Procesamiento de imágenes bajo Windows CE utilizando el procesador ARMv 4I

Jorge Andrés Álvarez Triana
Ing. Biomédico, Universidad Antonio Nariño
Docente Tiempo Completo, Investigador Grupo GEPRO,
Universidad Antonio Nariño UAN Ibagué, Colombia
grindzor@gmail.com

José Armando Fernández Gallego
MSc. Automatización Industrial,
Universidad Nacional de Colombia sede Manizales
Docente Tiempo Completo, Investigador Grupo GEPRO,
Universidad Antonio Nariño UAN Ibagué, Colombia
fernandezgallego@gmail.com

Resumen— En este artículo se presenta una metodología para el desarrollo de aplicaciones en dispositivos embebidos y móviles para el procesamiento digital de imágenes. Se muestra un ejemplo de desarrollo de una máscara sobel para detección de bordes aplicada sobre la imagen de Lena en el dispositivo Zeus Epic 520 que cuenta con el procesador ARMV4I con sistema operativo Windows CE.

Palabras clave— Dispositivo embebido, Procesador ARMv 4I, procesamiento de imágenes, Windows CE.

Abstract— This paper presents a methodology for the design of applications in embedded and mobile dispositives for digital image processing. Shows an example of design using a sobel mask for edge detection on the image of Lena in the dispositive Zeus Epic 520 which count with ARMv 4I processor with Windows CE operative system.

Keywords— Embedded dispositive, ARMv 4I processor, Image processing, Windows CE.

I. INTRODUCCIÓN

Actualmente los sistemas de cómputo utilizan procesadores programados en lenguaje C/C++ como un estándar adoptado en la industria con el que se abre la posibilidad de utilizar códigos de programación en diferentes arquitecturas de dispositivos mediante el compilador apropiado para cada uno. Como una primera etapa natural para un programador es realizar sus desarrollos en lenguaje C/C++ con compiladores para Windows o Linux, con los que es posible realizar cada una de las funciones implementadas a nivel de hardware con éxito, hasta este momento es transparente para el desarrollador la forma exacta en que los datos van a ser manipulados por el procesador.

Cuando se requiere desarrollar aplicaciones en una arquitectura diferente a la de un PC convencional, es necesario ahora centrarse en el conocimiento de la arquitectura del nuevo dispositivo, debido a que este es ahora un nuevo espacio de trabajo con diferentes leyes de operación y que necesariamente para su programación requieren de un nuevo compilador que pueda traducir el lenguaje C/C++ a su respectivo conjunto de instrucciones. Dado que cada fabricante posee una forma diferente de configuración del compilador, de sus dispositivos, su interfaz e interconexión con su plataforma de desarrollo puede llegar a ser confuso la forma de emprender una implementación con éxito en este nuevo sistema. En este artículo se presenta la metodología utilizada para programar el procesador ARMV 4I mediante el compilador Embedded Visual C++ 4.0 para aplicaciones bajo Windows CE 5.

II. MATERIALES Y MÉTODOS

A. Sistemas Embebidos

Son dispositivos de propósito específico y dedicado, a diferencia de un PC convencional son diseñados para ejecutar un número disminuido de operaciones [1]. Estos sistemas se caracterizan por su tamaño reducido, su bajo consumo de potencia y su bajo costo.

En [1] el autor explica las diferencias específicas con un sistema de cómputo convencional con los siguientes argumentos:

- Generalmente son de costo reducido
- La mayoría poseen restricciones de tiempo-real
- Existen múltiples arquitecturas de procesadores (MIPS, ARM, PowerPC y otros) cuyas características varían según la aplicación específica del sistema embebido (procesamiento de imágenes, transmisión de datos, entre otros)
- Poseen recursos limitados de memoria RAM, ROM u otros dispositivos de Entrada /Salida

(E/S) en comparación con una computadora tipo PC

- Un sistema embebido es diseñado desde dos perspectivas, hardware y software, habida cuenta su aplicación específica.

Este tipo de sistemas suele imponer límites de interacción con el usuario o restringirlo con la finalidad de evitar las modificaciones al sistema base.

