Základní informace pro práci s IO kartami v laboratoři MIT

Tomáš Koloušek

podpůrný materiál pro studenty V4 v rámci laboratorních cvičení předmětu praxe

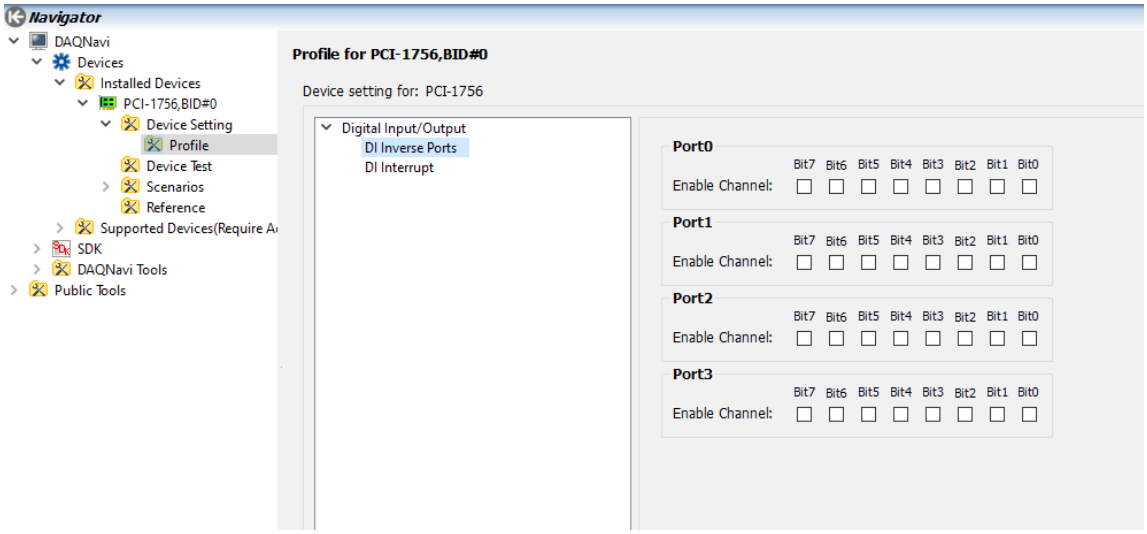
IO karty PCIe-1730 a PCI-1756 od fy Advantech jsou nainstalované v počítači. Jelikož v rámci signálů IO rozhraní používají průmyslovou logiku (0/24V s rozhodovací úrovní 10V), je pro jejich použití v rámci laboratoře vytvořeno připojovací rozhraní, které provádí konverzi na úrovně TTL 0-5V. Toto rozhraní je napájeno z počítače a **po zkontrolování zapojení** si musí student zapojit napájení pomocí vypínače umístěného na bracketu na zadní straně počítače. Zapojení napájení je indikováno oranžovou LED (12V z počítače) a modrou LED (+5V pro IO), umístěnými na desce rozhraní.

Pro snadnější ladění kódu je každý IO signál indikován svou červenou LED spolu s popisem signálu. Indikační LED jsou v pozitivní logice, tj. **svítící LED indikuje přítomnost log.1 na IO lince**.

K připojení periferie slouží šroubovací svorky, na každém portu je taktéž k dispozici digitální zem. Jednotlivé digitální země jsou vnitřně propojeny, k periferii stačí připojit pouze jednu z nich. Porty jsou popsány ve vztahu k IO kartě/počítači: P1 a P2 jsou výstupní porty z počítače (k zapojení do vstupu periferie), P3 a P4 jsou pak vstupní (k zapojení do výstupu periferie).

Šroubovací svorky jsou kompatibilní s „Jihlava konektory“, utáhnout je stačí s citem, žádné velké výkony přes ně nejsou přenášeny. Jedná se o třmenové svorky, před připojením vodiče musí být svorka dostatečně povolena aby následně s utahováním vodič sevřela. Jednotlivé bity IO portu jsou popsány na svorkovnici (dle pořadí na původních IO kartách), pořadí signálu na rozhraní nemá vliv na fyzické mapování IO linky do binární hodnoty při přístupu z programového kódu (bit 0 je tedy v hodnotě vždy vpravo jako LSB, nejvyšší bit 7 potom vlevo).

V rámci instalace IO karet je k dispozici i základní ovládací a testovací nástroj *Navigator*, pomocí kterého je možné ručně nastavovat stav výstupů a číst aktuální stav vstupů v rámci *Device Tests* záložky navigačního stromečku pro konkrétní IO kartu. Je tak možné si např. v rychlosti ověřit princip fungování periferie, správnost zapojení jednotlivých ovládacích signálů periferie na jednotlivé IO linky karty nebo odpovídající reakci vstupu na měnící se výstupní informaci z periferie.



