como son la transmisión instrumental y las características de reflexión, juegan un papel importante y deben ser considerados; en cambio, en el trabajo de banda media, estos factores requieren de poca o ninguna consideración en la reducción de los datos.

1.1.1 Los fotomultiplicadores

Los fotomultiplicadores actuales son dispositivos fotosensitivos extremadamente versátiles utilizados ampliamente en la detección y medición de energía radiante en el ultravioleta visible y las regiones cercanas al infrarrojo del espectro electromagnético. En este rango, los fotomultiplicadores son los detectores de energía radiante más sensibles de los que pueden disponerse, incluídos los sistemas CCD. La razón primordial es la utilización de amplificación de emisión secundaria, lo que hace posible para los fotomultiplicadores aproximarse las características de un dispositivo ideal limitado sólo por el ruido.
Otras características relevantes incluyen la elección de áreas fotosensibles que varían desde unas cuantas décimas de centímetro cuadrado hasta cientos de centímetros cuadrados, las amplificaciones son internas y su rango varía desde 10 hasta 10^9, su respuesta con tiempos de respuesta tan cortos como un nanosegundo y el nivel de la salida compatible con una electrónica auxiliar sin la necesidad de una amplificación previa.

Son utilizados mayormente en la detección y medición de los rayos X, gama y partículas energéticas, encuentran uso en la investigación nuclear, control industrial y exploración espacial. La existencia de diversos tipos de láseres ha abierto muchos campos de aplicación para los fotomultiplicadores en la espectroscopia de Raman, *ranging* láser, mediciones de precisión y comunicaciones.

1.1.2 Corriente obscura

En un fotomultiplicador, la corriente fluye en el ánodo aún cuando el tubo es operado en una oscuridad completa. La componente de CD de esta corriente es llamada corriente obscura de ánodo. Esta corriente y su ruido resultante delimitan el nivel mínimo de detección de luz del fotomultiplicador. Es generada por tres fuentes principales: pérdidas óhmicas, emisión de electrones desde el cátodo a otros elementos del tubo y efectos regenerativos. La primera es debida a fallas en la manufactura del tubo como contaminación. La emisión de electrones ocurre cuando el potencial en los dínodos se incrementa. Los efectos regenerativos ocurren cuando un electrón o un ión salta al dinodo anterior provocando otra avalancha de electrones esto sucede a voltajes elevados..

1.1.3 La eficiencia cuántica del cátodo

La eficiencia cuántica del cátodo puede definirse como el promedio de fotoelectrones emitidos por fotón incidente y depende de la naturaleza y construcción de cada detector.

1.1.4 Los pulsos de carga

Los pulsos de carga generados en los detectores son amplificados y regulados por amplificadores discriminadores en cada canal y enviados vía cable coaxial a los contadores de alta velocidad localizados en la interfase en la computadora de control.

1.1.5 Fotoemisión y emisión secundaria:

En esta sección será tratada la manera en la que la energía radiante es convertida a energía eléctrica en un tubo fotomultiplicador.

Estos tubos convierten la radiación incidente en las regiones del visible, infrarrojo y ultravioleta en señales eléctricas por medio del fenómeno de fotoemisión y luego son amplificadas por medio de emisión secundaria.

Un arreglo típico de un fotomultiplicador moderno se muestra en la Figura 1.1.

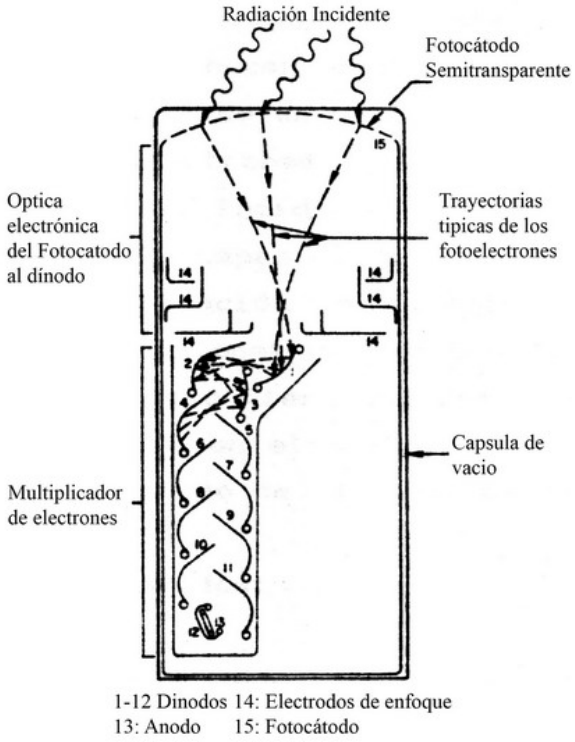


Figura 1.1

La energía radiante entra a través de la ventana que contiene un fotocátodo semitransparente depositado en su superficie interna. Este emite fotoelectrones debido al proceso de fotoemisión, a través de la interacción de la energía radiante incidente con los electrones del fotocátodo. Estos fotoelectrones son acelerados por un campo eléctrico externo de tal manera que golpean un área pequeña correspondiente al primer ánodo llamado dínodo. Los electrones secundarios resultantes del proceso de emisión secundaria, derivados por el impacto de los fotoelectrones con el primer dínodo, son acelerados hacia el segundo por el mismo campo eléctrico. Este proceso es repetido hasta que los electrones dejan el último dínodo chocando contra el ánodo, dejando el fotomultiplicador en la forma de un pulso de corriente.

Si, en promedio, cuatro electrones secundarios son liberados en cada dínodo por cada electrón incidente en él, la amplificación de un fotomultiplicador con 12 etapas es de 4^{12}, o aproximadamente 17 millones. Es decir, la liberación de un solo fotoelectrón en el fotocátodo resulta en 17 millones de electrones colectados en el ánodo. Debido a que este pulso tiene una duración de alrededor de 5 nanosegundos, la corriente en el ánodo es aproximadamente 1 miliamperio en el pico del pulso de la señal de salida.

1.2 Fotoemisión:

1.2.1 La fotoemisión

La fotoemisión es un proceso en el cual los electrones son liberados de la superficie de cierto material por la interacción de fotones de energía radiante con dicho material. La energía de un fotón está dada por la siguiente ecuación:

$$E_{\upsilon} = h\upsilon = \frac{h \cdot c}{\lambda}$$

Donde υ es la frecuencia de la radiación incidente, λ es la longitud de onda de la radiación incidente, h es la constante de Planck y c es la velocidad de la luz. Resolviendo esta ecuación podemos determinar:unidades

$$E_{\upsilon} = \frac{1239.5}{\lambda}$$

Donde $E_{V.}$ está en electrón volts y λ está en nanómetros. Entonces, los fotones de la luz visible que se encuentran en un intervalo de longitud de onda entre 400 y 700 nanómetros tienen energía que varía entre 3. 1 y 1. 8 electrón volts.

1.2.2 Emisión secundaria:

Cuando los electrones golpean la superficie de un material con la suficiente energía cinética, los electrones secundarios son emitidos. La razón de la emisión secundaria, δ se define como:

$$\delta = \frac{N_s}{N_e}$$

Donde N_s es el número promedio de electrones secundarios emitidos por N_e electrones primarios incidentes en la superficie.

Los pasos involucrados en la emisión secundaria se pueden resumir como sigue:

1.-Los electrones incidentes interaccionan con los electrones del material y los excitan a un estado más elevado de energía.

2.-Algunos de estos electrones excitados se mueven a la interfase de vacío-sólido. Enseguida se muestra un diagrama de Energía en la Figura 1.2.

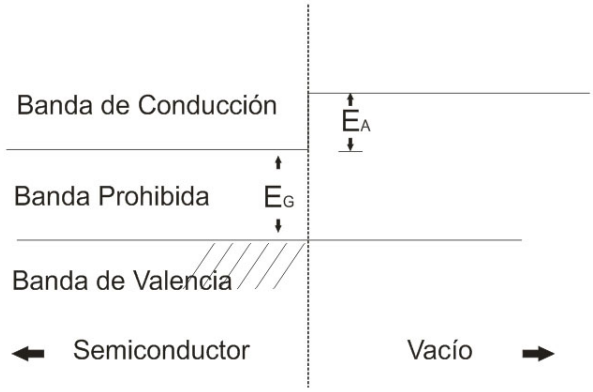


Figura 1.2

3.-Los electrones, que llegan a la superficie con mayor energía que la generada como barrera en la superficie son emitidos al vacío.

Cuando el rayo primario de electrones se impacta con el material del dínodo, su energía es disipada dentro del material y un número de electrones excitados se producen. Otra vez, el campo eléctrico externo los acelera ya en el vacío hacia el ánodo antes descrito.

Al hacer contacto con el ánodo, se produce un pulso de corriente que es convertido a niveles de voltajes reconocibles por la electrónica de adquisición, siendo sensado entonces dicho evento por la computadora de control.[5]

1.3 Descripción del fotómetro doble del Instituto de Astronomía.

Para realizar los estudios fotométricos descritos en la introducción, se utiliza un instrumento llamado fotómetro, capaz de recibir, procesar señales luminosas provenientes de una fuente emisora y generar un tren de pulsos cuya frecuencia es proporcionar al brillo de dicha fuente. A continuación se da una explicación de su funcionamiento.

1.3.1 Descripción general:

El fotómetro doble o de dos canales, es un instrumento diseñado para la medición de la intensidad de dos estrellas simultáneamente, cuyos resultados pueden guardarse en una unidad de disco y/o ser vistos de forma continua en un monitor. Esta forma de operación es llamada fotometría integral ó fotometría de alta velocidad, por su habilidad de utilizar escalas de tiempo cortas.

El fotómetro cuenta con un sistema de adquisición que registra los datos provenientes de los dos canales y los despliega en el monitor de la computadora. También almacena los datos en un archivo preseleccionado.

Se puede decir que el fotómetro lo forman dos grandes partes: el cabezal y el sistema de adquisición de datos.

El instrumento detector está formado esencialmente por un par de fototubos montados en un solo guiador excéntrico que trabajan con el principio de conteo de fotones; el canal principal, usualmente llamado canal A es utilizado normalmente para medir o sensar la estrella en estudio, mientras que el segundo fototubo, montado sobre el ocular del guiador excéntrico, puede ser utilizado para medir la intensidad de una estrella de referencia.

La razón de usar dos estrellas es para garantizar que las mediciones sean correctas, tomando en cuenta que la estrella de referencia tiene un brillo continuo y se debe considerar cualquier variación en este canal al interpretar los datos obtenidos de la observación de la estrella en estudio. Las fuentes de interferencia que pueden afectar una observación se deben principalmente a fenómenos atmosféricos, por ejemplo la presencia de nubes. La variación en ambos canales, es una comprobación de este tipo.

Si este guiador excéntrico no es necesario, el segundo fototubo puede utilizarse para medir la intensidad del cielo.
A continuación en la Figura 1.3 se muestra un dibujo esquemático del instrumento con una breve descripción de sus partes:

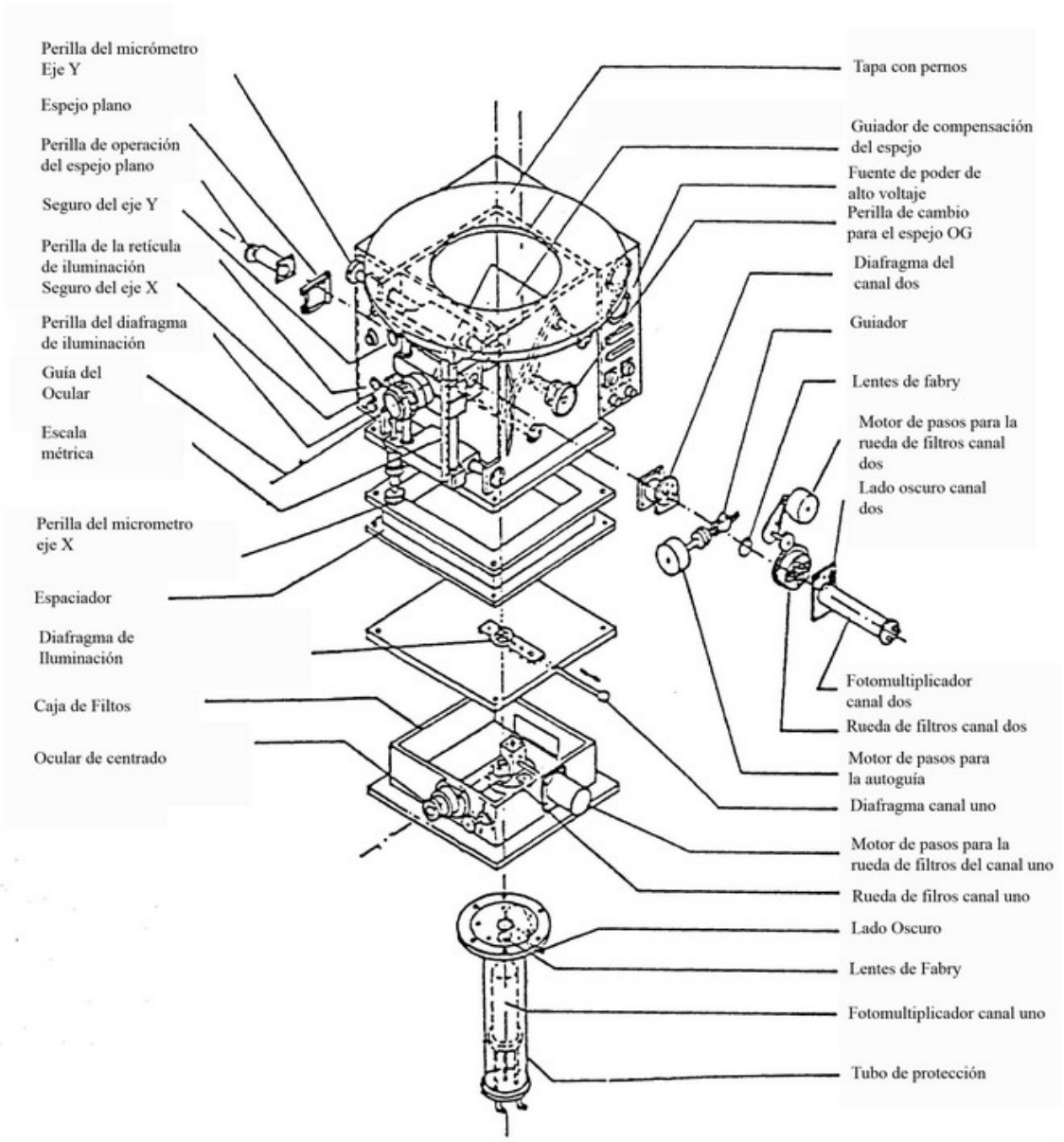


Figura 1.3 dibujo esquemático del fotómetro

1.3.2 Cabezal del fotómetro:

El sistema fotométrico completo se divide en dos partes: la primera contiene la parte óptica, las aperturas, filtros detectores manejados por motores de pasos, sus respectivos manejadores y la electrónica necesaria para manejar cables largos, mientras que la segunda parte consiste en la computadora personal y su interfaz.
La primera parte, el cabezal del fotómetro, debe estar sujeto al telescopio y ser operado desde ahí,

mientras que la segunda parte puede ser ubicada en cualquier lugar que se considere conveniente, generalmente en el cuarto de control.

El cabezal del fotómetro se muestra en la Figura 1.4. El guiador excéntrico tiene dos espejos móviles. El primero con dos posibles posiciones y un orificio central. La primera posición, es de 45 grados, permite a la luz emitida por la estrella llegar directamente al detector, sólo pasando a través de las lentes Fabry, por lo que ésta será, la posición de medición. En la segunda posición, el espejo refleja la imagen al ocular superior, que contiene una retícula iluminada que ayuda al centrado de la estrella en estudio y, en una etapa posterior de medición, al seguimiento de la estrella de referencia.

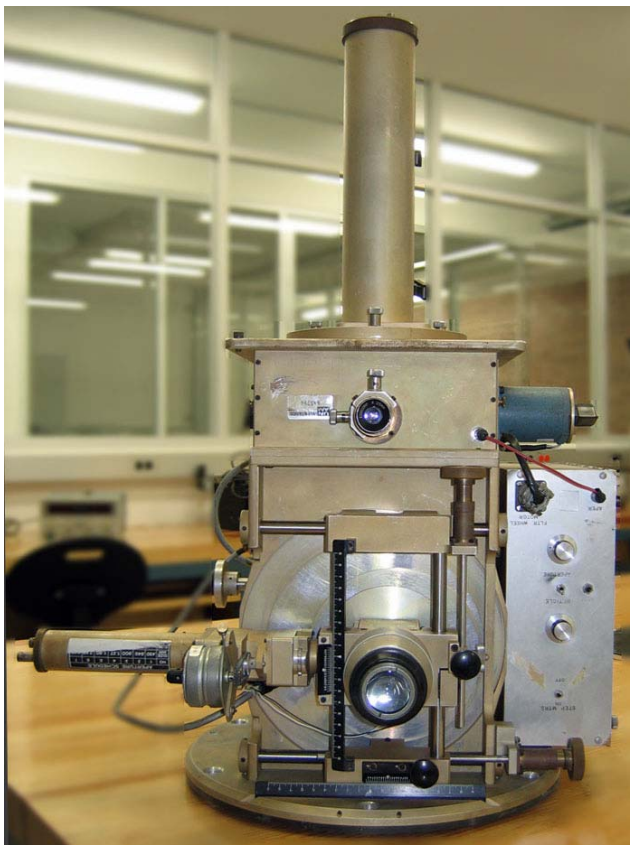


Figura 1.4 Cabezal del fotómetro

Cuando el fotómetro ha sido apropiadamente ajustado, el colocar la estrella exactamente en el centro de la retícula garantiza que también va a estar centrada en la apertura del canal uno.

La estrella puede ser vista a través del ocular inferior. Las aperturas son iluminadas por diodos emisores de luz (LED's) rojos cuyo brillo es controlado por perillas cercanas a los oculares.

Los filtros y diafragmas utilizados están colocados en dos en revólveres independientes, cada uno colocado frente a cada fototubo. Dependiendo de la medición deseada por el astrónomo, los filtros son seleccionados de manera directa o por control remoto desde la computadora. Esto es gracias a que,

junto con la electrónica, se habilitaron controles para los motores de pasos ya existentes en el instrumento.

1.3.3 Fototubos.

El fototubo es un RCA 8850 (Figura 1.5) de naturaleza bialcalina. Normalmente se opera sin refrigerar, ya que el conteo de la corriente obscura generada por la temperatura es menor a 100 cuentas/seg. Considerando que en un segundo puede haber alrededor de 20 millones de cuentas, vemos que la razón 100/20,000,000 es suficientemente pequeña y podemos despreciarla. Trabaja a un potencial de 1600 V.[14]

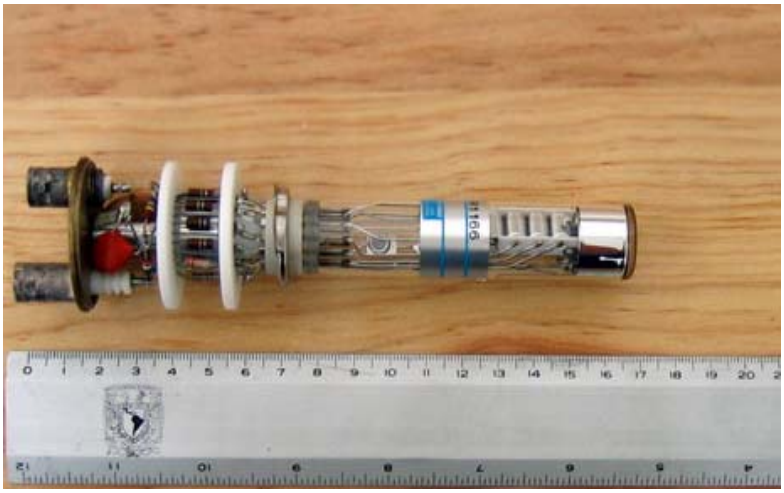


Figura 1.5 Fototubo

Capítulo 2 Planteamiento y análisis del problema

En este capítulo se discuten distintas soluciones para resolver el problema de la reactivación del fotómetro doble rápido del Instituto de Astronomía de la UNAM, describiendo las ventajas y desventajas de cada uno de acuerdo a las necesidades del Instituto y en general de los astrónomos que van a hacer uso de él.

2.1 Planteamiento general del problema.

El problema se divide en dos partes, primero desarrollar la electrónica de adquisición del instrumento dado que la electrónica con la que contaba ya no es funcional y segundo diseñar e implementar el software de control e interfaz. El sistema debe cumplir varias características como las expuestas a continuación.

El astrónomo necesita:

I. Registrar las variaciones de brillo de un objeto celeste. Para ello se utiliza la base optomecánica y fotoelectrónica del fotómetro doble.

II. Que la información adquirida a través del fotómetro sea desplegada gráficamente, en donde se represente en un eje el número de cuentas de los fotones y en el otro el tiempo universal. La estrella en observación y la estrella de referencia se deben mostrar en gráficas independientes.

III. Almacenar los datos, para después procesarlos, interpretarlos y compararlos contra otras observaciones.

IV. Que los datos almacenados incluyan las características de la observación, como son la hora y fecha, los filtros, el diafragma, el tiempo de integración, un espacio dentro del archivo para guardar comentarios de la observación, las cuentas de los fotones en tiempo real por canal.

Para satisfacer el primer requerimiento, se requiere de un instrumento capaz de sensar dichas variaciones. El fotómetro doble es capaz de llevar a cabo estas mediciones por medio de la óptica, mecánica y electrónica de la cual esta compuesto. El brillo emitido por las estrellas se puede medir a partir de los fotones. Por lo que hay que rehabilitar el instrumento para su correcto funcionamiento.

El segundo punto se descompone en el diseño de la electrónica de adquisición y en la interfaz de usuario (software).

Los puntos tres y cuatro son características que el software debe cumplir.

Analizando la parte de la electrónica de adquisición, el fotómetro doble rápido entrega una serie de pulsos, los cuales deben ser acondicionados eliminando posibles fuentes de ruido, por lo que se necesita algún método de calibración. Después de acondicionar los pulsos, hay que contarlos y posteriormente insertarlos en la Computadora Personal para su posterior procesamiento, almacenamiento y despliegue gráfico utilizando algún medio como el monitor de la PC.

La siguiente Figura 2.1 muestra gráficamente el problema.

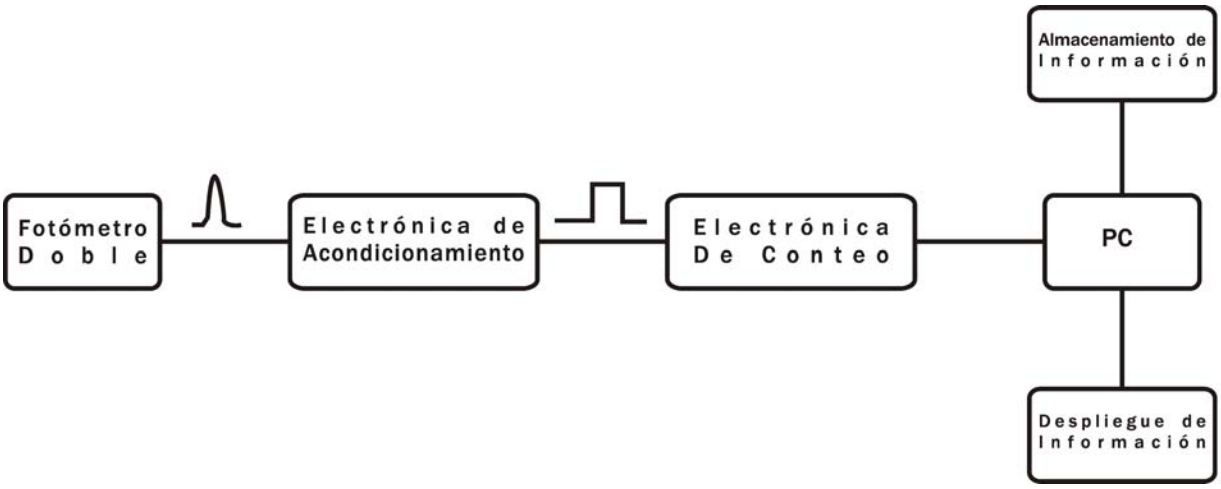


Figura 2.1: Descripción del Problema

2.2 Características del fotómetro doble del Instituto de Astronomía.

El fotómetro es capaz de recibir y procesar señales luminosas provenientes de una fuente emisora y generar un tren de pulsos cuya frecuencia es proporcionar al brillo de dicha fuente.

El tren de pulsos que entrega el fotómetro doble, son pulsos aperiódicos, con amplitud de 5V, dichos pulsos se pueden detectar con un intervalo mínimo de 27ns, que es el tiempo máximo en el que los fototubos pueden recuperar los electrones que pierden al haber llegado los fotones. La electrónica asociada de preamplificación cumple con esta especificación.

Tomando en cuenta estas características se requiere tener una medición del tren de pulsos de salida del

fotómetro, para ello se necesita el diseño del *hardware* y *software* que se muestra en la Figura 2.2.

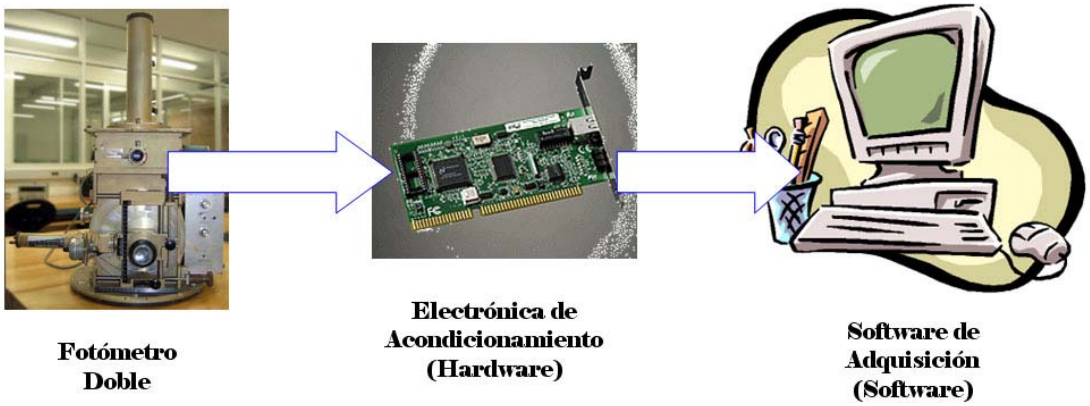


Figura 2.2: Diagrama de Bloques del problema

A continuación detallaremos los bloques de la electrónica y del software.

2.3 Electrónica de Acondicionamiento.

La electrónica de acondicionamiento debe ser capaz de preparar el tren de pulsos arrojado por el amplificador discriminador, tener un medio de calibración para eliminar posibles fuentes de ruido, de contar los pulsos que entrega el fotómetro, también debe poseer un reloj que genere un tiempo confiable para llevar a cabo las mediciones. Otro punto que debe cumplir la electrónica es conectar la parte de adquisición con un bus de datos en la computadora, en un formato adecuado para su posterior procesamiento vía software.

2.4 Software de Adquisición

El Software de Adquisición debe interpretar los datos que recibe de la electrónica de adquisición, conteo y comunicación en tiempo real duro, el software debe contar con una interfaz gráfica fácil de usar por el usuario, también con capacidades como almacenar los datos adquiridos, procesarlos y graficarlos.

A continuación se describen distintos métodos de solución para este problema.

2.5 Métodos de solución: análisis

Para el problema antes planteado las posibles soluciones son las siguientes:

1. Diseñar la electrónica de acondicionamiento, conteo y adquisición, diseñar e implementar el software de adquisición que se compone de un controlador, para un sistema operativo de código abierto, como es el caso de GNU/Linux. Diseñar e implementar la interfaz gráfica de usuario en algún entorno de programación visual para hacer el software amigable y fácil de usar, reduciendo el tiempo de aprendizaje en el uso del sistema.

2. Diseñar la electrónica de acondicionamiento, conteo y adquisición, diseñar e implementar el software de adquisición que se compone de un controlador, para un sistema operativo de código cerrado, como es el caso de Windows™. Diseñar e implementar la interfaz gráfica de usuario en algún entorno de programación visual para hacer el software amigable y fácil de usar, reduciendo el tiempo de aprendizaje en el uso del sistema.

3. Comprar una solución propietaria de hardware (tarjeta de adquisición de datos) y software comercial (LabView, por ejemplo), operando bajo el sistema operativo Windows™.

2.5.1 Primer método de solución diseñando la electrónica y utilizando software libre y de código abierto GNU/Linux.

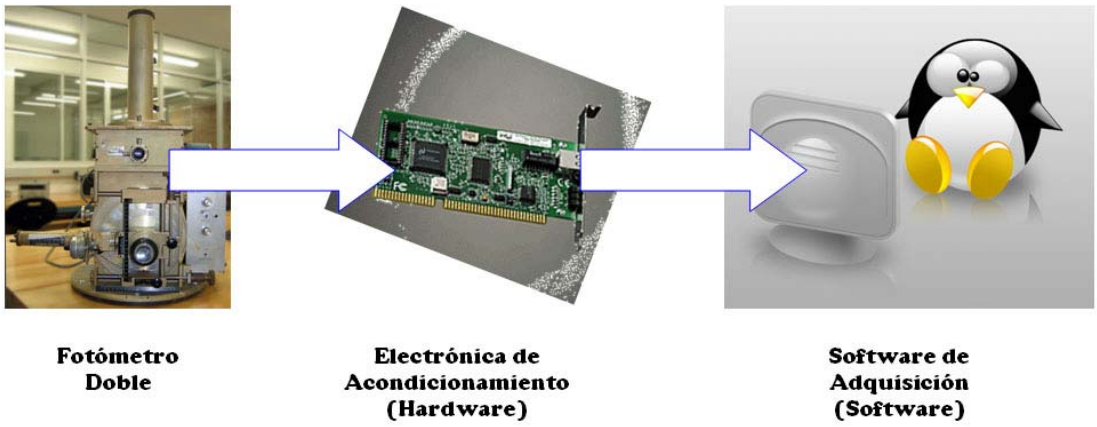


Figura 2.3

La solución planteada en la Figura 2.3 se basa primero en el diseño total de la electrónica que se compone de el modulo de acondicionamiento, conteo y adquisición, y del software que se compone de un controlador e interfaz gráfica de usuario. Al realizar el diseño e implementación total podremos dar una solución mas especifica, teniendo un mayor control de la solución.

En lo que respecta al software esta solución contempla el uso del sistema operativo GNU/Linux y de software de código abierto, las ventajas y desventajas de el uso de este sistema se enumeran a continuación.

2.5.1.1 Ventajas

- El costo del sistema operativo es prácticamente gratuito, al ser un sistema libre y de código abierto.
- Es relativamente fácil modificar el sistema para tener una solución mas personalizada.
- Es más confiable en cuanto a la seguridad en redes y estabilidad.
- Los requerimientos de la computadora son menores.
- La gama de herramientas y facilidades otorgadas por este sistema en cuanto a la programación es mayor.
- Un control más transparente en el uso de los puertos y buses de la computadora.

- Todo el sistema esta documentado, de esta manera es posible encontrar información de todas las partes que componen al sistema.
- Tiene una comunidad de desarrolladores muy activa, en varios idiomas y países.

2.5.1.2 Desventajas

- Falta de soporte de las compañías de *hardware*. sin embargo, existen algunos inconvenientes en el diseño de la electrónica de acondicionamiento debido a la falta de compatibilidad del *hardware* ya que no muchas compañías fabrican productos compatibles con este sistema operativo por lo que la implementación de la electrónica periférica es mas práctico si la realizamos.
- Conocer el funcionamiento del sistema operativo lleva más tiempo, debido a que hay mucha información.
- Son necesarios amplios conocimientos de programación.