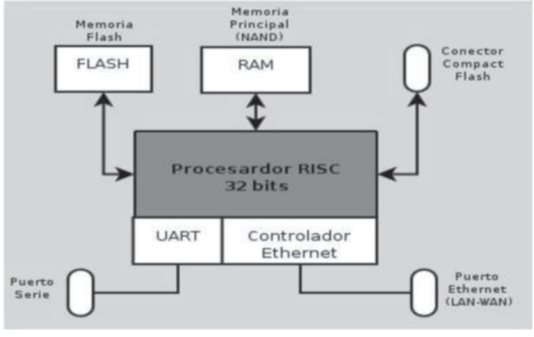
a). Arquitectura

Los sistemas embebidos son implementados en una sola tarjeta y en esta se encuentran contenidos los recursos necesarios y suficientes para su funcionamiento, tales como memoria RAM, microprocesador y puertos de comunicación, entre otros. Ver Fig. 2. La mayoría de estos recursos no son ampliables; y si existe la posibilidad no son de fácil acceso al usuario del común a excepción de los conectores y buses de datos que suministra el fabricante.

b). Procesadores integrados

Frecuentemente los sistemas embebidos tienen microprocesadores - integrados denominados *SOC* por sus siglas en inglés *system on chip.* Los procesadores tipo *SOC* poseen integrados componentes controladores *DRAM, USART, UART, ETHERNET, PCI, y otros.* La arquitectura habitual de los procesadores *SOC* es de tipo *RISC* por sus siglas en inglés *reduced Instruction set computer,* la cual puede apreciarse en la Fig. 1.

Fig. 1. ARQUITECTURA DE PROCESADOR SOC TIPO RISC TOMADO DE [1]



B. Metodología

La secuencia metodológica ordenada por pasos para el desarrollo de aplicaciones en procesamiento digital de imágenes en sistemas embebidos debe ser:

1) *Al abordar la programación de cualquier sistema embebido es el conocimiento de su arquitectura lo que dicta las pautas, normas y recursos disponibles para tal fin.*
2) *La comprensión del kernel o sistema operativo del dispositivo es fundamental, ya que incluidas dentro de este se encuentran disponibles las librerías necesarias para el manejo de imágenes; junto a estas es necesario revisar las incluidas por el fabricante. Si las librerías no son eficientes al aprovechar los recursos del sistema y consumen mayor tiempo máquina del esperado es necesario utilizar herramientas básicas del lenguaje C/C++ apoyadas por la base matemática de las técnicas del procesamiento digital de imágenes.*
3) *Es necesario entender las limitaciones del sistema tanto en software como en hardware para aprovechar al máximo los recursos disponibles.*
4) *El conocimiento de las herramientas de desarrollo es fundamental ya que estas difieren de las estándar siendo específicas para el dispositivo. El fabricante suele proporcionar los kits estándar de desarrollo que se encargan de enlazar la herramienta de desarrollo junto con las librerías del dispositivo y su procesador.*
5) *Es preciso un patrón de comparación de los resultados obtenidos contra una herramienta de software especializada en procesamiento digital de imágenes o librería para compiladores C/C++ estándar para realizar el ajuste finó de algoritmos.*

III. LIMITACIONES DEL SISTEMA

A. Procesador ARMv 4I

Este procesador tiene una arquitectura tipo *RISC* de 32 bits y preparado para trabajar en ambientes industriales.

Dentro las limitaciones del procesador ARMv 4I [1] se tienen:

1) *Punto flotante*

Su arquitectura es diseñada para aplicaciones móviles y embebidas optimizada para operaciones con valores enteros, pero no tiene soporte para operaciones de punto flotante [2], [3], [4], aunque es capaz de emularlas mediante instrucciones ilegales. Para utilizar este tipo de operacio-

nes el compilador C/C++ envía órdenes al procesador para que haga uso del emulador interno de operaciones punto flotante, lo que puede llevar a errores inesperados por exceso de operaciones ilegales y desborde de pila si existe exceso de datos de este tipo.

