Implementación y optimización de una biblioteca embebida para receptor GPS

R.Ghignone (*rghignone@frh.utn.edu.ar*), I. Castellucci Vidal (*icv@frh.utn.edu.ar)*, J.Giampetruzzi (*jgiampetruzzi@frh.utn.edu.ar)*, F. Larosa (*flarosa@frh.utn.edu.ar)*
Universidad Tecnológica Nacional - Facultad Regional Haedo

***Abstract* - El presente trabajo describe la implementación, validación y optimización de una biblioteca embebida para calcular la posición de usuario a partir de datos de la constelación GPS. Dicha librería fue escrita en lenguaje C sobre la plataforma de desarrollo EduCIAA. El algoritmo fue validado utilizando los datos de entrada provistos por un receptor GPS comercial; los errores promedio obtenidos fueron consistentes con los de la especificación del sistema GPS. Adicionalmente, se estudian soluciones para optimizar los tiempos de ejecución del algoritmo, mediante la implementación de las funciones en lenguaje ensamblador nativo del procesador ARM Cortex-M4F.**

***Índice de términos*— GPS, CIAA, algoritmo, posicionamiento, sistema embebido, assembly, Cortex M4F**

I. INTRODUCCIÓN

EL Sistema de Posicionamiento Global (*Global Positioning System*, GPS por sus siglas en inglés) es un sistema de posicionamiento creado por el Gobierno de los Estados Unidos de América (EUA) y mantenido por la Fuerza Aérea de los Estados Unidos (*United States Air Force*, USAF por sus siglas en inglés). Si bien su propósito principal fue proveer soporte a las actividades militares de los EUA, hoy en día es destacable su importancia para diversas actividades de la población civil [1].

El objetivo de este trabajo es diseñar, validar y optimizar una biblioteca embebida que implementa algoritmos [2] para el cálculo de la posición de usuario a partir de los datos de navegación y observables de la constelación GPS [1]. Esta biblioteca será aplicada en la implementación de un receptor GPS definido por software. Este tipo de receptores poseen una arquitectura más flexible que los receptores convencionales [3], lo cual permite entre otras cosas realizar mejoras progresivas en el diseño del mismo, agregar funcionalidades y facilitar la integración con módulos o plataformas a desarrollar a futuro.

A fin de validar la biblioteca se utilizó como fuente de datos un receptor comercial NEO-6M de la empresa UBlox y una placa EDU-CIAA-NXP, la cual tiene como núcleo un microcontrolador LPC4337 de la empresa NXP Semiconductors, basado en un procesador ARM Cortex-M4F como núcleo principal.

II. ARQUITECTURA GENERAL DEL SOFTWARE

Teniendo en cuenta que a futuro se utilizará como fuente de datos un receptor GPS de desarrollo propio, se estructuró el software con vistas a abstraer la operación de la biblioteca del hardware utilizado.
El software embebido consta de cuatro capas claramente definidas: el sistema operativo (SO), la capa de mediación, la capa de procesamiento y la capa de abstracción de hardware (HAL, por sus siglas en inglés, *Hardware Abstraction Layer*). El sistema operativo se encarga de ejecutar las tareas que comandan la toma de datos del receptor GPS y su procesamiento. La capa de mediación responde a un patrón de software comúnmente llamado "mediador" [4] que permite abstraer al sistema operativo de las interfaces y operaciones involucradas en la toma de datos y el procesamiento, y simplemente presenta la solución de usuario. La biblioteca GPSLIB recibe los datos de navegación y observables en un formato independiente del hardware utilizado y calcula la posición del usuario. Finalmente, la HAL provee datos del receptor UBLOX a través de un buffer abstrayendo completamente a las funciones de mayor nivel de las particularidades del receptor empleado.

Los módulos que componen la biblioteca embebida y las relaciones entre ellos se muestran de forma general en la Figura 1. Cada módulo está escrito en lenguaje C y comprende un archivo .h y un archivo .c. A continuación, describiremos brevemente los distintos módulos, variables y funciones necesarios para el cálculo de la posición de usuario.

