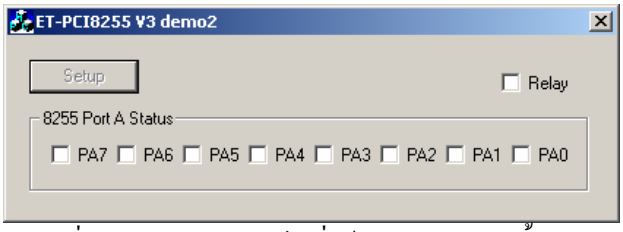
การเขียนโปรแกรมติดต่อกับ ET-PCI8255 V3
ตอนที่ 2 การส่งออกโดยใช้ตัวตั้งเวลา

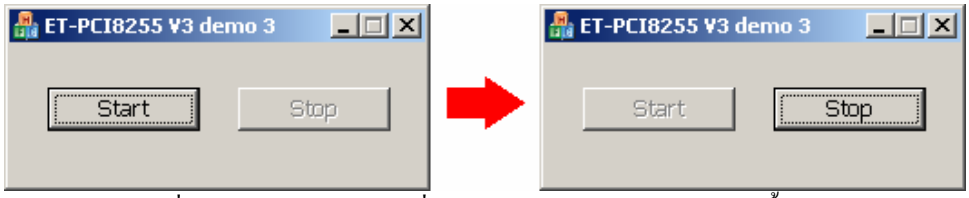
ศุภชัย บุศราทิจ
คณะเทคโนโลยีสารสนเทศ มหาวิทยาลัยราชภัฏเพชรบุรี

จากตัวอย่างของบทความครั้งที่แล้วผมได้ให้ตัวอย่างการส่งข้อมูลไปที่ PA1 ของ ET-PCI8255 V3 โดยเขียนด้วย Microsoft Visual C++ 6 คงจะพอเห็นแนวทางว่าจะนำไปปรับปรุงต่อไปอย่างไร และในบทความนั้นผมได้ลงท้ายว่าตอนต่อไปจะเป็นตัวอย่างการควบคุมโดยการส่งข้อมูลจากตัวตั้งเวลา (Timer) แทนการส่งทุกครั้งที่มีการคลิกบนคอนโทรล

เพื่อให้บทความมีประโยชน์มากขึ้น ในบทความตอนที่ 2 นี้ผมเขียนตัวอย่างโปรแกรมขึ้นมา 2 ตัวอย่าง โดยตัวอย่างแรกนั้นเป็นการปรับเปลี่ยนเล็กน้อยๆจากบทความตอนก่อนหน้านี้ (รูปที่ 1) และอีก 1 ตัวอย่างเป็นการเขียนด้วย Microsoft Visual C++ .NET 2003 เพื่อทำการสั่งไฟวิ่งจาก PA1 ไป PB1 และไปยัง PC1 หลังจากนั้นวนกลับมาเริ่มต้นที่ PA1 อีกครั้งแล้วทำต่อไปเรื่อยๆจนกว่าจะสั่งปิดโปรแกรมหรือคลิกที่ปุ่มจบโปรแกรม (รูปที่ 2)



รูปที่ 1 ตัวอย่างโปรแกรมไฟวิ่งด้วย PA1 โดยการตั้งเวลา



รูปที่ 2 ตัวอย่างโปรแกรมไฟวิ่งด้วย PA1 PB1 และ PC1 โดยการตั้งเวลา

รูปที่ 3 ตัวอย่างการทำงานของโปรแกรมที่ 2

อุปกรณ์ประกอบการทดลอง

1. เครื่องคอมพิวเตอร์ส่วนบุคคลที่มีช่องเสียบการ์ดแบบ PCI เหลืออยู่จำนวน 1 ช่อง
2. การ์ด ET-PCI8255 V3
3. บอร์ด ET-Test I/O

ตัวอย่างโปรแกรมที่ 1

ตัวอย่างนี้ปรับเปลี่ยนจากบทความครั้งที่แล้วในเรื่องของวิธีการส่งข้อมูลที่เปลี่ยนจากการส่งทุกครั้งที่มีการคลิกคอนโทรลที่ใช้ติดต่อกับบิตของ PA1 มาเป็นทำการส่งเป็นระยะเวลาที่ตั้งไว้