2.5.2 Segundo método de solución diseñando la electrónica y utilizando software de código cerrado Windows™

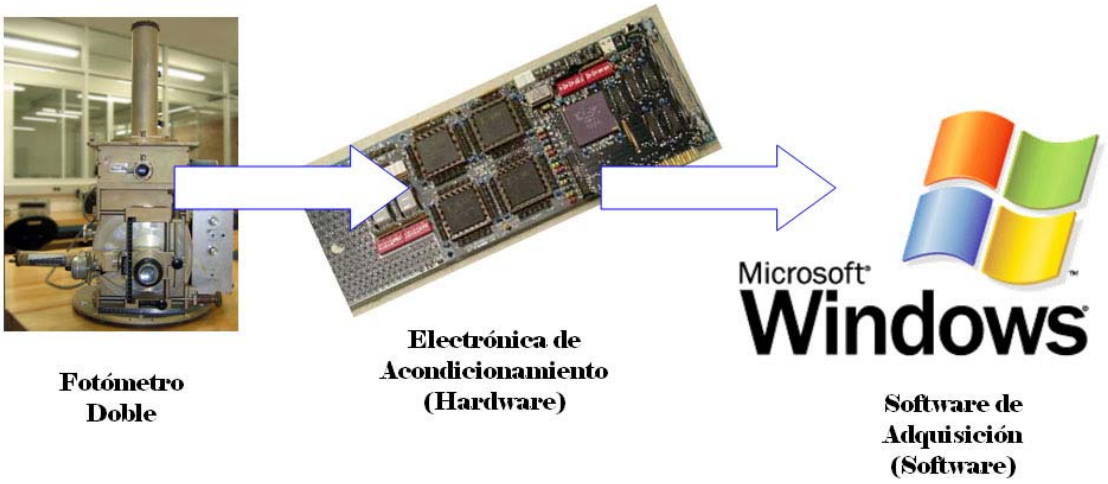


Figura 2.4 Diseño de la electrónica, implementación bajo el sistema operativo Windows™

La solución mostrada en la Figura 2.4 se basa en el diseño total de la electrónica que se compone de el modulo de acondicionamiento, conteo y adquisición, y del software que se compone de un controlador

e interfaz gráfica de usuario. Al realizar el diseño e implementación total podremos dar una solución mas especifica, teniendo un mayor control de la solución.

En lo que respecta al software esta solución contempla el uso del sistema operativo Windows™. y de software de código cerrado, las ventajas y desventajas de el uso de este sistema se enumeran a continuación.

2.5.2.1 Ventajas

- El ambiente Windows™. es manejado por una gran cantidad de personas, por lo que el tiempo de aprendizaje seria menor.
- Es compatible con la mayoría del hardware del mercado.

2.5.2.2 Desventajas

- Este sistema operativo necesita de manejadores o *drivers* para poder implementar la electrónica de adquisición, Debido a que la compañía Microsoft™. no comparte el código fuente no podemos llevar a cabo la comunicación entre la tarjeta y el sistema operativo por la falta de un manejador.

- Windows™. no es un SO en tiempo real,lo cual dificulta llevar a cabo las mediciones del fotómetro en tal sistema.

- El sistema operativo Windows presenta vulnerabilidades de seguridad, por lo que el sistema ademas necesita de soluciones de seguridad como son antivirus y cortafuegos esto hace que necesitemos mayores recursos en el *hardware*.

- El costo de las licencias asociadas a el sistema operativo y al entorno de programación.

2.5.3 Tercer método de solución, adquirir una solución propietaria de hardware y software, LabVIEW.

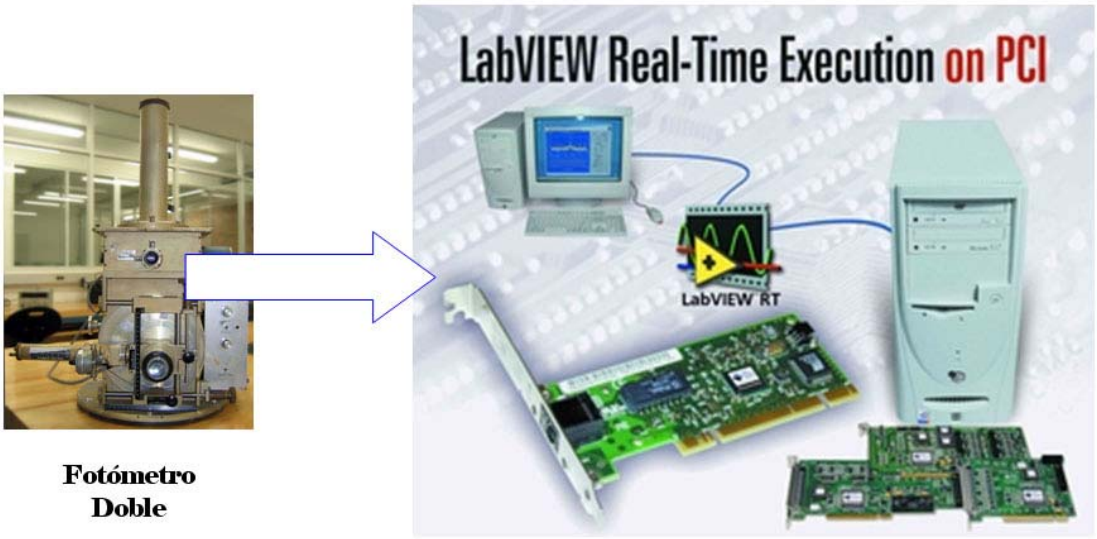


Figura 2.5

En la Figura 2.5 se muestra el módulo de LabVIEW para aplicaciones en tiempo real, el cual consta de una tarjeta PCI para comunicarse con una PC bajo el sistema operativo Windows™, el software se debe adquirir por separado ya que no esta incluido en el módulo, la PC debe tener los siguientes requerimientos de software y hardware:

Windows 2000/XP
- 128 MB RAM, 256 MB recomendado
- Pentium III/Celeron 600 MHz;
- 250 MB de espacio en disco duro, 670 MB recomendado
- manejadores o drivers TCP instalados

Necesita LabView completo o profesional instalado en el sistema.

Para evaluar la factibilidad de este método se solicitó la siguiente cotización.

1	LabVIEW Real-Time Module (ETS) for Windows	$ 2,495.00 USD
1	Standard Service Program for LabVIEW Real-Time (ETS) for Windows	$ 620.00 USD
1	LabVIEW Professional System for Win 2000/NT/XP (English)	$ 4,370.00 USD
		$ 7,485.00 USD

Los precios incluyen costos de importación, IVA y envío a México.

2.5.3.1 Ventajas

- Mayor velocidad en la implementación

Esta opción presenta probablemente la mayor velocidad de implementación y también un mayor costo en la misma, exige una computadora muy específica debido a los requerimientos mínimos de tales programas, así como una instalación voluminosa y comprometida. Por otro lado, no es fácil integrar funciones como remotización vía Internet y la curva de aprendizaje del software comercial retrasaría las últimas etapas.

2.5.3.2 Desventajas

- Su implementación tiene un costo muy elevado.
- Exige una computadora especifica debido a los requerimientos mínimos del software.
- Necesita instalación voluminosa y comprometida.
- No es fácil integrar funciones como remotización vía internet.
- La curva de aprendizaje del *software* retrasaría las ultimas etapas.
- Cuando se compra tecnología patentada por alguna empresa se crea una dependencia tecnológica hacia la misma, ya que si dicha compañía decide hacer modificaciones, actualizaciones o simplemente descontinuar el *software* o *hardware* del cual se estaba haciendo uso, ocasiona tener que hacer un nuevo diseño sobre otro de sus productos al no encontrar las refacciones apropiadas con la correspondiente pérdida de tiempo hasta la nueva puesta en

marcha del proyecto, así como también si el *hardware* o software requieren más capacidades de la PC o si ya no es compatible con el sistema operativo, o si el sistema operativo cambia y ya no es compatible con el *software*.

2.5.4 Resultado del análisis de los métodos de solución.

Tomando en cuanta las ventajas y desventajas de los tres métodos propuestos de solución, el método que elegimos es el primero, ya que nos permite un mayor control y personalización del sistema de adquisición tanto en la parte electrónica como en la parte del software, en el capítulo tres y cuatro desarrollamos la parte de la electrónica y el software. Cabe señalar que esta solución si se requiere en un futuro también puede ser migrada a el segundo método de solución, adaptando el software a este sistema operativo.

Capítulo 3 Diseño e implementación de la Electrónica de Acondicionamiento y Acoplamiento temporal.

En este capítulo se desarrolla la electrónica de adquisición para el fotómetro doble rápido del Instituto de Astronomía de la UNAM en base a la primer método de solución del capítulo anterior que consiste en diseñar toda la electrónica y usar el sistema operativo GNU Linux, ya que esta solución es la que nos permite tener un mayor control del proyecto, en cuanto a personalización y optimización del sistema de adquisición, tanto en el sistema operativo como en la electrónica, ademas es la solución que nos permite practicar los conocimientos adquiridos durante nuestra formación académica y también adquirir conocimientos en áreas donde normalmente el ingeniero electrónico no incursiona.

Como se explicó anteriormente los fotones inciden en el fotocátodo del fotomultiplicador, En el fotocátodo, la luz incidente produce una emisión de electrones (fotoelectrones) que son acelerados y focalizados sobre el primer dínodo. Estos electrones al chocar con el dínodo, ceden parte de su energía en arrancar nuevos electrones, de tal forma que por cada uno que llega salen varios (efecto multiplicador), los cuales a su vez son acelerados hacia el segundo dínodo y al chocar con el mismo se produce de nuevo un efecto multiplicador y así sucesivamente. Esto se observa mas claramente en la Figura 3.1.[21]

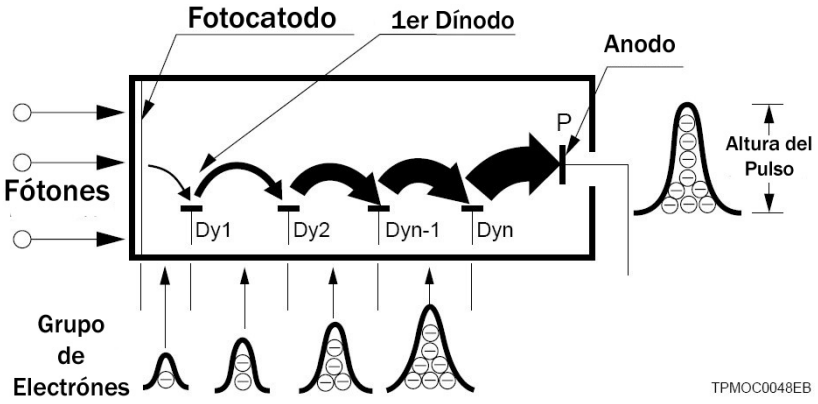


Figura 3.1: Aquí se Observa el grupo de electrones provenientes del efecto multiplicador

El punto de partida en el diseño de la electrónica son los voltajes recibidos del fotómetro después de sus etapas de amplificación, como ya se explico anteriormente, los pulsos tienen las siguientes características:

Son pulsos aperiódicos, con amplitud de 5V, dichos pulsos se pueden detectar con un intervalo mínimo de 27ns, que es el tiempo máximo en el que los fototubos pueden recuperar los electrones que pierden al haber llegado los fotones. Por lo cual empezamos con la electrónica de acondicionamiento.

3.1 Electrónica de Acondicionamiento

El fotómetro entrega un tren de pulsos que tienen que ser medidos con la electrónica de acondicionamiento, primero la señal se compara para poder eliminar posibles cuentas espurias causadas por ruido en el detector. El voltaje positivo de entrada del comparador tiene un umbral para eliminar dicho ruido y puede ser calibrado, este valor es tomado de la referencia de voltaje U1 , la cual es muy estable, U1 en el diagrama es el circuito LM385Z. En la Figura 3.2 se muestra la parte del diagrama esquemático correspondiente a esta descripción, los capacitores que aparecen son de desacoplo.

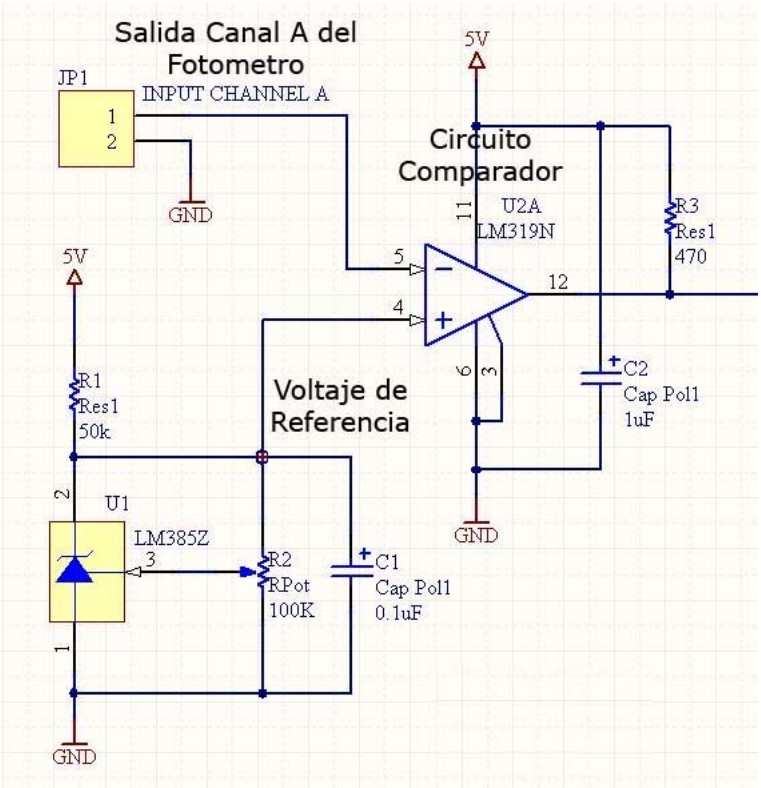


Figura 3.2 Electrónica de Acondicionamiento.

Una vez acondicionados los pulsos, continuamos con la electrónica de conteo.

3.2 Electrónica de Conteo

La señal de salida del comparador va a la entrada del contador de 8 bits que esta conectado en cascada con otro contador para así entregar un arreglo de 16 bits, la señal del reloj de 1ms estará siempre activa para poder tomar las mediciones en dicho tiempo, paralelamente otro circuito (DM74S138N) controla los contadores para poder habilitarlos y deshabilitarlos según el canal que se requiera leer y así poder adquirir los datos provenientes del fotómetro. En la Figura 3.3 se muestra la parte del diagrama esquemático correspondiente a esta descripción.

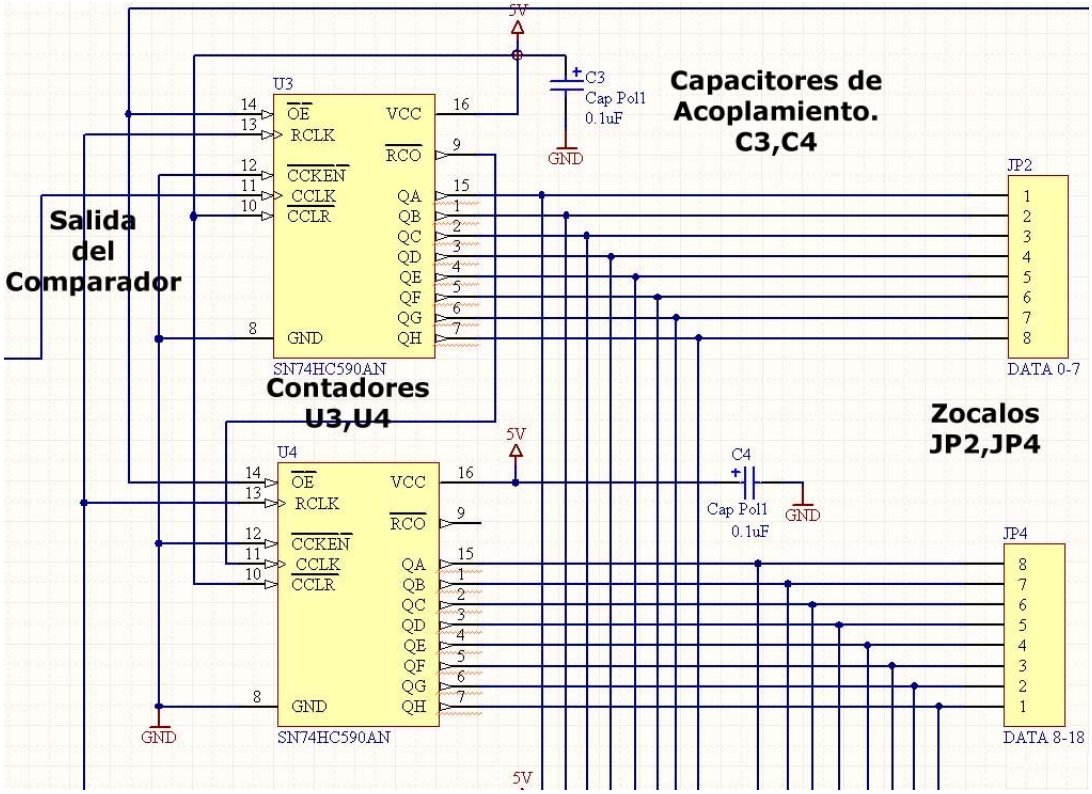


Figura 3.3 Electrónica de Conteo.

El motivo de que el contador sea de 16 bit es que se necesitan mínimo un promedio de 37 000 cuentas por milisegundo y con este arreglo garantizamos al menos 65 535 cuentas.

Las salidas de los contadores son acopladas a dos zócalos (*headers)* de la tarjeta JDR *Microdevices*, la cual se encarga de realizar la comunicación con el bus ISA, esta tarjeta cuenta con electrónica que se encarga de la decodificación del bus y permite la escritura de los puertos. Por medio la tarjeta JDR vamos a obtener los datos de salida de los contadores y también podremos controlar los mismos.

La tarjeta JDR se muestra en la Figura 3.4.

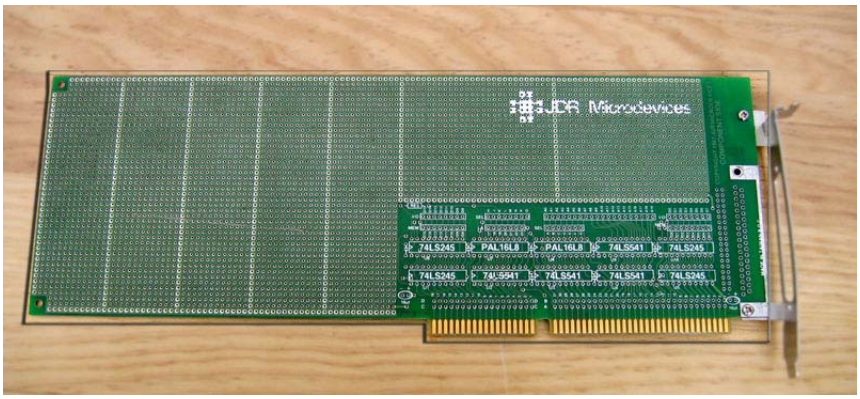


Figura 3.4 Tarjeta JDR *Microdevices*

El reloj de 1ms se obtiene de un cristal de 2MHz, al pasar por tres contadores de década se reduce a una frecuencia de 2kHz o un periodo de 500 µs y por ultimo con un contador que esta configurado para dividir la frecuencia en dos obteniendo un periodo de 1ms como se muestra en la Figura 3.5

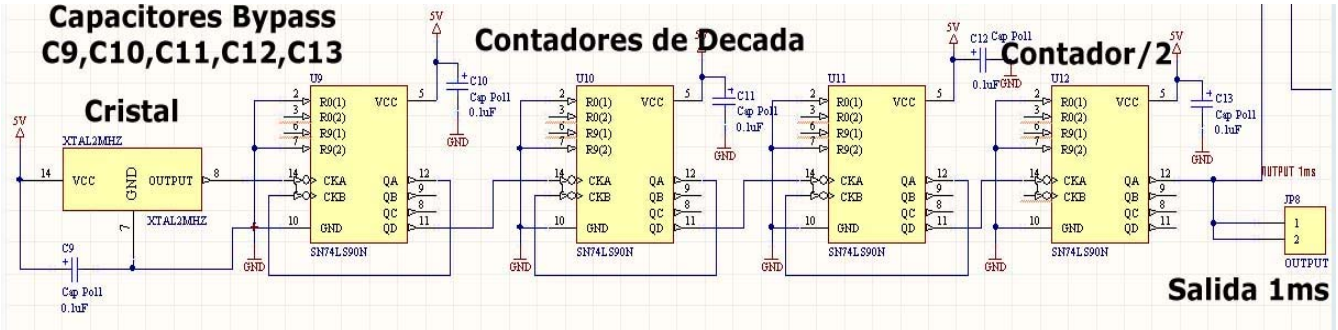


Figura 3.5

A continuación se muestra el diagrama completo de la electrónica de acondicionamiento y conteo, que fue explicado.

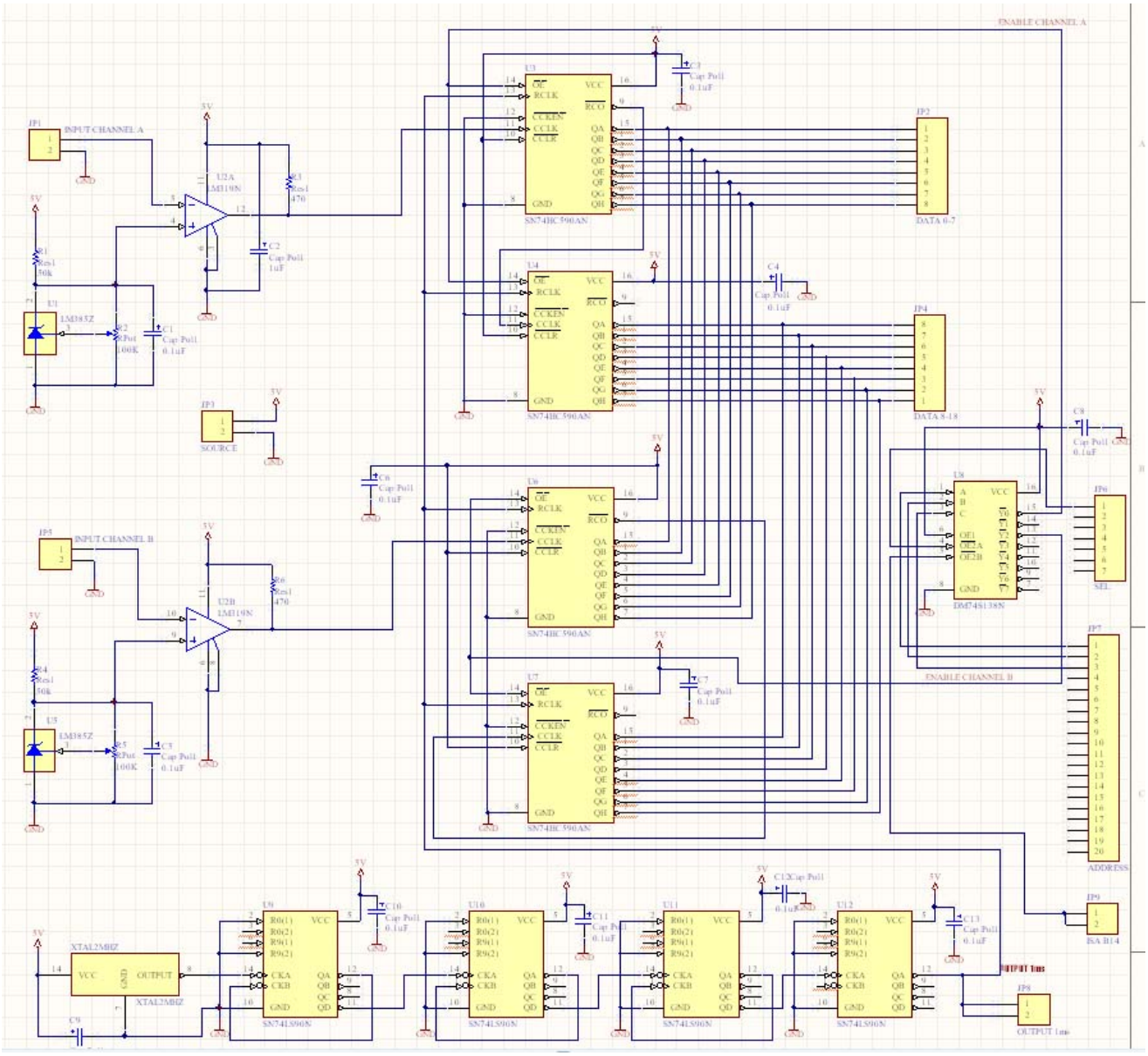


Figura 3.6 Diagrama esquemático completo de la Electrónica de Acondicionamiento y Conteo (*Para ver este diagrama amplificado ver la sección de Anexos).*

Una vez realizado el diagrama esquemático de la electrónica de acondicionamiento diseñamos el circuito impreso usando un asistente para la configuración básica de la placa en cuanto a las unidades, el número de capas, el área del impreso, ubicación de los componentes, entre otras cosas. Después se exporta la lista de componentes con las respectivas conexiones entre ellos, se define el tipo de empaque para cada componente, la distancia entre estos, la ubicación de los mismos y las trayectorias de las pistas (*ruteo*), Los componentes se ordenan con la finalidad de obtener un arreglo óptimo para aprovechar el espacio en la placa obteniendo el menor numero de pistas y de conexiones entre ambos lados de la tarjeta, esto se debe a las limitaciones de la maquina que construye los circuitos impresos con la que cuenta el Instituto de Astronomía, estas limitaciones son listadas a continuación.

- El surco que realiza el espacio entre pistas es muy reducido, se deben de colocar las pistas lo más separado que se pueda, para evitar cortocircuitos.

- Esta tarjeta no contiene *thru-holes*, por lo que el trazado de rutas se realizó a manera de evitar los mismos, y fue elaborada sin el ayudante de trazado de rutas (fue elaborada completamente por nosotros), por lo cual tuvo un mayor grado de dificultad, en las especificaciones y cuidado de no cometer errores.

- Tampoco cuenta con una capa que inhibe la oxidación del metal, si la contuviera facilitaría el soldado de la tarjeta, ya que al no contar con el se debe tener cuidado de que la soldadura no invada pistas cercanas y donas.

- El tiempo de manufactura es demasiado largo ya que es necesaria la revisión de las especificaciones antes mencionadas y la revisión detallada para cerciorarse que no contenga errores así como el cuidado de la maquinaria.

- Se realiza el devastado de cobre alrededor de las donas para soldar, el área es muy pequeña.
  En el Anexo 6 se muestra la distribución de los componentes, con las características antes descritas.

Después de la distribución adecuada de los componentes se realizaron las trayectorias de las pistas (*ruteo)* como se muestra en el Anexo 7, Anexo 8 y Anexo 9.
pistas en la cara inferior

Al tener el diseño de la tarjeta en circuito impreso se envían para su manufactura al laboratorio de electrónica del Instituto de Astronomía donde se lleva a cabo la fabricación de la misma por medio de la maquina de circuitos impresos, se hicieron algunos prototipos hasta obtener la versión final que se muestra en el Anexo 10 y en el Anexo 11.

En el Anexo 12 se muestra una zona difícil de soldar debido a que existen donas y pistas al rededor, y se observa también la diminuta distancia entre pistas.

Una vez que el laboratorio de electrónica del Instituto de Astronomía manufacturó el PCB continuamos poblando la tarjeta con los componentes electrónicos, ya poblado el PCB se acopló a la tarjeta JDR *Microdevices*, para comunicar la electrónica de adquisición por medio del bus ISA con la computadora bajo el sistema operativo GNU/Linux.

3.3 Interrupciones

Debido a la necesidad de que el sistema operativo atienda la tarjeta de adquisición cada milisegundo y garantizar a su vez que no haya pérdida de información en las lecturas requerimos del uso de interrupciones para lograrlo debido a que están concebidas justamente para obligar a un CPU a responder en un momento específico a una petición de proceso. A continuación detallaremos el concepto y manejo de las interrupciones.

Una interrupción se genera cuando se quiere que la CPU deje de ejecutar el proceso en curso y ejecute una función específica de quien produce la interrupción. Cuando se ejecuta esta función específica decimos que la CPU está atendiendo la interrupción. Podemos realizar una clasificación de las interrupciones, atendiendo a la fuente que las produce.

Interrupción *software*: se produce cuando un usuario solicita una llamada del sistema.

Interrupciones *hardware*: son causadas cuando un dispositivo *hardware* requiere la atención de la CPU para que se ejecute su manejador.

Excepciones, son interrupciones causadas por la propia CPU, cuando ocurre algo no deseado, por ejemplo una división por cero.

Las interrupciones *hardware* son producidas por varias fuentes, por ejemplo del teclado, cada vez que se presiona una tecla se genera una interrupción. Otras interrupciones son originadas por el reloj, la impresora, el puerto serie, el floppy, etcétera. Una interrupción de tipo *hardware* es una señal producida por un dispositivo físico del ordenador. Esta señal informa a la CPU que el dispositivo requiere su atención.

La CPU parará el proceso que está ejecutando para atender la interrupción. Cuando la interrupción termina, la CPU reanuda la ejecución en donde fue interrumpida, pudiendo ejecutar el proceso parado originalmente o bien otro proceso.

Existe un *hardware* específico, para que los dispositivos puedan interrumpir lo que está haciendo la CPU. La propia CPU, tiene entradas específicas para ser interrumpida: INT y NMI, cuando se activa esta entrada INT, la CPU para lo que está haciendo y activa la salida para reconocer la interrupción INTA, y comienza a ejecutar el código especial que maneja la interrupción. Algunas CPU´s disponen de un conjunto especial de registros, que solo son utilizados en el modo de ejecución de interrupciones, lo que facilita el trabajo de controlar las interrupciones.

La *mother board* utiliza un controlador para decodificar las interrupciones que no son mas que señales eléctricas producidas por los dispositivos, coloca en el bus de datos información de que dispositivo interrumpió y activa la entrada INT de interrupción a la CPU. Este chip controlador protege a la CPU y la aísla de los dispositivos que interrumpen, además de proporcionar flexibilidad al diseño del sistema. El controlador de interrupciones tiene un registro de estado para habilitar o inhibir las interrupciones en el sistema.

3.4 Sistema de Interrupciones basados en Controladores de Interrupciones Programables (arquitectura i386).

Existe diversas formas de implementar un controlador de interrupciones, los computadores IBM PC o compatibles, utilizan el controlador de interrupciones programable de Intel 82C59A-2 CMOS o sus chips compatibles. Este controlador ha sido utilizado desde los comienzos del IBM PC, y es bien conocido el espacio de direccionamiento de sus registros en la arquitectura ISA. Incluso en chips más modernos se ha mantenido la misma localización, cabe resaltar que este CIP esta embebido en las computadoras personales.

En la Figura 3.7, se muestran dos controladores de 8 entradas, cada uno de ellos tiene una máscara y un registro de estatus de interrupción, un CIP1(controlador de interrupción programable) y un CIP2. Los registros de máscara están en los direccionamientos 0x21 y 0xA1 y los registros del estatus están en 0x20 y 0xA0.

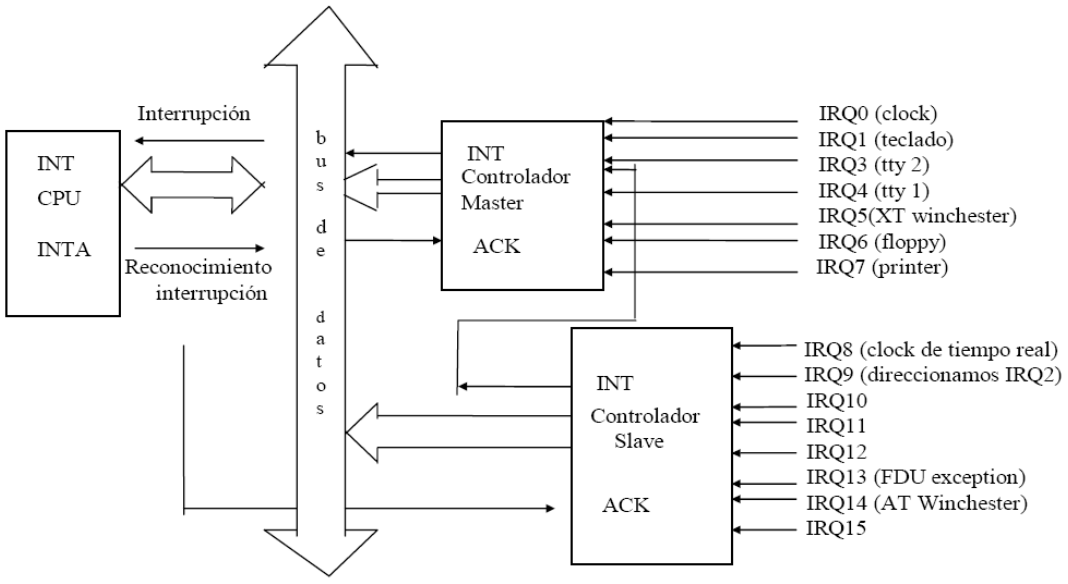


Figura 3.7

Al escribir en un bit determinado del registro de máscara permite una interrupción, escribiendo un cero se invalida esta interrupción. Así pues, escribir un uno en la entrada 3 permite la interrupción 3, escribiendo cero se invalida. Los registros de máscara de interrupción son solamente de escritura, por lo tanto Linux debe guardar una copia local de lo que se ha escrito en los registros de máscara.