2) Velocidad

La velocidad establecida para el reloj de este procesador es 520 MHz. El procesamiento en tiempo real es limitado por esta velocidad y las aplicaciones robustas deben ser puntuales, precisas y eficientes. Es recomendable no hacer uso de gran cantidad de variables flotantes para no aumentar el tiempo de respuesta.

B. Windows CE 5

Dentro de las limitaciones para esta versión embebida de Windows encontramos:

1) Módulos

Windows CE 5 a diferencia de las versiones para PC convencional está constituido por módulos, la inclusión de estos varía según el fabricante del dispositivo en donde está instalado. Así el fabricante del sistema podría prescindir de funciones necesarias para el procesamiento gráfico y otras funciones requeridas.

2) Procesos

Windows CE 5 es multitarea, cada proceso ejecutado tiene un límite de 32 MB y restringe los recursos disponibles para la aplicación que va a desarrollar [6], [7], [8].

3) Actualización del sistema

La forma de actualización del sistema requiere escribir la memoria Flash o Flash NAND. Si este proceso se realiza de forma incorrecta se corre el riesgo de inutilizar el dispositivo [9].

IV. SINCRONIZACIÓN WINDOWS CE 5 – WINDOWS XP

Al desarrollar una aplicación para cualquier dispositivo embebido es necesario sincronizar datos entre el sistema de desarrollo y el dispositivo sobre el cual se programa.

Para el caso del software de desarrollo Embedded Visual C++ la sincronización de datos es puntual a la versión del software requiriendo de Windows XP o inferior mínimo Windows 2000. Es necesaria la instalación de Microsoft Active Sync 3.75 o superior.

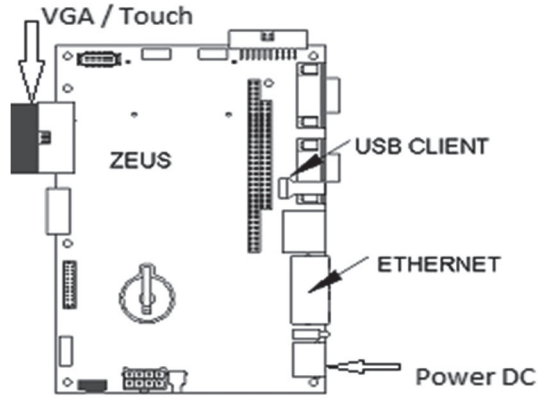
Puede realizarse sincronización por medio de conexión Ethernet / LAN, vía bluethoot y vía USB. El programa hace uso del protocolo IP v4 en su versión 3.75, donde el host es Windows XP y el cliente es Windows CE permitiendo importar y exportar datos entre los dos sistemas. Microsoft Active Sync se encarga de la compatibilidad entre archivos debido a que las cabeceras de los mismos son distintas en Windows XP y Windows CE [10].

V. DISPOSITIVO DE DESARROLLO

Características como bajo consumo de potencia e integración en una sola placa son propias de los sistemas embebidos. El sistema ZEUS EPIC 520 de la empresa Eurotech cumple con estos lineamientos básicos y posee características necesarias y suficientes de procesamiento y velocidad.

Este dispositivo posee un procesador ARMv 4I como el usado por celulares, GPS y Tablet PC. Cuenta con una memoria RAM de 256 MB en board, memoria flash 64MB, 256kB de memoria RAM de respaldo con batería y un socket para memorias SD y MMC. A nivel video y gráficos ostenta un controlador para su respectiva pantalla táctil, un controlador de captura rápida de video y controlador de video de salida VGA. Interfaces de comunicación entrada salida como Ethernet, RS 232, RS 422, RS 485, USB 1.1 y soporte para modem wireless [5]. En la Fig. 2 se muestra el diseño esquemático del dispositivo.

Fig. 2. DISEÑO ESQUEMÁTICO DEL DISPOSITIVO EMBEBIDO ZEUS EPIC 520



VI. SOFTWARE DE DESARROLLO

Al desarrollar una aplicación para un dispositivo embebido es necesaria la instalación del set estándar de desarrollo, por *su sigla* en inglés *SDK,* proporcionadas por el fabricante. El software de desarrollo debe ser capaz de interpretar el *SDK* y permitir seleccionarlo de una lista.