การตั้งเวลาใน MFC นั้นมี 3 ส่วนที่ต้องทำความเข้าใจคือ การสร้างตัวตั้งเวลา การตอบสนองเหตุการณ์ที่เกิดจากการตั้งเวลา และการลบตัวตั้งเวลา

การตั้งเวลา

การตั้งเวลานั้นจะต้องใช้คำสั่งของ Win32 API (Application Programming Interface) ดังรูปแบบต่อไปนี้

```
UINT_PTR  SetTimer( UINT_PTR nIDEvent,
                    UINT nElapse,
                    void (CALLBACK* lpfnTimer)( HWND,
                                                UINT,
                                                UINT_PTR,
                                                DWORD)
                  );
```

โดยที่ nIDEvent คือ หมายเลขตัวตั้งเวลาที่เรากำหนดขึ้นเพื่อใช้งาน ซึ่งเป็นค่าใดๆที่ไม่ใช่ศูนย์
nElapse คือ ค่าเวลาที่ตั้งให้ทำงาน มีหน่วยเป็นมิลลิวินาที (millisecond) เช่น ถ้ากำหนดเป็น 1000 จะหมายความว่ามีการเรียกฟังก์ชั่นตอบสนองการตั้งเวลาทุกๆ 1 วินาที
lpfnTimer คือ ฟังก์ชั่นที่ตอบสนองการตั้งเวลา แต่ในที่นี้ให้กำหนดเป็น 0

จากรูปแบบ เมื่อนำมาใช้กับโปรแกรมตัวอย่างผู้เขียนได้กำหนดค่าเอาให้ตัวตั้งเวลามีหมายเลขเป็น 1 และทำงานทุก 50 มิลลิวินาที จะสามารถเขียนโค้ดโปรแกรมได้ดังนี้

```
m_timer = SetTimer(1, 50, 0);
```

การตอบสนองเหตุการณ์ที่เกิดจากการตั้งเวลา

หลังจากกำหนดเวลาให้ตัวตั้งเวลาทำงานตามที่ต้องการเสร็จแล้ว ระบบปฏิบัติการจะส่งเหตุการณ์กับมาที่โปรแกรมของเราทุกครั้งเมื่อถึงเวลาที่เราตั้งเอาไว้ ซึ่งการดักเหตุการณ์นั้นจะต้องดักที่ WM_TIMER เมื่อใช้ผ่านทางคลาสวิซาร์ด (Class Wizard) จะได้โค้ดต้นแบบดังนี้

```
void CEt8255v3dem1Dlg::OnTimer(UINT nIDEvent)
{
	// TODO: Add your message handler code here and/or …
	CDialog::OnTimer(nIDEvent);
}
```

หน้าที่ของเราเหลือเพียงแค่เพิ่มคำสั่งที่ต้องการหลังบรรทัด // TODO:… เท่านั้น โดย nIDEvent นั้นจะเป็นหมายเลขตัวตั้งเวลาที่จะต้องตอบสนอง นั่นหมายความว่าถ้ามีการตั้งเวลาหลายตัว เราจะต้องตรวจสอบก่อนว่า nIDEvent เท่ากับหมายเลขตัวตั้งเวลาที่กำหนดไว้หรือไม่ ไม่เช่นนั้นโปรแกรมจะทำงานผิดพลาดได้

ในที่นี้เราต้องการให้มีการส่งข้อมูลไปที่ PA1 ทุกครั้งที่ถึงเวลาที่กำหนดไว้ จะสามารถเขียนโค้ดได้ดังนี้

```
void CEt8255v3dem1Dlg::OnTimer(UINT nIDEvent)
{
	// TODO: Add your message handler code here and/or call default
	Out32(BaseAddress+PA1,InOutData);

	CDialog::OnTimer(nIDEvent);
}
```

การลบตัวตั้งเวลา

การลบตัวตั้งเวลานั้นมีคำสั่งดังรูปแบบต่อไปนี้

```
BOOL KillTimer( UINT_PTR  nIDEvent );
```

โดยที่ nIDEvent คือ ค่าหมายเลขตัวตั้งเวลาที่เรากำหนดในตอนสร้างด้วย SetTimer(…)