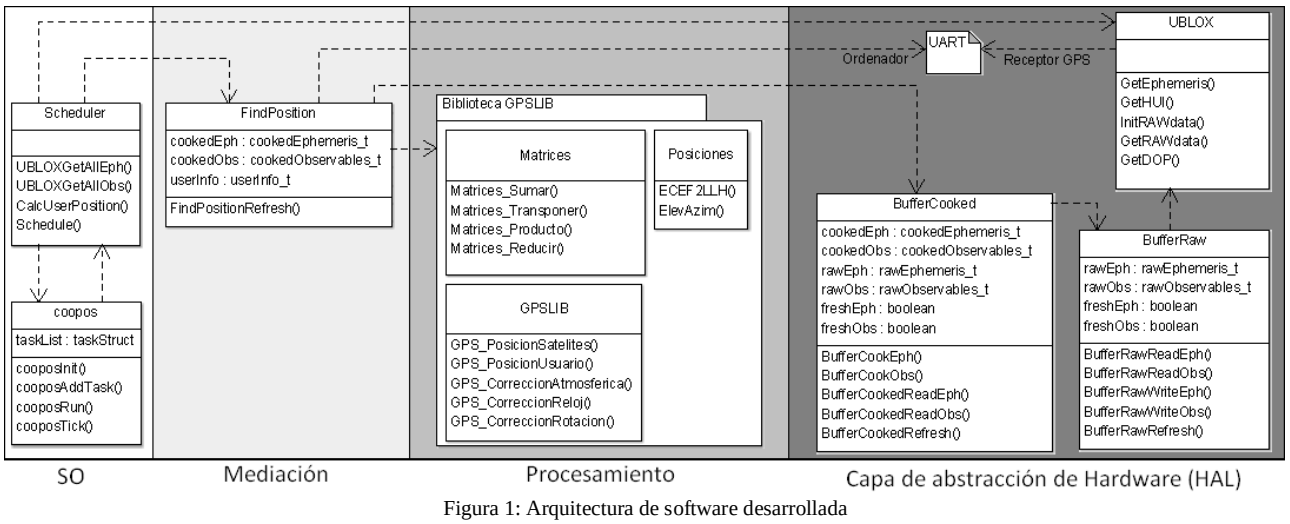

Figura 1: Arquitectura de software desarrollada

A. UBLOX

Éste módulo contiene las funciones para inicializar el receptor GPS externo y obtener de él los datos de entrada a la librería, implementando una interfaz para el protocolo UBX [5] del dispositivo. Dicho dispositivo está conectado a la EDU-CIAA mediante una de las cuatro UARTs que dispone la placa. Los drivers que manejan las UARTs del LPC4337 se hallan en el módulo UART. A continuación, se describen los tipos de datos y funciones implementadas:

- *GetEphemeris() :*Recibe las efemérides y factores de corrección que posibilitan calcular la posición de los satélites. Estos datos están contenidos en las palabras 3 a 10 de los subframes 1 a 3 del mensaje de navegación [1] para cada satélite.

- *GetRAWData() :*Obtiene las mediciones, previa habilitación mediante la función *ActivarRAWData()*, correspondientes a: distancia a cada satélite (pseudorango), fase de portadora, efecto Doppler y relación señal/ruido.

- *GetHUI():*Obtiene los datos HUI (*Health, Utc, Ionospheric*), es decir, salud (*health*) de los satélites, los factores de tiempo UTC (Coordinated Universal Time) y los parámetros ionosféricos para el modelo de Klobuchar [6] que permiten corregir los pseudorangos debido al retardo ionosférico.

- *GetDOP():* Obtiene las llamadas "diluciones de precisión" (*Dilution of Precision*, DOP por las siglas en inglés), que permiten estimar el error cometido debido a la distribución geométrica de los satélites respecto al receptor [7].