A. Embedded Visual C++

Este software de desarrollo está orientado a aplicaciones móviles o embebidas. Su interfaz gráfica es semejante a la de Visual C++ 6.0, enfocada a la programación en el *API (Aplicativo de interfaz de programación)* de Windows CE en sus diferentes versiones, sin dejar de lado el desarrollo estándar C++.

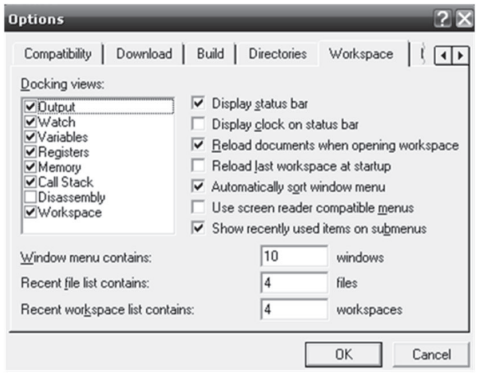
1) Configuración del software

El programa puede configurarse desde el menú Tools. Los submenús necesarios para la configuración son los siguientes:

a) Menú Options

Muestra diferentes opciones organizadas por pestañas como se muestra en la Fig. 3. Entre las cuales encontramos configuración del *editor*, tabulaciones, ajuste (*debug), compatibilidad, descarga (download), construir (build), directorios, espacio de trabajo, macros* y *forma.* Donde las pestañas fundamentales para la configuración son *descarga, directorios* y *espacio de trabajo*.

Fig. 3. CUADRO DE DIÁLOGO DEL MENÚ OPCIONES



La pestaña *directorios* permite enlazar *librerías* y los *SDK* al procesador y plataforma de desarrollo. Los enlaces se logran al escribir la ruta del directorio que contiene los archivos necesarios según sea el tipo incluir (*include), librerías (library), origen (source) o ejecutable*. Para enlazar librerías además de especificar las rutas de directorio es necesario una vez creado un proyecto ir al menú proyecto (*Project)* pestaña enlace (*link)* e incluir los nombres de los módulos de librería *.lib* en la opción *Object/library module*s.

La pestaña descarga (*download)* permite administrar que datos propios del ejecutable son enviados al dispositivo y cambiar el tiempo de espera para detectar la sincronización con el dispositivo.

En la pestaña espacio de trabajo (*workspace)* encontramos las opciones que nos permiten vigilar el programa mientras corre en modo ajuste (debug). Las esenciales son salida (*output)* que visualiza lo compilado y posibles errores, observar (*watch)* que permite vigilar la variables, memoria (*memory)* que permite ver recursos disponibles y en cola (*call stack)* que permite vigilar el estado de pila. La importancia de estas opciones radica en la limitación de recursos que se tiene por parte del dispositivo por lo cual es imperioso vigilar y observar para prevenir y corregir errores.

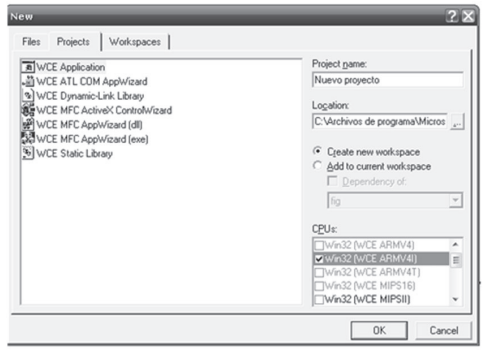
b) Menú Configurar director de plataforma (configure Platform Manager)

Desde este menú es posible administrar los dispositivos instalados por el *SDK* del fabricante así como los emuladores de pocket PC o de Windows CE, que permiten configurar el método de sincronización con el dispositivo. Es recomendable ajustarlo a *Microsoft Active Sync* y hacer el *test de sincronización.*

B. Proyecto en Embedded Visual C++.

Como es tradición en los ambientes de trabajo de Microsoft visual studio para ejecutar el código es necesario crear un nuevo proyecto, la diferencia radica en la posibilidad de elegir el procesador con el cual se desea trabajar [10] en este caso particular el procesador ARMv 4I ver Fig.4.