ถ้าฟังก์ชั่นทำงานถูกต้องจะคืนค่ากลับมาเป็นจริง หรือ TRUE แต่ถ้าค่าที่เรากำหนดไปนั้นไม่มีการตั้งเอาไว้หรือลบไปแล้วจะคืนค่ากลับมาเป็นเท็จหรือ FALSE

จากตัวอย่างตั้งแต่การตั้งเวลาและการตอบสนองเหตุการณ์ โค้ดสำหรับการลบตัวตั้งเวลาสามารถเขียนได้ดังต่อไปนี้

```
if (m_timer) {
	InOutData = 0x00;
	Out32(BaseAddress+PA1,InOutData);

	KillTimer(m_timer);
}
```

สาเหตุที่ต้องมีการเปรียบเทียบก่อนลบนั้นเพื่อสร้างความมั่นใจว่า m_timer นั้นได้ถูกสร้างขึ้นมาแล้ว เพราะถ้ายังไม่ถูกสร้าง จะมีความเสี่ยงต่อ 2 คำสั่งต่อไปที่จะส่งค่า 0 ไปที่พอร์ต PA1

ตัวอย่างโปรแกรมที่ 2

ในตัวอย่างนี้มีการทำงานดังที่ได้กล่าวเอาไว้ก่อนหน้านี้แล้ว และได้ทำการปรับเปลี่ยนโดยการแยกโค้ดโปรแกรมสำหรับติดต่อกับฮาร์ดแวร์ออกมาเป็นแฟ้มชื่อว่า pio32.h ซึ่งมีรายละเอียดดังต่อไปนี้