Cuando se produce una señal de interrupción, El código de manejo de la interrupción lee dos registros de estatus de interrupción (ISRs). Trata el ISR en *0x20* como los ocho bits inferiores, y el ISR en *0xA0* como los ocho bits superiores. Así pues, una interrupción en el dígito binario 1 del ISR en *0xA0* será tratada como la interrupción 9 del sistema. El segundo bit de CIP1 no es utilizado ya que sirve para encadenar las interrupciones del controlador CIP2, por lo tanto cualquier interrupción del controlador CIP2 se pasa al bit 2 del controlador CIP1.

El controlador de interrupción programable 8259 (CIP en la placa base) maneja todas las interrupciones *hardware*. Estos controladores toman las señales de los dispositivos y los convierten a las interrupciones específicas en el procesador.

Los IRQ o **i**nterrupt request (Pedido de Interrupción), son las notificaciones de las interrupciones enviadas desde los dispositivos *hardware* a la CPU, en respuesta a la IRQ, la CPU salta a una dirección – una **r**utina de **s**ervicio de **i**nterrupción (ISR), comúnmente llamada *Interrupt handler* (Manejador de interrupciones) - Que se encuentra como una función dentro del software manejador de ese dispositivo formando parte del núcleo. Así, una función manejadora de interrupciones es una función del núcleo que ejecuta el servicio de esa interrupción.
Los IRQ se encuentran numerados, y cada dispositivo *hardware* se encuentra asociado a un número IRQ (ver Figura 3.8). En la arquitectura IBM PC y compatibles, por ejemplo, IRQ 0 se encuentra asociado al reloj o temporizador, el cual genera 100 interrupciones por segundo, el disquete la interrupción 6, los discos IDE las interrupciones 14 y 15. Se puede compartir un IRQ entre varios dispositivos.
La Figura 3.8 muestra las interrupciones *hardware* y su correspondiente puerto en el Controlador Programable de Interrupciones (CIP). No se deben confundir los números IRQ con los números de la interrupción. Los CIP se pueden programar para generar diversos números de interrupción para cada IRQ.
Los Controladores también controlan la prioridad de las interrupciones. Por ejemplo, el reloj (en IRQ 0) tiene una prioridad más alta que el teclado (IRQ 1). Si el procesador está atendiendo una interrupción del reloj, el CIP no generará una interrupción para el teclado hasta que ISR del reloj reajusta el CIP. Por otra parte, el reloj puede interrumpir ISR del teclado. El CIP se puede programar para utilizar una variedad de esquemas de la prioridad, pero no se suele hacer esto.

Se debe de tener en cuenta que el IRQ 2 del primer CIP, valida o invalida las entradas del Segundo CIP (8 a 15).

Algunas interrupciones son fijadas por convenio en la configuración del PC, así es que los manejadores de los dispositivos solicitan simplemente la interrupción cuando se inicializan. [4]

00H	-	división por cero o desbordamiento
02H	-	NMI (interrupción no-enmascarable)
04H	-	desbordamiento (EN)
08H	0	Temporizador del sistema
09H	1	Teclado
0AH	2	Interrupción del segundo PIC
0BH	3	COM2
0CH	4	COM1
0DH	5	LPT2
0EH	6	disquete
0FH	7	LPT1
70H	8	Reloj
71H	9	I/o general
72H	10	I/o general
73H	11	I/o general
74H	12	I/o general
75H	13	Coprocesador
76H	14	Disco duro
77H	15	I/o general

Figura 3.8 Descripción de los números de interrupciones

3.5 Interrupciones hardware en Linux.

Una de las principales tareas del sistema de manejo de interrupciones es llevar las diferentes interrupciones a los códigos de manejo de esas interrupciones.

Cuando se activa el contacto 6 del controlador de interrupciones, se debe reconocer cual es la interrupción asociada a ese contacto, por ejemplo el controlador del dispositivo disquete solicita siempre la IRQ 6,por lo tanto el sistema de manejo de interrupciones debe encaminar a la rutina que trata esta interrupción, para ello Linux proporciona un conjunto de estructuras de datos y tablas, y un conjunto de funciones que las inicializan y las manejan.
El sistema de interrupciones es muy dependiente de la arquitectura, Linux en la medida
de lo posible, tratará de que sea independiente de la máquina en la que reside el sistema, para ello el sistema de interrupciones se va a implementar mediante una serie de estructuras de datos y funciones en lenguaje C que facilitarán la portabilidad.

Empleamos la interrupción 7 que es la asignada al puerto paralelo (LPT1), debido a que necesitamos que el sistema responda en un tiempo muy corto.

Capítulo 4 Diseño e implementación del Software de Adquisición (GUI)

En este capítulo se desarrolla todo lo referente al software para el sistema de adquisición, desde la introducción al sistema operativo GNU/Linux, el kernel o núcleo del sistema, los controladores, tipos de controladores según el tipo de dispositivo, la explicación e implementación del código fuente del controlador o módulo y finalmente el desarrollo e implementación de la interfaz gráfica de usuario.

El diseño del software se divide en dos partes:

- El diseño del módulo o controlador.
- El diseño de la interfaz gráfica de usuario.

Las funciones del controlador o modulo, se dividen en:

- Inicializar el hardware reservando un espacio en memoria, asignar los puertos de comunicación de la tarjeta de adquisición, entre otros.
- Realizar operaciones de lectura para adquirir los datos de la tarjeta e insertarlos en el espacio de memoria desde donde puedan ser usados por el usuario.
- Asignar una interrupción para ejecutar la lectura cada que esta se manifieste.
- Una vez que el módulo no se ocupe, liberar todos los recursos anteriormente asignados a este.

Los requerimientos de la interfaz gráfica de usuario se abordan en el apartado correspondiente en este capítulo.

4.1 El Sistema Operativo Unix

Desde su creación en 1969 a manos de Dennis Ritchie y Ken Thompson, el sistema operativo Unix se sigue manteniendo como uno de los más poderosos y elegantes sistemas operativos.

Unix se creo a partir de Multics, un proyecto fallido de un sistema operativo multiusuario en el que los Laboratorios Bell estaban involucrados. En el verano de 1969, los programadores de los Laboratorios Bell diagramaron un sistema de archivos (*file system*) el cual se convirtió en UNIX. En 1973, el sistema

operativo se reescribió en el lenguaje de programación C, un paso sin precedentes en aquel tiempo, pero que pavimentó el camino para la portabilidad en el futuro.

La primera versión de UNIX usada ampliamente fuera de los Laboratorios Bell, fue el Sistema Unix, Sexta Edición, comúnmente llamada V6. Otras compañías migraron UNIX a nuevas maquinas lo que resulto en variaciones del sistema operativo. En 1977, los Laboratorios Bell lanzaron una combinación de todas estas variaciones en un solo sistema operativo, UNIX System III. Se rumora que la versión IV del sistema fue solo de desarrollo y finalmente en 1982, AT&T lanzó el System V.

La simplicidad en el diseño de UNIX, acompañado del factor de que era distribuido con el código fuente, lideró a otras organizaciones a su desarrollo. El más influyente de estos contribuidores fue la Universidad de California en Berkeley. Las variaciones de UNIX de Berkeley son llamadas Distribuciones de Software de Berkeley (*BSD, Berkeley Software Distributions*). Estas versiones agregaron memoria virtual, TCP/IP entre otros avances. Actualmente el desarrollo de BSD continúa con los sistemas Darwin, Dragonfly BSD, FreeBSD, NetBSD, y OpenBSD.

En los 80s y 90s, múltiples compañías dedicadas a la construcción de servidores y estaciones de trabajo introdujeron su propia versión comercial de UNIX. Estos sistemas eran típicamente basados en las versiones de Berkeley o AT&T, con características avanzadas y desarrolladas para aprovechar sus equipos. Entre estos sistemas encontramos HP-UX de Hewlett Packard, AIX de IBM, DYNIX/ptx de Sequent, IRIX de SGI y Solaris de Sun.

El diseño elegante y original del sistema UNIX lo han convertido en un sistema robusto, poderoso y estable. Su fortaleza descansa en las decisiones tomadas por Dennis Ritchie, Ken Thompson y algunos otros desarrolladores.

4.2 El nacimiento del Sistema Operativo GNU/Linux

El Proyecto GNU fue iniciado en 1984 con el propósito de desarrollar un sistema operativo compatible con UNIX. Hacia 1991, cuando la primera versión del núcleo Linux fue liberada, el proyecto GNU había producido varios de los componentes del sistema, incluyendo un intérprete de instrucciones, una biblioteca C y un compilador.

El sistema operativo GNU Linux nació de la unión del proyecto GNU con el núcleo o *kernel* desarrollado por Linus Torvalds en la Universidad de Helsinki.

Linus Torvalds compartía el objetivo de crear un sistema operativo compatible con UNIX pero para computadoras con procesador Intel 80386 el cual para aquel tiempo era un procesador nuevo y avanzado. Linus comenzó escribiendo un simple emulador de Terminal, el cual usaba para conectarse a un sistema UNIX más grande en su escuela. Su emulador de Terminal mejoró y evolucionó en un sistema operativo inmaduro pero con varias características de UNIX. A finales de 1991 Linus publicó su sistema en Internet.

Rápidamente Linus atrajo muchos usuarios y desarrolladores los cuales modificaban y mejoraban su

código. Debido a su Licencia Linux se convirtió en un proyecto de colaboración desarrollado por personas de todo el mundo.

En este momento Linux es un sistema operativo completo el cual corre en plataformas AMD x86-64, ARM, Compaq Alpha, CRIS, DEC VAX, H8/300, Hitachi SuperH, HP PA-RISC, IBM S/390, Intel IA-64, MIPS, Motorola 68000, PowerPC, SPARC, UltraSPARC, y v850. Corre en sistemas tan pequeños como un reloj y tan grandes como clusters de super-computadoras que ocupan cuartos completos. Hoy el interés comercial en Linux es fuerte. Empresas relativamente nuevas como Montanista y Red Hat, y compañías grandes como IBM y Novell, proveen soluciones basadas en Linux para escritorio y servidores.

Linux es un clon de UNIX pero no es UNIX. Esto es, aunque Linux tome prestadas muchas ideas de UNIX y una implementación de la API (del inglés *Application Programming Interface* - Interfaz de Programación de Aplicaciones) de UNIX, no es descendiente del código fuente de UNIX como otros sistemas basados en UNIX.

Una de las características más importantes de Linux es que no es un producto comercial, en cambio es un proyecto colaborativo desarrollado a través de Internet, por lo que el *kernel* es software libre y abierto. Específicamente, el kernel de Linux se encuentra bajo los términos de la licencia GNU licencia pública general (del inglés, *General Public License* o GPL).

Consecuentemente uno es libre de descargar el código fuente y hacer las modificaciones que uno crea convenientes, la única condición es, si se quieren distribuir estas modificaciones se hagan bajo los mismos términos, incluyendo la disponibilidad del código fuente.

Linux es muchas cosas para distintas gentes. Los fundamentos de Linux son, el *kernel*, las librerías de C, el compilador, la cadena de herramientas y las utilerías básicas del sistema, como el proceso de registro y el *shell*.

Los sistemas más modernos basados en Linux tienen incorporado el sistema de ventanas X (X *Window*) con un sistema de escritorio lleno de características, como GNOME o KDE. Miles de aplicaciones libres y comerciales existen hoy en día para Linux.[8]

4.3 El Kernel (Núcleo).

El *kernel* (El *kernel* es referido frecuentemente como núcleo en español) es el componente central de cualquier sistema operativo. Todos los sistemas operativos constan de una parte encargada de gestionar los diferentes procesos y las posibles comunicaciones entre el *hardware* de un ordenador con los programas que están en funcionamiento, entre otras y variadas tareas. Es, por ejemplo, el que facilita el acceso a datos en los distintos soportes posibles (CD-ROM, unidad de disco duro, etc.), o el que arranca el ordenador, o el que reinicia todos los dispositivos que sean necesarios.

La principal propiedad de un *kernel* es que todas estas operaciones de manejo de memoria o de dispositivos, son, desde un punto de vista de usuario, totalmente transparentes, esto es, no es necesario saber como trabajar a bajo nivel con el procesador para realizar las operaciones que sean necesarias, ya que será el *kernel*, a través de una serie de instrucciones ya implementadas el que lo hará por nosotros, a estas instrucciones se les conocen como *drivers*, módulos o controladores de dispositivo.

Los *kernels* de Windows NT o Minix son de tipo micro-*kernel*, caracterizado porque proveen al sistema de un estado mínimo necesario de funcionalidad, cargando el resto de funciones necesarias en procesos autónomos e independientes unos de otros, comunicándose con este micro-*kernel* a través de una interfaz bien definida.

Este tipo de estructura es más fácil de mantener y el desarrollo de nuevos componentes es mucho más simple, dando a su vez una mayor estabilidad al sistema. Por otro lado, debido a la estructura rígida de la interfaz, estos tipos de *kernel* son mucho más complicados de reestructurar, y además, debido a las arquitecturas del hardware actual, el proceso de intercomunicación dentro del micro-*kernel* es mucho más que una simple llamada, por lo que hace que esta estructura sea más lenta que los *kernels* de tipo monolíticos o macro-*kernels.*

Linux ha sido desarrollado gracias a diferentes programadores de todo el mundo. Debido a esto, una estructura de micro-kernel es prácticamente inconcebible, aunque esto no quiere decir que el *kernel* de linux sea una simple lista de instrucciones sin estructura de programación. A pesar de la estructura de macro-*kernel*, se ha intentado igualar su velocidad utilizando código optimizado en velocidad (aunque complicado de entender), y se ha recuperado algunas de las mejores características de la estructura de micro-*kernel*, como puede ser la carga de los diferentes controladores necesarios como módulos independientes, siempre sin olvidar la estructura monolítica original.

En el caso de Linux, la mayor parte del *kernel* está escrito en C, existiendo también instrucciones en ensamblador, aunque estas últimas se usan mayoritariamente en los procesos de arranque y en el control de co-procesador aritmético.

Como dato curioso cabe comentar el significado de la serie de números que acompañan al núcleo, tanto compilado como al directorio que contiene las fuentes de éste, que, a pesar de no ser necesarios, se suelen incluir porque aportan una mayor información. Este conjunto de cifras tienen el formato X.X.XX y su significado no es más que la versión del núcleo a la que corresponde dicho archivo, aunque no es simplemente así. Como se puede suponer, la variación en una unidad del primer grupo de cifras significa un cambio muy importante en el *kernel*, siendo ésta menor conforme el grupo de cifras que varía está más hacia la derecha. El último grupo de cifras tiene, además del significado anterior como indicador de versión, un significado añadido, que nos dice si la cifra es par, esa versión de *kernel* se considera como una versión estable, si, en cambio ésta es impar, se considera que la versión del núcleo es una versión en fase *beta* o de desarrollo.[8]

4.4 Controlador de dispositivo ó módulo.

Uno de los problemas más serios que presentan los sistemas operativos es que deben contar con el código necesario para poder interaccionar con el *hardware* de la computadora. Esto se soluciona en Linux mediante pequeños programas llamados módulos. En el fondo, un módulo es un pequeño programa escrito para permitir al núcleo del sistema operativo interaccionar con el *hardware* y proporcionar los recursos de éste al *software* de aplicación y al usuario.

Ahora bien, el problema consiste en que un módulo debe ser escrito por alguien que conozca claramente cómo funciona el dispositivo de *hardware* y cómo funciona el sistema operativo para el que se está escribiendo. En Linux esto puede ser fácil desde el punto de vista del sistema operativo, ya que su código fuente y las herramientas de desarrollo están a libre disposición del público.

4.5 Drivers o módulos en Linux

Un controlador de dispositivo (llamado normalmente controlador, módulo o *driver*) es un programa informático que permite al sistema operativo interaccionar con un periférico, haciendo una abstracción del *hardware* y proporcionando una interfaz posiblemente estandarizada para usarlo.

El concepto de módulo en Linux, de por sí es bastante simple, ya que se trata de un código objeto compilado adecuadamente que se agrega al *kernel*, permitiendo al usuario interaccionar con el nuevo dispositivo a través de un puntero asociado en el directorio:

/dev.

Es como un manual de instrucciones que le indica cómo debe controlar y comunicarse con un dispositivo en particular. Por tanto, es una pieza esencial, sin la cual no se podría usar el *hardware*.

A diferencia de una aplicación que desempeña una tarea de principio a fin, cuando un módulo o *driver* se registra es para atender futuras peticiones, por lo tanto, la función init_module (el punto de entrada del *driver*), es preparar al módulo para futuras invocaciones de sus funciones. El segundo punto de entrada de un *driver*, cleanup_module, cuando es invocado le indica al *kernel* que tanto el *driver* como sus funciones ya no están disponibles.

El *kernel* ofrece una serie de subrutinas o funciones en el espacio de usuario que permiten al programador de aplicaciones finales interaccionar con el *hardware*. Habitualmente, en sistemas UNIX o Linux, este diálogo se hace a través de las funciones o subrutinas para leer y escribir en ficheros, ya que en estos sistemas los propios dispositivos se ven desde el punto de vista del usuario como ficheros.

Hay dos formas principales de que un módulo del núcleo se comunique con los procesos. Una es a través de los ficheros de dispositivos (como los que están en el directorio /dev) y la otra es usar el sistema de ficheros proc.

El sistema de ficheros /proc es un mecanismo para que el núcleo y los módulos del núcleo envíen información a los procesos, fue diseñado para permitir un fácil acceso a la información de los mismos.

El propósito original de los ficheros de dispositivo es permitir a los procesos comunicarse con los controladores de dispositivos en el núcleo, y a través de ellos con los dispositivos físicos (módems, terminales, etc.).

La forma en la que esto se implementa es, a cada controlador de dispositivo, responsable de algún tipo de *hardware*, se le asigna su propio número mayor. La lista de los controladores y de sus números mayores está disponible en /proc/devices, para el caso del *driver* para la tarjeta electrónica del fotómetro el número mayor seleccionado es el 60, este número es usado para prototipos. A cada dispositivo físico administrado por un controlador de dispositivo se le asigna un número menor, por ejemplo en los discos duros comparten el número mayor, pero su número menor es diferente.

En el directorio /dev se incluye un fichero especial, llamado fichero de dispositivo, para cada uno de estos dispositivos, tanto si está realmente instalado en el sistema como si no, en el caso de la tarjeta electrónica del fotómetro este fichero es /dev/fotometro.

Los sistemas Unix-Linux distinguen a los dispositivos en tres tipos principales, por lo que cada módulo usualmente es una implementación de uno de estos tipos:

Módulo de carácter, módulo de bloque y módulo *network.*

Esta división de módulos en diferentes tipos o clases no es rígida, el programador puede elegir en construir grandes módulos, integrando diferentes tipos de *drivers* en un solo bloque de código.

4.5.1 Dispositivos de carácter ó tipo char.

Un dispositivo de carácter (*char*) es aquel que puede ser accedido como un flujo de *bytes* (como un archivo). Este tipo de driver usualmente implementa al menos las llamadas de sistema ***open***, ***close***, ***read*** y ***write***. La consola de texto (/dev/console) y el puerto serie (/dev/ttySO) son ejemplos de *drivers* de carácter, los cuales son bien representados por la abstracción de flujo de bytes.

Los dispositivos de carácter son accedidos por nodos del sistema de archivos, por ejemplo: /dev/tty1 *y* /dev/lp0. La única diferencia relevante entre un dispositivo de carácter y un archivo regular es que el archivo siempre se puede mover hacia adelante y hacia atrás, y los dispositivos de carácter, son solo canales de datos, los cuales solo se pueden acceder en secuencia.

4.5.2 Dispositivos de Bloque.

Como en los dispositivos de carácter, los dispositivos de bloque pueden ser accedidos desde el sistema de archivos como un nodo en el directorio: /dev. Un dispositivo de bloque es aquel que puede albergar un sistema de archivos, por ejemplo un disco. En la mayoría de los sistemas Unix, un dispositivo de bloque solo puede ser usado como múltiplos de bloque, donde un bloque es usualmente 1Kb de datos o alguna otra potencia de dos.

Linux permite a la aplicación leer y escribir el dispositivo como un dispositivo de carácter, permite la transferencia de cualquier número de bytes en un tiempo. Como resultado, un dispositivo de carácter difiere solo en la manera en la que los datos son tratados internamente por el *kernel*, así como en la interfaz de software *kernel/driver*.

Las diferencias entre un dispositivo de carácter y uno de bloque permanece transparente al usuario.

4.5.3 Dispositivos de Red (Network)

Cualquier transacción de red se realiza a través de una interfaz, esto es, un dispositivo que es capaz de intercambiar información con un servidor (*host*). Usualmente, la interfaz es un dispositivo de *hardware*, pero también puede ser un dispositivo únicamente de software como una interfaz *loopback* (autorespuesta).

Una interfaz de red está encargada de enviar y recibir paquetes, de la red de trabajo al subsistema del *kernel*, sin conocer como las transacciones de cada paquete se llevan acabo.

Aunque las conexiones del telnet y del ftp son orientadas a un flujo de *bytes*, transmiten con el mismo dispositivo; el dispositivo no considera los flujos individuales, sino solamente los paquetes de los datos. Al no ser como tal un dispositivo orientado al flujo de bytes, una interfaz de la red no se mapea fácilmente a un nodo en el sistema de archivos, como /dev/tty1. La manera de Unix para proporcionar el acceso a la interfaz es asignándoles un nombre único (como eth0), pero ese nombre no tiene una entrada correspondiente en el sistema de archivos. La comunicación entre el núcleo y un *driver* de dispositivo de la red es totalmente diferente del que utiliza con los dispositivos tipo carácter y de bloque. En lugar de las funciones read y write, se utilizan funciones del *kernel* relacionadas con la transmisión de paquetes.

4.5.4 Espacio de usuario ("*user space*") y espacio *kernel* ("*kernel space*")

El sistema operativo Linux y en especial su *kernel* se ocupan de gestionar los recursos de hardware de la máquina de una forma eficiente y sencilla, ofreciendo al usuario una interfaz de programación simple y uniforme. El *kernel*, y en especial sus drivers, constituyen así un puente o interfaz entre el programador de aplicaciones para el usuario final y el hardware. Toda subrutina que forma parte del

kernel tales como los módulos o *drivers* se consideran que están en el espacio del *kernel* ("*kernel space*").

Entre sus componentes típicos se encuentran:

- Los *interrupt handlers* (manejadores de interrupciones) para atender las peticiones de interrupción.
- Un *scheduler* (administrador de recursos y de tiempo del procesador) para compartir o administrar tiempo de procesador entre los múltiples procesos que estén en ejecución.
- Un *memory management* (manejador de memoria) para controlar espacios de direcciones de proceso.
- *Sistem services* (servicios de sistema) como comunicación inter-proceso y *networking*

En los sistemas modernos con unidades de modo protegido, el *kernel* reside en un estado elevado del sistema en comparación con las aplicaciones normales de usuario. Este estado incluye un espacio de memoria protegido y acceso total al hardware. A este estado de sistema y al espacio de memoria es colectivamente referido como espacio *kernel*.

Los programas que utiliza el usuario final, tales como las "*shell*" u otras aplicaciones con ventanas como por ejemplo "*kpresenter*", se ejecutan en el espacio de usuario ("*user space*"). Estos programas ven un subconjunto de recursos de máquina disponibles y no tienen la capacidad para realizar ciertas funciones como acceder directamente al hardware, como es lógico estas aplicaciones necesitan interaccionar con el hardware del sistema, pero no lo hacen directamente, sino a través de las funciones de llamadas al sistema (ver Figura 4.1) que soporta el *kernel*, por ejemplo, la librería de C que depende de la interfaz de llamadas al sistema instruyen al *kernel* a realizar tareas en su favor.

Cuando una aplicación ejecuta una llamada al sistema, se dice que el *kernel* esta ejecutando a favor de la aplicación, del mismo modo, se dice que la aplicación esta ejecutando una llamada al espacio *kernel* y el *kernel* esta corriendo el contexto del proceso. Esta relación donde las aplicaciones llaman al *kernel* vía la interfaz de llamadas del sistema es la manera fundamental en la que las aplicaciones realizan su trabajo.[13]

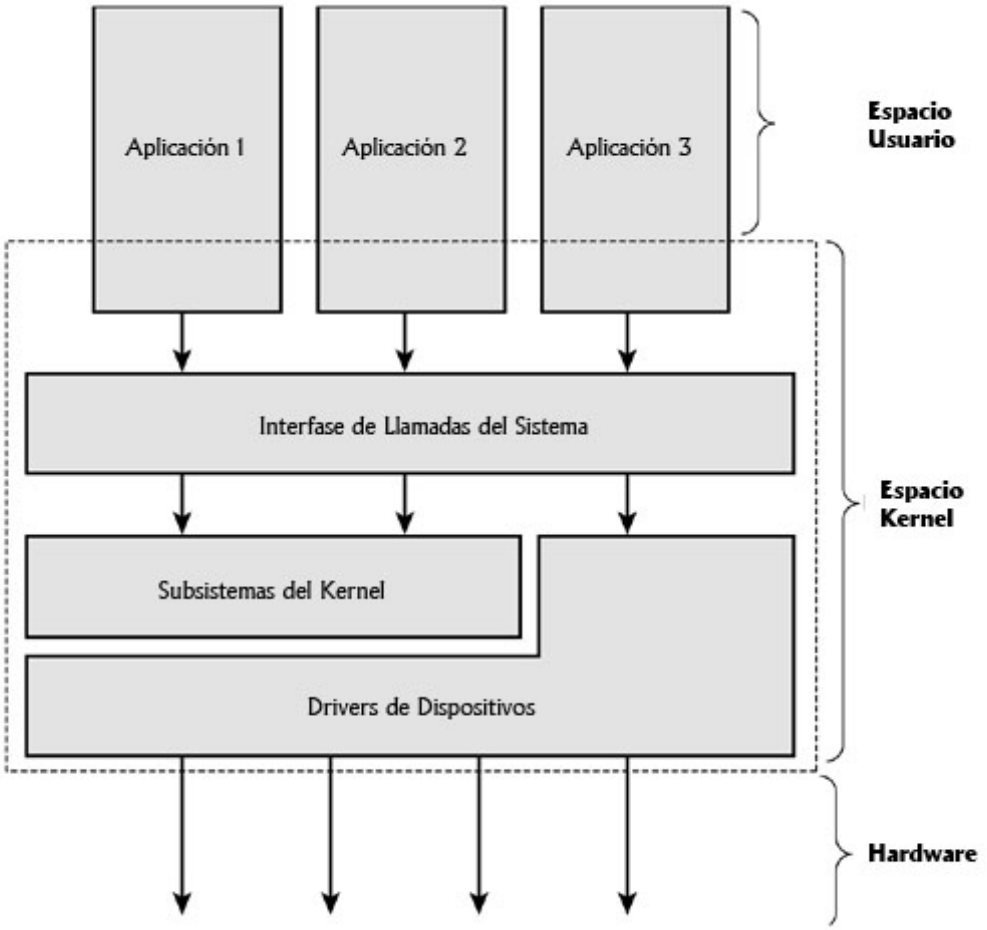


Figura 4.1 Relación entre las aplicaciones, el kernel y el hardware.

4.5.5 Operaciones de Archivo

Un dispositivo abierto es identificado internamente por una estructura de archivo, y el *kernel* utiliza la estructura de operaciones de archivo para acceder a las funciones del *driver*. La estructura esta definida en <Linux/fs.h>, es un arreglo de punteros a funciones.
Cada archivo es asociado a su propio conjunto de funciones (al incluir un campo llamado f_op el cual apunta a la estructura file_operations). La mayoría de las veces, las operaciones se encargan de implementar llamadas del sistema las cuales reciben el nombre de open, read, etc.

Se puede considerar al archivo como un "objeto" y a las funciones operando en él, como los "métodos", usando la terminología de la programación orientada a objetos para denotar acciones declaradas por un objeto que actúan en el mismo.
Convencionalmente, una estructura file_operations o un puntero a la estructura recibe el nombre de fops (o alguna variación). Cada campo en la estructura debe apuntar a la función en el driver que implementa una operación específica, o se debe dejar en NULL (nulo) para operaciones no soportadas.

4.5.6 La estructura de archivo (file).

La estructura de archivo definida en <Linux/fs.h>, es la segunda estructura más importante utilizada en los módulos de dispositivo. Nótese que una estructura archivo (o en inglés *file)* no tiene nada que ver con archivos de los programas en el espacio usuario. Un archivo esta definido en la librería de C y nunca aparece en el código del *kernel*. Y por otro lado, una estructura de archivo, es una estructura kernel que nunca aparece en los programas de usuario.

La estructura de archivo representa un archivo abierto (no es específico de los *drivers* de dispositivo; todo archivo abierto en el sistema tiene una estructura de archivo asociada en el espacio *kernel*). Es creada en el *kernel* al ejecutar la función, open y se pasa a cualquier función que opera al archivo, hasta que se ejecuta la función close. Después de que todas las instancias al archivo se cierran, el *kernel* libera la estructura. Un archivo abierto es diferente de un archivo de disco, representado por la estructura inode.

En los archivos fuente del kernel, un puntero a la estructura archivo es llamado usualmente file o filp (apuntador de archivo "file pointer"). Entonces con file nos referimos a la estructura y con filp a el apuntador a la estructura.

4.6 Código Fuente del driver Fotómetro.

4.6.1 Declaración de las librerías de encabezado para la creación del módulo Fotómetro.

En el encabezado del código fuente se encuentran las declaraciones de librería para crear un *driver*. Estos archivos se encuentran por lo regular en la siguiente ruta:

/usr/src/linux
/usr/src/asm

Destacan:

- fs.h en el cual define la estructura de archivo para el *driver*.
- slab.h define reservar memoria en el espacio *kernel* por numero de *bytes* o por páginas.
- module.h define la creación del módulo o *driver*.
- kernel.h define funciones como *printk* para enviar información hacia el espacio usuario.
- io.h e ioport.h define el acceso de entrada y salida a puertos de *hardware*.
- uaccess.h define las funciones para el envío de información de el espacio *kernel* al espacio usuario y viceversa.
- interrupt.h define funciones para el control de interrupciones de *hardware* o *software*.

```
#define MODULE
#define __KERNEL__

#include <linux/config.h>          /*Estas librerias estan explicadas arriba y todas estan incluidas en el código fuente de LINUX*/
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/slab.h>
#include <linux/fs.h>
#include <linux/errno.h>          /*manejo de errores */
#include <linux/types.h>          /* tipos de datos */
#include <linux/ioport.h>
#include <asm/system.h>
#include <asm/uaccess.h>
#include <asm/io.h>
#include <linux/interrupt.h>
#include <linux/tqueue.h>          /*Interrupciones*/
#include <linux/sched.h>          /*Asignación de tiempo de procesador */
```

Con esta parte del código se definen aspectos como la licencia del *driver*, el dispositivo asociado a él, el nombre del dispositivo, una breve descripción y los autores. Esta información se despliega al momento de insertar el módulo en el *kernel* o en los archivos de información correspondientes ubicados en el directorio:

/proc

```
/* definiciones del dispositivo, como nombre, autores, dispositivo asociado, licencia y una breve descripción */
#define DEVICE_NAME "fotometro"
#define DRIVER_AUTHOR "Helios Alvarez y Jorge Morales"
#define DRIVER_DESC "Driver para Fotometro de doble Canal"
MODULE_LICENSE("GPL");
MODULE_AUTHOR(DRIVER_AUTHOR);
MODULE_DESCRIPTION(DRIVER_DESC);
MODULE_SUPPORTED_DEVICE("fotometro");
```

4.6.2 Declaración de los métodos básicos del fotómetro.

El controlador del dispositivo fotómetro se compone de cuatro métodos, las cuales son llamados cuando alguien intenta interaccionar con el archivo de dispositivo. La forma en que el núcleo sabe cómo llamarlas es a través de la estructura file_operations ó fops. Esta estructura incluye punteros a estos cuatro métodos.

```
/* Son 4 métodos principales, el open abre el dispositivo para trabajar, y adquiere un espacio de memoria y otros recursos, el release
libera al dispositivo y a los recursos asociados, el read lee la información del dispositivo y el write escribe información al dispositivo */

struct file_operations fotometro_fops = {
  read: fotometro_read,
  write: fotometro_write,
  open: fotometro_open,
  release: fotometro_release
};
```

El método fotómetro_open es de tipo entero y como parámetros tiene un apuntador a una estructura de tipo inode y otro apuntador a una estructura tipo archivo.