B. BufferRaw

Este módulo implementa dos buffers (*rawEph[],rawObs[])* tipo FIFO (*First In – FirstOut*), en forma de colas circulares, para almacenar los juegos de efemérides y de observables (llamaremos así al conjunto de mediciones, DOP y datos HUI) que se van obteniendo del receptor. Estos buffers permiten adaptar las velocidades entre el extremo productor (receptor GPS) y el consumidor (biblioteca de procesamiento GPSLIB). Los tipos de datos *rawEphemeris_t* y *rawObservables_t* están definidos en la cabecera *GPSPositionLibrary.h*, incluida en todos los módulos. A continuación, se describen los tipos de datos y funciones implementadas:

- *freshEph, freshObs:* Estas variables de estado indican la presencia o no de juegos de efemérides u observables listos para ser procesados en la fase siguiente.

- *BufferRawReadEph(),BufferRawReadObs():*Estas funciones agregan un nuevo juego de datos al buffer correspondiente, y actualizan las variables de estado.

- *BufferRawWriteEph(),BufferRawWriteObs():*Estas funciones copian el elemento a la cabeza del buffer a un destino especificado y actualizan las variables de estado.

- *BufferRawRefresh()*: Esta función permite actualizar los buffers del módulo de forma permanente e independiente al resto del sistema, ante la presencia de nuevos datos del receptor.

C. BufferCooked

El propósito de éste módulo es transformar los datos enviados por el receptor (*rawEph, rawObs)* del formato en que se encuentran (dependiente del receptor GPS utilizado) al formato en que se utilizan en el algoritmo final de cálculo (*cookedEph, cookedObs)*.A continuación, se describen los tipos de datos y funciones implementadas:

- *BufferCookEph() :* Ésta función se encarga de seccionar la trama de datos en formato binario proveniente del receptor GPS, obtener las efemérides en formato de punto fijo y aplicar las constantes de escala definidas por el estándar [1] a fin de obtener las efemérides en punto flotante que servirán como entrada para la biblioteca GPSLIB.

- *BufferCookObs() :* Esta función *c*umple el mismo propósito que la función anterior para los observables.

- *BufferCookedReadEph(), BufferCookedReadObs() :* Estas funciones copian los datos formateados (*cookedEph*o *cookedObs)* a una dirección especificada como argumento.

- *BufferCookedRefresh() :* Esta función actualiza el estado del buffer, recurriendo a las funciones *BufferRawWriteEph*() y *BufferRawWriteObs()* para obtener nuevos datos y a *BufferCook* para transformarlos al formato de destino.

D. FindPosition

Este módulo recibe los datos de entrada provistos por *BufferCooked* y lleva a cabo el algoritmo matemático para obtener la posición del usuario. Para ello recurre a la biblioteca GPSLIB [2], que implementa los pasos matemáticos del algoritmo. A continuación, se describen los tipos de datos y funciones implementadas:

- *userInfo_t:* Esta estructura reúne la información de la posición obtenida y los cálculos realizados.

- *FindPositionRefresh():* Esta función implementa el algoritmo matemático para obtener la posición del usuario. Adicionalmente, envía estos resultados por conexión USB a través de una de las UARTs. Así, a través de una computadora conectada a la placa podemos analizar el comportamiento de la librería (tiempo de ejecución, errores, dispersión de las mediciones, debugging, etc.).

E. coopos

Éste módulo implementa un sistema operativo cooperativo el cual soporta la ejecución de una cantidad N tareas temporizadas (N se define en tiempo de compilación).

- *taskList:* Es un vector de punteros a función que almacena la dirección de inicio de cada tarea.

- *cooposInit() :* Esta función inicializa el módulo y la lista de tareas *taskList[]*.

- *cooposAddTask() :* Esta función agrega una nueva tarea a la lista, el período de ejecución de la misma, el desplazamiento inicial y la cantidad de veces que se ejecuta la tarea o si se ejecuta indefinidamente.