แฟ้ม pio32.h

```
/*
 * Filename     : pio32.h
 * Author       : ศุภชัย บุศราทิจ (Supachai Busaratid)
 *                raek@etteam.com
 * Compiler     : Microsoft Visual C++ .NET 2003
 * OS           : Microsoft Windows 2000 Professional
 * PC Hardware : AMD 3200+
 * Controller   : ET-PCI8255 v3
 * I/O          : ET-Test I/O
 * Date         : ๑๘ พฤษภาคม ๒๕๔๙ (May 18, 2006)
 */

#include <windows.h>

#pragma once
#pragma comment(lib,"inpout32.lib")

/* ฟังก์ชั่นใน inpout32.dll */
short _stdcall Inp32(short PortAddress);
void _stdcall Out32(short PortAddress, short data);

/* ตัวแปรภายนอกสำหรับ ET-PCI8255 V3     */
SHORT BaseAddress;
BYTE  InOutData, CheckData;
SHORT IO_BaseAddress;
      // Tiger-320 Register offset
BYTE PIB,  // Reset & PIB cycle
       AUXC, // AUX Direction port
       AUXD, // AUX Data port
       PA1, PB1, PC1, PCC1, // 8255 Port A/B/C/Control #1
       PA2, PB2, PC2, PCC2, // 8255 Port A/B/C/Control #2
       PA3, PB3, PC3, PCC3, // 8255 Port A/B/C/Control #3
       ON_Bit0,  // XXXX XXXX OR 0000 0001 = XXXX XXX1
       OFF_Bit0, // XXXX XXXX AND 1111 1110 = XXXX XXX0
       ON_Bit1,  // XXXX XXXX OR 0000 0010 = XXXX XX1X
       OFF_Bit1, // XXXX XXXX AND 1111 1101 = XXXX XX0X
       ON_Bit2,  // XXXX XXXX OR 0000 0100 = XXXX X1XX
       OFF_Bit2, // XXXX XXXX AND 1111 1011 = XXXX X0XX
       ON_Bit3,  // XXXX XXXX OR 0000 1000 = XXXX 1XXX
       OFF_Bit3, // XXXX XXXX AND 1111 0111 = XXXX 0XXX
       ON_Bit4,  // XXXX XXXX OR 0001 0000 = XXX1 XXXX
       OFF_Bit4, // XXXX XXXX AND 1110 1111 = XXX0 XXXX
```

```
        ON_Bit5,  // XXXX XXXX OR 0010 0000 = XX1X XXXX
        OFF_Bit5, // XXXX XXXX AND 1101 1111 = XX0X XXXX
        ON_Bit6,  // XXXX XXXX OR 0100 0000 = X1XX XXXX
        OFF_Bit6, // XXXX XXXX AND 1011 1111 = X0XX XXXX
        ON_Bit7,  // XXXX XXXX OR 1000 0000 = 1XXX XXXX
        OFF_Bit7; // XXXX XXXX AND 0111 1111 = 0XXX XXXX



/* ฟังก์ชั่นการทำงาน */

void pio32_init(SHORT base)
{
      SHORT SetupData;

      IO_BaseAddress = base; // I/O Base address for ET-PCI8255 V3

      PIB       = 0x00; // Reset & PIB cycle
      AUXC      = 0x02; // AUX Direction port
      AUXD      = 0x03; // AUX Data port

      PA1       = 0xC0;
      PB1       = 0xC4;
      PC1       = 0xC8;
      PCC1      = 0xCC; // 8255 Port A/B/C/Control #1
      PA2       = 0xD0;
      PB2       = 0xD4;
      PC2       = 0xD8;
      PCC2      = 0xDC; // 8255 Port A/B/C/Control #2
      PA3       = 0xE0;
      PB3       = 0xE4;
      PC3       = 0xE8;
      PCC3      = 0xEC; // 8255 Port A/B/C/Control #3

      ON_Bit0  = 0x01; // XXXX XXXX OR 0000 0001 = XXXX XXX1
      OFF_Bit0 = 0xFE; // XXXX XXXX AND 1111 1110 = XXXX XXX0
      ON_Bit1  = 0x02; // XXXX XXXX OR 0000 0010 = XXXX XX1X
      OFF_Bit1 = 0xFD; // XXXX XXXX AND 1111 1101 = XXXX XX0X
      ON_Bit2  = 0x04; // XXXX XXXX OR 0000 0100 = XXXX X1XX
      OFF_Bit2 = 0xFB; // XXXX XXXX AND 1111 1011 = XXXX X0XX
      ON_Bit3  = 0x08; // XXXX XXXX OR 0000 1000 = XXXX 1XXX
      OFF_Bit3 = 0xF7; // XXXX XXXX AND 1111 0111 = XXXX 0XXX
      ON_Bit4  = 0x10; // XXXX XXXX OR 0001 0000 = XXX1 XXXX
      OFF_Bit4 = 0xEF; // XXXX XXXX AND 1110 1111 = XXX0 XXXX
      ON_Bit5  = 0x20; // XXXX XXXX OR 0010 0000 = XX1X XXXX
      OFF_Bit5 = 0xDF; // XXXX XXXX AND 1101 1111 = XX0X XXXX
      ON_Bit6  = 0x40; // XXXX XXXX OR 0100 0000 = X1XX XXXX
      OFF_Bit6 = 0xBF; // XXXX XXXX AND 1011 1111 = X0XX XXXX
      ON_Bit7  = 0x80; // XXXX XXXX OR 1000 0000 = 1XXX XXXX
      OFF_Bit7 = 0x7F; // XXXX XXXX AND 0111 1111 = 0XXX XXXX


            BaseAddress = IO_BaseAddress;
      // Initial Reset and 8255 Bus Cycle
      SetupData = Inp32(BaseAddress+PIB); // Read PIB Reset Port
      SetupData = SetupData & OFF_Bit0; // Bit0 = EXTRST# = "0" (Reset:RES#)
      SetupData = SetupData | ON_Bit5;    // Bit5:4 = 11 = PIB Cycle Slowest
      SetupData = SetupData | ON_Bit4;
      Out32(BaseAddress+PIB,SetupData);   // Active RES# & Relay
```

```
	// Initial Data (AUX) For CS# and Relay //
	SetupData = Inp32(BaseAddress+AUXD); // Read Aux Data Port
	SetupData = SetupData & OFF_Bit0; // Bit0 = Aux0 = "0" (Enable CS)
	SetupData = SetupData | ON_Bit4; // Bit4 = Aux4 = "1" (Relay OFF)
	Out32(BaseAddress+AUXD,SetupData); // Active Chips Select & Relay

	// Initial Direction (AUX) For CS# and Relay //
	SetupData = Inp32(BaseAddress+AUXC); // Read Aux Port Direction
	SetupData = SetupData | ON_Bit4; // Aux4 = "1" = Output
	SetupData = SetupData | ON_Bit0; // Aux0 = "1" = Output
	Out32(BaseAddress+AUXC,SetupData); // Setup Aux Direction
}

BYTE pio32_read(BYTE portx)
{
	InOutData = Inp32(BaseAddress+portx); // Read Output Latch Port-X
	return InOutData;
}

void pio32_write(BYTE portx, BYTE data)
{
	Out32(BaseAddress+portx,data); // Update Port-X
}
```

