



Münchener Str. 4a
D-82131 Gauting / Germany
Tel. +49-89-8931043/45
E-Mail: contact@crst.de
Web: www.crst.de

AP-Note

Firmware-Simulation auf Windows-PC

Thomas Criegee CRST GmbH

Einleitung

Bei der Entwicklung von Firmware (in C oder C++) für "embedded" Geräte stellt sich vielfach die Frage, wie man möglichst schnell und kostengünstig zu einem funktionierenden und stabilen Code kommt.

Oft muss die Firmware-Entwicklung schon beginnen, bevor die entsprechende Zielhardware zur Verfügung steht. Vielfach soll auch einem wichtigen Kunden schon möglichst früh ein funktionierendes Modell des Gerätes vorgestellt werden.

Auch bringen die Hersteller von "embedded" Prozessoren in immer kürzeren Zeitabständen neue Tools für ihre neuesten Prozessoren auf den Markt. Nicht immer sind diese Tools stabil und ausgereift.

Somit ergibt sich die Situation, dass Entwickler, die termingerecht Firmware für "embedded" Geräte fertigstellen sollen, bei Beginn der Entwicklung noch keine funktionsfähige Hardware zur Verfügung haben. Sofern aber schon eine funktionsfähige Hardware zur Verfügung steht, wird beim Debugging auf der "embedded" Zielhardware oft mehr Zeit im Kampf mit unausgereiften Tools als mit der Entwicklung der eigentlichen Firmware verbracht.

Eine Möglichkeit, diesen Zustand zu beseitigen besteht darin, die eigentliche Firmware mit ausgereiften Standard-Tools (MS Visual Studio® o.ä.) auf einem PC zu entwickeln, soweit wie möglich zu testen und damit die Zeit zum Debugging auf der eigentlichen Zielhardware zu minimieren ("Software Rapid Prototyping"). Voraussetzung hierfür ist die Kapselung aller hardwareabhängigen Komponenten und die Nachbildung dieser Komponenten auf dem PC. Auch sollte ein ausgereifter C-Compiler für die „embedded“ CPU vorhanden sein.

Wie die Praxis zeigt, funktioniert dieses Verfahren nicht nur für neu zu entwickelnde Firmware sondern auch für Firmware-Änderungen an bestehenden Geräten.

Die oben beschriebene Vorgehensweise soll im folgenden für die Firmware einer kleinen Brandmeldezentrale beschrieben werden.

Die Brandmeldezentrale besteht aus einem "embedded" Prozessor, RAM, Flash-EPROM, LCD-Anzeige, mehreren Tasten, Timern sowie vielfachen I/O-Möglichkeiten. Die Firmware der Brandmeldezentrale (geschrieben in C) war bereits vorhanden, sollte aber aufgrund neuer Anforderungen für drahtlosen Betrieb (Stichwort "Wireless") um den Anschluss von Funkkomponenten erweitert werden.

Bestandteile der Simulation

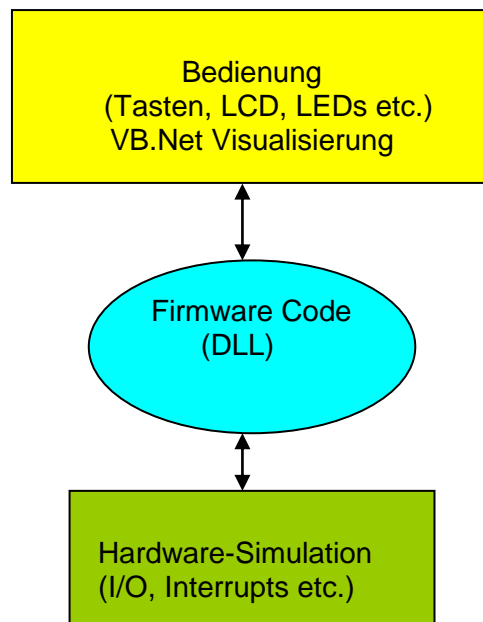
Um eine schnellere und komfortablere Durchführung der nötigen Firmware-Änderungen in einer Umgebung ohne "embedded" Hardware zu gewährleisten, wurde eine unter Windows® XP laufende, aus zwei Komponenten bestehende, Simulation erstellt.