- *cooposRun():* Esta función implementa el bucle principal del programa, espera las interrupciones del procesador para ejecutar las tareas asignadas.

- *cooposTick():* Esta función se encarga de que en cada interrupción del timer *SysTick* del procesador se actualice el contador de tareas y se las ejecute, si corresponde.

F. Scheduler

Este módulo agrega las tareas a la lista correspondiente, a través de la función *cooposAddTask()*. En nuestra configuración experimental, las tareas están dadas por las siguientes funciones:

- *UBLOXGetAllEph():*Llama a *GetEphemeris()* para obtener las efemérides de todos los satélites y las almacena en *BufferRaw.*

-*UBLOXGetAllObs():*Obtiene los observables de la constelación a través de las funciones *GetHUI(), GetRAWdata() y GetDOP(),* y los almacena en *BufferRaw.*

-*CalcUserPosition():* Esta función llama a *FindPositionRefresh*(), que realiza las tareas correspondientes al módulo *FindPosition.*

III. Biblioteca GPSLIB

Este módulo contiene las funciones para calcular la posición de usuario a partir de las efemérides y observables y funciones auxiliares requeridas [2].

A. Archivos cabecera (.h)

Constantes.h: Definiciones de constantes a usar según el estándar WGS-84 [8]
Angulos.h: Macros para el pasaje entre grados sexagesimales, semicírculos y radianes

B. Matrices

En este módulo se define una estructura básica *Matriz*, que facilita el manejo de dichos elementos y la estructura de las funciones. Se implementan las siguientes funciones:

-*Matrices_Producto():*Multiplica dos matrices y almacena el resultado en una tercera.

-*Matrices_Transponer():* Transpone una matriz y almacena el resultado en otra

-*Matrices_Sumar():*Suma dos matrices y almacena el resultado en una tercera.

-*Matrices_Reducir():* Aplica la reducción de Gauss-Jordan a una matriz, y almacena el resultado en otra.

C. Posiciones

Este módulo define los tipos de datos y funciones para el pasaje de posiciones en sistema de coordenadas ECEF (Earth-Centered, Earth-Fixed) hacia el sistema LLH (Latitude, Longitude, Height), y cálculos de ángulos de elevación y azimut [9].

- *ECEF2LLH ():* Obtiene latitud, longitud y altura de un punto a partir de sus coordenadas en el sistema ECEF

- *ElevAzim():* Calcula la elevación y el ángulo acimutal de un satélite con respecto al usuario.

D. GPSLIB

Éste módulo contiene las funciones que implementan los diferentes pasos del algoritmo de cálculo y corrección de la posición del receptor a partir de los datos de la constelación de satélites GPS

- *GPS_PosicionSatelite():* Esta función calcula la posición de un satélite determinado a partir de las efemérides y del pseudorango asociado, conociendo el tiempo GPS actual. Así se obtiene la posición de dicho satélite en el sistema ECEF, calculando primero su posición en el marco de su órbita, y luego transformando dicha órbita al sistema mencionado.

- *GPS_PosicionUsuario():* Esta función calcula la posición del receptor en el sistema ECEF a partir de los vectores de pseudorangos y posiciones de cada satélite.

- *GPS_CorreccionAtmosferica():* Esta función calcula el factor de corrección (en metros) que se aplicará el pseudorango debido al retardo ionosférico según el modelo de Klobuchar [6] para cada satélite.

- *GPS_CorreccionReloj():* Esta función calcula el factor de corrección que debe sumarse al pseudorango medido para compensar el efecto de la desincronización de reloj del satélite, dada por los parámetros *af*, *Toc* y *Tgd* en el mensaje de navegación

- *GPS_CorreccionRotacion():* Esta función aplica una rotación sobre la posición del satélite para corregir la desviación causada por la rotación de la Tierra durante el tiempo en que la señal viaja del mismo hasta el receptor.