Fig. 4. CUADRO DE DIÁLOGO NUEVO PROYECTO



También posee la alternativa de elegir una plantilla de aplicación previamente configurada, o elegir una guía de configuración de la aplicación. Aunque venga previamente configurado o se utilice la guía de configuración es recomendable configurarlo manualmente como se ha descrito anteriormente.

Una vez creado el proyecto podemos seleccionar el *SDK, procesador* y *dispositivo* a utilizar desde la barra de herramientas configuración de *WCE,* dentro de esta barra encontramos dos modos de compilación / ejecución; un modo ajuste (*Debug)* y un modo (lanzamiento) (*release).* El modo ajuste (debug) compila y ejecuta conjuntamente el embedded visual C++ con el dispositivo. El modo lanzamiento (release) ejecuta totalmente el código en el dispositivo y este sirve para guardar en la memoria del dispositivo él ejecutable.

C. Punto de entrada Winmain en Windows CE.

A diferencia de C/C++ estándar donde se utiliza la función *main* en el compilador embedded visual c++ es necesario utilizar *Winmain*. Los parámetros de esta función son definidos por el sistema operativo y son universales para todas las clases de arquitectura de Windows. Cada parámetro pasa al sistema operativo como código de salida. Winmain se encarga de traducir y trasladar mensajes, y de cargar los aceleradores internos de Windows para ejecutar procedimientos de ventana. [7], [8] y [10].

D. Portabilidad del código

El código no es portable ni compatible con otras versiones del mismo, por ejemplo un código escrito para la versión 5 puede no ser ejecutado en la versión 6.

La portabilidad del código entre procesadores asumiendo la misma versión de Windows y los mismos contenidos, depende que sean de la misma familia o compatibles, por ejemplo un código compilado con un procesador ARMv 4I no es portable a un procesador x86, pero si es compatible con un ARMv 4T.

E. Archivo ejecutable

El archivo *.exe* generado por el compilador es descargado al sistema de ficheros del dispositivo. La permanencia del mismo depende del dispositivo en sí; para el caso del ZEUS EPIC 520 [5], [10] este es guardado en directorio raíz a la parte de ficheros en RAM, por lo que al cortar la alimentación del dispositivo el ejecutable se pierde. Esto se soluciona mediante la entrada que posee el dispositivo para memorias externas SD o USB que retengan los datos de forma permanente. Para caso de dispositivos móviles como Pocket PC o teléfonos móviles es guardado en la memoria removible del dispositivo.

VII. LECTURA Y ESCRITURA DE LA CABECERA DEL FORMATO BMP

Las librerías contenidas en Windows CE para carga de imágenes son poco eficientes a nivel velocidad por lo tanto es necesario utilizar un métodos alternativos de carga como lo es leer la imagen directamente desde un buffer binario. Durante este trabajo es utilizado el formato BMP; formato de archivo básico usado para imágenes digitales en Windows. Este es soportado por la mayoría de versiones de Windows.

El formato BMP es compuesto por:

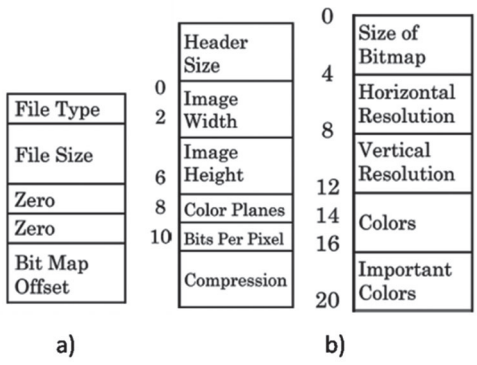
1) Cabecera de archivo.
2) Cabecera de mapa de bits
3) Tabla de color
4) Datos crudos de la imagen.

La cabecera del archivo mostrada en la Figura 4 es formada por los primeros 14 bytes. Donde los primeros dos bytes indican el tipo del archivo que pueden ser en formato hexadecimal (*4D42)* ó en ASCII (BM). Los siguientes cuatro bytes indican el tamaño del archivo, seguidos de dos bytes reservados. Los bytes restantes dan un valor *offset* a los datos de imagen [6].