จากโค้ดของ pio32.h จะมีฟังก์ชั่นสำคัญ 3 ส่วนคือ pio32_init( ) pio32_read( ) และ pio32_write( ) ที่ทำหน้าที่ดังนี้

pio32_init(SHORT base)

ทำหน้าที่กำหนดค่าเริ่มต้นในการทำงาน โดยเราจะต้องกำหนดค่าตำแหน่ง base ของการ์ด PCI เป็นพารามิเตอร์ ส่วนวิธีการดูค่าตำแหน่งนั้นให้ดูจากรูปที่ 4

pio32_read(BYTE portx)

ทำหน้าที่อ่านข้อมูลจากแลทช์ (Latch) ของไมโครคอนโทรลเลอร์แล้วคืนค่ากลับมา ในการใช้งานนั้นจะต้องส่งพารามิเตอร์เป็นหมายเลขพอร์ตที่ต้องการติดต่อ อันได้แก่ AUXC AUXD PA1 PB1 PC1 PCC1 PA2 PB2 PC2 PCC2 PA3 PB3 PC3 และ PCC3

pio32_write(BYTE portx, BYTE data)

ทำหน้าที่ส่งข้อมูลไปที่แลทช์ (Latch) ของไมโครคอนโทรลเลอร์ ในการใช้งานจะต้องส่งพารามิเตอร์ 2 ตัว คือ portx ที่เป็นตำแหน่งของพอร์ตที่เราต้องการส่งข้อมูล และ data ที่เป็นค่าที่เราต้องการส่งไป