E. Aplicación del algoritmo de posicionamiento

La forma en que se aplican las funciones de la librería GPSLIB, en la función *FindPositionRefresh(),*es la siguiente:

- Una vez obtenidas las efemérides y los observables de *BufferCooked*, se extraen los satélites cuya relación señal/ruido supera los 20 dB/Hz, para evitar usar señales reflejadas de satélites fuera de vista que pueden alterar el cálculo.

-Para éstos satélites, se aplica sobre los pseudorangos la corrección por desincronización de reloj (*GPS_CorreccionReloj*)
- Luego, se calcula la posición de los satélites (*GPS_PosicionSatelite)* y se corrigen dichas posiciones por rotación terrestre (*GPS_CorreccionRotacion*)
- Se calcula la posición del usuario (*GPS_PosicionUsuario)*, chequeando la convergencia del algoritmo
- A partir de dicha posición se obtienen y aplican las correcciones de carácter ionosférico (*GPS_CorreccionAtmosférica*), y se recalcula la posición definitiva para el receptor

IV. RESULTADOS Y MEDICIONES DE LA BIBLIOTECA

El entorno experimental para validar la biblioteca consistió en:
- El receptor GPS comercial UBLOX NEO-6M conectado vía UART a la EDU-CIAA
- La placa EDU-CIAA conectada vía USB a una PC, que recibe los datos registrados

Los factores a medir fueron:
- *Error y dispersión* en las posiciones calculadas por la biblioteca. El estándar GPS [9] establece como máximo un error de 30 m, lo cual constituye un requerimiento funcional para el proyecto.
- *Tiempo de ejecución* de las funciones de la biblioteca. Se pretende optimizar el código para mejorar la tasa de resultados por segundo, aunque este no es un requisito funcional, sino de performance. Como referencia, la mayoría de los receptores comerciales ofrecen una tasa de 50 soluciones por segundo (1 solución cada 20 mseg) [5].

A. Error y dispersión

Los resultados obtenidos de la biblioteca desarrollada se compararon contra la posición "esperada", es decir, la indicada por el receptor GPS comercial.

Se tomaron varios conjuntos de posiciones calculadas, las cuales se enviaron por USB al ordenador conectado a la placa, y se aplicaron diversos ensayos estadísticos *offline* para identificar errores y dispersiones, y reajustar el algoritmo.
Para estudiar la versión de la librería descrita por este documento se tomó un conjunto de 760 muestras. Mediante un análisis estadístico se obtuvieron los resultados que se muestran en la Tabla I. La diferencia entre la posición media obtenida y la posición esperada fue de 9.097 m

TABLA I: ANÁLISIS ESTADÍSTICO DE LAS MUESTRAS TOMADAS

Coordenadas ECEF	Valor esperado [m]	Valor medio obtenido [m]	Desvío Estándar [m]
X	2696817.56	2696824.61	7.1484
Y	-4512032.80	-4512038.53	9.7384
Z	-3600366.33	-3.600366.04	5.7659

En la figura 2 se muestra la distribución de las muestras contenidas a medida que nos alejamos de la posición real (de forma similar a una distribución acumulada de probabilidad).

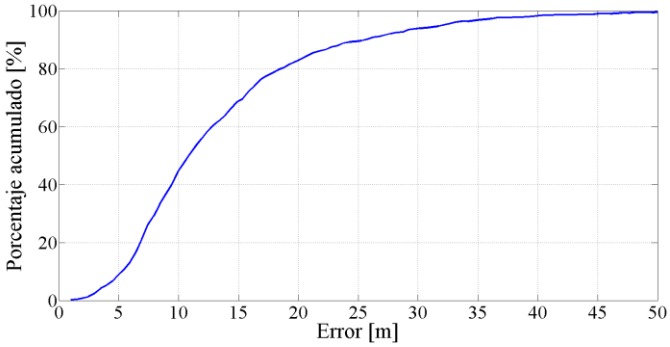

Figura 2: Proporción de las muestras contenidas en función de la distancia a la posición "real"

Se puede apreciar que cerca del 95% de las muestras presentan un error menor o igual a los 30 m que especifica el estándar GPS como error máximo.