Die erste Komponente besteht aus der VB.Net® Bedienoberfläche, die die Tasten, LCD-Anzeige, LEDs etc. der Brandmeldezentrale simuliert.

Die zweite Komponente besteht aus dem (Original) Firmware-Code der Brandmeldezentrale, der anhand von "Compiler-Switches" so modifiziert wird, daß er auch in einer 32-Bit Windows® Umgebung übersetzbar und ablauffähig (z.B. als DLL) ist.

Mit Hilfe dieser Simulation ist ein „Debugging“ der Firmware auch ohne vorhandene Hardware leicht möglich. Außerdem können externe Ereignisse (Alarmer, Störungen und Wartungsanforderungen) simuliert werden.

Die folgende Abbildung zeigt eine Übersicht der Simulation:



Die Bedienung der Brandmeldezentrale wurde mit einer VB.Net® Oberfläche nachgebildet. Die in C geschriebene Firmware wurde in in einer Win32 DLL "verpackt", die hardwareabhängigen Komponenten durch WIN32 hardwareabhängige Komponenten ersetzt.

Selbstverständlich ist auch eine Simulation "aus einem Guss", also ohne DLL, möglich, z.B. mit Hilfe einer kompletten „C-Anwendung“ oder der Nachbildung der Bedienung in einer WIN32 Konsolen-Anwendung ("DOS-Box").

Simulation der Bedienung

Die Bedienung der Brandmeldezentrale wurde mit einer VB.Net® Oberfläche realisiert (siehe Abbildung). Die LCD-Anzeige wurde mit einer Listbox (4 Zeilen à 20 Spalten), die Tasten mit Command-Buttons, die LEDs mit farbigen Labels simuliert. Je nach Anforderung können natürlich auch andere Elemente verwendet werden.

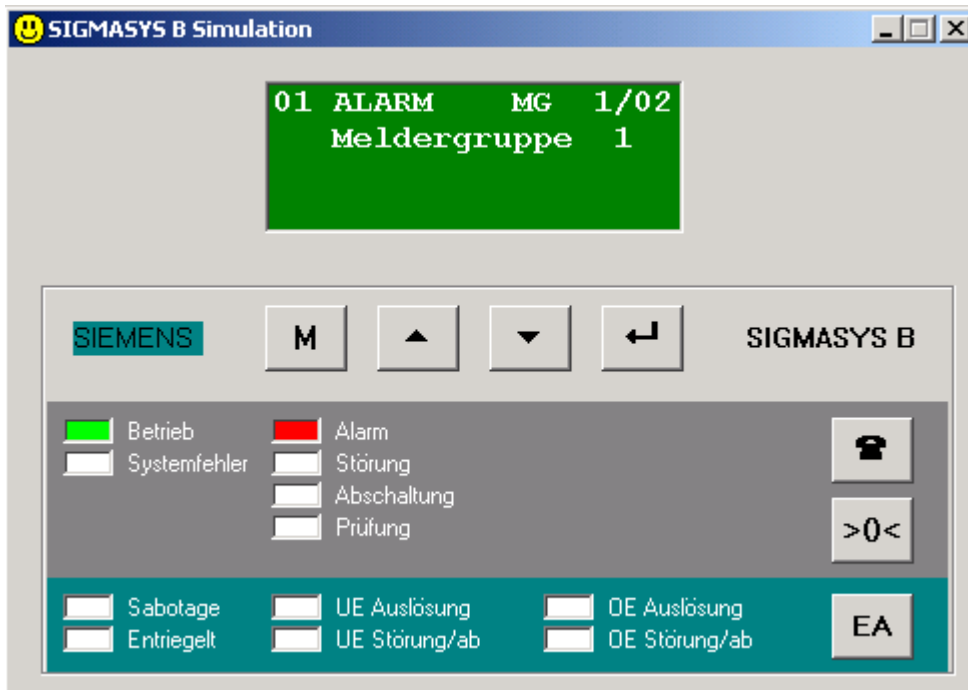


Bild 1: Simulation der Bedienung auf Windows-PC

Über ein kleines Textfenster können zudem über einfache Kommandos externe Ereignisse (Alarmer, Störungen und Wartungsanforderungen) simuliert sowie die Konfiguration der Anlage angezeigt werden.