La cabecera de mapa de bits es conformada por 40 bytes mostrados en la *Figura 4 b)*. Inicia con el tamaño de la cabecera siempre de 40 seguida del tamaño de la imagen en alto y ancho; si el valor del alto es un número negativo significa que la imagen está invertida o fue almacenada de abajo hacia arriba. El siguiente campo contiene los planos de color donde 1 es representa imágenes blanco y negro, y 3 para imágenes de color RGB. Los siguientes 2 bytes

indican los bits por pixel, por ejemplo, 8 es para 256 niveles de grises. El campo que corresponde a compresión activa la compresión si está en uno y desactiva en cero. Los siguientes dos campos expresan la resolución horizontal y vertical, están medidos en pixeles por metro. Por último los dos campos relacionados con color ayudan a descomprimir e interpretar los colores junto con la tabla de color [11].

Fig 5. ESTRUCTURA DE LA TRAMA DE DATOS DE LAS CABECERAS DEL FORMATO BMP. A) CABECERA DE ARCHIVO, B) CABECERA DE MAPA DE BITS [6]



Más allá de las cabeceras se cuenta con la tabla de color, que funciona reasignando un valor de entrada a un valor de color.

La parte final de la trama corresponde a los datos almacenados de la imagen en la extensión BMP, datos que deben ser interpretados con la tabla de color.

Para aumentar la velocidad de procesamiento, la lectura del archivo se realiza con la función *fopen* para la carga del archivo y *fread* con su respectiva cabecera <*commctrl.h*>.

Para escritura se utiliza la función *fopen* con la bandera *w* para escribir un nuevo archivo y adicionalmente se utiliza *fwrite* para agregar nuevos datos a la trama del archivo [11].

VIII. RESULTADOS

En el dispositivo de desarrollo se realizó un ejercicio de implementación con el filtro de *SOBEL* que se muestra en la Fig. 6. Las ecuaciones utilizadas para obtener el valor total del gradiente son (1) y (2). En donde las operaciones básicas desarrolladas son suma, resta potenciación y radicación al procesar los datos obtenidos por medio de la memoria flash y la memoria del programa.

Fig. 6. MÁSCARAS DE SOBEL VERTICAL Y HORIZONTAL

$$Gy=\begin{bmatrix}+1 & +2 & +1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix} \qquad Gx=\begin{bmatrix}+1 & 0 & -1\\ +2 & 0 & -2\\ +1 & 0 & -1\end{bmatrix}$$

La fig. 6 muestra la matriz *I* que representa la imagen de entrada suponiendo que es de 3x3. Las ecuaciones 1 y 2 muestran como se calculan los gradientes Gm_x y Gm_y [12], [13].

Fig. 7. MATRIZ *I* QUE REPRESENTA LA IMAGEN DE ENTRADA

$$I=\begin{bmatrix}a & b & c\\ d & e & f\\ g & h & i\end{bmatrix}$$

1.

$$\begin{aligned}G_{mx}=&\left(a*Gx(1,1)+b*Gx(1,2)+c*Gx(1,3)+\ldots\right.\\&\ldots+d*Gx(2,1)+e*Gx(2,2)+f*Gx(2,3)+\ldots\\&\left.\ldots+g*Gx(3,1)+h*Gx(3,2)+i*Gx(3,3)\right)/9\end{aligned}$$

2.

$$\begin{aligned}G_{my}=&\left(a*Gy(1,1)+b*Gy(1,2)+c*Gy(1,3)+\ldots\right.\\&\ldots+d*Gy(2,1)+e*Gy(2,2)+f*Gy(2,3)+\ldots\\&\left.\ldots+g*Gy(3,1)+h*Gy(3,2)+i*Gy(3,3)\right)/9\end{aligned}$$

3.

$$Gp=\sqrt{Gm_x^2+Gm_y^2}$$

Para obtener el gradiente total de salida se utiliza la ecuación 3 donde *Gp* es la imagen del gradiente total calculado [12], [13].