B. Tiempos de ejecución

Resulta de interés conocer el tiempo que tarda cada parte del algoritmo en ser ejecutada, a fines de hallar aquellos pasos que representan "cuellos de botella" y enfocar en ellos el esfuerzo por mejorar el consumo de ciclos.

Para esta tarea se agregó a la librería GPSLIB una biblioteca de macros *DWT.h*, que permiten acceder al registro de control (CTRL) y al contador de ciclos (CYCCNT) del DWT (*Data Watchpoint and Trace*), una unidad del procesador destinada a funciones auxiliares de debugging [10]. El procedimiento para medir ciclos consumidos con esta unidad es el siguiente:

- Se pone a cero el registro contador CYCCNT
- En el principio del código a medir, se pone en 1 el bit menos significativo del registro de control, lo cual inicia el conteo de ciclos

- Al final del código bajo prueba, se pone en 0 el bit antes mencionado, lo cual detiene la cuenta, y se guarda el estado del registro contador, que contiene los ciclos de procesador transcurridos desde su activación.

El reloj del procesador ARM Cortex-M4F fue configurado a una frecuencia de 204MHz, con lo cual cada ciclo medido representa un tiempo aproximado de 4.9 ns.

Se ejecutaron 480 búsquedas de solución, de las cuales se midieron los ciclos correspondientes a los siguientes pasos del algoritmo:
- GPS_PosicionSatelite
- GPS_PosicionUsuario (construccion de matriz alfa)
- GPS_PosicionUsuario (preinversion)
- GPS_PosicionUsuario (inversion)
- GPS_PosicionUsuario (postinversion)
- GPS_CorreccionAtmosferica

Los tiempos fueron medidos sobre las funciones con mayor consumo de ciclos de la biblioteca GPSLIB. A fines de apreciar mejor como se distribuye dicho consumo, la función *GPS_PosicionUsuario()* fue dividida en cuatro partes:

- C*onstrucción de matriz alfa*: La matriz alfa representa el jacobiano [11] del sistema no lineal de ecuaciones que relacionan la posición del usuario con la de los satélites.

- *Preinversion:* El cálculo de la posición del usuario requiere la pseudoinversión [12] de la matriz alfa.

- *Inversion:* La matriz del paso anterior se somete a reducción de Gauss-Jordan para efectuar la inversión matricial

- *Postinversion:* Comprende la extracción de la matriz resultante de la inversión y otros productos matriciales

Los tiempos medidos se exponen en la Tabla II

TABLA II: TIEMPOS DE EJECUCIÓN MEDIDOS

Sección	Tiempo de ejecución [µseg]	
	Valor medio	Desvío estándar
GPS_PosicionSatelite	389,81	1,85
GPS_PosicionUsuario (construccion de matriz alfa)	48,01	0,259
GPS_PosicionUsuario (preinversion)	63,71	0,108
GPS_PosicionUsuario (inversion)	118,85	0,14
GPS_PosicionUsuario (postinversion)	75,71	0,147
GPS_CorreccionAtmosferica	492,47	1,805

En un caso típico se requiere hallar la posición de seis satélites y se realizan en promedio siete iteraciones para calcular la posición del usuario, de esta manera el consumo de tiempos queda como se muestra en la tabla III.