El método fotómetro_release es de tipo entero y como parámetros tiene un apuntador a una estructura de tipo inode y otro apuntador a una estructura tipo archivo.

El método fotómetro_read es de tipo ssize_t y como parámetros un apuntador a una estructura tipo archivo, un buffer de tipo char, el numero de *bytes* a enviar de tipo size_t y un apuntador de tipo loff_t para el corrimiento.

El método fotómetro_write es de tipo ssize_t y como parámetros un apuntador a una estructura tipo archivo, un *buffer* de tipo const char, el numero de bytes a enviar de tipo size_t y un apuntador de tipo loff_t para el corrimiento.

El método cleanup_module que es de tipo void y no requiere parámetros.

```
/*Declaración de que tipo es cada método, entero, ssize o void */
int fotometro_open (struct inode *inode, struct file *filp);
int fotometro_release (struct inode *inode, struct file *filp);
ssize_t fotometro_read (struct file *filp, char *buf,   size_t count, loff_t *f_pos);
ssize_t fotometro_write (struct file *filp, const char *buf,   size_t count, loff_t *f_pos);
void cleanup_module (void);
```

4.6.3 Init_module, y cleanup_module.

El método init_module llama a module_register_chrdev para añadir el controlador de dispositivo a la tabla de controladores de dispositivos de carácter del núcleo. También devuelve el número mayor que usará el controlador. El método cleanup_module libera el dispositivo.

Registrar y liberar el módulo es la acción principal de estos dos métodos.
En el núcleo los métodos no se activan por propia iniciativa, sino que son llamadas por procesos a través de las llamadas al sistema, por los dispositivos de hardware a través de las interrupciones y/o por otras partes del *kernel*. Como resultado, cuando se añade código al núcleo, se supone que es para registrarlo como parte de un manejador o para un cierto tipo de evento y cuando se quita, todas las funciones de este y los recursos ocupados por el mismo se liberan.

En el método init_mudule la función register_chardev se utiliza para añadir el *driver* al espacio *kernel* y requiere como parámetros:

- El número mayor asignado al dispositivo, el cual se usa como identificador único del hardware y el *driver*. La lista de los controladores y de sus números mayores está disponible en el directorio /proc/devices, en este caso el número mayor seleccionado es el 60 que es usado para prototipos.
- El nombre del dispositivo.
- La dirección de la estructura file_operations del dispositivo, en este caso fotometro_fops definida anteriormente.

En caso de éxito register_chardev regresa como valor cero o un número positivo, en caso de error el valor de retorno es negativo.

La función check_region declarada en ioport.h es llamada para ver si un rango de puertos esta disponible para disponer de ellos. Si la operación no tiene éxito regresa como respuesta un valor negativo como –EBUSY o –EINVAL (-error dispositivo ocupado ó –error operación no válida).

En caso de que los puertos estén disponibles es usada la función request_region declarada en ioport.h para asignar los puertos al módulo fotómetro.

La función request_irq declarada en sched.h es usada para asignar la interrupción al controlador con la cual el hardware se comunica cuando requiere atención. Si la solicitud de la interrupción procede exitosamente regresa como valor cero, en caso contrario el valor es negativo. Los parámetros que utiliza son:

- El número entero sin signo de la interrupción en solicitud.
- El puntero de tipo void a la función encargada de manejar la interrupción.

- Banderas: SA_INTERRUPT cuando la interrupción es de tipo rápido ya que ejecuta la interrupción desactivando temporalmente las demás. SA_IRQ cuando la interrupción va a ser compartida.
- La cadena con el nombre del dispositivo, se usa en /proc/interrupts para mostrar el propietario de la interrupción.
- Puntero de tipo void utilizado para compartir líneas de interrupción. Cuando no es el caso debe ser asignado un valor nulo.

La función get_free_pages declarada en slab.h es usada para asignar recursos de memoria al módulo. La función regresa un puntero al primer *byte* del área de memoria asignada. Los parámetros que utiliza son:

- Banderas: GFP_KERNEL es la asignación normal de memoria. GFP_ATOMIC se usa para asignar memoria para manejadores de interrupción y código fuera del contexto de un proceso, este tipo de memoria es muy limitada.
- Orden: es el número de páginas en logaritmo base dos a solicitar ($\log_2 N$), por ejemplo, el orden es 0 si se solicita una pagina y 3 si se solicitan 8 paginas. Cada página tiene 4kb de memoria. Si el orden es muy grande (no existan suficientes áreas continuas del tamaño solicitado), la asignación de páginas fallará. El orden máximo es 9 (que corresponde a 512 páginas o 2MB).

```
int init_module(void) {
  int result;

  /* Registrando dispositivo */
  result = register_chrdev(fotometro_mayor, DEVICE_NAME, &fotometro_fops);
  if (result < 0)
            {
     printk("<1>fotometro: no puedo obtener numero mayor %d\n", fotometro_mayor);
     return result;
    }

   /* Registrando puertos Canal A 0x300-0x301 Canal B 0x302-0x303*/
   port = check_region(0x300, 4);
   if (port)
             {
     printk("<1>fotometro: no puedo reservar 0x302\n");
     result = port;
     goto fallo;
     }
   request_region(0x300, 4, DEVICE_NAME);

   /* Registrando la interrupción */
   result = request_irq(irq, fotometro_interrupt,SA_INTERRUPT,"fotometro",NULL);
            if (result)
                        {
       printk(KERN_INFO "fotometro: no puede obtener el irq asignado\n");
      }

  /**
  * Reservando memoria
  * 1 pagina = 4kb = tamaño del buffer
  * 1 canal = 16 bits
  * 128 lecturas a almacenar máximo  */
```

```
  fotometro_buffer = __get_free_pages(GFP_ATOMIC,0);
  if (!fotometro_buffer) return -ENOMEM;

  /*Inicializando puenteros del buffer */
  fotometro_head = fotometro_tail = fotometro_buffer;
  printk("<1>Insertando modulo\n");
  return 0;

  fallo:
    cleanup_module();
    return result;
}
```

Para remover el módulo correctamente, todos los recursos solicitados al momento de registrarlo con la función init_module se debe de liberar. Para eso se usa la función cleanup_module.

En el método cleanup_module, la primera función es unregister_module. Esta rutina es la encargada de extraer la información del dispositivo del espacio *kernel*. Tiene como parámetros:

- El número mayor asociado al módulo y al dispositivo.
- El nombre del dispositivo.

En orden para que la subrutina tenga éxito, el módulo no debe estar en uso.

La función release_region declarada en ioport.h es llamada para liberar un rango de puertos. Sus parámetros son:

- La dirección del puerto.
- El rango, que se utilizó con anterioridad en la subrutina request_region.

La función free_page declarada en slab.h es usada para liberar el recurso de memoria del módulo. Su parámetro es un puntero al primer *byte* de la memoria asignada.

Finalmente para liberar la interrupción se usa la subrutina free_irq. Sus parámetros son:

- El número de la línea de interrupción a liberar.
- Puntero de tipo void utilizado para compartir líneas de interrupción. Si esta opción no fue utilizada se le asigna el valor nulo

```
void cleanup_module(void) {

  /* Liberamos el dispositivo con el numero mayor */
  unregister_chrdev(fotometro_mayor, DEVICE_NAME);

  /* Liberando puerto */
  if (!port)  {
        release_region(0x300,4);
   }

  if (fotometro_buffer) {
      free_page(fotometro_buffer);
  }

  free_irq(irq, NULL);
  printk("<1>Quitando modulo\n");

}
```

Un punto importante a recordar es que no se puede permitir que el módulo del núcleo sea borrado cuando *root* quiera. El motivo es que si el fichero del dispositivo es abierto por un proceso y entonces quitamos el módulo del núcleo, el uso del fichero causaría una llamada a la posición de memoria donde el método read y/o write usado debería estar. Si tenemos suerte, ningún otro código fue cargado allí, y obtendremos un feo mensaje. Si no tenemos suerte, otro módulo del núcleo fue cargado en la misma posición, lo que significará un salto en mitad de otra función del núcleo. El resultado sería imposible de predecir, pero no sería positivo.

Normalmente, cuando no se permite algo, se devuelve un código de error (un número negativo) desde el método que se supone que lo tendría que hacer. Con cleanup_module esto es imposible porque es una función void por lo que no regresa algún valor. Una vez que se llama a cleanup_module, el módulo está muerto. En todo caso, hay un contador que cuenta si otros módulos del núcleo o procesos están usando el módulo, llamado contador de referencia (que es el último número de la línea en /proc/modules). Si este número es distinto de cero, rmmod que es la función para desinstalar el módulo fallará. La cuenta de referencia del módulo está disponible en la variable mod_use_count_. Como hay macros definidos para manejar esta variable (MOD_INC_USE_COUNT y MOD_DEC_USE_COUNT), es preferible controlar esta variable con ellos, que utilizar mod_use_count_ directamente, por lo tanto será más seguro si la implementación cambia en el futuro.

El método encargado de abrir el dispositivo es fotometro_open y su función es incrementar la cuenta de uso para referencia al momento eliminar el módulo del núcleo.
Para ello usa el macro MOD_INC_USE_COUNT el cual incrementa en una unidad la cuenta según el número de veces que es convocada.

```
int fotometro_open(struct inode *inode, struct file *filp) {
  /* Aumentamos la cuenta de uso */
  MOD_INC_USE_COUNT;
  /* Exito */
  return 0;

}
```

El método encargado de cerrar el dispositivo es fotometro_release y su función es decrementar la cuenta de uso para referencia al momento de eliminar el módulo del núcleo.
Para ello usa el macro MOD_DEC_USE_COUNT el cual decrementa en una unidad la cuenta según el número de veces que es convocada.

```
int fotometro_release(struct inode *inode, struct file *filp) {
  /* Decrementamos la cuenta de uso */
  MOD_DEC_USE_COUNT;
  /* Exito */
  return 0;
}
```

4.6.4 Lectura y escritura.

Los métodos read y write desempeñan una función parecida, que es, copiar información desde y hacia el código de aplicación. Como aparece en la declaración de las funciones tienen una estructura de este tipo.

ssize_t read(estructura tipo archivo *filp, char *buff, size_t count, loff_t *f_pos);
ssize_t write(estructura tipo archivo *filp, const char *buff, size_t count, loff_t *f_pos);

Para los dos métodos, filp es un puntero y count es el tamaño de información a solicitar ó a transferir. El argumento buff apunta al *buffer* de usuario que contiene la información que se va a escribir ó el *buffer* vacío donde los datos nuevos van a ser depositados. Finalmente f_pos es un apuntador de tipo corrimiento largo (long offset) este objeto indica la posición del archivo que el usuario esta accediendo.

El valor de retorno de los dos métodos es de tamaño y signo (ssize_t, en inglés, signed size type). Esto quiere decir que el valor es un número negativo es por que ocurrió un error. Un valor igual o mayor a cero indica, al programa que realizo la llamada, cuantos *bytes* de información se transmitieron de manera exitosa. Si algo de la información es transferida correctamente y un error ocurre, el valor retornado debe ser la cuenta de *bytes* transferidos correctamente y el error no se reporta sino hasta la siguiente vez que la función es llamada.

En lo que a transferencia de datos concierne, la principal función de estos dos métodos es la necesidad de transferir información entre el espacio de direcciones del *kernel* y el espacio de direcciones del usuario.

La transferencia de información en Linux es desempeñada por funciones especiales definidas en <asm/uaccess.h>.

```
unsigned long copy_to_user(void *to, const void *from, unsigned long count);
unsigned long copy_from_user(void *to, const void *from, unsigned long count);
```

El papel de estas dos funciones no esta limitado a copiar información desde y hacia el espacio de usuario, también revisan si los apuntadores del espacio de usuario son válidos. Si el apuntador no es válido, no se realiza ninguna transferencia; si por otra parte si una dirección no válida es encontrada, solo una parte de la información es transferida. En ambos casos el valor retornado es la cantidad de información esperando a ser transferida.

Tanto como a los métodos actuales de dispositivo concierne, la tarea de el método read es copiar información de el dispositivo hacia el espacio de usuario utilizando la función copy_to_user, mientras que con write el método copia información desde el espacio de usuario hacia el dispositivo usando la función copy_to_user.

Cualquiera que sea la cantidad de información transferida con estos métodos, se debe actualizar la posición del archivo en el apuntador *f_pos para representar correctamente la posición actual del archivo después de completar una exitosa llamada de sistema.

En el método fotómetro_read se prepara el envío de información del buffer fotometro_buffer hacia el espacio usuario, primero se obtiene la cuenta de bytes a enviar al restar fotometro_head con fotómetro_tail.

La información se envía con la función copy_to_user definida en uaccess.h, requiere como parámetros el *buffer* destino, un puntero del *buffer* origen y el número de *bytes* a enviar.

Para poder comparar el número de *bytes* enviados con los recibidos en el espacio usuario usamos la función printk enviando la cuenta de *bytes* a enviar.

La nueva posición del *buffer* se actualiza con la función fotómetro_incr_bp, comentada en el apartado de interrupciones.

```
ssize_t fotometro_read(struct file *filp, char *buf, size_t count, loff_t *f_pos)
{
    int count0;
    /* Cuenta del numero de bits a enviar a espacio uisuario */
    count0 = fotometro_head - fotometro_tail;
```

```
    if (count0 < 0) // si el head esta atrás de tail
        count0 = fotometro_buffer + PAGE_SIZE - fotometro_tail;
    if (count0 < count) count = count0;
    /* Copiando información a espacio usuario */
    if (copy_to_user(buf, (char *)fotometro_tail, count))
            return -EFAULT;
    printk("Cuenta:%d \n", count);
    /* Actualizando buffer */
    fotometro_incr_bp (&fotometro_tail, count);
    return count;

}
```

El método de escritura para el dispositivo fotómetro esta deshabilitado, ya que la función de la tarjeta de adquisición es de solo lectura.

```
ssize_t fotometro_write( struct file *filp, const char *buf, size_t count, loff_t *f_pos) {


printk("<1>Escritura no soportada para el driver fotometro\n");
return -EINVAL;

}
```

4.6.5 Manejo de interrupciones para el Módulo.

Hay dos tipos de interacción entre el CPU y el resto del hardware de la computadora. El primer tipo es cuando el CPU da órdenes al hardware, el otro es cuando el hardware necesita decirle algo al CPU. La segunda, llamada interrupción, es mucho más difícil de implementar porque hay que tratar con ella cuando le conviene al hardware, no al CPU. Los dispositivos hardware típicamente tienen una pequeña cantidad de RAM, y si su información no es leída cuando está disponible, esta se pierde.

Bajo Linux, las interrupciones hardware se llaman IRQs (abreviatura de **I**nterrupt **R**e**q**uests). Hay dos tipos de IRQs, cortas y largas. Una IRQ corta es la que se espera que dure un periodo de tiempo muy corto, durante el cual el resto de la máquina estará bloqueada y ninguna otra interrupción será manejada. Una IRQ larga es una que puede durar más tiempo y durante la cual otras interrupciones pueden ocurrir (pero no interrupciones que vengan del mismo dispositivo). Si es posible, siempre es mejor declarar un manejador de interrupciones como largo.

Cuando el CPU recibe una interrupción, detiene lo que quiera que esté haciendo (a menos que se encuentre procesando una interrupción más prioritaria, en cuyo caso tratará con esta interrupción sólo cuando la más prioritaria se haya acabado), salva parámetros en la pila y llama al manejador de interrupciones. Esto significa que ciertas cosas no se permiten dentro del propio manejador de

interrupciones, porque el sistema se encuentra en un estado desconocido. La solución a este problema es que el manejador de interrupciones haga lo que necesite hacer inmediatamente, normalmente leer algo desde el *hardware* o enviar algo al *hardware*, y después planificar el manejo de la nueva información en un tiempo posterior (esto se llama `bottom half') y retorna. El núcleo garantiza que llamará al bottom half tan pronto como sea posible; y cuando lo haga, todo lo que está permitido en los módulos del núcleo seguirá activo.

La forma de implementar esto es llamar a request_irq() para que este a su vez, llame al manejador de interrupciones cuando se reciba la interrupción (hay 15 de ellas, más una que se utiliza para disponer en cascada los controladores de interrupción, en las plataformas Intel).

Esta función recibe el número de IRQ, el nombre de la función, banderas, un nombre para /proc/interrupts y un parámetro para pasar al manejador de interrupciones. Las banderas pueden incluir SA_SHIRQ para indicar que estás permitiendo compartir la IRQ con otro manejador de interrupciones (normalmente porque un número de dispositivos hardware están en la misma IRQ) y SA_INTERRUPT para indicar que esta es una interrupción rápida. Esta función sólo tendrá éxito si no hay ya un manejador para esta IRQ, o si se esta compartiendo.

En el módulo para el fotómetro la interrupción se maneja a través de las siguientes funciones.

Declaración del parámetro queue para el manejo de las interrupciones.

```
DECLARE_WAIT_QUEUE_HEAD(fotometro_queue);
```

Declaración de las variables para el control del buffer circular.

```
unsigned long fotometro_buffer = 0;
unsigned long volatile fotometro_head;
volatile unsigned long fotometro_tail;
```

Declaración de variables para el control de interrupciones.

```
unsigned long fotometro_base = 0;
static int irq = 7;
```

En esta función a la llegada de la interrupción es leída la información de los puertos 0x300 y 0x302, los cuales corresponden a cada canal en el fotómetro, esta información es convertida a una cadena de caracteres e insertada en el *buffer*, se actualizan los apuntadores para su envío hacia el espacio usuario en el método read.

```
void fotometro_interrupt(int irq, void *dev_id, struct pt_regs *regs)
{

   int written;
   static unsigned short int canA, canB;

   canA = inw(0x300);
   canB = inw(0x302);
   printk("1:%5d \n", canA);
   printk("2:%5d \n", canB);
   written = sprintf((char *)fotometro_head,"1:%5d\n2:%5d\n",canA,canB);
   fotometro_incr_bp(&fotometro_head, written);
   wake_up_interruptible(&fotometro_queue); /* awake any reading process */
}
```

La función fotometro_incr_bp es importante en el manejo del *buffer* ya que actualiza los apuntadores head y tail. Cuando opera sobre head es al momento de ingresar nueva información revisando no salir del área de memoria asignada, y cuando opera sobre tail es para calcular la información a enviar y actualizando el apuntador de acuerdo a la cantidad enviada a el espacio usuario.[13]

```
static inline void fotometro_incr_bp(volatile unsigned long *index, int delta)
{
    unsigned long new = *index + delta;
    barrier ();  /* No optimizar estos al mismo tiempo */
    *index = (new >= (fotometro_buffer + PAGE_SIZE)) ? fotometro_buffer : new;
}
```

4.7 Compilación e instalación del Módulo

Antes de compilar el módulo es necesario crear en el directorio de dispositivos un archivo en el cual, los demás procesos o programas tengan acceso al hardware haciendo peticiones al módulo. Ya que desde el punto de vista de Linux todos los dispositivos de hardware se representan como archivos, para esto hay que crear en el directorio /dev la entrada correspondiente asociando a el módulo con el numeró mayor y si es el caso también con el número menor.

El comando para crear un nodo de dispositivo en el sistema de archivos es mknod; para poder hacer uso de mknod es necesario tener permisos de superusuario o root. El comando toma tres argumentos además del nombre para poder crear el archivo. En el caso del fotómetro el comando es:

```
$ mknod /dev/fotometro c 60 0
```

El comando crea un dispositivo tipo char o de carácter, con número mayor 60 y número mayor 0.

Este archivo se conserva a menos que se borre con la siguiente instrucción:

```
$ rm /dev/fotometro
```

Para compilar un módulo en Linux hay que posicionarse en el directorio donde esta el archivo fuente fotómetro.c, después se llama al compilador gcc con algunas banderas para la optimización del código y como parámetro adicional hay que indicar al compilador la dirección de los archivos fuente o include del *kernel* con la opción –I, como se muestra a continuación.

Si los archivos include del *kernel* de Linux están en /usr/src/linux/include el comando es:

```
$ gcc -I/usr/src/linux/include -O -Wall -c fotometro.c
```

En el caso de la maquina donde se compiló el módulo los archivos fuente del *kernel* de Linux se encuentran en el directorio /usr/src/linux-2.4.29/include, por lo que el comando es:

```
$ gcc -I/usr/src/linux-2.4.29/include -O -Wall -c  fotometro.c
```

Después de compilar el archivo fuente del fotómetro, hay que cargarlo a el espació *kernel*, para eso hay que cambiar de usuario a superusuario o root, con el comando:

```
$su
```

Una vez ingresada la contraseña en la línea de comando hay que posicionarse en el directorio donde compilamos el módulo y buscar el archivo que se generó después de compilar el archivo fuente, que en nuestro caso es fotometro.o, e ingresarlo a el espacio *kernel* con el siguiente comando.

```
$insmod fotometro.o
```

Para ver si el módulo quedo registrado podemos usar el comando:

```
$lsmod
```

Con este comando obtenemos una lista de todos los módulos instalados en la computadora.

Finalmente podemos eliminar el módulo del *kernel* con el comando:

```
$ rmmod fotometro
```

Podemos comprobar que el módulo ya no está instalado de nuevo con el comando:

```
$ lsmod
```

4.8 Diseño de la interfaz de Usuario

Los requerimientos iniciales para la interfaz de usuario son:

- Intuitivo (Fácil de usar).
- Aprovechar las capacidades del sistema gráfico de Linux.
- Permitir la toma de decisiones durante la ejecución del programa, por ejemplo, detener la observación, comentar las observaciones, etc.
- Guardar las lecturas obtenidas durante la observación.
- Graficar en tiempo real las lecturas por canal del fotómetro.

Para cumplir con estos requerimientos de la interfaz de usuario, se utilizó el entorno de programación Borland Kylix 3 open edition, por la facilidad de crear programas en un entorno visual y sólido.

4.8.1 Diagrama de Flujo para el diseño de la Interfaz de Usuario.

En el inicio del diagrama se solicita el nombre del archivo en el que se van a almacenar los datos de la observación. Si no existe se le asigna nombre al archivo, el programa no continúa.
Una vez asignado un nombre para el archivo, el programa abre el *buffer* del fotómetro como archivo.

Este *buffer* es el que se creó con el módulo y se encuentra en la siguiente ruta:
/dev/fotometro

En el siguiente bloque, hace una lectura al *buffer* como una cadena de datos, en esta cadena aparecen los dos canales del fotómetro.

Al analizar la cadena se separa la información proveniente de los dos canales, para diferenciar entre el canal A y el canal B del fotómetro, al inicio de la cadena aparece un uno o un dos según corresponda.

De esta cadena se extraen solamente el número de cuentas y se convierte de cadena de caracteres a un número entero, este paso es el mismo para los dos canales.

Estos datos se almacenan en un *buffer* según el canal, en la casilla Canal A o Canal B, según corresponda.

En la siguiente casilla “Recorre Gráficos” se actualiza la posición del píxel a dibujar dentro de la pantalla.

En la casilla “Print Canal A Canal B” se imprime el píxel correspondiente en la pantalla.

Después en la casilla de decisión, se cuestiona si la observación ha concluido, si es el caso, se guarda toda la información en el archivo. Si no es el caso se lee el siguiente dato del *buffer* del fotómetro. Como se muestra en la Figura 4.2.

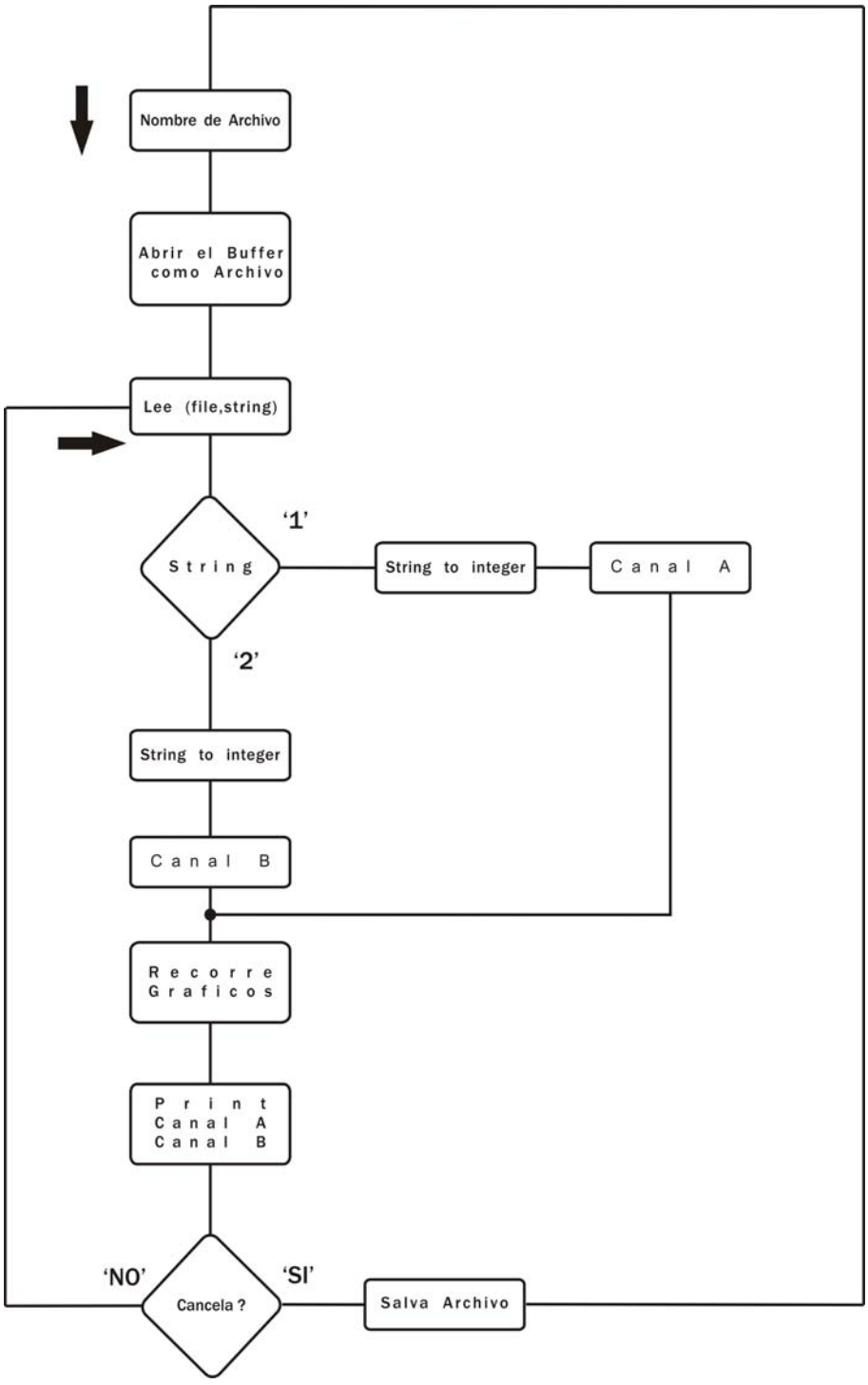


Figura 4.2 Diagrama de Flujo

Debido a que el sistema gráfico toma demasiado tiempo en actualizar la pantalla, y para garantizar que todas las lecturas sean correctas se utilizó un diseño de programa alternativo, haciendo uso de

programación multi-hilos, separando la parte de lectura del *buffer* del fotómetro y la parte gráfica. Como se muestra en la Figura 4.3.

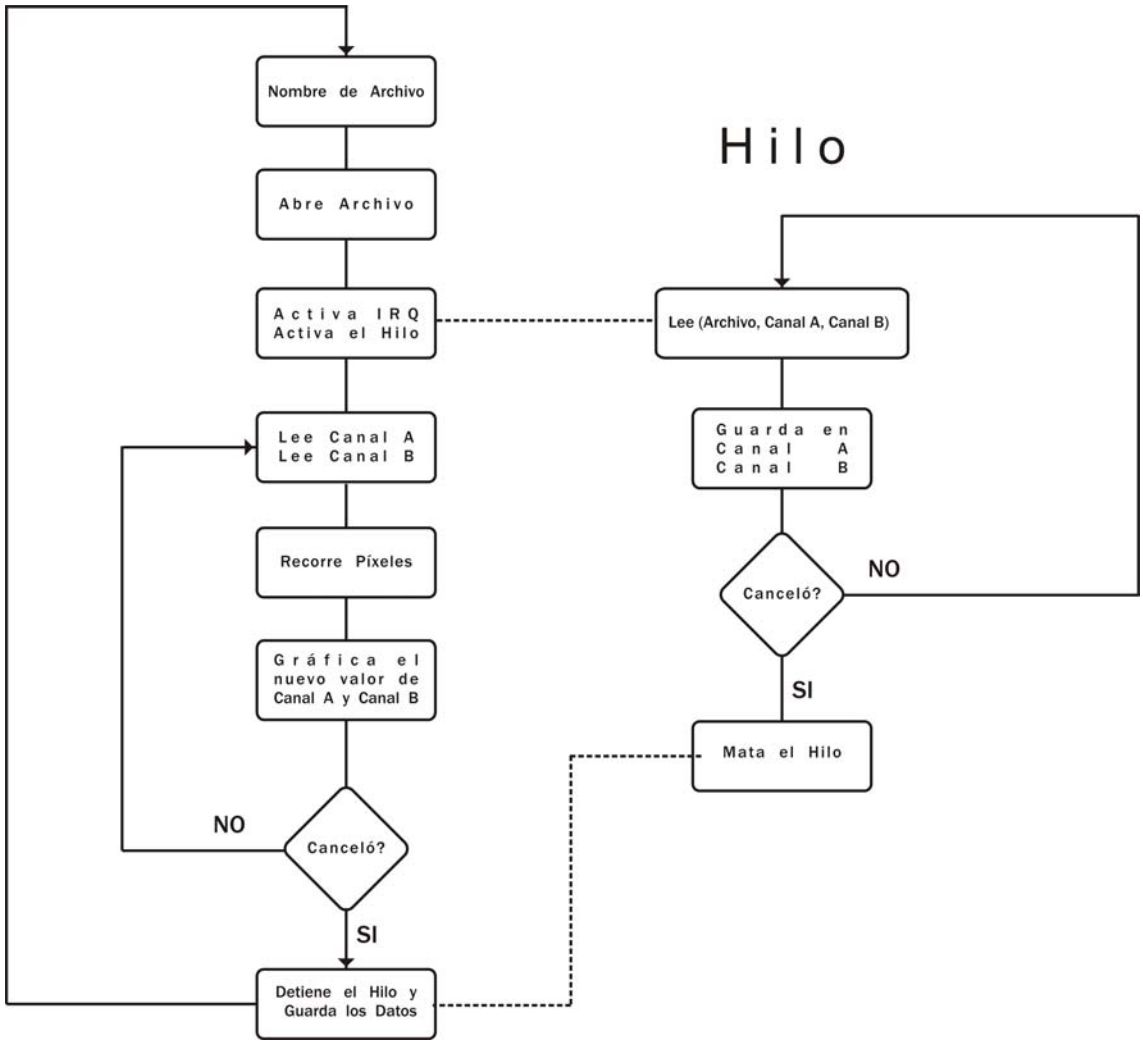


Figura 4.3 Diagrama de Bloques del Programa Alternativo

Primero se tiene que asignar un nombre de archivo en el que se almacenaran las observaciones, si no se asigna un nombre el programa no continúa. Después abre el *buffer* del fotómetro como archivo.