นอกจากนี้ยังมีการเรียกไดเร็กทีฟ (Directive) พิเศษอีก 2 คำสั่งคือ

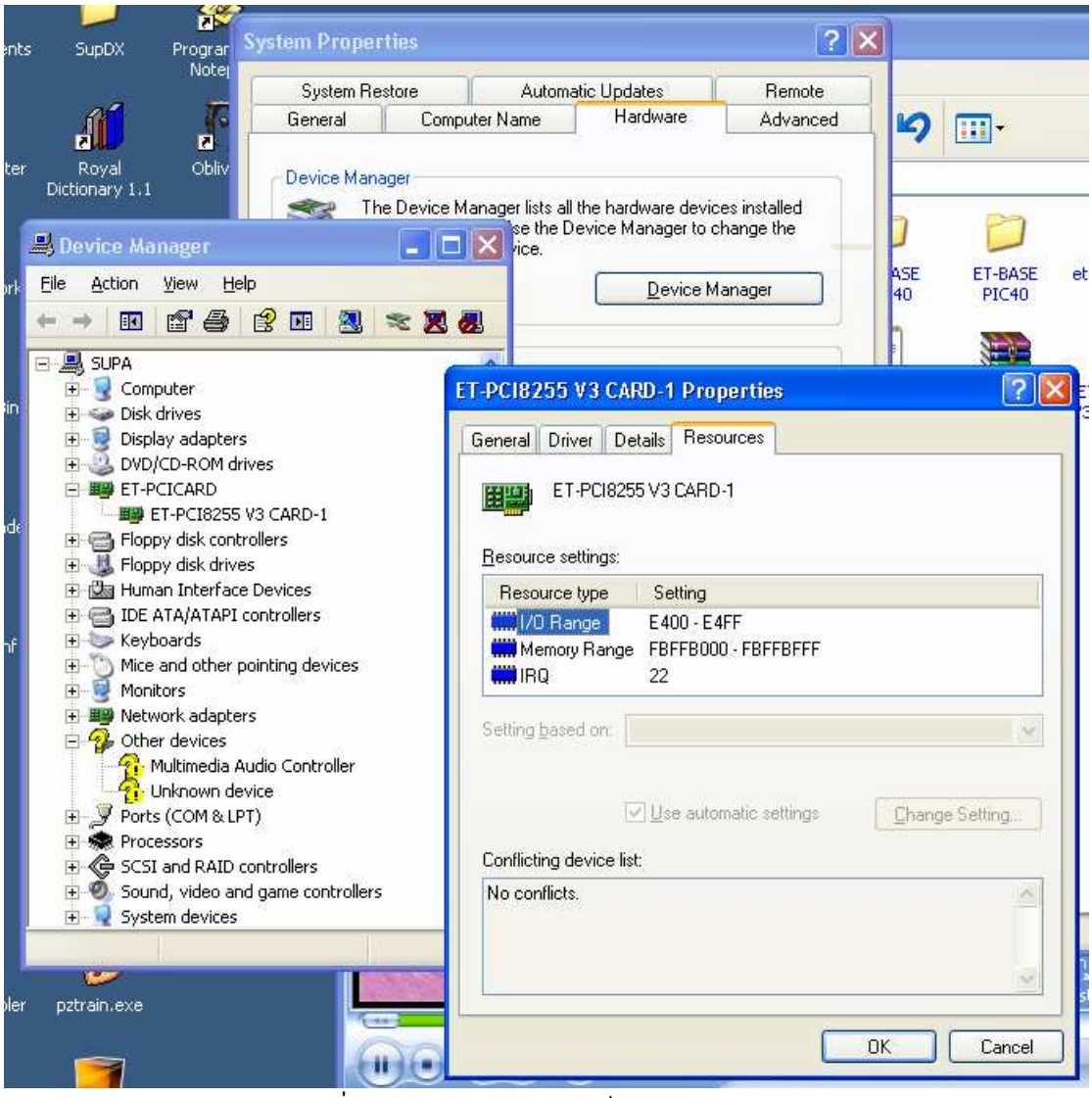
#pragma once

มีความหมายว่าให้เรียกแฟ้มนี้เพียงครั้งเดียว ซึ่งมีประโยชน์เหมือนกับการใช้

```
#ifndef _PIO32_H
#define _PIO32_H
…
#endif // _PIO32_H
```

#pragma comment(lib,"inpout32.lib")

มีความหมายว่าในการเชื่อมโยงแฟ้มเพื่อสร้างโค้ดปลายทางให้นำแฟ้ม inpout32.lib มาทำการเชื่อมโยงด้วย ข้อดีของคำสั่งนี้ก็คือ ทำให้เราไม่ต้องเพิ่มแฟ้ม inpout32.lib เข้าในแฟ้มโครงการ (Project File) เพราะเป็นกำหนดให้เรียกโดยอัตโนมัติ



รูปที่ 4 การดูค่าตำแหน่งของการ์ด ET-PCI8255 V3

แฟ้ม **etPCI8255demo3Dlg.h**

```
// etPCI8255demo3Dlg.h : header file
//

#pragma once
#include "afxwin.h"


// CetPCI8255demo3Dlg dialog
class CetPCI8255demo3Dlg : public CDialog
{
// Construction
public:
	CetPCI8255demo3Dlg(CWnd* pParent = NULL);	// standard constructor

// Dialog Data
	enum { IDD = IDD_ETPCI8255DEMO3_DIALOG };

	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support


// Implementation
protected:
	HICON m_hIcon;

	// Generated message map functions
	virtual BOOL OnInitDialog();
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	DECLARE_MESSAGE_MAP()
public:
	afx_msg void OnBnClickedButton1();
	afx_msg void OnBnClickedButton2();
	CButton btnStop;
	CButton btnStart;
	int m_timer;
	afx_msg void OnTimer(UINT nIDEvent);
	afx_msg void OnClose();
	unsigned char m_state;
	int m_port;
};
```