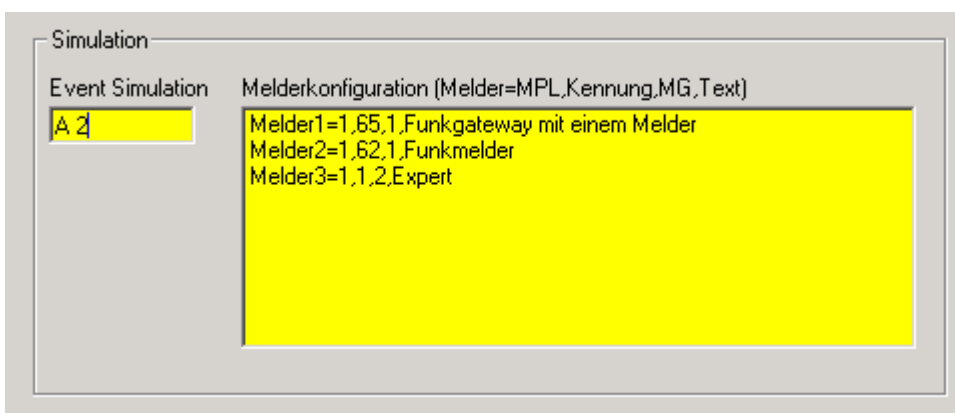


Bild 2: Simulation externer Ereignisse

Für die externen Ereignisse stehen folgende Ereigniskommandos zur Verfügung:

Kommando	Beispiel	Beschreibung
A<n>	A1	Alarm von Melder Nummer 1
S<n>	S3	Störung von Melder Nummer 3
W<n>	W4	Wartungsanforderung Melder Nummer 4

Die folgende Tabelle zeigt, wie die hardwareabhängigen Komponenten der Brandmeldezentrale anhand von Win32-Funktionen nachgebildet werden:

Hardware	Win32-Funktion
LCD-Anzeige	4x20 Zeilen Listbox
Tasten	Command-Buttons
LEDs	Labels (Farbänderung)
Alarmer/Störungen, Wartungen	Über Kommandos in Eingabefeld.
Anlagenkonfiguration	Einlesen aus Konfigurationsdatei

Firmware als DLL

Die gesamte in C geschriebene Firmware der Brandmeldezentrale wird (durch Conditional Compile #if WIN32 etc.) so erweitert, dass sie auch auf PCs unter Windows®-XP ablauffähig ist und mit der Visual Studio Entwicklungsumgebung übersetzt werden kann. Dazu wurde folgendermaßen vorgegangen:

1. Alle hardwareabhängigen Komponenten werden gekapselt und mit #if WIN32 bzw. #ifndef WIN32 „separiert“.
2. Für alle hardwareabhängigen Komponenten werden entsprechende WIN32 Funktionen erstellt.
3. Alle für den Ablauf unter Windows nötigen Funktionen werden in zwei Modulen (win32.c, win32.h) gekapselt.
4. Der Firmware-Code der Brandmeldezentrale wird in eine Windows DLL („Firmware.dll“) gepackt.
5. Die Kommunikation der VB Bedienoberfläche mit der Firmware-DLL erfolgt über ein definiertes API.

Kommunikation der VB.Net Bedienoberfläche mit der Firmware-DLL

Die Kommunikation der VB-Bedienoberfläche mit der Firmware-DLL erfolgt über die folgenden API-Funktionen:

API-Funktion	Bedeutung
SIM_Init	Initialisierung der DLL, Übergabe der Handles für LCD-Anzeige, LEDs etc.
SIM_KeyPress	Übergabe der entsprechenden „Tastencodes“ an DLL
SIM_InitMain	Aufruf der Firmware-Initialisierung
SIM_RunMain	Aufruf der Firmware-Hauptschleife (main()) Von Windows-Timer gesteuert (z.B. Aufruf alle 250 ms).
SIM_ParseCommand	Übergabe der eingegebenen "externen Events" (Alarmer, Störungen, Wartungen) an die Firmware-DLL.

Die Kommunikation der Firmware-DLL mit der Bedienoberfläche erfolgt über WIN32-Messages an die entsprechenden Elemente der VB-Bedienoberfläche (LCD-Anzeige und LEDs).