Este *buffer* es el que se creó con el módulo y se encuentra en la siguiente ruta:
/dev/fotometro

En el siguiente bloque se activa la interrupción y arranca el hilo de adquisición.
La función de este hilo es la de leer el *buffer* del dispositivo **fotómetro**, leyendo la cadena de caracteres y separando la información según corresponda a cada canal, esta información se almacena en dos *buffer*, Canal A y Canal B, de donde el programa encargado de graficar toma los datos. Este hilo continúa adquiriendo datos hasta que el proceso padre es cancelado.

En el bloque “Lee Canal A, Canal B” los datos son leídos del *buffer* que comparten el hilo de adquisición con el programa principal.

En el Bloque “Recorre Píxeles” en realidad crea una imagen la cual se imprime en pantalla en el espacio correspondiente a cada canal, del bloque “Gráfica el nuevo valor de Canal A y Canal B”.

Después en la casilla de decisión, se cuestiona si la observación ha concluido, si es el caso, primero se detiene el hilo de adquisición y posteriormente se guarda toda la información de los *buffer* Canal A Canal B en el archivo. Si no es el caso se lee el siguiente dato del *buffer* Canal A y del *buffer* Canal B.

El código con comentarios se encuentra en la sección de anexos para un mejor entendimiento del mismo ver Anexo 13.

Capítulo 5

5.1 Integración del Sistema

Al tener finalizada la tarjeta electrónica de acoplamiento y acondicionamiento, y el software de Adquisición (GUI), continuamos con la integración del Sistema.

El fotómetro debe ser conectado como se muestra en el diagrama de bloques de la Figura 5.1

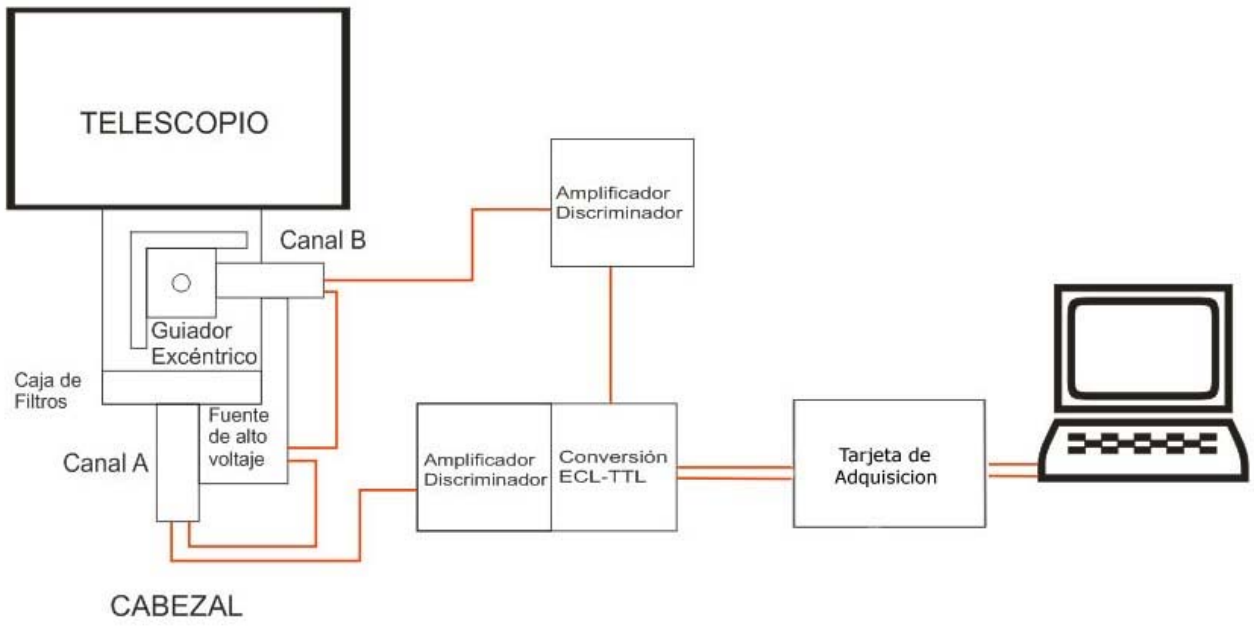


Figura 5.1 Diagrama de Bloques para la conexión del fotómetro.

Debemos de tener mucho cuidado en la conexión de los cables coaxiales del fotómetro, ya que se trabaja con alto voltaje, cabe aclarar que los conectores coaxiales de alto voltaje son diferentes a los conectores coaxiales comunes, por lo que no se debe forzar la conexión de los mismos, si se encuentra con un caso así es porque este se esta conectando de una manera que no es la correcta, por lo cual en las siguientes figuras explicaremos la forma adecuada de la conexión del fotómetro. En la Figura 5.2 se muestra la conexión del fototubo del Canal A.

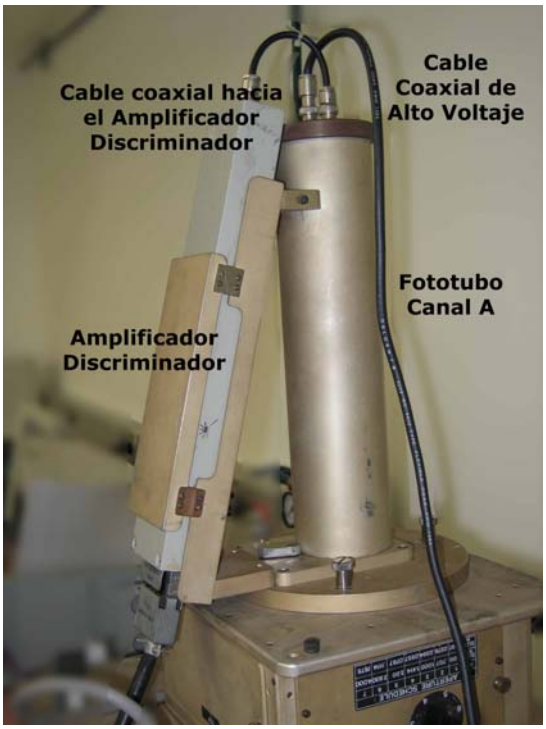


Figura 5.2: muestra que una de las conexiones del fototubo del Canal A se debe conectar a la salida de la fuente de alto voltaje y la otra conexión va conectado al amplificador discriminador.

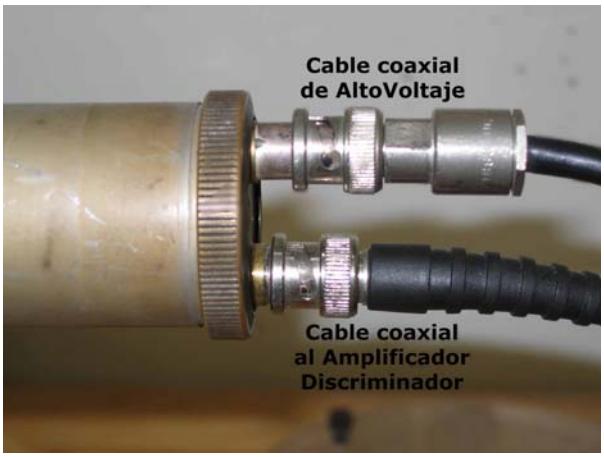


Figura 5.3: Aquí se observa como los conectores coaxiales de alto voltaje son diferentes a los comunes, los conectores coaxiales de alto voltaje son más largos.

En la Figura 5.3 se mostró la adecuada conexión del fototubo del Canal B y a continuación se muestra un Amplificador discriminador (Figura 5.4), así como sus conectores (Figura 5.5).



Figura 5.4: por un lado el amplificador tiene un conector coaxial que es la entrada al Amplificador y por el otro un conector especial que es la salida del Amplificador.

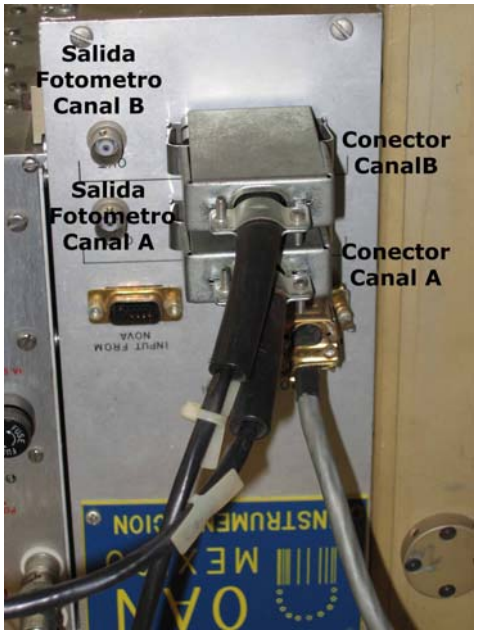


Figura 5.5 Aquí son mostrados los conectores de la salida de los Amplificadores Discriminadores, como se puede observar estos conectores son especiales, tienen solo una forma de conectarse por lo que no se puede conectar de manera incorrecta.

La salida del los Amplificadores Discriminadores, se deben conectar al fotómetro, estos conectores son especiales por lo que no existe problema para conectarlos, en la Figura 5.5 podemos ver las salidas del fotómetro de los Canales A y B.

5.2 Pruebas y calibración en laboratorio

En seguida describiremos el proceso para operar la interfaz de usuario.

Primero se inserta el módulo en el núcleo, se inicia sesión con la cuenta de *root*, estando en el directorio donde se encuentra el archivo del módulo, y con la instrucción ***insmod*** y el nombre del archivo fotómetro.o, después se verifica que se haya cargado el módulo con la instrucción ***lsmod***. Estos pasos son descritos en la Figura 5.6 y Figura 5.7.



Figura 5.6 Aquí nos situamos en la ruta donde esta el programa del módulo y después se inserta.

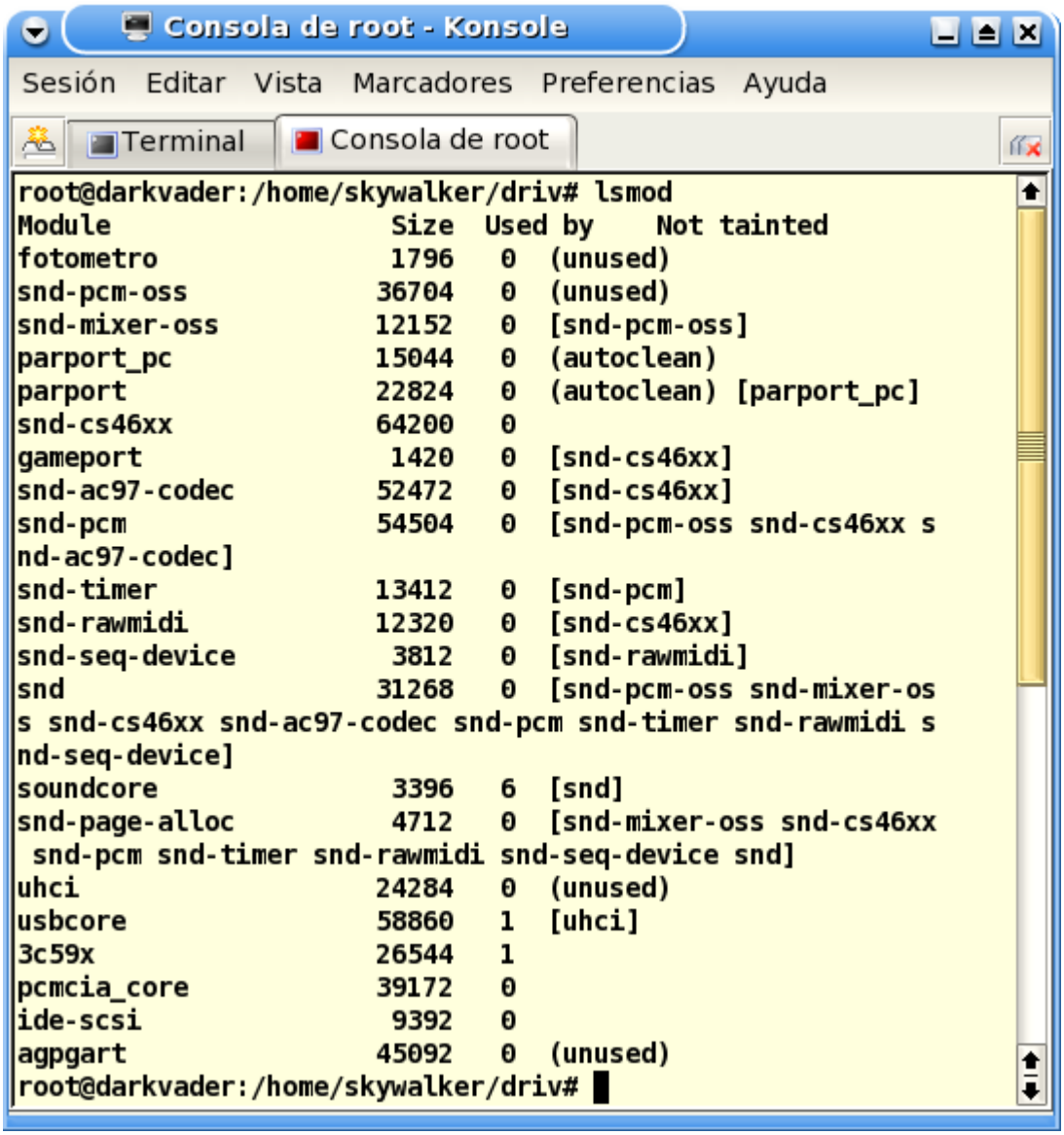


Figura 5.7 Aquí se observa que el Módulo del fotómetro ha sido cargado

Con la instrucción cat/proc/interrupts se despliegan tres columnas que detallan las interrupciones, la primera columna muestra el numero de interrupción, la segunda muestra el numero de interrupciones que ha sido usado por el dispositivo, y en la tercer columna el nombre del dispositivo. Esto se muestra en la Figura 5.8.

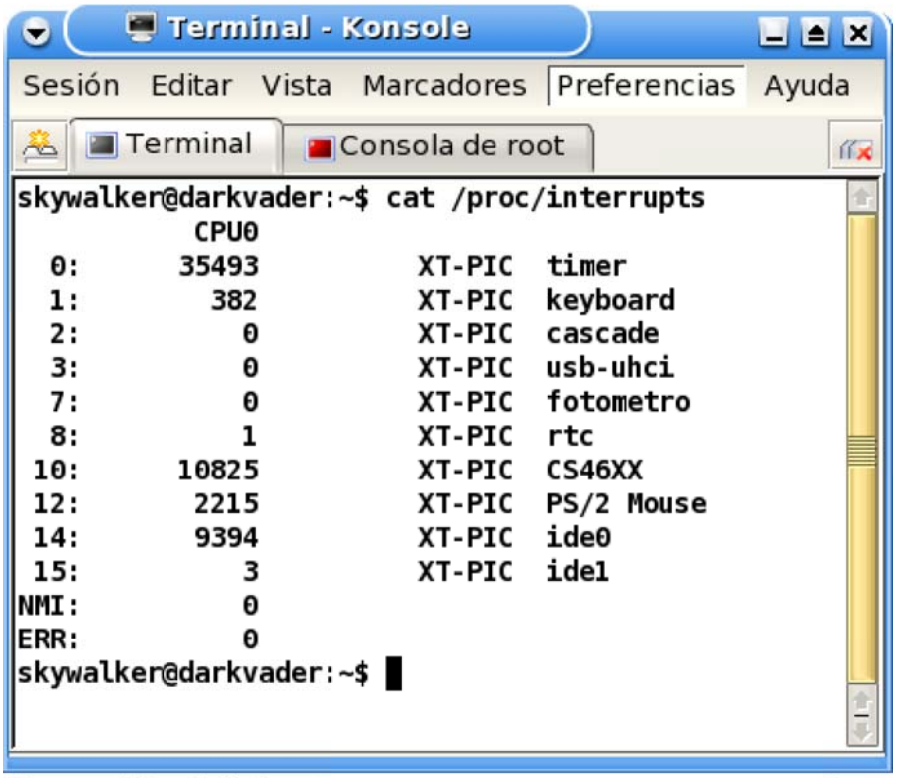


Figura 5.8: el fotómetro esta utilizando la interrupción 7, se observa que no ha sido usada.

Ahora que el módulo ha sido insertado y verificado su funcionamiento, continuamos para ejecutar la interfaz de usuario y describiremos el manejo de la misma.

Primero en la cuenta de usuario se escribe la instrucción xhost + para que desde la cuenta de usuario se pueda ejecutar la interfaz gráfica como *root* y así poder abrir la interfaz desde la cuenta de usuario, ver Figura 5.9. Después con la instrucción *su* se cambia el usuario por el de *root* tecleando la contraseña. Luego nos situamos en el directorio donde esta el archivo ejecutable de la interfaz, ver Figura 5.10. Una vez situados en el directorio ejecutamos el programa de la interfaz llamado fotod, ver Figura 5.11.

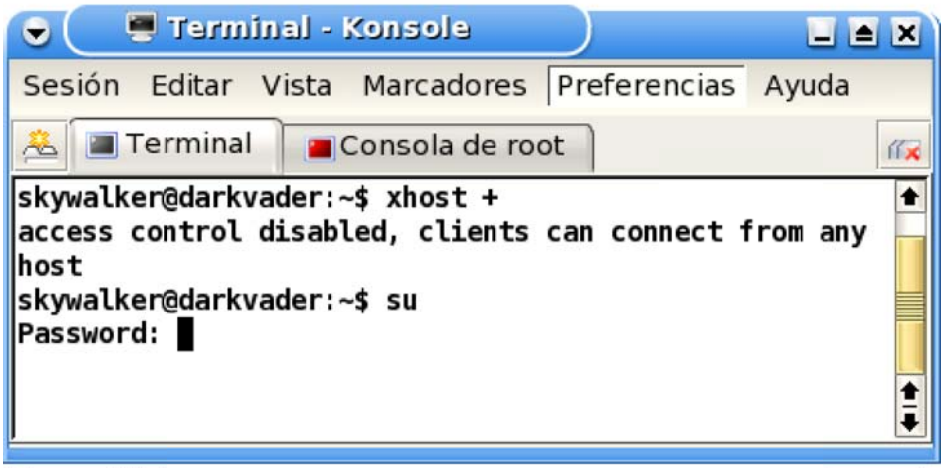


Figura 5.9: instrucción xhost + para acceder a la interfaz del usuario desde *root*

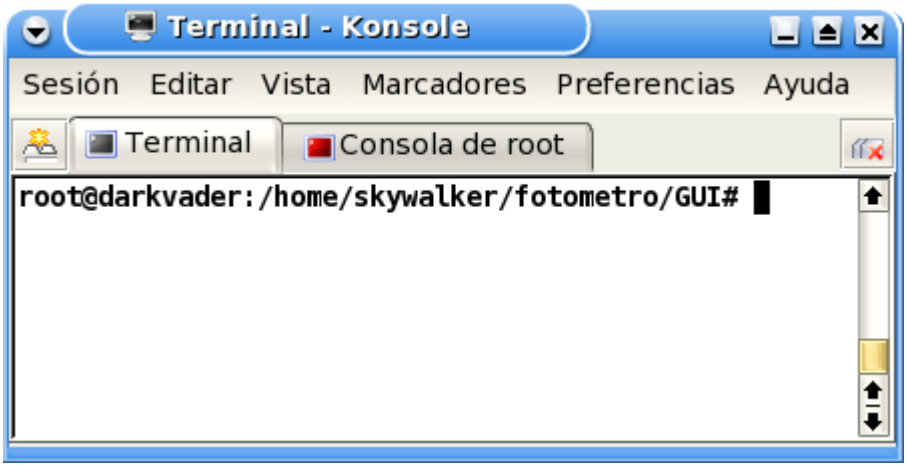


Figura 5.10 Aquí nos situamos en el directorio donde se encuentra el ejecutable fotod

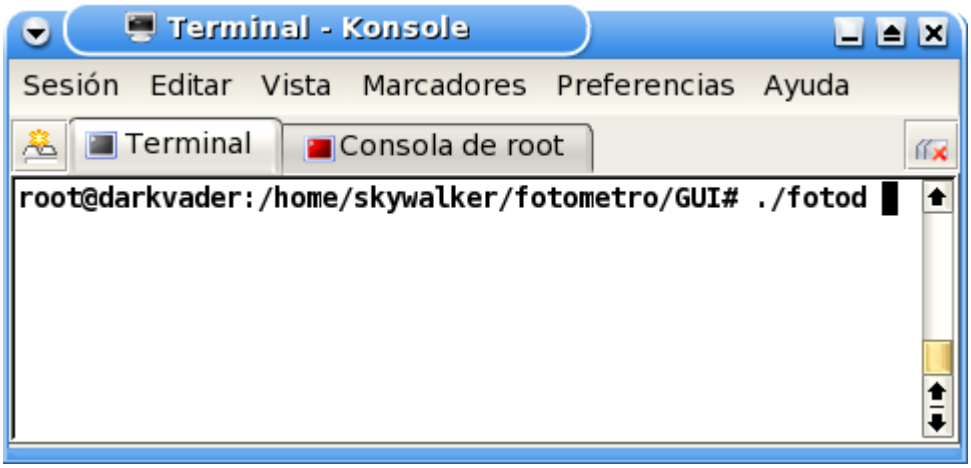


Figura 5.11 Se ejecuta la interfaz de usuario.

Una vez ejecutado el programa de usuario se abre una ventana como la siguiente, ver Figura 5.12. El programa no puede iniciar hasta que se asigne un nombre de archivo para guardar los datos.

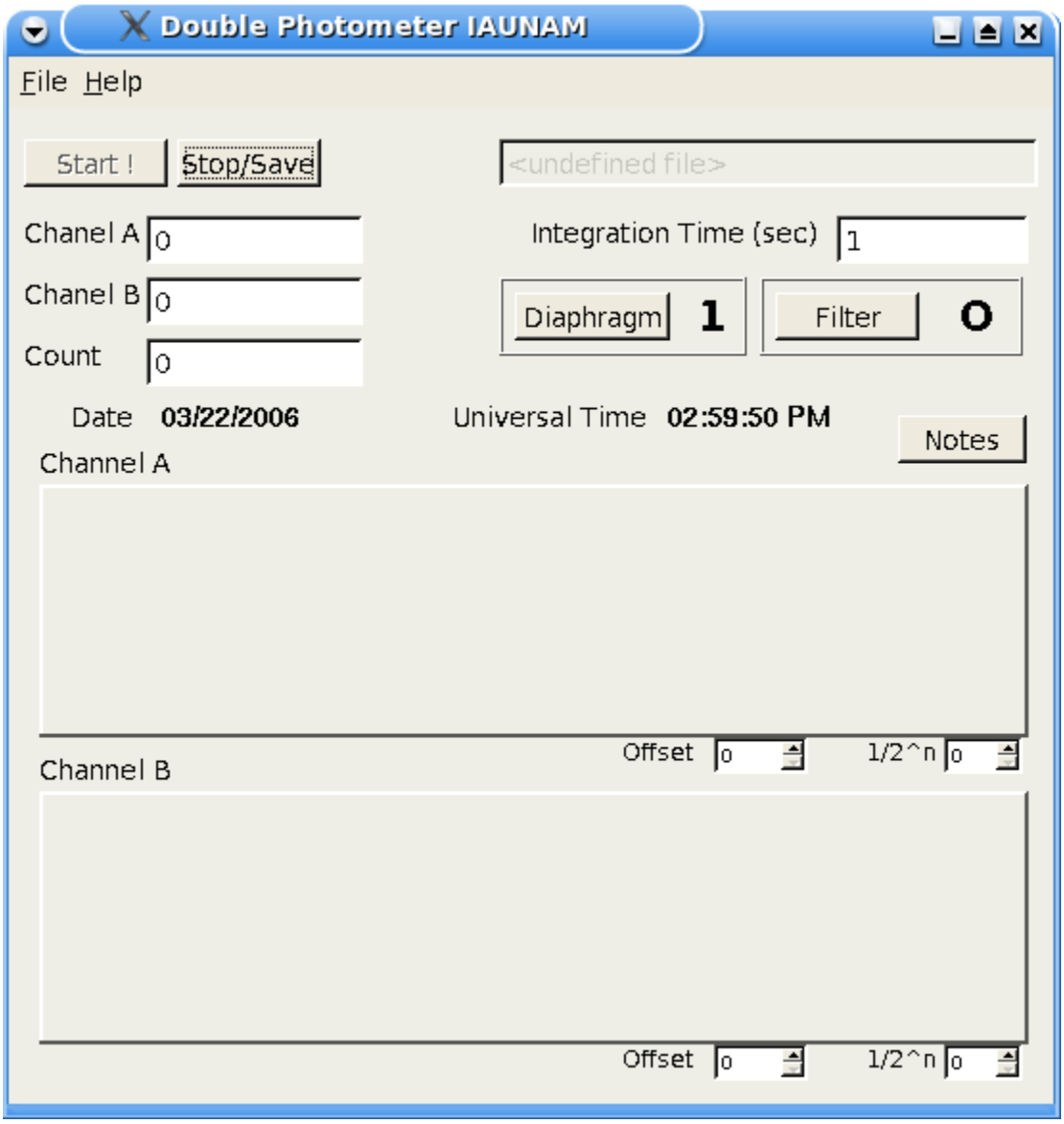


Figura 5.12 Ventana con el botón de Start! Inactivo, debido a que aun no se asigna un archivo donde guardar los datos de las lecturas.

En el menú File, al seleccionar la opción de guardar archivo despliega una ventana en la cual podemos escoger el directorio en donde deseamos guardar el archivo en el cual serán guardados los datos de las lecturas, ver Figura 5.13.

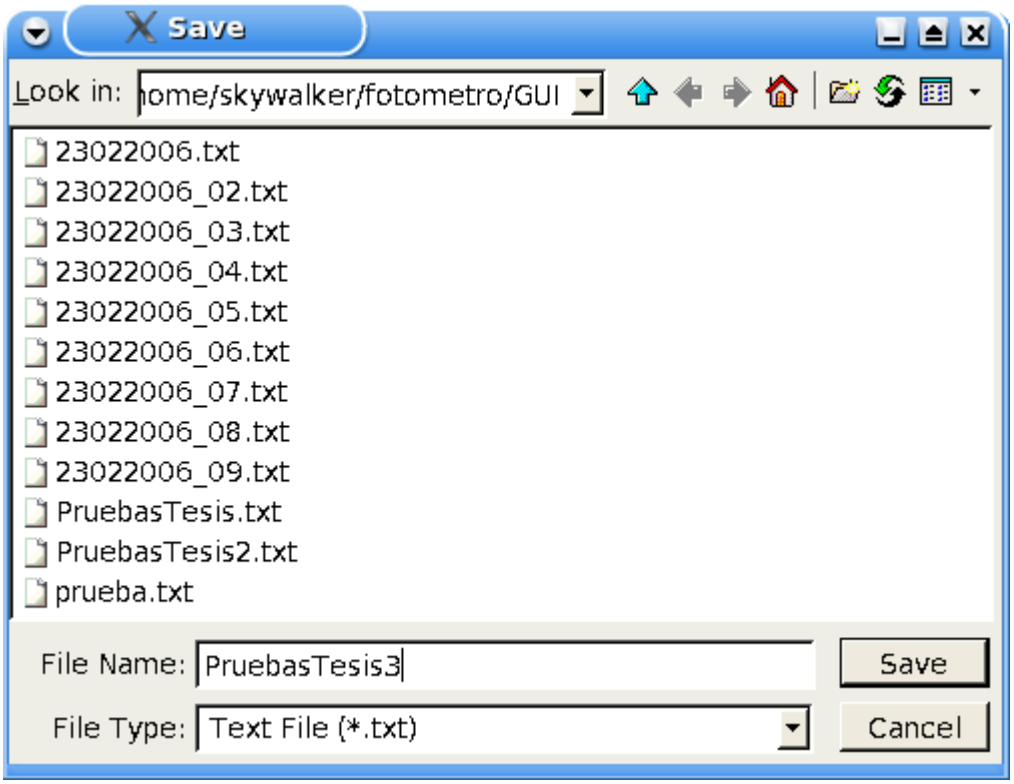


Figura 5.13 Aquí se asigna un nombre de archivo para guardar las observaciones.

Si el nombre del archivo elegido ya existe aparecerá una ventana como en la Figura 5.14.

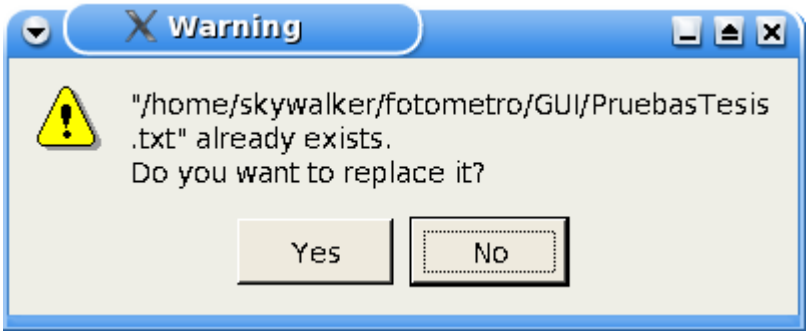


Figura 5.14 Ventana que confirma si queremos remplazar el archivo con el nombre ya existente.

Una vez que se eligió el nombre de archivo, el botón *Start!* estará activo para iniciar con las lecturas de los datos provenientes de las entradas de los canales. Como se muestra en la Figura 5.15.

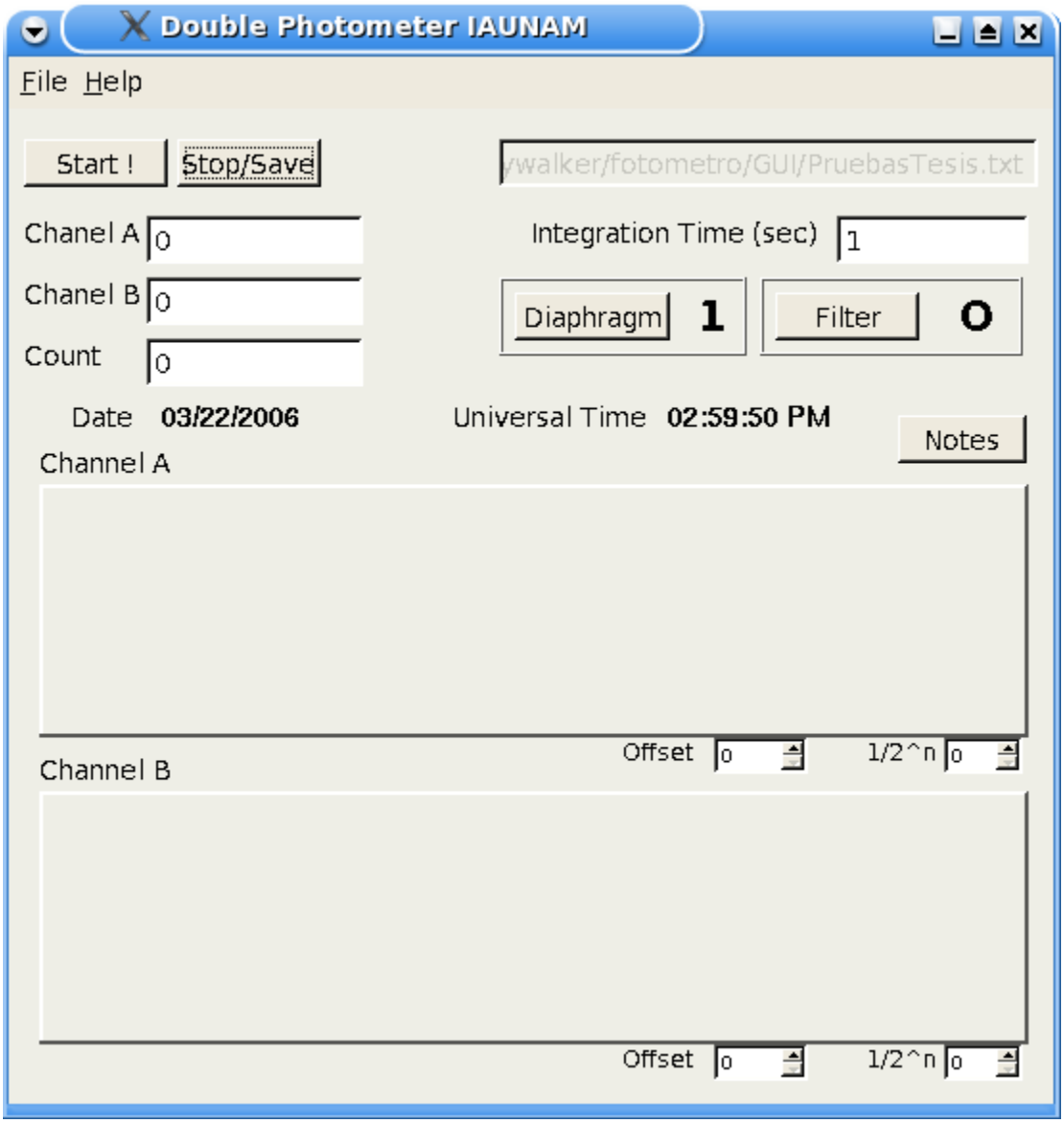


Figura 5.15 Botón *Start!* Activo para iniciar lecturas.