แฟ้ม **etPCI8255demo3Dlg.cpp**

```
// etPCI8255demo3Dlg.cpp : implementation file
//

#include "stdafx.h"
#include "etPCI8255demo3.h"
#include "etPCI8255demo3Dlg.h"
#include ".\etpci8255demo3dlg.h"

#include "pio32.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif
```

```
// CetPCI8255demo3Dlg dialog



CetPCI8255demo3Dlg::CetPCI8255demo3Dlg(CWnd* pParent /*=NULL*/)
	: CDialog(CetPCI8255demo3Dlg::IDD, pParent)
	, m_timer(0)
	, m_state(0)
	, m_port(0)
{
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CetPCI8255demo3Dlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_BUTTON2, btnStop);
	DDX_Control(pDX, IDC_BUTTON1, btnStart);
}

BEGIN_MESSAGE_MAP(CetPCI8255demo3Dlg, CDialog)
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	//}}AFX_MSG_MAP
	ON_BN_CLICKED(IDC_BUTTON1, OnBnClickedButton1)
	ON_BN_CLICKED(IDC_BUTTON2, OnBnClickedButton2)
	ON_WM_TIMER()
	ON_WM_CLOSE()
END_MESSAGE_MAP()



// CetPCI8255demo3Dlg message handlers

BOOL CetPCI8255demo3Dlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here

	return TRUE;  // return TRUE  unless you set the focus to a control
}



// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CetPCI8255demo3Dlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND,
reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
```

```
	else
	{
		CDialog::OnPaint();
	}
}



// The system calls this function to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CetPCI8255demo3Dlg::OnQueryDragIcon()
{
	return static_cast<HCURSOR>(m_hIcon);
}


void CetPCI8255demo3Dlg::OnBnClickedButton1()
{
	// TODO: Add your control notification handler code here
	btnStop.EnableWindow(1);
	btnStart.EnableWindow(0);
	btnStop.SetFocus();
	m_timer = 0;
	m_timer = SetTimer(1,100,0);
	if (m_timer) {
		m_port = 1; /* 1 = PA1, 2 = PB1, 3 = PC1 */
		m_state = 0x01;
		pio32_init(0xE400);
		pio32_write(PCC1,0x80);
	}
}



void CetPCI8255demo3Dlg::OnBnClickedButton2()
{
	// TODO: Add your control notification handler code here
	btnStart.EnableWindow(1);
	btnStop.EnableWindow(0);
	btnStart.SetFocus();
	if (m_timer) {
		KillTimer(m_timer);
		m_timer = 0;
		pio32_write(PA1,0x00);
		pio32_write(PB1,0x00);
		pio32_write(PC1,0x00);
	}
}


void CetPCI8255demo3Dlg::OnTimer(UINT nIDEvent)
{
	// TODO: Add your message handler code here and/or call default
	if (m_timer) {
		//...
		pio32_write(PA1,0x00);
		pio32_write(PB1,0x00);
		pio32_write(PC1,0x00);
		switch (m_port) {
			case 1: pio32_write(PA1,m_state); break;
			case 2: pio32_write(PB1,m_state); break;
			case 3: pio32_write(PC1,m_state); break;
		}
		m_state = m_state << 1;
		if (m_state == 0x00) {
			m_state = 0x01;
			m_port = m_port+1;
			if (m_port > 3) m_port = 1;
		}
	}
	CDialog::OnTimer(nIDEvent);
}
```

```
void CetPCI8255demo3Dlg::OnClose()
{
	// TODO: Add your message handler code here and/or call default
	if (m_timer) {
		KillTimer(m_timer);
		m_timer = 0;
		pio32_write(PA1,0x00);
		pio32_write(PB1,0x00);
		pio32_write(PC1,0x00);
	}
	CDialog::OnClose();
}
```

สรุป

จากบทความนี้หวังว่าคงพอทำให้นำไปประยุกต์ใช้ได้กันมากขึ้นนะครับ เพราะเรามีฟังก์ชั่นการติดต่อให้ใช้งานกัน 3 ฟังก์ชั่น นั่นคือ pio32_init( ) pio32_read( ) และ pio32_write( ) ส่วนบทความในครั้งต่อไปจะเป็นเรื่องของการรับข้อมูลจาก 8255 อีกตัวหนึ่งแล้วส่งออกไปยัง 8255 อีกตัวหนึ่ง ซึ่งถ้าใครไปลองทำก่อนก็คงไม่ยากครับ

สุดท้ายต้องขอขอบคุณครอบครัวและทีมงานอีทีที โดยเฉพาะคุณกอบกิจ เติมผาติ อย่างมากครับ ที่ยังได้ให้โอกาสผมเขียนบทความต่อไป