TABLA III: TIEMPOS DE EJECUCIÓN TOTALES

De las tablas II y III se observa que:

Sección	Tiempo [ms]	N	Total [ms]	Porc.
GPS_PosicionSatelite	0,389	6	2,33	47%
GPS_PosicionUsuario (construccion de matriz alfa)	0,048	7	2,135	43%
GPS_PosicionUsuario (preinversion)	0,064			
GPS_PosicionUsuario (inversion)	0,118			
GPS_PosicionUsuario (postinversion)	0,075			
GPS_CorreccionAtmosferica	0,492	1	0,492	10%
Total			4,957	100

-Las operaciones individuales que consumen más tiempo son el cálculo de la posición de los satélites, la inversión o reducción matricial, y la aplicación de correcciones atmosféricas

-En el consumo total, debido a los cálculos iterativos, las correcciones atmosféricas pierden peso relativo, concentrando el algoritmo de posicionamiento de los satélites la mayor parte del tiempo consumido.

-El tiempo total de cálculo estimado (4,957 mseg) no define la tasa de soluciones del dispositivo, ya que el tiempo se reparte entre otras tareas como la recepción de los datos, el manejo de los buffers, la comunicación con el ordenador, las cuales reducen la tasa final.

V. OPTIMIZACIÓN DE LA BIBLIOTECA

Cualquier estrategia orientada a mejorar el rendimiento del algoritmo debe enfocarse en optimizar en primera instancia las operaciones globalmente más costosas en tiempos.

En nuestro caso, se decidió reimplementar las operaciones más costosas con funciones escritas en el lenguaje ensamblador del procesador ARM Cortex-M4F, aprovechando la Unidad de Punto Flotante (*FPU* por sus siglas en inglés) que se incorpora en hardware de forma nativa. [10]

A. Entorno de programación

En el núcleo del procesador ARM CORTEX M4F encontramos 13 registros de propósito general de 32 bits (*r0-r12*), además de los siguientes registros con funciones especiales:

- *Stack Pointer (r13, SP)*
- *Link Register (r14, LR)*
- *Program Counter (r15, PC)*
- *Program Status Register (r16, xPSR)*

La FPU, por su parte, posee 32 registros de propósito general (*S0-S31*) para punto flotante en precisión simple (32 bits), que pueden combinarse en pares para ser utilizados como 16 registros (*D0-D15*) para punto flotante en precisión doble (64 bits). Además, cuenta con un Registro de Estados (*Floating Point Status Control Register, FPSCR)* análogo al *xPSR*, junto con otros registros de sistema:

- *Floating Point Context Control Register (FPCCR)*
- *Floating Point Context Address Register (FPCCR)*
- *Floating Point Default Status Control Register (FPDSCR)*

Los registros del núcleo, junto con el banco de registros de propósito general de la FPU, se muestran en la figura 3.

Analizando el código generado automáticamente por el compilador utilizado, se encontró un bajo uso de registros y una alta cantidad de instrucciones de lectura/escritura sobre la memoria (instrucciones *ldr/str*), lo cual aumenta considerablemente los tiempos de ejecución. Por lo tanto, la optimización en lenguaje ensamblador se enfoca en gran medida en aprovechar los registros de propósito general disponibles para disminuir al mínimo los accesos a memoria.

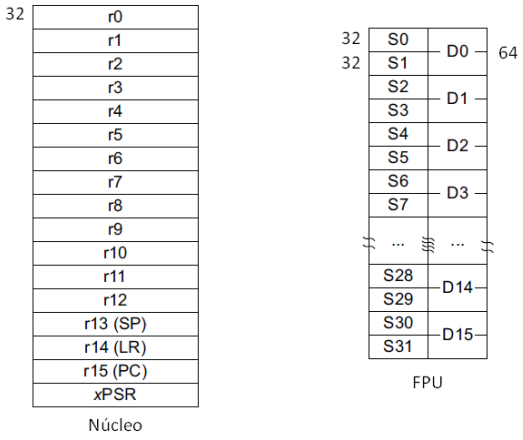

Figura 3: Registros utilizados del ARM Cortex-M4F

B. Optimización y resultados

Se reemplazaron las funciones de la biblioteca GPSLIB escritas en C por sus equivalentes en lenguaje ensamblador (*assembly)*, definidas en archivos .S (código fuente en lenguaje ensamblador) individuales.