5.2.1 Interfaz de Usuario

La ventana de la Interfaz de usuario cuenta con las siguientes características:

- Botón Diaphragm: Este botón selecciona entre 7 de los diafragmas con los que cuenta el fotómetro.

- Botón Filter: Este botón selecciona entre 4 filtros con los que cuenta el fotómetro, que son O,

U,V,B

- Etiqueta Date: En esta etiqueta nos dice la fecha en la que estamos realizando las observaciones.

- Etiqueta Universal Time: Esta etiqueta nos muestra la hora en la que se esta realizando las observaciones.

- Integration Time: Aquí podemos asignar el tiempo en el cual se tomaran las lecturas es decir cada cuanto tiempo se tomaran las cuentas provenientes de los fototubos.

- Botón Notes: Este botón Despliega una ventana donde podemos ingresar notas a cerca de las observaciones, esta opción con la que cuenta el astrónomo es muy útil debido a que en el transcurso de la toma de lecturas se puede abrir constantemente e ir haciendo anotaciones, se puede cerrar la ventana y las anotaciones seguirán guardadas, hasta que se finalice con el botón Stop. En la Figura 5.16

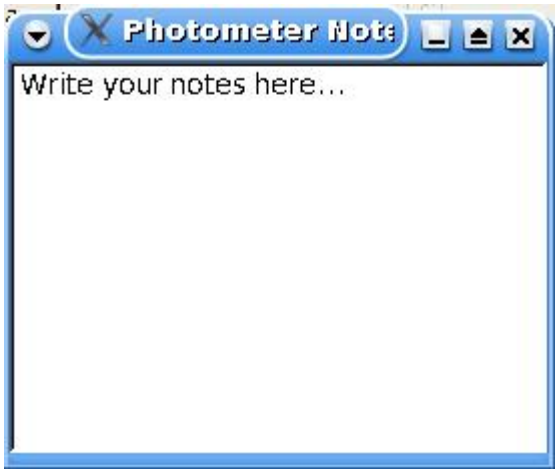


Figura 5.16 Ventana para Realizar Anotaciones

- Channel A y Channel B: Aquí se muestra el número de cuentas que estamos obteniendo por el tiempo de integración que hayamos asignado.

- Counts: Aquí iremos observando el número de cuentas que lleva el sistema desde que iniciamos las lecturas de los datos.

- Gráfica Channel A y Channel B, aquí estarán graficadas las cuentas por el tiempo de integración que hayamos elegido.

- Offset: Este botón desplaza la gráfica hacia abajo.

Una vez que el programa ha iniciado a tomar lecturas al intentar cerrar con la opción de cerrar el programa, este no se cerrara hasta que hayamos detenido el programa correctamente y guardado los

datos de las lecturas.

Ahora realizamos las pruebas de la interfaz de usuario con ayuda de un generador de funciones.

Asignamos un nombre de archivo para que nos permita iniciar el programa, a la entrada de la tarjeta de adquisición colocamos la entrada de un generador de funciones a 100 Hz, con un tiempo de integración de 0.1 segundos.

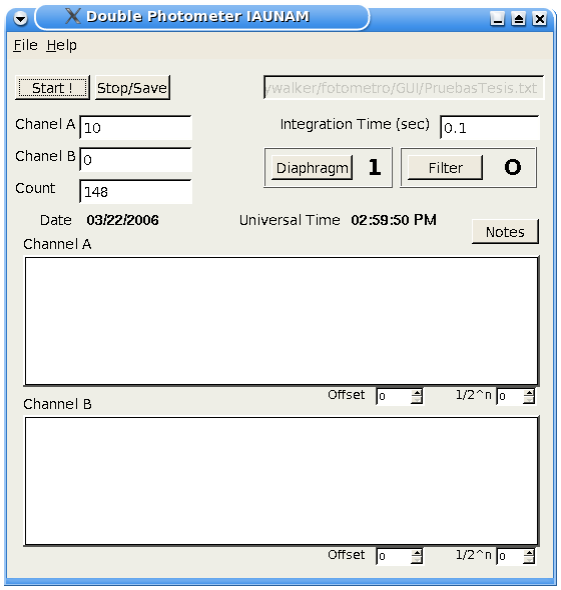


Figura 5.17 Entrada de lecturas a 100Hz con tiempo de integración de 0.1 segundos.

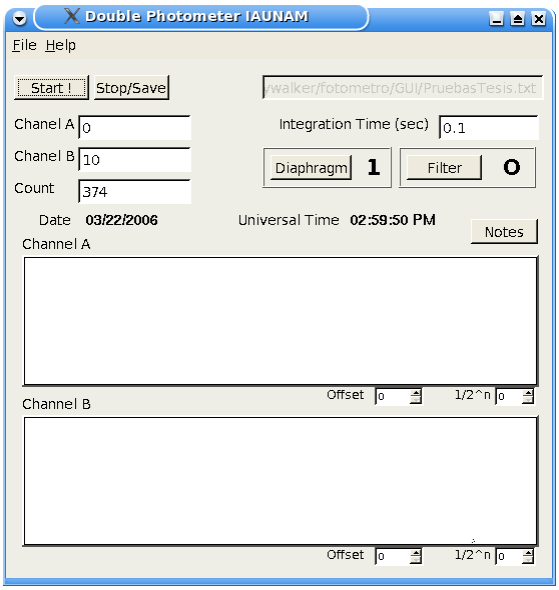


Figura 5.18 Canal B con 10 cuentas por segundo y un total de 374

En la Figura 5.17 se observa que el canal A esta recibiendo 10 cuentas cada 0.1 segundos y en el despliegue de las cuentas lleva un total de 148.

Ahora cambiamos la entrada del Canal A por la del Canal B, y los datos obtenidos los observamos en la Figura 5.18.

Modificamos la frecuencia del generador de funciones a 1KHz y conectamos primero al Canal A y luego al Canal B esto se ve en (Figura 5.19,Figura 5.20)

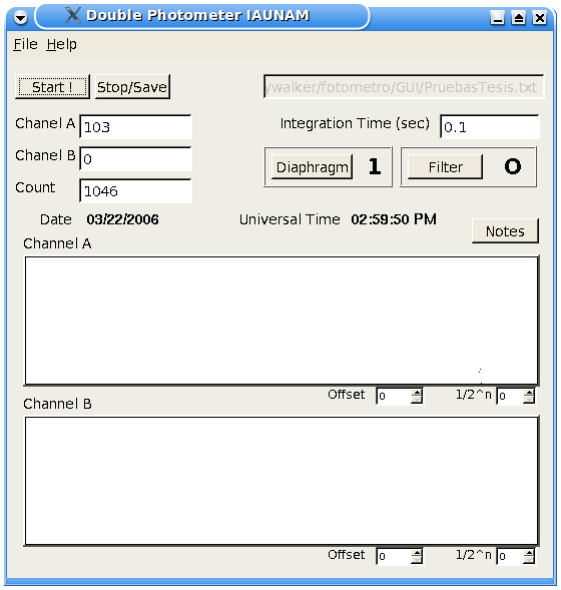


Figura 5.19 Canal A a 1Khz

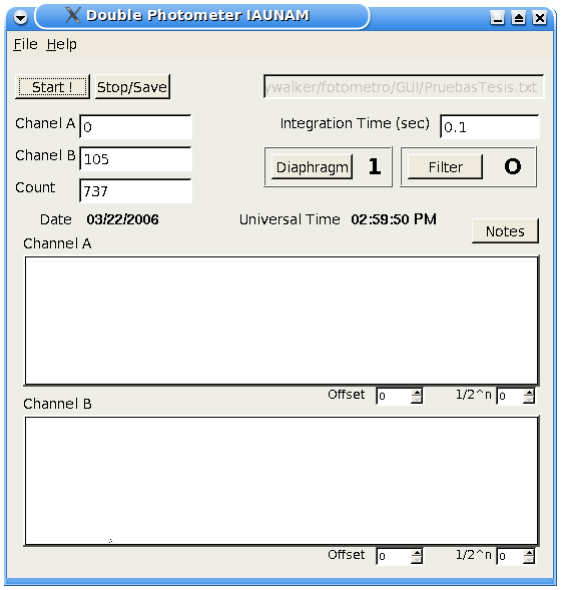


Figura 5.20 Canal B a 1KHz

Aumentamos 10 veces la frecuencia del generador de funciones observando. La Figura 5.21y la Figura 5.22

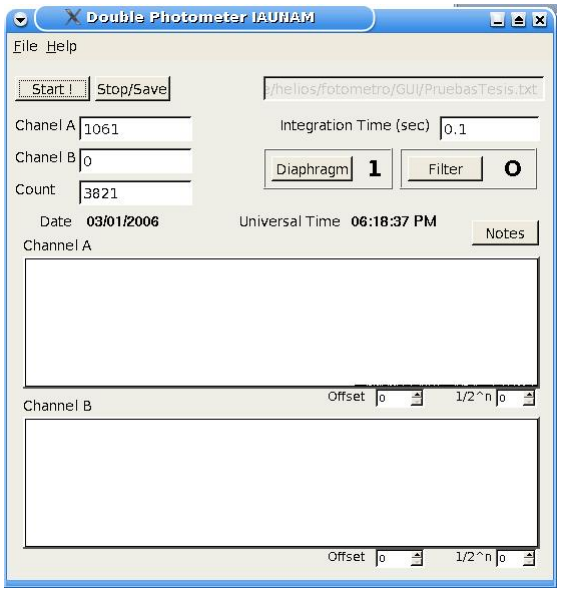


Figura 5.21 Canal A a 10KHz

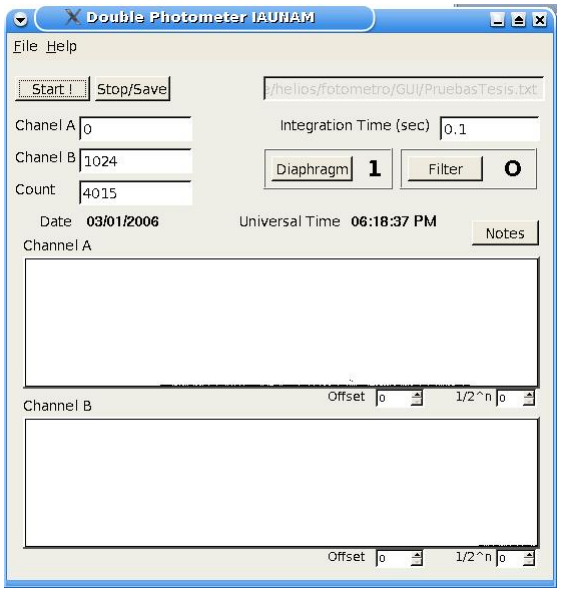


Figura 5.22 Canal B a 10KHz

De las Figura 5.21Figura 5.22 vemos que aumentaron 10 veces el número de las cuentas en el despliegue de los canales con respecto a las figuras anteriores, teniendo el despliegue del Canal A 1061 y el Canal B 1024 cuentas.

Ahora aumentamos la frecuencia del generador de funciones a 100KHz, y observamos algunos puntos

en el despliegue de la gráfica, ver Figura 5.23 el despliegue de las cuentas del canal A se observa que están a 10529, después cambiamos el conector al canal B para observar las cuentas, esto se muestra en la Figura 5.24.

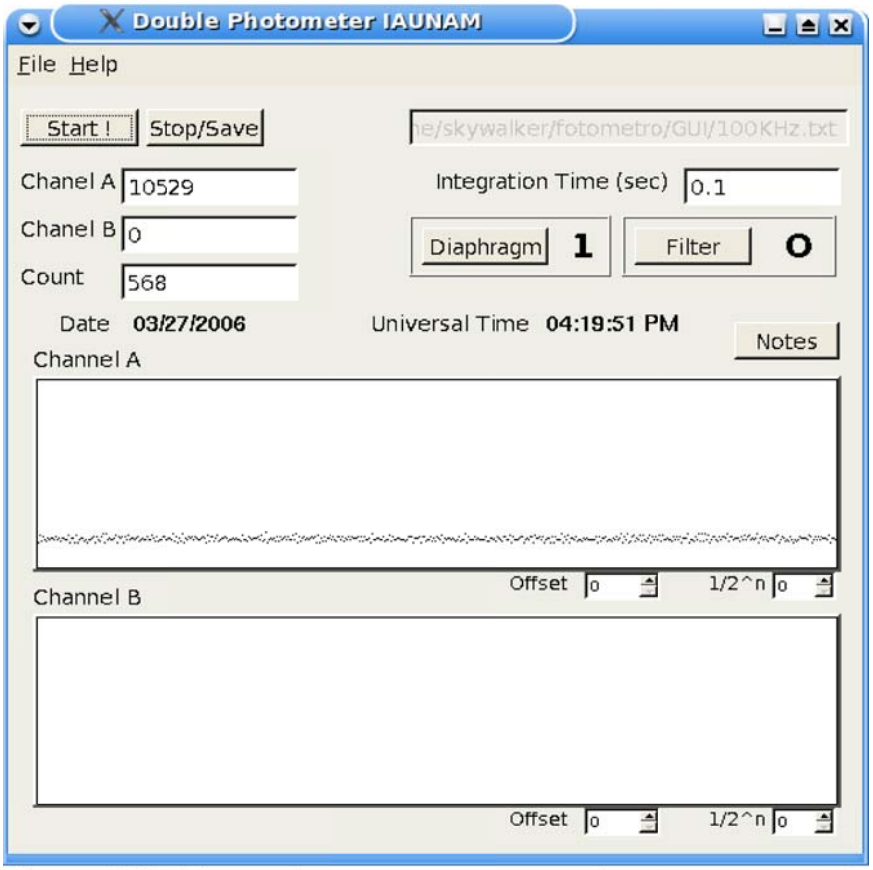


Figura 5.23: En esta imagen Observamos algunos puntos en la gráfica del Canal A

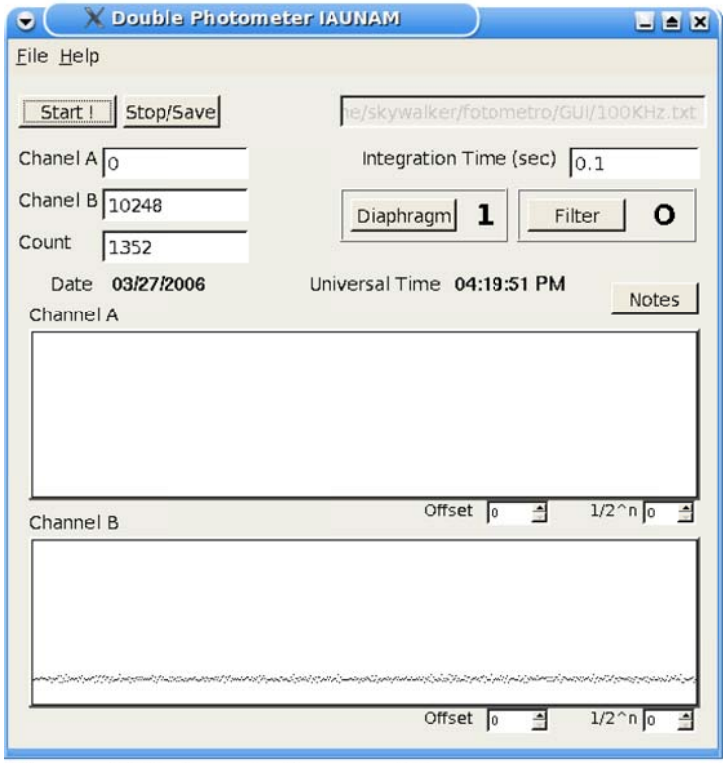


Figura 5.24: En esta imagen Observamos algunos puntos en la gráfica del Canal B

Realizamos variaciones de frecuencia para verificar que en las gráficas también se observen estas variaciones.

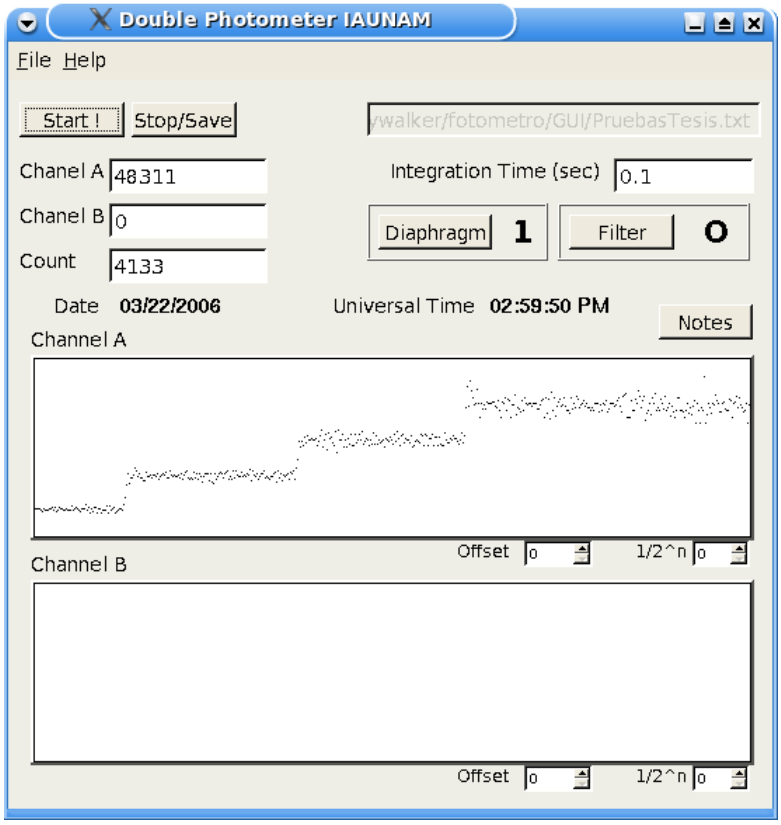


Figura 5.25 Aquí Se variaron las frecuencias de 100 KHz a 400KHz en el Canal A

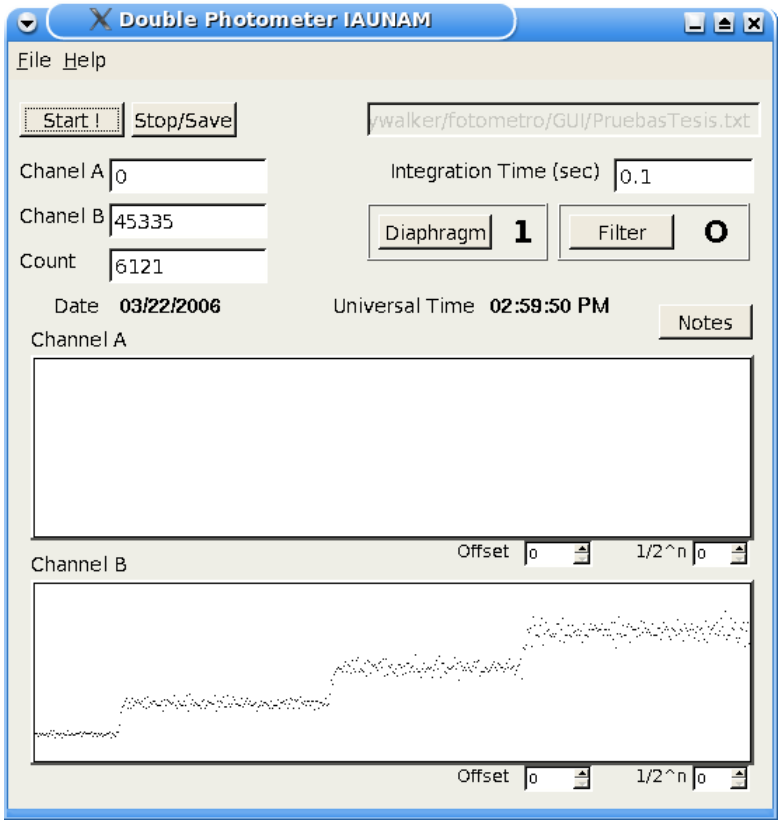


Figura 5.26 En el Canal B también se hicieron variaciones de frecuencias de 100 KHz a 400Khz.

5.3 Pruebas en Sitio Astronómico

5.3.1 Instalación en el Telescopio

El cabezal del fotómetro se instaló y operó en el telescopio, mientras que el sistema de adquisición se instaló sobre la plataforma, ambos se conectan a través de cables coaxiales. Para posteriores observaciones se aconseja que el sistema de adquisición este cerca para comodidad del Astrónomo para que pueda efectuar los procedimientos de alineación y ajuste en el cabezal del fotómetro.

Para instalar el cabezal se colocó la parte circular del mismo alineada a la platina del telescopio, orientando los buscadores según lo requirió el observador. El fototubo del canal A se coloca en la parte inferior del cabezal, este tiene tres tornillos de sujeción, los cuales no se pueden retirar de la base del fototubo. El fototubo del canal B se coloca en la parte derecha del guiador excéntrico, el fotómetro cuenta con cuatro tornillos de tipo *allen* para sujetarlo a la platina.

Una vez montado el fotómetro en la platina se procede a la conexión del mismo como se explico al inicio de este capítulo.

Ya conectado correctamente el fotómetro (ver Figura 5.28), se abre la cúpula del observatorio para empezar con las pruebas de fotometría, continuamos con la localización de la estrella VZcnc, ingresando las coordenadas a la computadora (ver Figura 5.27) que controla el movimiento del telescopio, Una vez ingresadas las coordenadas el telescopio se mueve en la dirección de la estrella a localizar, la computadora esta programada para que siga el movimiento de la estrella VZcnc, esto es que a pesar del movimiento de rotación de la tierra, el telescopio no pierda a la estrella y siga apuntando a la dirección de la misma.

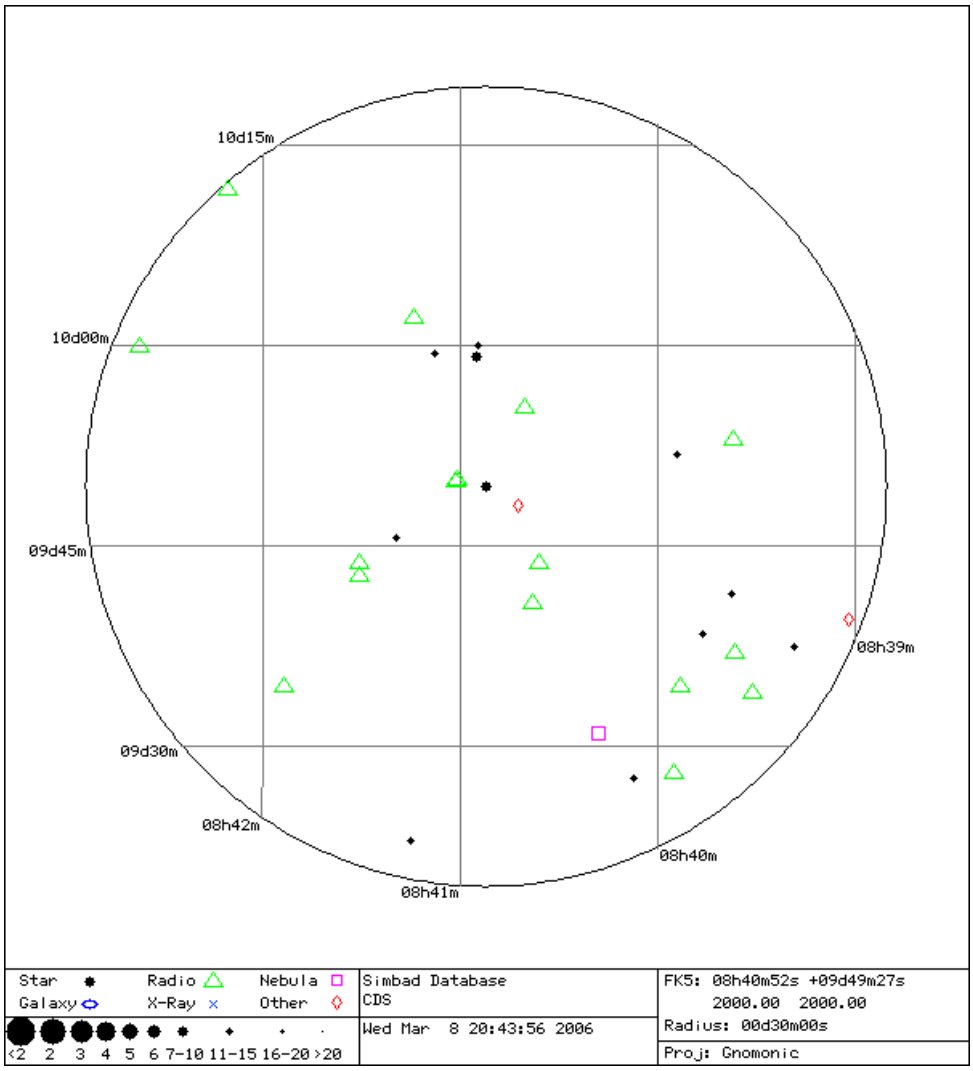


Figura 5.27 Coordenadas de la estrella VZcnc

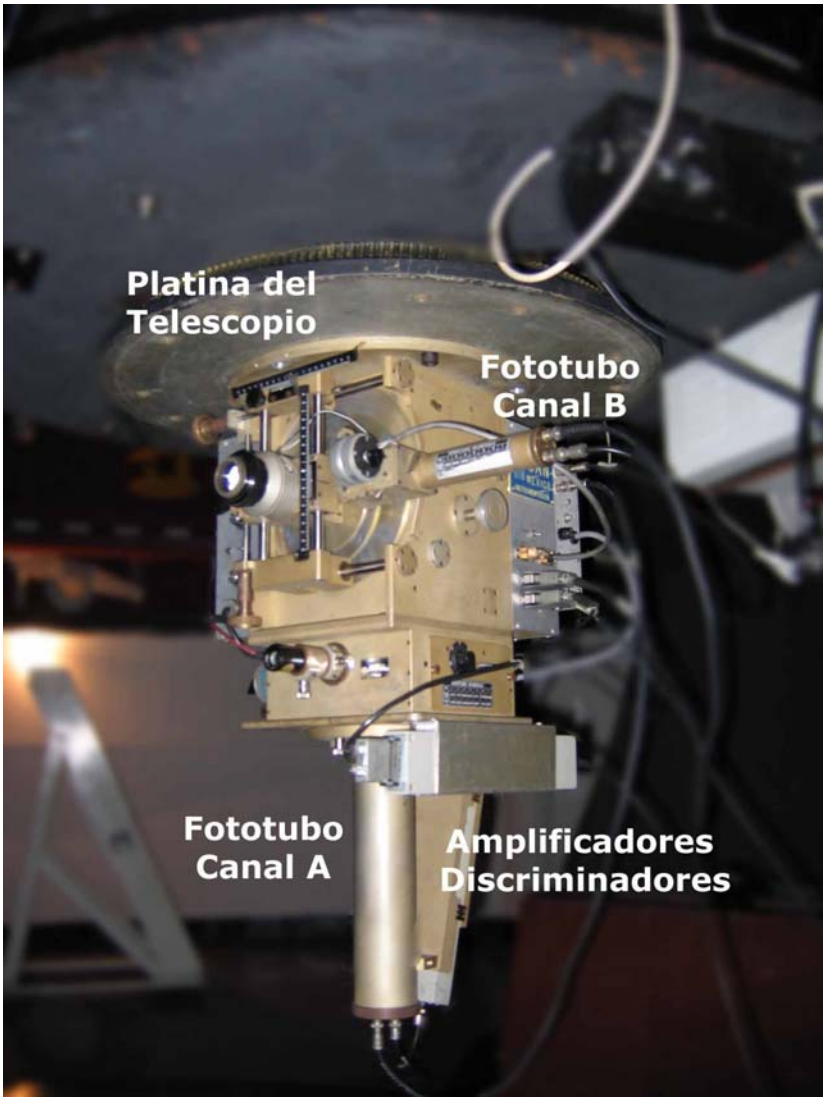


Figura 5.28: Fotómetro montado en la platina del telescopio

Para realizar la fotometría en el canal A se debe tener en cuenta que el espejo plano a 45º se encuentre en la posición donde el agujero central esté alineado con el fototubo (La perilla debe estar en posición afuera).

El periscopio esté en la posición donde no obstruya el paso de la luz al detector.

El filtro y el diafragma seleccionado según los requerimientos observacionales.

El espejo plano a 45º con agujero central del guiador excéntrico puede moverse con la perilla colocada a un costado del cabezal, esta tiene dos posiciones aseguradas. Para pasar de una posición a otra, la perilla se gira un cuarto de vuelta contra el sentido de las manecillas del reloj, se extrae o empuja hasta

la posición siguiente y para asegurarla se gira un cuarto de vuelta en sentido contrario al anterior. En la Figura 5.29 se muestra la perilla del espejo a 45º.

Para asegurar que la estrella está centrada en el diafragma se utiliza el guiador excéntrico.



Figura 5.29 Perilla del espejo a 45º

5.3.2 Proceso de encendido del Fotómetro

Ya localizada la estrella y centrada en el guiador continuamos con el encendido del fotómetro

- Ajustamos los voltajes de operación a 1700 Volts.
- Colocamos el interruptor general en la posición de encendido, esto permite que todo el sistema este alimentado.
- Por ultimo se prende la fuente de alto voltaje.

Ya alimentado el fotómetro verificamos que esté arrojando datos los canales del Fotómetro, con un osciloscopio tomamos mediciones en las salidas de los canales del fotómetro, para verificar el conteo en los fototubos y después continuar con la conexión de la interfaz de adquisición.

Una vez que comprobamos que los canales del fotómetro estaban arrojando datos, procedimos con la conexión a tierra. Medimos con un Multimetro que la tierra del fotómetro y la tierra de la computadora se encuentren al mismo voltaje. Lo cual no fue así, ya que existió una diferencia de voltaje de 75 Volts, por lo que conectamos el chasis del fotómetro al chasis del CPU.

Corrigiendo este problema de tierras, seguimos con la conexión de la interfaz de adquisición.

Montamos la tarjeta electrónica de acoplamiento a la computadora en el bus ISA.

Empleamos los pasos anteriormente descritos en este mismo capítulo para ejecutar la interfaz de usuario de la computadora de adquisición.

Describiéndolos una vez más en breve.

- Insertar el módulo del dispositivo.
- Verificar la inserción del módulo.
- Ejecutar la interfaz de usuario
- Asignar nombre al archivo en el que se guardaran las observaciones.

Ya que está insertado el módulo y el programa de adquisición ejecutándose, conectamos los canales a la tarjeta de adquisición y procedemos con la toma de lecturas.

5.3.3 Proceso de apagado del fotómetro.

Una vez terminada la sesión de observación, se apaga la fuente de alto voltaje y después el interruptor general.

En la Figura 5.30 se observa el sistema completo (Telescopio, Fotómetro y Sistema de adquisición).