Simulation der Firmware-Interrupts

Eine ganz wesentliche Rolle in "embedded" Geräten spielen durch Interrupts signalisierte asynchrone Ereignisse. Darum ist die Nachbildung von Interrupts bzw. Interrupt-Handlern bei der Simulation von "embedded" Geräten unabdingbar.

Jeder Firmware-Interrupt sollte durch eine entsprechende Win32 Funktion simuliert werden können. Dem Erfindungsreichtum des Entwicklers sind hier in einer Windows-Umgebung keinerlei Grenzen gesetzt. Zwei Beispiele seien erwähnt, Timer-Interrupts und serielle Interrupts, die Daten von externen seriellen Komponenten empfangen sollen.

Timer-Interrupts können durch (fast identische) Win32 Callback-Funktionen ersetzt werden, die periodisch von Windows-Timern aufgerufen werden.

Beispiel:

```
#if WIN32
void CALLBACK tmrIRQHandler(HWND hWnd, // window handle for timer msg
                           UINT uiMsg, // WM_TIMER message
                           UINT uiEvent, // timer identifier
                           DWORD dwTime) // current system time
#else
interrupt void tmrIRQHandler (void)
#endif
{
    // Code
}
```

Ein serieller Interrupt kann z.B. durch einen separaten Win32-Thread nachgebildet werden. Dieser Thread hängt mit WaitCommEvent() an der jeweiligen PC COM-Schnittstelle, sobald Zeichen eintreffen, werden diese "aufgesammelt" und an die entsprechende Verarbeitungsfunktion weitergereicht.

Beispiel:

```
#if WIN32
void comReadThread (void)
{
    while (1)
    {
        // Wait timed for serial event
        rt = WaitCommEvent(hCom, &dwComMask, NULL );
        if (rt)
        {
            rt = ReadFile(hCom, &ch, 1,&dwBytesRead, NULL );
            // Code
        }
    }
}
#else
interrupt void comIRQHandler (void)
{
    // Code
}
#endif
```

Simulation weiterer Hardware-Komponenten

Auch für die Simulation anderer Hardware-Komponenten lässt sich meist eine adäquate und praxisnahe Lösung finden. Die folgende Tabelle, die die zuvor schon erwähnten und darüberhinaus andere vielfach eingesetzte Hardwarekomponenten den entsprechenden "Simulationskomponenten" gegenüberstellt, erhebt keinen Anspruch auf Vollständigkeit, sie soll lediglich als Beispiel für die beschriebene Vorgehensweise dienen.

Hardwarekomponente	"Simulationskomponente"	Bemerkungen
CPU-Register	"Globale" C-Variable/Macros o.ä.	
Statisches RAM	Array	
Flash-EEPROM	ASCII oder Binärfile	Inhalt bleibt auch nach Ausschalten des Systems erhalten
CMOS-RAM (batteriegepuffert)	Binärfile	Inhalt bleibt auch nach Ausschalten des Systems erhalten
Timer	Windows-Timer	
DMA	DMA-Thread + "DMA-Memory"	Läuft im Hintergrund (wie DMA) und schreibt Daten in "DMA-Memory" (Array).
Dual-Prozessor-System	Zwei Threads oder zwei getrennte WIN32-Anwendungen	
LCD-Anzeige	VB.Net List- oder Textbox	
LEDs	Farbiges VB-Label	
Tasten	PC-Keyboard oder "Command Buttons"	
RTC-Baustein	"PC-Uhr"	SYSTEMTIME GetLocalTime() SetLocalTime()

Zusammenfassung

Mit Hilfe des beschriebenen Vorgehens ist eine zeit- und kostenoptimale Entwicklung von Firmware für "embedded" Geräte auch ohne die eigentliche Hardware möglich. Ebenso lassen sich Änderungen an bereits existierender Firmware wesentlich vereinfachen.

Was bleibt, ist das Programmieren und Testen der hardwareabhängigen Komponenten (Registerzugriffe, Interrupts, DMA etc.) auf der eigentlichen Zielhardware. Da dafür aber oftmals Beispielprogramme oder "Evaluations-Boards" der CPU-Hersteller vorliegen, ist dieser Schritt zwar oft immer noch zeitintensiv, sollte aber kein unüberwindliches Hindernis auf dem Weg zu einer funktionsfähigen Firmware mehr darstellen.