Ve stejném nástroji je nutné před vlastním použitím v kódu vytvořit *profil* použití karty. V rámci profilu je možné např. invertovat některé signály (tj. log.1 v kódu odpovídá log.0 na fyzickém rozhraní a naopak). Doporučil bych však v rámci hw profilu karty nic neměnit, ponechat všechny checkboxy ve výchozím nezaškrtnutém stavu a všechny inverzní signály aktivní log.0 řešit přímo na

úrovni programového kódu. Profil karty je potřeba uložit do xml souboru pomocí tlačítka *„save as“* pro pozdější načtení do aplikace.

Pro přístup ke kartě ze C# je používána knihovna *Automation.BDAQ*, která je dostupná buď z lokálního adresáře v počítači s kartou (standardně C:\Program Files\Advantech\DAQNavi\Automation.Bdaq\4.0.0.0), příp. je stejná knihovna dostupná v rámci nástroje NuGet z vývojového prostředí.

Pro přístup ke kartě je nutné mít v rámci kódu instanci třídy *DeviceInformation*, která popisuje na jaké kartě v rámci aplikace chceme pracovat. V rámci jedné aplikace je možné pracovat i s několika kartami naráz, tento setup však není nikde v rámci laboratoře MIT použit. Instanci této třídy je nutné jednoznačně identifikovat používanou kartu pomocí property *DeviceInformation*, do které je nutné nastavit přesné označení karty z Navigatoru. Pro karty PCI-1756 je nutné dále pomocí property *DeviceMode* nastavit režim pro zápis pomocí výčtové hodnoty *AccessMode.ModeWrite*. Bez tohoto nastavení na kartě PCI-1756 budou veškeré pokusy o zápis končit s chybovým kódem (defaultní nastavení na *AccessMode.ModeRead*):

```
DeviceInformation ioDevice = new DeviceInformation();  // instance IO karty

ioDevice.Description = "PCI-1756,BID#0";       // identifikace karty v systému
ioDevice.DeviceMode = AccessMode.ModeWrite; // nastavení přístupu pro zápis
```

Pro přístup na IO rozhraní karty slouží instance tříd *InstantDiCtrl* (pro vstup do počítače) a *InstantDoCtrl* (pro výstup z počítače). Před použitím instance pro vlastní přístup na IO rozhraní je nutné ji nejprve propojit s konkrétní IO kartou, se kterou bude komunikovat. To je možné přiřazením do property *SelectedDevice*.

```
InstantDoCtrl ioOutput = new InstantDoCtrl(); // instance pro práci s výstupem
InstantDiCtrl ioInput = new InstantDiCtrl();  // dtto pro vstup

ioOutput.SelectedDevice = ioDevice; // spojení výstupu s konkrétní kartou
ioInput.SelectedDevice = ioDevice;  // to samé pro vstup
```

Poté je ještě nutné pro každou instanci třídy pro přístup k IO rozhraní načíst XML profil popisující nastavení aktivních úrovní jednotlivých IO linek, který byl předtím vytvořen v rámci aplikace Navigator:

```
ioOutput.LoadProfile("PCI-1756_profile.xml"); // načtení profilu použité karty
ioInput.LoadProfile("PCI-1756_profile.xml");  // povšimněte si relativní cesty
```

Třídy *InstantDiCtrl* a *InstantDoCtrl* obsahují pro vlastní přenos hodnot metody *Read(int index_portu, out byte hodnota_na_portu)* a *Write(int index_portu, byte hodnota_k_zapisu)*. Zatímco zápis hodnoty na vstupní port nemá praktický význam, čtení hodnoty je možné i z výstupního portu, kde se vrátí aktuální hodnota přítomná na portu (poslední hodnota, která byla na výstup portu zapsána). To se může hodit např. pro bitové operace, kdy je třeba nastavit nebo vynulovat konkrétní bit na portu. Porty jsou indexovány vždy na konkrétní kartě a na daném směru, P1 je tedy výstup 0, P2 je výstup 1, P3 je **vstup 0** a P4 je vstup 1.

Možný příklad použití je v následujícím snippetu kódu. Povšimněte si nutnosti zpětné konverze výstupu logických operací na datový typ byte, který je logickou operací rozšířen na typ int, který ale není možné zpracovat jako parametr metody Write:

```
ioOutput.Write(0, 0b00000111);     // zápis přímé hodnoty na P1
ioOutput.Read(1, out byte stavP2); // načtení aktuálního stavu P2
// změna stavu bitu P2.2 (xor s log.1 v masce), nulování P2.0 (and s 0 v masce)
ioOutput.Write(1, (byte)( (stavP2 ^ (1 << 2)) & ~(1 << 0) ) );

ioInput.Read(0, out byte stavP3);  // načtení stavu P3 pro další zpracování
Console.Write("Tlačítko na P3.2 {0} stisknuté",
                    (stavP3 & (1 << 2)) == 0 ? "je" : "není");
```