Estrategias de desarrollo de proyectos con microcontroladores usando herramientas "open source" y generadores de código

Tropea, S.

Centro de Investigación y Desarrollo en Telecomunicaciones, Electrónica e Informática (CITEI)

En este trabajo se expone como metodología de trabajo el uso de herramientas *open source* y generadores de código para el desarrollo de aplicaciones con microcontroladores PIC de Microchip.

INTRODUCCIÓN

Se necesitaba desarrollar un medidor de gases (CO y CH4) domiciliario que incluyera un *display* alfanumérico, indicadores sonoros y luminosos y un botón de *reset* y configuración del dispositivo[1].

Por cuestiones de costo, volumen de producción y experiencia se decidió implementarlo utilizando un microcontrolador (MCU) PIC de Microchip. La MCU seleccionada fue la 16C716 que en ese momento era muy nueva y carecía de un soporte adecuado en todas las herramientas disponibles.

Método empleado

Teniendo en cuenta, entre otras cosas, la estabilidad y robustez del sistema de desarrollo, costos y flexibilidad se decidió utilizar herramientas *open source* para el desarrollo.

Debido a la necesidad de flexibilidad se decidió utilizar generadores de código, esto se explica con mayor detalle en *Lenguaje de programaci*ón.

HERRAMIENTA UTILIZADAS

Programador

Los programadores que disponíamos para DOS no soportaban esta MCU y el fabricante no podía ser contactado. El que poseíamos para Windows tampoco la soportaba y estaba discontinuado. De no haber elegido la metodología *open source* deberíamos haber adquirido uno nuevo. Esto no afectaría a este proyecto pero en muchas áreas, por ejemplo la educativa, es muy pesado comprar nuevos programadores cada vez que se use un nuevo dispositivo. Algunos programadores se pueden actualizar, sin embargo esto implica un costo y/o demoras considerables.

Usamos el prog84[2] y un hardware que ya teníamos. Prog84 es configurable de manera tal que no fue difícil hacer que lo aceptara. El 16C716 no era soportado, pero se solucionó haciendo pequeñas modificaciones en el programa, también fue necesario introducir cambios para soportar señales especiales que poseía nuestro hardware. Prog84 está escrito en lenguaje C.

Ensamblador

No fue difícil encontrar un ensamblador que fuera compatible con la sintaxis del compilador usado en DOS/Windows (MPASM). Usamos el gpasm[3] que ya tenía soporte para el 16C716 debido a que el mismo no agrega nuevas instrucciones.

Simulador

Con estas herramientas era posible desarrollar la aplicación propuesta pero siempre es aconsejable probar el desempeño del programa en un simulador de tiempo real.

Si bien esta es la herramienta más compleja existe un simulador llamado gpsim[4]. El mismo no sólo simula correctamente el comportamiento de las MCU de Microchip sino que soporta los dispositivos incluidos en las mismas. Adicionalmente soporta *plug-ins* para simular dispositivos externos, entre ellos *displays* LCD y 7 segmentos. La

velocidad de simulación es más de un orden de magnitud superior al simulador provisto por el fabricante convirtiéndolo en una excelente herramienta.

El simulador no soportaba el nuevo 16C716, hubo que agregarlo. Gpsim está escrito en C++ y utiliza una estructura de clases muy interesante para describir cada procesador. Se creó una clase derivada de otro controlador muy parecido que agrega las características más importantes de la MCU.

También fue necesario hacer algunos agregados al simulador de LCD para soportar los LCDs seleccionados para esta aplicación.

Si bien fue necesario invertir algo de tiempo el resultado fue más que excelente, en las estaciones de trabajo usadas (K7 750 MHz). La MCU, que usa un cristal 4,19 MHz, es simulada a una velocidad superior al tiempo real. Para verificar los tiempo de ejecución y demoras se incorporaron al simulador los comandos e interfaz de usuarios necesarios para medir tiempos.

LENGUAJE DE PROGRAMACIÓN

Las MCU de Microchip son muy pequeñas y el desarrollo en lenguajes de alto nivel se complica cuando necesitamos explotarlas al máximo. Por otro lado la programación en *assembler* hace los programas muy difíciles de modificar. Por cuestiones de interfaz de usuario y caracterización del sensor a usar era necesario que el programa fuera flexible. Para solucionar esto se optó por usar generadores de código para las partes del programa que debían ser flexibles y *assembler* para las rutinas de interrupción y apoyo.

Se usaron herramientas que ya habíamos desarrollado con anterioridad y herramientas especialmente desarrolladas para este caso.

Statem: generador de máquinas de estado

Statem recibe como entrada un archivo de texto que describe los estados, acciones asociadas y condiciones de transición de una máquina de estados. El lenguaje soporta "supercondiciones" de una manera similar al Graphset de los PLC y operaciones varias para describir las acciones asociadas a cada estado. El mismo puede generar código en lenguaje *assembler* o C.

Mifit: Generador de código para aproximar un juego de datos

Mifit[5] permite generar código para aproximar un juego de datos utilizando ecuaciones simples (rectas y parábolas). El mismo permite evaluar el error introducido en la aproximación. Actualmente los datos de entrada son de 8 bits y los cálculos están limitados a 24 bits de resolución.

CONCLUSIONES

El uso de herramientas *open source* permite una gran flexibilidad gracias a poseer el código fuente. Esto nos permitió agregar soporte para una MCU nueva, ajustar el comportamiento de las herramientas a nuestras necesidades y corregir problemas en las mismas. Adicionalmente son de extrema utilidad para cuando es necesario reducir costos, muy común en ambientes educativos de nuestra región. La calidad de las herramientas es muy buena, igual o mejor que las comerciales. Como contrapartida es necesario invertir tiempo para ajustarlas y adaptarlas. En cada situación se deberá evaluar si esto es o no un problema.

Sabíamos que el uso de herramientas de alto nivel en MCUs tan pequeñas no era recomendable, y pudimos verificar que generadores de código creados para resolver puntos particulares son de gran ayuda y permiten realizar cambios importantes con mayor facilidad que usando herramientas tales como un compilador de C. El código generado puede optimizarse hasta ser comparable o mejor que el escrito por un humano, con la ventaja de que esto es realizado en una fracción de segundo.

Referencias

[1] Tropea, S.; Roberti, M. Medidor domiciliario de gases tóxicos. También presentado en estas jornadas.

[2] F. Damgaard, W. Lewis y otros.

http://home3.inet.tele.dk/frda/picasm/prog.html

[3] J. Bowman, S. Dattalo, C. Franklin, J. Cameron y otros.

http://gpasm.sourceforge.net/

[4] S. Dattalo, R. Forsberg, D. Schudel y otros.

http://www.dattalo.com/gnupic/gpsim.html

[5] Salvador E. Tropea, http://mifit.utic.com.ar/

Para mayor información contactarse con:

Ing. Salvador E. Tropea - salvador@inti.gov.ar

Volver a página **principal**