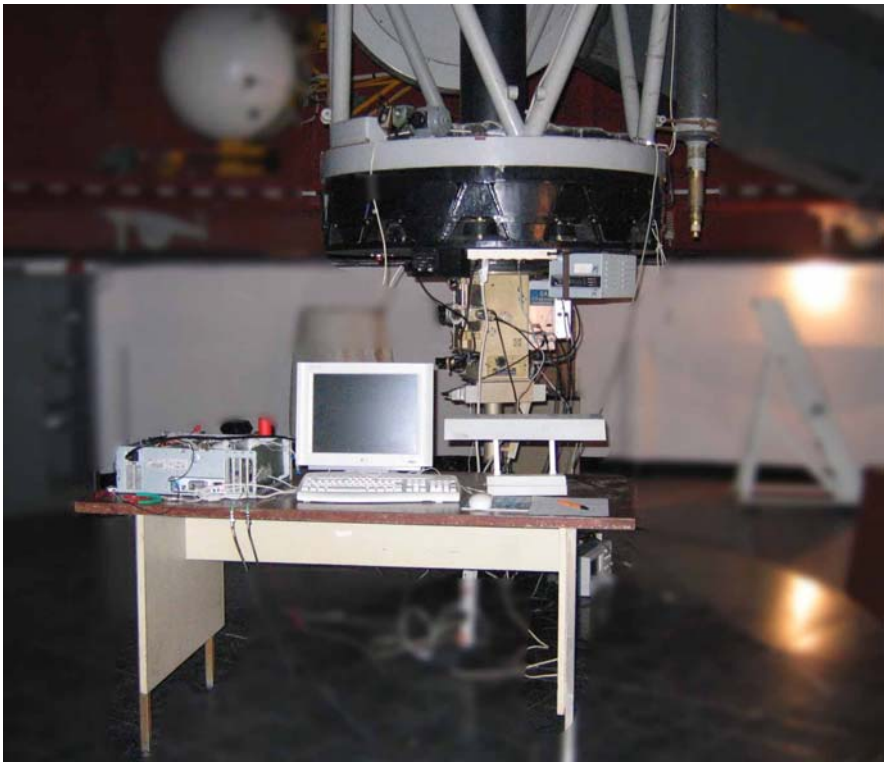


Figura 5.30: Sistema Completo.

Con las gráficas mostradas en la Figura 5.31 el Astrónomo José H. Peña verificó el correcto funcionamiento del fotómetro y del sistema de adquisición.

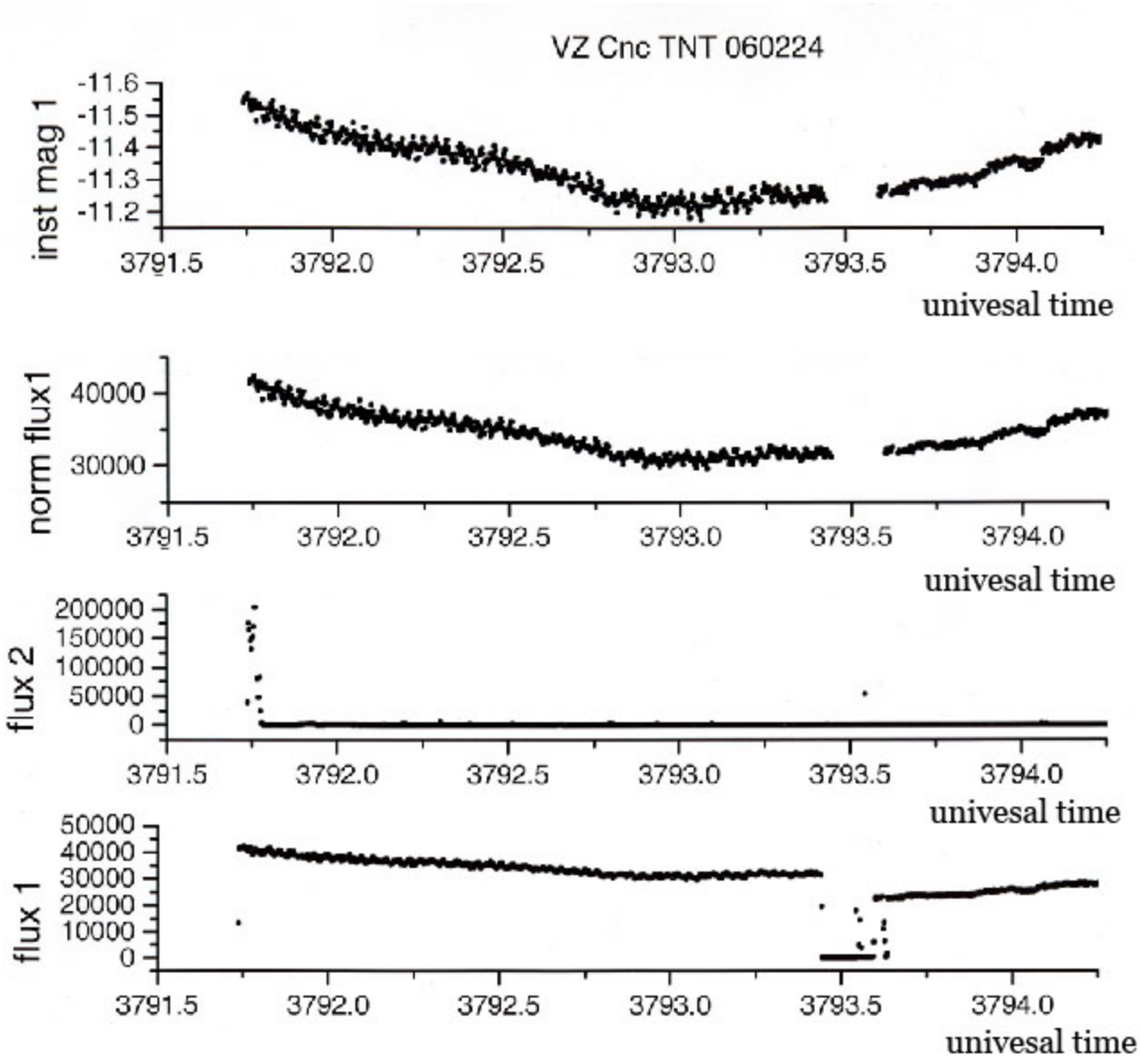


Figura 5.31: Gráfica elaborada por el Astrónomo J. H. Peña para comprobar el funcionamiento del Sistema de Adquisición.

Capítulo 6 Resultados y Conclusiones

Con el presente trabajo de tesis el fotómetro doble rápido del Instituto de Astronomía de la UNAM se ha rehabilitado a un estado de funcionamiento, actualizando la electrónica de adquisición, la cual era inservible y obsoleta. También actualizamos la computadora en la que la tarjeta de adquisición reside, los programas que controlan la tarjeta, y el software que despliega gráficamente los datos adquiridos por este sistema y los presenta de una manera amigable al usuario.

Por lo tanto realizamos la implementación, puesta a punto y operación de la electrónica de Acondicionamiento y Acoplamiento (*hardware)* con sus correspondientes partes de software el módulo y la interfaz gráfica de usuario, con las cuales obtuvimos los datos arrojados por el fotómetro.

Mediante lecturas adquiridas por el fotómetro tanto en el laboratorio de electrónica, como en el Observatorio de Tonantzintla del Instituto de Astronomía de la UNAM, comprobamos el correcto funcionamiento, en tiempo real transfiriendo las lecturas del fotómetro a la computadora, posteriormente almacenando las lecturas en el disco duro de la computadora, con lo cual comprobamos el correcto funcionamiento del Sistema de Adquisición, con esto verificamos la operación del instrumento.

Corroboramos la veracidad de las lecturas con la gráficas de referencia de la estrella VZcnc con los resultados obtenidos, y después del análisis realizado, concluimos que los resultados son correctos y con ello verificamos la confiabilidad del Sistema.

El fotómetro doble rápido puede ser usado como un instrumento de prácticas en la formación de astrónomos del Instituto de Astronomía de la UNAM.

El sistema se encuentra listo para ser controlado de manera remota, agregando software para este fin, utilizando toda las ventajas que el sistema operativo GNU/Linux ofrece.

La electrónica de acondicionamiento, conteo, adquisición y comunicación con el ducto de la computadora se fabricaron en México y los diagramas se encuentran en el Instituto de Astronomía, por lo que, no es dependiente de terceros y nos brinda cierto nivel de independencia tecnológica.

Toda la electrónica y el software se diseño e implemento en los laboratorios del Instituto de Astronomía de la UNAM, por lo que no fue necesario la manufactura por parte de otra empresa.

El sistema fue diseñado y desarrollado para trabajar con software libre y de código abierto, por lo que se abolieron costos de licencias y dependencia tecnológica. Por lo que los costos del sistema se redujeron al mínimo.

Este trabajo de tesis puede servir:

- Como referencia para la programación de módulos bajo el sistema GNU/Linux.
- Para futuras actualizaciones del sistema de adquisición.
- Como referencia en el manejo y control del fotómetro para los astrónomos.
- Como auxiliar al astrónomo en el procedimiento de observación, con el sistema de adquisición del fotómetro.

La ingeniería electrónica abarca campos tan diversos que nos permitió aplicar nuestros conocimientos en el campo de la astronomía.

Como resultado de este proyecto, obtuvimos otro tipo de conocimientos durante el desarrollo de la implementación y puesta en marcha del sistema, conocimientos como el manejo del sistema operativo, GNU/Linux y de su estructura, también, que para llevar a cabo las lecturas en tiempo real es necesario la creación de un módulo debido a que programando a nivel usuario no se obtienen todas las lecturas debido a la velocidad de adquisición que este sistema demanda. Esto es porque el administrador de recursos y de tiempo del procesador del Sistema Operativo Linux, tiene una velocidad de atención para cada proceso de aproximadamente 10ms, y el sistema de adquisición requiere de una velocidad de al menos 1ms para la lectura de los datos.

El conjunto del sistema de adquisición tiene espacio para mejorar. por ejemplo, haciendo que el sistema pueda ser operado remotamente a través de Internet, teniendo en cuenta las implementaciones necesarias para el control de seguridad del sistema, esto con el fin de evitar que personal ajeno al Instituto de Astronomía tengan acceso al mismo.

Bibliografía

[1] Bandel David, Napier Robert, "Edicion Especial Linux", 2001
[2] Calbet Xavier, "Breve Tutorial para escribir drivers en Linux",
[3] Cogdell J.R., "Fundamentos de Electrónica", 2000
[4] Eggebrettch Lewis C, "Interfacing to the IBM Personal Computer", 1987
[5] Henden Arne A. & Kaitchuck Ronald H., "Astronomical photometry", 1990
[6] Jay Salzman, Ori Pomerantz, "The Linux Kernel Module Programming Guide",
[7] Kernighan Brian W. & Ritchie Dennis M., "The C Programming Language",
[8] Love Robert, "Linux Kernel Development", 2005
[9] Martin Armendariz Alvaro, "Sistema de Control y Adquisición de datos para un Fotómetro doble rápido en aplicaciones Astronomicas", 1989
[10] Olaf Kirch, "Guia del Nucleo",
[11] Ori Pomerantz, "Guia de Programación de Módulos del Núcleo Linux",
[12] Peña Saint Martin Jose H., "Comunicación Personal" ,
[13] Rubini Alessandro, Jonathan Corbet, "Linux Device Drivers", 2001
[14] Sanchez Beatriz, Angeles Fernando, Iriarte Arturo, "Fotómetro rápido de dos canales", 1994
[15] http://www.rae.es
[16] ftp://ftp.slackbook.org/pub/slackbook/slackbook-2.0.pdf
[17] http://es.tldp.org/Manuales-LuCAS/doc-progmodlinux/doc-progmodlinux-html
[18] http://www.tldp.org/LDP/lkmpg/2.4/html/
[19] http://es.tldp.org/Manuales-LuCAS/ENL/enl-0.8.2-0.2.tar.gz
[20] http://es.tldp.org/Presentaciones/200103hispalinux/calbet/pdf/driv_tut_last.pdf
[21] http://www.hamamatsu.com

Anexos

Hojas de especificaciones.

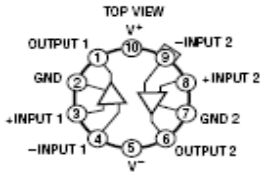
Anexo 1: Hoja de especificaciones del circuito LM319

LT119A/LT319A
LM119/LM319

ABSOLUTE MAXIMUM RATINGS

Supply Voltage 36V
Output to Negative Supply Voltage 36V
Ground to Negative Supply Voltage 25V
Ground to Positive Supply Voltage 18V
Differential Input Voltage (Note 5) ..... ±5V
Differential Input Current (Note 5) .....±5mA
Input Voltage (Note 1)
Output Short-Circuit Duration 10s
Operating Temperature Range
LT119A, LM119 ..... −55°C to 125°C
LT319A, LM319 ..... 0°C to 70°C
Storage Temperature Range ..... −65°C to 150°C
Lead Temperature (Soldering, 10 sec)..... 300°C

PACKAGE/ORDER INFORMATION

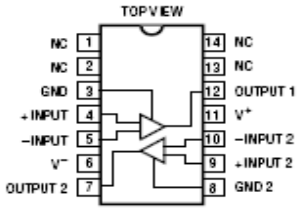


H PACKAGE
10-LEAD TO-5 METAL CAN

T_{JMAX} = 150°C, θ_{JA} = 150°C/W, θ_{JC} = 45°C/W

ORDER PART NUMBER

LT119AH
LM119H
LT319AH
LM319H



J PACKAGE
14-LEAD CERAMIC DIP

N PACKAGE
14-LEAD PLASTIC DIP

T_{JMAX} = 150°C, θ_{JA} = 100°C/W (J)
T_{JMAX} = 85°C, θ_{JA} = 100°C/W (N)

LT119AJ
LM119J
LT319AJ
LM319J
LT319AN
LM319N

Consult factory for Industrial and Military grade parts.

ELECTRICAL CHARACTERISTICS (Note 2)

SYMBOL	PARAMETER	CONDITIONS		LT119A MIN	LT119A TYP	LT119A MAX	LT119 MIN	LT119 TYP	LT119 MAX	UNITS
V_{OS}	Input Offset Voltage	V_S = ±15V, V_{CM} = 0			0.3	0.5			4	mV
V_{OS}	Input Offset Voltage	(Note 3)			0.5	1.0		0.7	4	mV
			●		1.2	2.0			7	mV
CMRR	Common-Mode Rejection Ratio			90	106					dB
I_{OS}	Input Offset Current	(Note 3)			20	40		30	75	nA
			●			75			100	nA
I_B	Input Bias Current	(Note 3)			150	500		150	500	nA
			●			1000			1000	nA
A_V	Voltage Gain			20	40		10	40		V/mV
	Response Time	(Note 4)			80			80		ns
V_{SAT}	Saturation Voltage	$V_{IN} \le -5mV$, I_O = 25mA $V^+ \ge 4.5V$, $V^- = 0V$			0.75	1.5		0.75	1.5	V
		$V_{IN} \le -6mA$, $I_{SINK} \le 3.2mA$ $T_A \ge 0°C$			0.23	0.4		0.23	0.4	V
		$T_A \le 0°C$				0.6			0.6	V
	Output Leakage Current	$V_{IN} \ge 5mV$, V_{OUT} = 35V			0.2	2		0.2	2	µA
			●		1	10		1	10	µA
	Input Voltage Range	V_S = ±15V	●	−12	±13	12	−12	±13	12	V
		V^+ = 5V, V^- = 0V	●	1		3	1		3	V

Anexo 2: Hojas de especificaciones del circuito LM385

National Semiconductor

February 2000

LM185/LM285/LM385
Adjustable Micropower Voltage References

General Description

The LM185/LM285/LM385 are micropower 3-terminal adjustable band-gap voltage reference diodes. Operating from 1.24 to 5.3V and over a 10µA to 20mA current range, they feature exceptionally low dynamic impedance and good temperature stability. On-chip trimming is used to provide tight voltage tolerance. Since the LM185 band-gap reference uses only transistors and resistors, low noise and good long-term stability result.

Careful design of the LM185 has made the device tolerant of capacitive loading, making it easy to use in almost any reference application. The wide dynamic operating range allows its use with widely varying supplies with excellent regulation.

The extremely low power drain of the LM185 makes it useful for micropower circuitry. This voltage reference can be used to make portable meters, regulators or general purpose analog circuitry with battery life approaching shelf life. Further, the wide operating current allows it to replace older references with a tighter tolerance part.

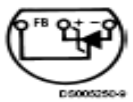
The LM185 is rated for operation over a −55°C to 125°C temperature range, while the LM285 is rated −40°C to 85°C and the LM385 0°C to 70°C. The LM185 is available in a hermetic TO-46 package and a leadless chip carrier package, while the LM285/LM385 are available in a low-cost TO-92 molded package, as well as S.O.

Features

- Adjustable from 1.24V to 5.30V
- Operating current of 10µA to 20mA
- 1% and 2% initial tolerance
- 1Ω dynamic impedance
- Low temperature coefficient

Connection Diagrams

TO-92
Plastic Package



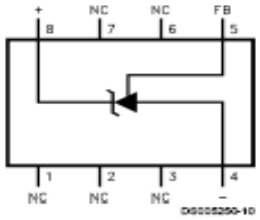
Bottom View

TO-46
Metal Can Package



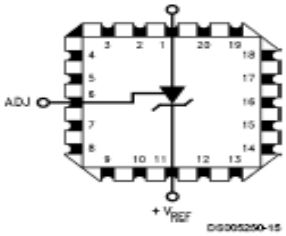
Bottom View

SOIC Package



Top View

20-Leadless Chip Carrier

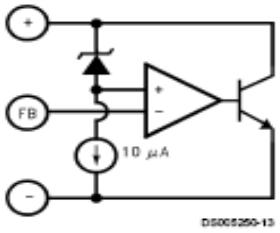


Top View

Ordering Information

Package	Temperature Range			NSC Drawing
	−55°C to 125°C	−40°C to 85°C	0°C to 70°C	
TO-46	LM185BH			H03H
	LM185BH/883			
	LM185BYH			
	LM185BYH/883			
TO-92		LM285BXZ	LM385BXZ	Z03A
		LM285BYZ	LM385BYZ	
		LM285Z	LM385BZ	
			LM385Z	
8-Pin SOIC		LM285M	LM385M	M08A
		LM285BYM	LM385BM	
20-Leadless Chip Carrier	LM185BE/883			E20A

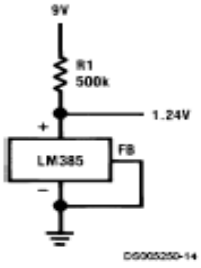
Block Diagram



DS005250-13

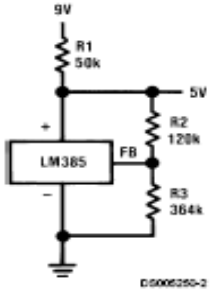
Typical Applications

1.2V Reference



DS005250-14

5.0V Reference



DS005250-2

$$V_{OUT} = 1.24 \left(\frac{R3}{R2} + 1 \right)$$

Anexo 3: Hojas de especificaciones del circuito 74HC590

TOSHIBA TC74HC590AP/AF

TOSHIBA CMOS DIGITAL INTEGRATED CIRCUIT SILICON MONOLITHIC

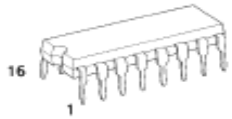
TC74HC590AP, TC74HC590AF

8 – BIT BINARY COUNTER / REGISTER WITH 3 – STATE OUTPUTS

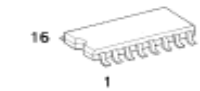
The TC74HC590A is a high speed CMOS 8-BIT COUNTER/REGISTER fabricated with silicon gate C²MOS technology.
It achieves the high speed operation similar to equivalent LSTTL while maintaining the CMOS low power dissipation.
The internal counter counts on the positive going edge of Counter Clock (CCK) when Counter Clock Enable ($\overline{CCKEN}$) is low. When Counter Clear ($\overline{CCLR}$) is low, the internal counter is cleared asynchronously to the clock.
Data in the internal counter are loaded into the register at positive going edge of Register Clock (RCK), and the register outputs are countrolled by enable input ($\overline{G}$).
All inputs are equipped with protection circuits against static discharge or transient excess voltage.

FEATURES :

- High Speed ····· f_{MAX} = 62MHz(typ.) at V_{CC} = 5V
- Low Power Dissipation ····· I_{CC} = 4μA(Max.) at Ta = 25°C
- High Noise Immunity ····· $V_{NIH} = V_{NIL}$ = 28% V_{CC} (Min.)
- Output Drive Capability ····· 15 LSTTL Loads For QA ~ QH
  10 LSTTL Loads For $\overline{RCO}$
- Symmetrical Output Impedance ···
  $|I_{OH}| = I_{OL}$ = 6mA(Min.) For QA ~ QH
  $|I_{OH}| = I_{OL}$ = 4mA(Min.) For $\overline{RCO}$
- Balanced Propagation Delays ····· $t_{pLH} = t_{pHL}$
- Wide Operating Voltage Range ····· VCC (opr.) = 2V ~ 6V
- Pin and Function Compatible with 74LS590

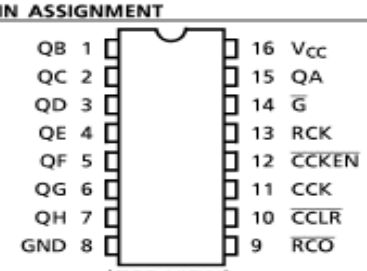


P (DIP16-P-300-2.54A)
Weight : 1.00g (Typ.)



F (SOP16-P-300-1.27)
Weight : 0.18g (Typ.)

PIN ASSIGNMENT



(TOP VIEW)

TRUTH TABLE

INPUT					FUNCTION
G	RCK	$\overline{CCLR}$	$\overline{CCKEN}$	CCK	
H	X	X	X	X	Q OUTPUTS DISABLE
L	X	X	X	X	Q OUTPUTS ENABLE
X	↑	X	X	X	COUNTER DATA IS STORED INTO REGISTER
X	↓	X	X	X	REGISTER STATE IS NOT CHANGED
X	X	L	X	X	COUNTER CLEAR
X	X	H	L	↑	ADVANCE ONE COUNT
X	X	H	L	↓	NO COUNT
X	X	H	H	X	NO COUNT

X : Don't care

$\overline{RCO} = \overline{QA'} \cdot QB' \cdot QC' \cdot QD' \cdot QE' \cdot QF' \cdot QG' \cdot QH'$

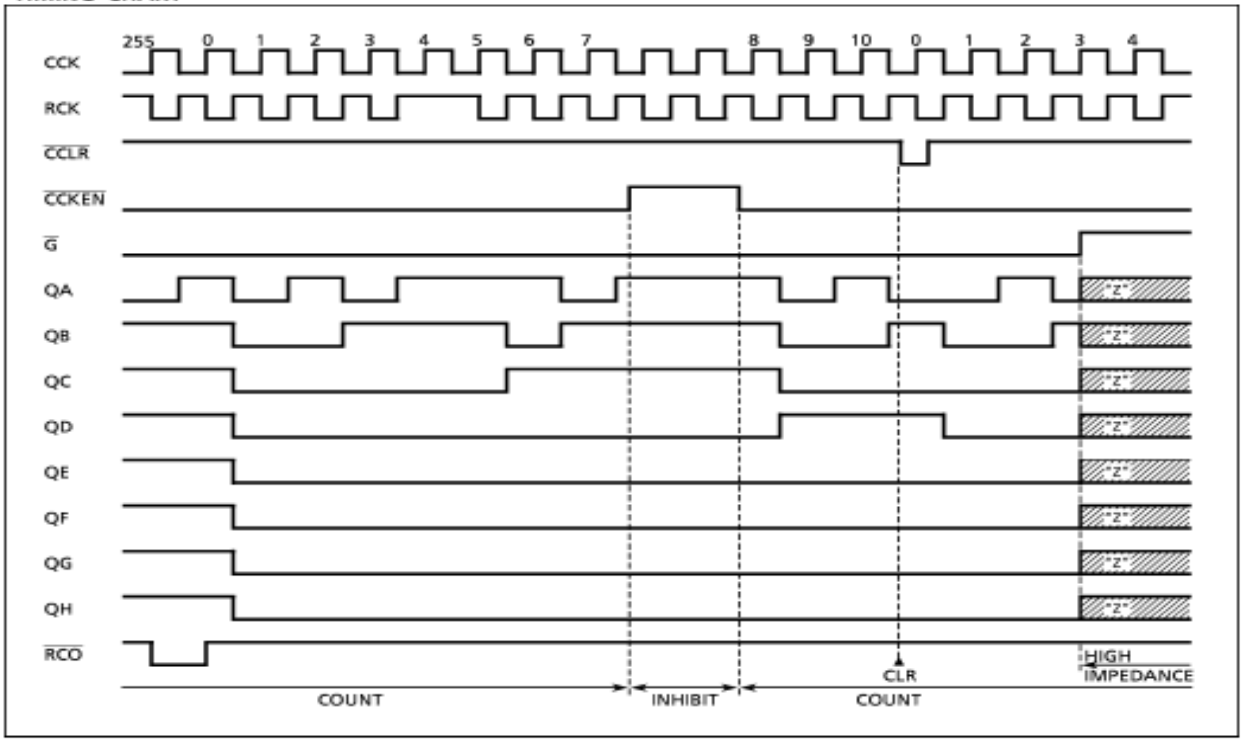
(QA'~QH' : Internal outputs of the counter)

980508EBA2

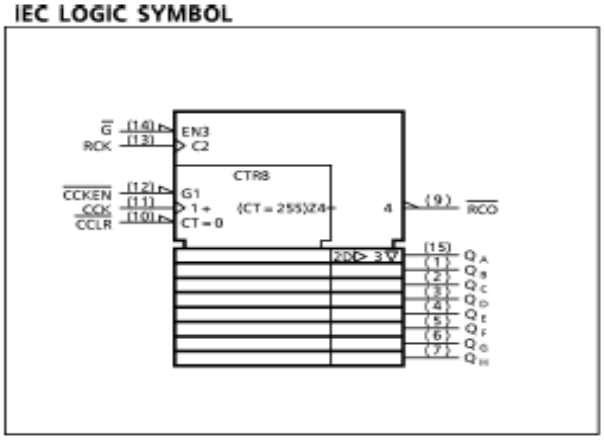
● TOSHIBA is continually working to improve the quality and the reliability of its products. Nevertheless, semiconductor devices in general can malfunction or fail due to their inherent electrical sensitivity and vulnerability to physical stress. It is the responsibility of the buyer, when utilizing TOSHIBA products, to observe standards of safety, and to avoid situations in which a malfunction or failure of a TOSHIBA product could cause loss of human life, bodily injury or damage to property. In developing your designs, please ensure that TOSHIBA products are used within specified operating ranges as set forth in the most recent products specifications. Also, please keep in mind the precautions and conditions set forth in the TOSHIBA Semiconductor Reliability Handbook.

TOSHIBA TC74HC590AP/AF

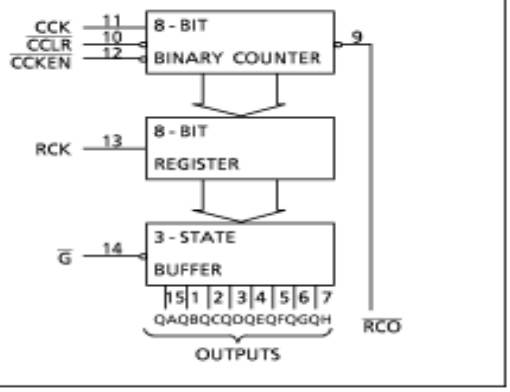
TIMING CHART



IEC LOGIC SYMBOL



BLOCK DIAGRAM



98050BEBA2

- The products described in this document are subject to foreign exchange and foreign trade laws.
- The information contained herein is presented only as a guide for the applications of our products. No responsibility is assumed by TOSHIBA CORPORATION for any infringements of intellectual property or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any intellectual property or other rights of TOSHIBA CORPORATION or others.
- The information contained herein is subject to change without notice.

TOSHIBA TC74HC590AP/AF

ABSOLUTE MAXIMUM RATINGS

PARAMETER	SYMBOL	VALUE	UNIT
Supply Voltage Range	V_{CC}	−0.5~7	V
DC Input Voltage	V_{IN}	−0.5~V_{CC}+0.5	V
DC Output Voltage	V_{OUT}	−0.5~V_{CC}+0.5	V
Input Diode Current	I_{IK}	±20	mA
Output Diode Current	I_{OK}	±20	mA
DC Output Current (RCO) (Q_A~Q_H)	I_{OUT}	±25 ±35	mA
DC V_{CC}/Ground Current	I_{CC}	±75	mA
Power Dissipation	P_D	500 (DIP)* / 180 (SOP)	mW
Storage Temperature	T_{stg}	−65~150	°C

*500mW in the range of Ta= −40°C~65°C. From Ta=65°C to 85°C a derating factor of −10mW/°C shall be applied until 300mW.

RECOMMENDED OPERATING CONDITIONS

PARAMETER	SYMBOL	VALUE	UNIT
Supply Voltage	V_{CC}	2~6	V
Input Voltage	V_{IN}	0~V_{CC}	V
Output Voltage	V_{OUT}	0~V_{CC}	V
Operating Temperature	T_{opr}	−40~85	°C
Input Rise and Fall Time	t_r, t_f	0~1000 (V_{CC}=2.0V) 0~500 (V_{CC}=4.5V) 0~400 (V_{CC}=6.0V)	ns

Anexo 4: Hoja de especificaciones del circuito 74LS90

MOTOROLA

DECADE COUNTER; DIVIDE-BY-TWELVE COUNTER; 4-BIT BINARY COUNTER

SN54/74LS90
SN54/74LS92
SN54/74LS93

DECADE COUNTER;
DIVIDE-BY-TWELVE COUNTER;
4-BIT BINARY COUNTER

LOW POWER SCHOTTKY

The SN54/74LS90, SN54/74LS92 and SN54/74LS93 are high-speed 4-bit ripple type counters partitioned into two sections. Each counter has a divide-by-two section and either a divide-by-five (LS90), divide-by-six (LS92) or divide-by-eight (LS93) section which are triggered by a HIGH-to-LOW transition on the clock inputs. Each section can be used separately or tied together (Q to CP) to form BCD, bi-quinary, modulo-12, or modulo-16 counters. All of the counters have a 2-input gated Master Reset (Clear), and the LS90 also has a 2-input gated Master Set (Preset 9).

- Low Power Consumption . . . Typically 45 mW
- High Count Rates . . . Typically 42 MHz
- Choice of Counting Modes . . . BCD, Bi-Quinary, Divide-by-Twelve, Binary
- Input Clamp Diodes Limit High Speed Termination Effects



J SUFFIX
CERAMIC
CASE 632-08



N SUFFIX
PLASTIC
CASE 646-06



D SUFFIX
SOIC
CASE 751A-02

ORDERING INFORMATION

SN54LSXXJ	Ceramic
SN74LSXXN	Plastic
SN74LSXXD	SOIC

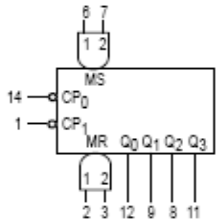
PIN NAMES

		LOADING (Note a) HIGH	LOW
$\overline{CP_0}$	Clock (Active LOW going edge) Input to ÷2 Section	0.5 U.L.	1.5 U.L.
$\overline{CP_1}$	Clock (Active LOW going edge) Input to ÷5 Section (LS90), ÷6 Section (LS92)	0.5 U.L.	2.0 U.L.
$\overline{CP_1}$	Clock (Active LOW going edge) Input to ÷8 Section (LS93)	0.5 U.L.	1.0 U.L.
MR_1, MR_2	Master Reset (Clear) Inputs	0.5 U.L.	0.25 U.L.
MS_1, MS_2	Master Set (Preset-9, LS90) Inputs	0.5 U.L.	0.25 U.L.
Q_0	Output from ÷2 Section (Notes b & c)	10 U.L.	5 (2.5) U.L.
Q_1, Q_2, Q_3	Outputs from ÷5 (LS90), ÷6 (LS92), ÷8 (LS93) Sections (Note b)	10 U.L.	5 (2.5) U.L.

NOTES:
a. 1 TTL Unit Load (U.L.) = 40 µA HIGH/1.6 mA LOW.
b. The Output LOW drive factor is 2.5 U.L. for Military, (54) and 5 U.L. for commercial (74) Temperature Ranges.
c. The Q_0 Outputs are guaranteed to drive the full fan-out plus the $\overline{CP_1}$ input of the device.
d. To insure proper operation the rise (t_r) and fall time (t_f) of the clock must be less than 100 ns.