La Fig.8 muestra la imagen de entrada *I* y las imágenes de salida Gm_x, Gm_y y *Gp* con 8 bits de resolución en escala de grises y con 8 bits de resolución. En el primer cuadrante se muestra la imagen de entrada Lenna en formato BMP; en el segundo y tercer cuadrante se muestra el resultado de aplicar el operador sobel vertical y horizontal. La imagen del cuarto cuadrante muestra el resultado final de la implementación del filtro sobel mediante C/C++ que hace uso del embedded visual C++ para el sistema embebido Zeus Epic 520 con procesador ARMv 4I.

Fig. 8. RESULTADO DE APLICAR EL OPERADOR SOBEL SOBRE LA IMAGEN EN EL DISPOSITIVO ZEUS EPIC 520

Los datos de salida del filtro deben ajustarse a la resolución máxima de 8 bits.

El tiempo empleado por el dispositivo al leer, detectar bordes y escribir la imagen de salida en formato BMP fue de 1,89 s.

XI. CONCLUSIONES

Los sistemas de procesamiento actuales de tipo embebido cuentan con una arquitectura limitada y diferente a los sistemas de cómputo tradicional, las implementaciones en esta nueva arquitectura deben estar soportadas por el estudio previo del sistema donde se quiere realizar el procesamiento de los datos con el objetivo de conocer sus herramientas y limitaciones propias. Antes de elegir un dispositivo embebido es una tarea fundamental conocer la aplicación específica que se quiere realizar, el entorno de trabajo donde se alojará el dispositivo y las garantías de programación tales como librerías, programas de sincronización y desarrollo que garanticen un acompañamiento al programador al emprender su tarea.

En el ejemplo de aplicación que se muestra en este artículo se recopila no solamente la realización de una máscara espacial a una imagen, sino una tarea de investigación y documentación previa que debe seguirse al momento de programar una arquitectura embebida. El filtro de sobel implementado no cuenta con limitaciones numéricas de procesamiento y su resultado es el mismo que se obtendría con otro sistema de procesamiento clásico, este resultado se logra con el total conocimiento de la herramienta utilizada y su respectivo entorno de desarrollo.

REFERENCIAS

[1] C. Hallinan,"Embedded Linux primer", Prentice hall, september 2006, capítulo 2.

[2] ARMv 5 Reference manual, ARM ltda, 2005 disponible en http://infocenter. arm.com/help/index.jsp?topic=/com.arm.doc.ddi0100i/index.html, 2007 copyrigth.

[3] Devian, "ArmEabiPort", disponible en http://wiki.debian.org/ArmEabiPort, capítulo 4.

[4] ARM,"Procedure Call Standard for the ARM® Architecture", ARM IHI 0042D, current through ABI release 2.08, October 2009.

[5] ARCOM, "ZEUS Windows CE Development kit Manual", 2001 ARCOM.

[6] Microsoft msdn, "Windows CE 5.0", disponible en http://msdn.microsoft. com/enus/library/gg144991%28v= WinEmbedded.0%29.aspx, 2011 Microsoft.

[7] S.Phung, "Profesional Windows CE 6.0", Wiley Publishing inc, 2009.

[8] S. Pavlov, P. Belevsky, "Windows CE 6.0 Fundamentals", Microsoft Press. 1998.

[9] Microsoft msdn, "Platform builder Users Guide Windows CE 5.0", disponible en http://msdn.microsoft. com/en-us/library/aa448756.aspx, 2011 Microsoft.

[10] P. González, "Manual básico de Embedded Visual C++ 4.0", Universidad de Oviedo, 2004.

[11] D. Phillips, "Image processing in C", "2nd ed. R&D Publications, 2000, pp 7-21 , pp 47 -56.

[12] W. K. Pratt, "digital image procesing", Wiley-Interscience Publication,JOHN WILEY & SONS, INC, 3th edition, pp 453 – 454.

[13] Q. Ying-Donga, C. Cheng-Songa, C. San-Benb, L. Jin-Quana," A fast subpixel edge detection method using Sobel–Zernike moments operator", Image and Vision Computing 23, 2005, pp 11–17.