Se midió el tiempo de ejecución nuevamente, y se obtuvieron los valores medios que se muestran en la Tabla IV, comparando los mismos con los resultados previos.

TABLA IV: TIEMPOS DE EJECUCIÓN SIN Y CON OPTIMIZACIÓN

Sección	Sin optimizar **[µseg]**	Optimizado **[µseg]**	Mejora
GPS_PosicionSatelite	389,81	21,76	94,42%
GPS_PosicionUsuario (construccion de matriz alfa)	48,01	N/A	N/A
GPS_PosicionUsuario (preinversion)	63,71	21,15	66,8%
GPS_PosicionUsuario (inversion)	118,85	34,39	71,06%
GPS_PosicionUsuario (postinversion)	75,71	25,24	66,66%
GPS_CorreccionAtmosferica	492,47	N/A	N/A

Observamos en las zonas optimizadas correspondientes a operaciones matriciales una reducción del tiempo consumido aproximadamente a un tercio del valor original. Por otro lado, el tiempo de cálculo para la posición de los satélites en la versión optimizada resultó el 5,58% del valor sin optimizar.

Con estos nuevos valores, y aplicando el mismo razonamiento que en la Tabla III, el tiempo de ejecución estimado se reduce a 1,52 ms, es decir, un 30,66% del tiempo total estimado anteriormente.

VI. CONCLUSIONES

Se logró diseñar y construir una biblioteca de funciones capaz de procesar las efemérides y observables de los satélites de la constelación GPS para obtener la posición de usuario de acuerdo a las especificaciones del sistema. La implementación realizada resultó independiente de la plataforma de hardware utilizada y será aplicada en la construcción de un GPS definido por software para aplicaciones en navegación.

Asimismo, se logró optimizar el tiempo de procesamiento empleado en secciones críticas del mismo aproximadamente a un tercio de su valor original, siendo este resultado promisorio para aplicaciones con altas tasas de muestreo y/o bajo consumo.

VII. AGRADECIMIENTOS

Los autores agradecen a los Ingenieros Edgardo Fernández Vescovo y Edgardo Comas por su constante apoyo y asistencia en las tareas realizadas.

Los autores desean agradecer el apoyo de la Universidad Tecnológica Nacional que a través del Proyecto de Investigación y Desarrollo (PID) N° 3636 ha financiado parte de este trabajo.

VIII. REFERENCIAS

[1] United States Air Force, "The GPS Standard Positioning System", *2008.*

[2] F. Larosa et al, “Desarrollo y validación de algoritmos en C para cálculo de posición de usuario en el sistema GPS”, Congreso Argentino de Ingeniería (CADI) 2016

[3] W. Tuttlebee, “Software defined radio”, John Wiley & Sons, Chapter 1.

[4] E. Gamma et al, Patrones de Diseño, Addison Wesley, pp. 251-259

[5] UBLOX 6 Receiver Description, v 7.03

[6] J. Klobuchar, Ionospheric Time-Delay Algorithm for Single-Frecuency GPS Users, IEEE Transactions on Aerospace and Electronic Systems, Vol. AES-23, Number 3, May 1987.

[7] E. Kaplan, C. Hegarty, Understanding GPS: Principles and Applications, Artech House

[8] National Imagery and Mapping Agency, Department of Defense World Geodetic System 1984, Third Edition, 3 de Enero de 2000

[9] J. Bao Yen Tsui, Fundamental of Global Positioning System Receivers: A software approach, Second Edition, Wiley & Sons, Chapter 2.

[10] Cortex M4F, Technical Reference Manual, Revision R0P0, ARM Limited

[11] Wolfram Math World, Jacobian, [Online], Disponible: http://mathworld.wolfram.com/Jacobian.html

[12] Wolfram Math World, Pseudoinverse, [Online]. Disponible: http://mathworld.wolfram.com/Pseudoinverse.html