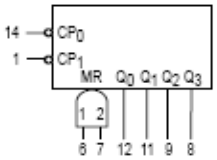
LOGIC SYMBOL

LS90



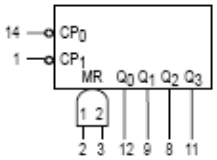
V_{CC} = PIN 5
GND = PIN 10
NC = PINS 4, 13

LS92



V_{CC} = PIN 5
GND = PIN 10
NC = PINS 2, 3, 4, 13

LS93



V_{CC} = PIN 5
GND = PIN 10
NC = PIN 4, 6, 7, 13

Anexo 5: Hoja de especificaciones del circuito 74S139

FAIRCHILD
SEMICONDUCTOR™

August 1986
Revised April 2000

DM74S138 • DM74S139
Decoder/Demultiplexer

General Description

These Schottky-clamped circuits are designed to be used in high-performance memory-decoding or data-routing applications, requiring very short propagation delay times. In high-performance memory systems these decoders can be used to minimize the effects of system decoding. When used with high-speed memories, the delay times of these decoders are usually less than the typical access time of the memory. This means that the effective system delay introduced by the decoder is negligible.

The DM74S138 decodes one-of-eight lines, based upon the conditions at the three binary select inputs and the three enable inputs. Two active-LOW and one active-HIGH enable inputs reduce the need for external gates or inverters when expanding. A 24-line decoder can be implemented with no external inverters, and a 32-line decoder requires only one inverter. An enable input can be used as a data input for demultiplexing applications.

The DM74S139 comprises two separate two-line-to-four-line decoders in a single package. The active-LOW enable input can be used as a data line in demultiplexing applications.

All of these decoders/demultiplexers feature fully buffered inputs, presenting only one normalized load to its driving circuit. All inputs are clamped with high-performance Schottky diodes to suppress line-ringing and simplify system design.

Features

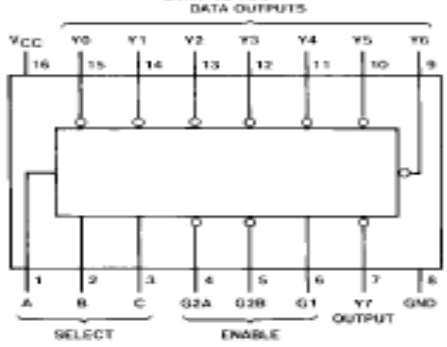
- Designed specifically for high speed:
  - Memory decoders
  - Data transmission systems
- DM74S138 3-to-8-line decoders incorporates 3 enable inputs to simplify cascading and/or data reception
- DM74S139 contains two fully independent 2-to-4-line decoders/demultiplexers
- Schottky clamped for high performance
- Typical propagation delay time (3 levels of logic)
  - DM74S138 8 ns
  - DM74S139 7.5 ns
- Typical power dissipation
  - DM74S138 245 mW
  - DM74S139 300 mW

Ordering Code:

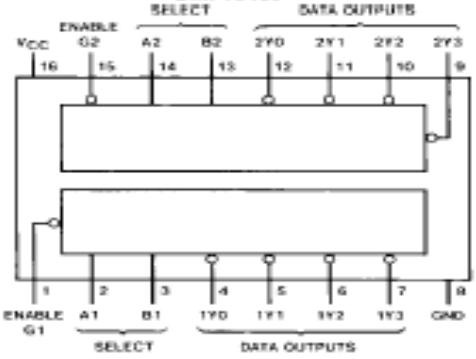
Order Number	Package Number	Package Description
DM74S138N	N16E	16-Lead Plastic Dual-In-Line Package (PDIP), JEDEC MS-001, 0.300 Wide
DM74S139N	N16E	16-Lead Plastic Dual-In-Line Package (PDIP), JEDEC MS-001, 0.300 Wide

Connection Diagrams

DM74S138



DM74S139



Function Tables

DM74S138

Inputs					Outputs							
Enable		Select										
G1	G2*	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

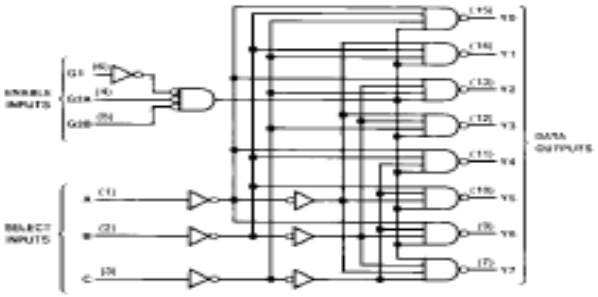
DM74S139

Inputs			Outputs			
Enable	Select					
G	B	A	Y0	Y1	Y2	Y3
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

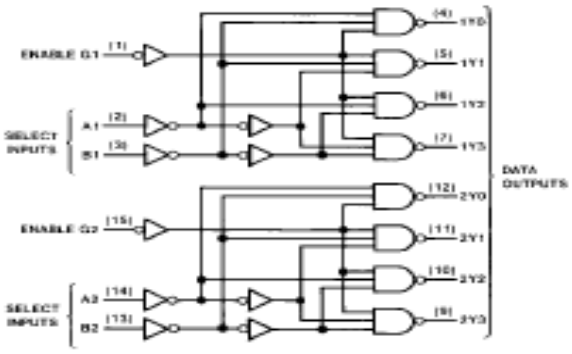
* G2 = G2A + G2B
H = HIGH level
L = LOW level
X = don't care (either LOW or HIGH logic level)

Logic Diagrams

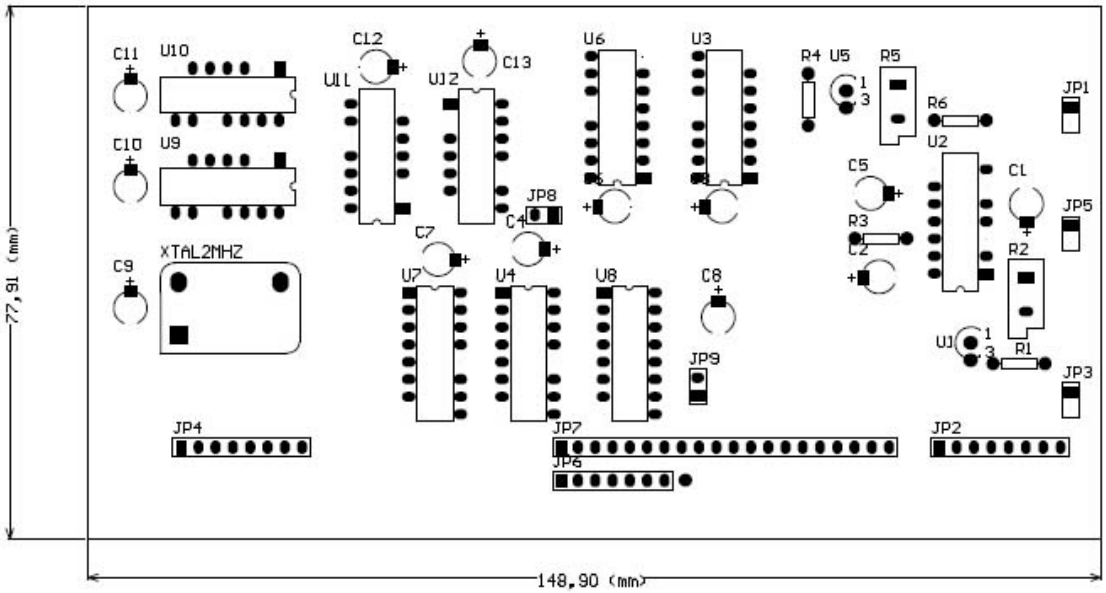
DM74S138



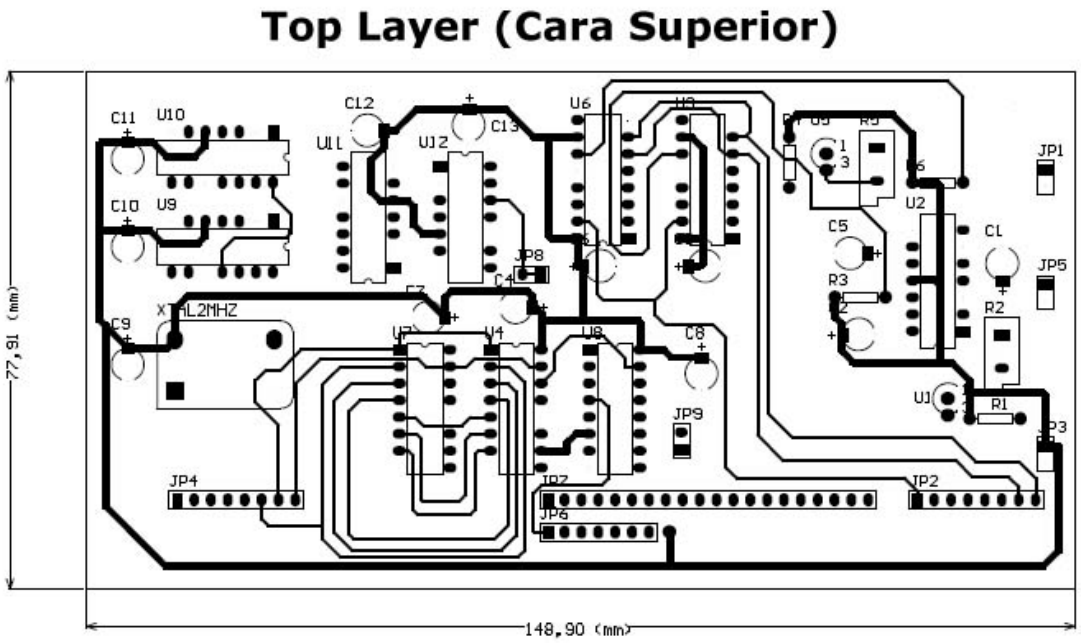
DM74S139



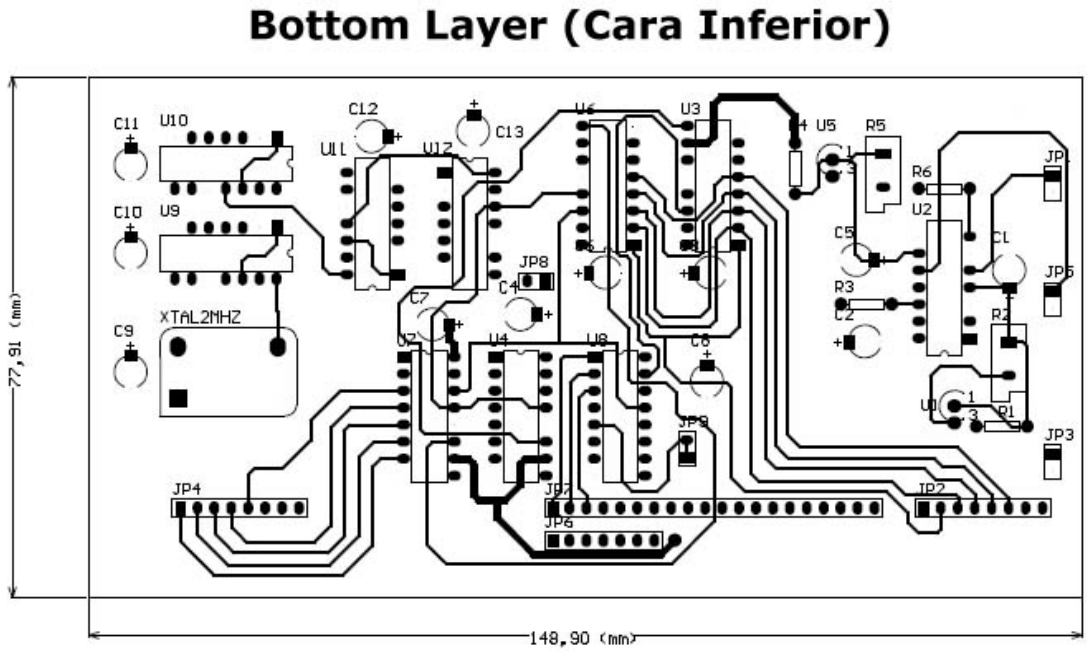
Circuito Impreso



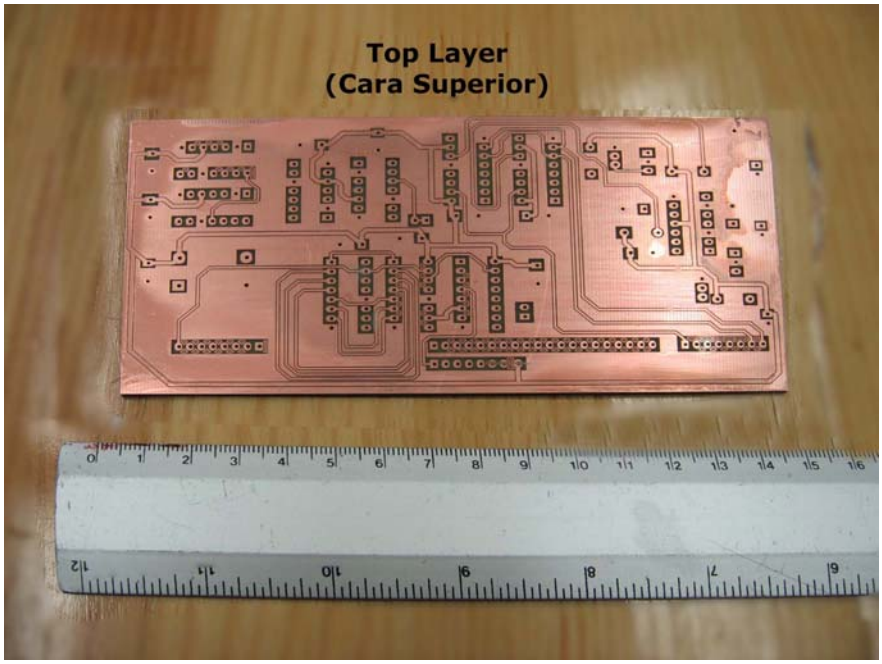
Anexo 6: Distribución de los componentes de la Electrónica de Adquisición



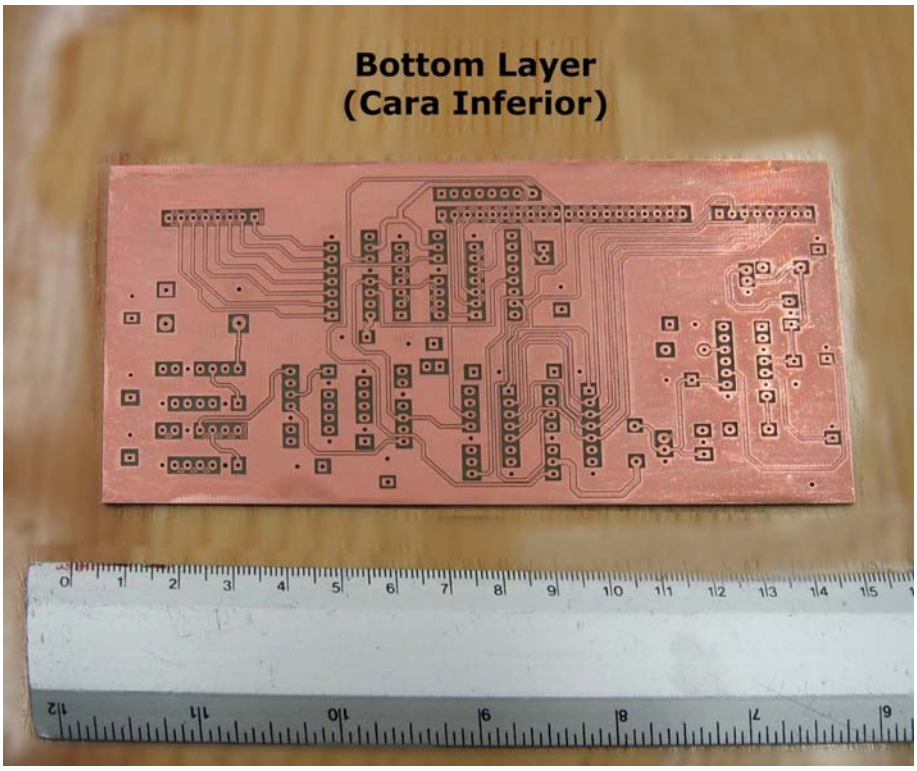
Anexo 7: Trayectoria de las pistas en la cara superior



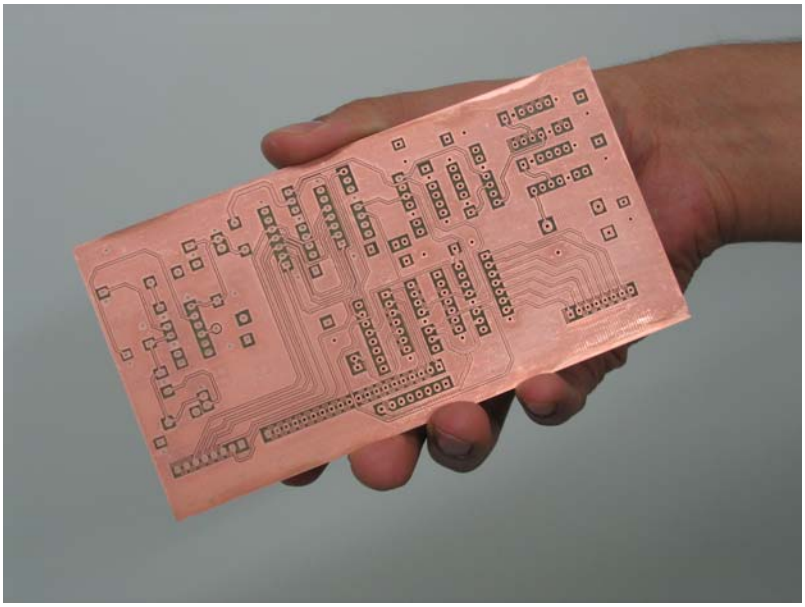
Anexo 8: Trayectoria de las pistas en la cara inferior



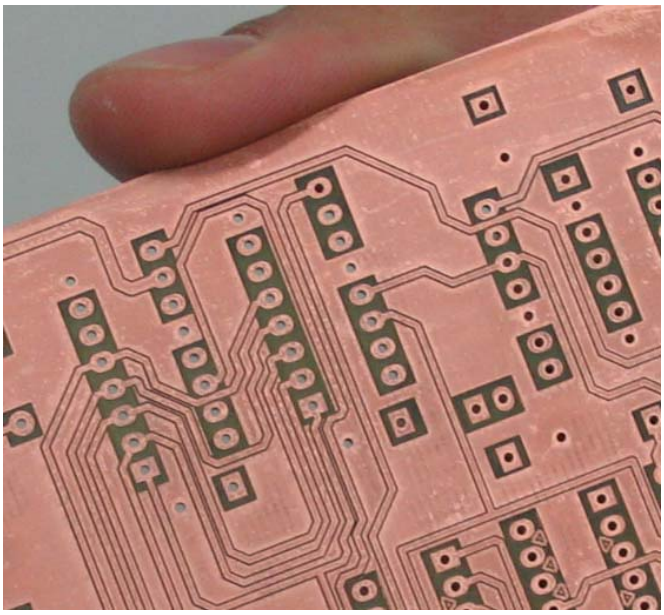
Anexo 9: circuito impreso final cara superior



Anexo 10: circuito impreso final cara inferior



Anexo 11: Comparación de la Tarjeta



Anexo 12: Zona con dificultad para soldar

Código Fuente

Anexo 13: Código fuente de la Interfaz de usuario

```
unit UnitMain;

interface

uses
  SysUtils, Types, Classes, Variants, QTypes, QGraphics, QControls, QForms,
  QDialogs, QStdCtrls, QButtons, LibC, QComCtrls, QExtCtrls, QMenus, QMask,
  UnitGlobal, UnitAcq, UnitNotes;

Const IRQBase = $378;
      FilterSet : Array[1..4] of Char = ('O','U','V','B'); // Arreglo de Caracteres Para el Metodo Filter
type
  TFormMain = class(TForm)                  /** TODAS ESTAS VARIABLES SON CREADAS POR EL ENTORNO DE PROGRAMACION */
    BtnStart: TbitBtn;                      /** Y SON REFERENTES A LAS VENTANAS, BOTONES, CASILLAS DE TEXTO, ETC,      */
    BtnStop: TBitBtn;                       /** DEL AMBIENTE GRÁFICO       */
    EditCanalA: TEdit;
    EditCanalB: TEdit;
    LabelA: TLabel;
    LabelB: TLabel;
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    SetFileName1: TMenuItem;
    Help1: TMenuItem;
    About1: TMenuItem;
    Help2: TMenuItem;
    SaveDlg: TSaveDialog;
    LabelIntTime: TLabel;
    EditIntTime: TEdit;
    PanelGraph: TPanel;
    BtnFilter: TBitBtn;
    LabelFilter: TLabel;
    PanelGraphA: TPanel;
    PBGraphA: TPaintBox;
    EditFileName: TEdit;
    PanelGraphB: TPanel;
    PBGraphB: TPaintBox;
    LabelChanA: TLabel;
    LabelChanB: TLabel;
    N1: TMenuItem;
    Exit1: TMenuItem;
    PanelDiaf: TPanel;
    BtnDiaf: TBitBtn;
    LabelDateMsg: TLabel;
    LabelTimeMsg: TLabel;
    LabelDiaph: TLabel;
    SpinEdit1: TSpinEdit;
    SpinEdit2: TSpinEdit;
    SpinEdit3: TSpinEdit;
    SpinEdit4: TSpinEdit;
    LabelOffA: TLabel;
    LabelDivA: TLabel;
    LabelOffB: TLabel;
    LabelDivB: TLabel;
```

```
    LabelTime: TLabel;
    LabelDate: TLabel;
    BtnNotes: TBitBtn;
    Notes1: TMenuItem;
    Label1: TLabel;
    EditCount: TEdit;
    procedure BtnStartClick(Sender: TObject);
    procedure BtnStopClick(Sender: TObject);
    procedure SetFileName1Click(Sender: TObject);
    procedure BtnFilterClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure BtnDiafClick(Sender: TObject);
    procedure BtnNotesClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private declarations }
  public
    { Public declarations }
    FileReady : Boolean;  // Bandera nombre de archivo listo
    FilterIdx : Integer;  // Indice para el Arreglo de Filter
    DiaphIdx  : Integer;  // Indice para el Arreglo de Diaphragm
    BMA       : TBitMap;  // PaintBox Canal A
    BMB       : TBitMap;  // PaintBox Canal B
    IntTime   : Real;     // Tiempo de Integracion
    StartDate : TDateTime;// Fecha de Inicio
    StartTime : TDateTime;// Tiempo de Inicio
    Acqr      : TAcquire; // Hilo de Adquisición

    // Metodo para escribir en el puerto para generar la interrupción
    Procedure SetPortB(Port:Word; Value:Byte);
    Procedure SaveToFile; // Metodo para salvar el Archivo
  end;

var
  FormMain: TFormMain;

implementation

{$R *.xfm}

Procedure TFormMain.SetPortB(Port:Word; Value:Byte); // Metodo para escribir al puerto
Begin
   asm
      PUSH  AX         // Salva registros AX y DX en stack
      PUSH  DX         //
      MOV   DX,Port    // Mueve la palabra
      MOV   AL,Value   //
      OUT   DX,AL      // Saca los bits al Puerto
      POP   DX         // Recupera registros AX y DX del stack
      POP   AX         //
   end;
End;

procedure TFormMain.BtnStartClick(Sender: TObject); // Metodo al activar el botón Start

Var   I,J,
//      Yg,
      GPrev,GIdx,
      Delay   : Integer;
begin
   IntTime:=StrToFloat(EditIntTime.Text);
   Divisor:=Trunc(1000*IntTime); // no de ciclos a integrar
```

```
  //retraso en la atención a la GUI dependiente del tiempo de integración
  If IntTime<0.02 Then
    Delay:=20
  Else
    Delay:=1;

  AssignFile(F,'/dev/fotometro'); // Abre el Buffer Fotometro
  Reset(F);

  // Activa la interrupción
  ioperm(IRQBase,4,1);
  SetPortB(IRQBase+2,$10);

  Idx:=0;                // indice global de datos
  Cancel:=False;
  GPrev:=MaxData-512;
// Lee fecha y hora del sistema;
  StartDate:=Date;
  StartTime:=Time;
// Arranca el THREAD de adquisición
  Acqr.Resume;
  Repeat
// ciclo de graficado
    GIdx:=Idx-512;       // indice de graficacion(inicio de ventana)
    If GIdx<0 Then
      GIdx:=MaxData+GIdx;// Verifica rango de operación del indice gráfico
    If GIdx>MaxData Then
      GIdx:=GIdx-MaxData;

    If GIdx<>GPrev Then  // Si hay datos nuevos, refresca gráficas
      Begin
        BMA.Canvas.Rectangle(0,0,512,128);    // Borra los canvas
        BMB.Canvas.Rectangle(0,0,512,128);
        I:=GIdx;                          // Indice de graficado de dato
        For J:=1 to 512 Do
          Begin
            BMA.Canvas.Pixels[J,128-DataA[I] shr 9]:=clBlack; // Print canal A
            BMB.Canvas.Pixels[J,128-DataB[I] shr 9]:=clBlack; // Print canal B

            Inc(I);
            If I>MaxData Then
              I:=1;
          End;
        PBGraphA.Canvas.Draw(0,0,BMA);       // Muestra la imagen virtual sobre la pantalla
        PBGraphB.Canvas.Draw(0,0,BMB);       // Muestra el bitmap sobre el Canvas

        EditCanalA.Text:=IntToStr(DataA[Idx-1]);  // Muestra el dato anterior al que se encuentra el Indice CanA
        EditCanalB.Text:=IntToStr(DataB[Idx-1]);  // Muestra el dato anterior al que se encuentra el Indice CanB
        EditCount.Text:=IntToStr(Idx);            // Muestra el Indice en el editor Count

        GPrev:=GIdx;
      End;
    If Idx mod Delay = 0 Then  // Procesa eventos de la GUI
      Application.ProcessMessages;
  Until Cancel;                // Hasta que el usuario cancele
  Acqr.Suspend;                // Suspende el THREAD de lectura
  CloseFile(F);                // Cierra el dispositivo FOTOMETRO
  SaveToFile;                  // Guarda los datos en el archivo
end;

procedure TFormMain.BtnStopClick(Sender: TObject); // Metodo del Boton Stop
begin
  SetPortB(IRQBase+2,$0); // Deshabilita interrupciones
```

```
  Cancel:=True;
end;

procedure TFormMain.SetFileName1Click(Sender: TObject); // Metodo Asignar Nombre al Archivo
begin
  FileReady:=False;
  If SaveDlg.Execute Then
    Begin
      FileReady:=True;
      EditFileName.Text:=SaveDlg.FileName;
    End;
  BtnStart.Enabled:=FileReady;
end;

// Cambia la etiqueta del filtro
procedure TFormMain.BtnFilterClick(Sender: TObject); // Metodo Filter
begin
  Inc(FilterIdx);
  If FilterIdx>4 Then
    FilterIdx:=1;
  LabelFilter.Caption:=FilterSet[FilterIdx];
end;

// Preparación de variable durante la Creación de La Forma
procedure TFormMain.FormCreate(Sender: TObject);
begin
  FilterIdx:=1;
  DiaphIdx:=1;
  FileReady:=False;


  //Crea las Bitmap
  BMA:=TBitMap.Create;
  BMA.Height:=128;
  BMA.Width:=512;
  BMA.PixelFormat:=pf32Bit;

  BMB:=TBitMap.Create;
  BMB.Height:=128;
  BMB.Width:=512;
  BMB.PixelFormat:=pf32Bit;

  // Actualiza la hora y la fecha desde el sistema
  LabelDate.Caption:=DateToStr(Date);
  LabelTime.Caption:=TimeToStr(Time);

  // Crea el hilo de ejecución de lectura
  Acqr:=TAcquire.Create(True);
end;

procedure TFormMain.BtnDiafClick(Sender: TObject);  // Metodo Diaphragm
begin
  Inc(DiaphIdx);
  If DiaphIdx>7 Then
    DiaphIdx:=1;
  LabelDiaph.Caption:=IntToStr(DiaphIdx);
end;

procedure TFormMain.BtnNotesClick(Sender: TObject); // Metodo Notes
begin
  FormNotes.Show;
end;
```

```
Procedure TFormMain.SaveToFile;  // Metodo Salva Archivo

Var   DataFile : TextFile;
      I        : Integer;

Begin
   If MessageDlg('Save Data to File '+SaveDlg.FileName+'?',mtConfirmation,[mbOK,mbCancel],0)=mrOK Then
      Begin  // Si el Archivo es Guardado, Guarda los Datos siguientes
        AssignFile(DataFile,SaveDlg.FileName);
        Rewrite(DataFile);
        WriteLn(DataFile,'Integration Time:   ',EditIntTime.Text);
        WriteLn(DataFile,'Date:               ',DateToStr(StartDate));
        WriteLn(DataFile,'Time:               ',TimeToStr(StartTime));
        WriteLn(DataFile,'Diaphragm:          ',DiaphIdx);
        WriteLn(DataFile,'Filter:             ',FilterSet[FilterIdx]);

        For I:=0 to FormNotes.MemoNotes.Lines.Count-1 Do   //Recorre las lineas
          WriteLn(DataFile,'> ',FormNotes.MemoNotes.Lines[I]);

        For I:=1 to Idx Do
          WriteLn(DataFile,I:10,DataA[I]:10,DataB[I]:10);
        CloseFile(DataFile);

        EditFileName.Text:='<undefined file>';
        BtnStart.Enabled:=False;
        FileReady:=False;
      End;
End;

procedure TFormMain.FormClose(Sender: TObject; var Action: TCloseAction);
begin
   Acqr.Destroy; //Se Destruye el Hilo
end;

end.
```

```
// ************El código siguiente pertenece a la Unidad del Hilo

unit UnitAcq;

interface

uses
  Classes, SysUtils, UnitGlobal;

type
  TAcquire = class(TThread)
  private
    { Private declarations }
  protected
    procedure Execute; override;
  end;

implementation

{ TAcquire }

procedure TAcquire.Execute;

Var   I,
      SumaA,SumaB : Integer;
      CanalA,
```

```
    CanalB,
    AbsA,AbsB,
    PrevCanalA,
    PrevCanalB    : Word;
    ARdy, BRdy : Boolean;
    Defined : Boolean;

begin
  Defined:=False;
  PrevCanalA:=0;
  PrevCanalB:=0;
  While not Cancel and not Terminated Do
    Begin
      SumaA:=0;
      SumaB:=0;
      For I:=0 to Divisor-1 Do
        Begin
          CanalA:=0;
          CanalB:=0;

          ARdy:=False;
          BRdy:=False;

          Repeat
            Repeat
              ReadLn(F,Res);
            Until Res<>'';
            Case Res[1] of
              '1': Begin
                     CanalA:=StrToInt(Copy(Res,3,5));
                     ARdy:=True;
                   End;
              '2': Begin
                     CanalB:=StrToInt(Copy(Res,3,5));
                     BRdy:=True;
                   End;
            End;
          Until ARdy and BRdy;
          If Defined Then
            Begin
              AbsA:=CanalA-PrevCanalA;
              AbsB:=CanalB-PrevCanalB;
              SumaA:=SumaA+AbsA;
              SumaB:=SumaB+AbsB;
            End;
          PrevCanalA:=CanalA;
          PrevCanalB:=CanalB;
          Defined:=True;
        End;

      Inc(Idx);
      If Idx>MaxData Then
        Idx:=1;

      DataA[Idx]:=SumaA;
      DataB[Idx]:=SumaB;
    End;
end;

end.
```

```
// ************El Codigo siguiente pertenece a la Unidad Global
```

```
unit UnitGlobal;

interface

Const MaxData = 500000;

Var
      DataA     : Array[1..MaxData] of LongWord;
      DataB     : Array[1..MaxData] of LongWord;
      F         : TextFile;
      Cancel    : Boolean;
      Divisor   : Integer;
      Idx       : Integer;
      Res       : String;

implementation

end.
```

```
// ************El Codigo siguiente pertenece a la Unidad Notes
unit UnitNotes;

interface

uses
  SysUtils, Types, Classes, Variants, QTypes, QGraphics, QControls, QForms,
  QDialogs, QStdCtrls;

type
  TFormNotes = class(TForm)
    MemoNotes: TMemo;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  FormNotes: TFormNotes;

implementation

{$R *.xfm}

end.
```