PRESTO

PRESTO_APP01 – Popis presto.dll

Aplikační poznámka

ASIX s.r.o.
Na Popelce 38/17
150 00 Praha 5 - Košíře

www.asix.cz

podpora@asix.cz

obchod@asix.cz

ASIX s.r.o. si vyhrazuje právo změny tohoto dokumentu, jehož aktuální podobu naleznete na Internetu.

ASIX s.r.o. nenese žádnou zodpovědnost za škody způsobené použitím produktu firmy ASIX s.r.o.

11.4.2022

Obsah

1

presto.dll

1.1 Úvod

Funkce implementované v presto.dll umožňují na jednotlivých pinech programátoru nastavovat logické úrovně dle potřeby nebo číst jejich stav, takto lze vytvářet různé komunikační protokoly. Pro ovládání všech pinů, které umožňují výstup, je tu funkce *QSetPins()*, pro čtení pinů s možností vstupu funkce *QGetPins()*. Funkcí *QSendByte()* je možné rychle poslat SPI Byte na pinech data a clock, pokud je současně potřeba i číst, použije se funkce *QSendByte_OutIn()*. Dále jsou tu funkce pro nastavení vlastností programátoru, ovládání napětí a funkce pro čtení návratových hodnot.

Tuto knihovnu je možné použít se všemi programátory PRESTO bez ohledu na verzi hardware.

1.2 Značení pinů programátoru

V tomto dokumentu jsou pro větší přehlednost jednotlivé piny programátoru značeny zjednodušeně, jak je naznačeno v následující tabulce.

Název pinu	Označení	Funkce	Poznámka
P1 - VPP	P	I/O, 13 V	logické úrovně nebo programovací napětí
P2		klíč	
P3 - VDD	VDD	napájecí	výstup 5 V napájení nebo vstup externího napájení
P4 - GND	GND	napájecí	
P5 - DATA/ MOSI	D	I/O	rychlý výstup dat funkcí *QShiftByte()*
P6 - CLOCK	C	O	rychlý výstup hodin funkcí *QShiftByte()*
P7 - MISO	I	I	
P8 - LVP	L	I/O	

Tab. 1: Programovací konektor

Význam: I - vstupní pin, O - výstupní pin, I/O - vstupní i výstupní pin, 13 V - programovací napětí

1.3 Způsob práce s programátorem

Příkazy se z podstaty USB vykonávají ve frontě. Čekání na odpověď od každého jednotlivého volání, například *QGetPins()*, by práci velmi zpomalilo.

U některých příkazů, kde je to zvlášť žádoucí, např. *QOpenPresto()*, je vhodné nepokračovat dokud se výsledek nepotvrdí. Cyklus **příkaz → PRESTO → odpověď** trvá většinou okolo jednotek až desítek milisekund.

Ve stejném pořadí, v jakém byly do fronty zadávány příkazy (funkce *Q...()*), se vyčítají jejich odpovědi. Vrácená data je možné číst buď neblokujícím způsobem funkcí *AGet()* nebo blokujícím způsobem funkcí *AGetBlocking()*.

1.4 Seznam funkcí

```
void __stdcall QOpenPresto(int sn);
void __stdcall QClosePresto(void);
void __stdcall QSetPins(int pins);
void __stdcall QGetPins(void);
void __stdcall QPoweronVdd(int delayus);
void __stdcall QPoweroffVdd(void);
void __stdcall QDelay(int delayus);
```

```
void __stdcall QSetActiveLED(bool led);
bool __stdcall AGet(int *answer);
int __stdcall AGetBlocking(void);
void __stdcall AGetProgList(int *sn_list, int co
unt, int *count_returned);
void __stdcall AClearFatalError(void);
void __stdcall QShiftByte(int databyte, int mode
);
void __stdcall QShiftByte_OutIn(int databyte, in
t mode, int InputPin);
void __stdcall QCheckSupplyVoltage(void);
void __stdcall QPoweronVpp13V(void);
void __stdcall QPoweroffVpp13V(void);
void __stdcall QSetDPullup(bool dpullup_on);
void __stdcall QCheckGoButton(void);
void __stdcall QSetPrestoSpeed(int speed);
```

1.5 Popis funkcí

1.5.1 QOpenPresto

Funkce zkusí otevřít PRESTO. Pokud je parametr sn rovný -1, otevře jedno PRESTO bez ohledu na jeho sériové číslo. V případě, že sn není -1, pak jeho hodnota specifikuje sériové číslo programátoru tak, že pokud je sériové číslo PRESTA A6016789, pak sn by mělo být 0x6789. Sériové číslo definují poslední čtyři znaky v hexa tvaru.

void __stdcall QOpenPresto(int sn);

Parametr:

sn - Sériové číslo programátoru.

Návratové hodnoty:

OPEN_OK - otevření se zdařilo

OPEN_NOTFOUND - programátor nenalezen

OPEN_CANNOTOPEN - nelze otevřít

OPEN_ALREADYOPEN - nelze otevřít, programátor je již otevřený

Návratové hodnoty jsou vraceny prostřednictvím funkce AGet() nebo AGetBlocking().

Příklad:

```
QOpenPresto(0x6789); // otevře PRESTO SN A601678
9
```

1.5.2 QClosePresto

Zavře PRESTO a vypne napětí, pokud na jeho výstupu nějaká jsou.

void __stdcall QClosePresto(void);

Návratové hodnoty:

CLOSE_OK

CLOSE_CANNOTCLOSE - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkce AGet() nebo AGetBlocking().

1.5.3 QSetPins

Nastaví výstupní piny programátoru podle konstant. **Pozor:** Pokud je v jednom požadavku více změn pinů najednou, nejprve se zároveň nastaví piny D a C a pak L a P. Nelze tedy udělat hranu na např. L a C zároveň, ale na D a C ano, toho lze využít u sériových komunikací.

void __stdcall QSetPins(int pins);

Parametr:

pins - Definuje požadované hodnoty na pinech programátoru.

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkce AGet() nebo AGetBlocking().

Příklad:

Chci-li nastavit D do log.1, C do log. 0 a stav ostatních signálů nechat nezměněný, zavolám funkci

```
QSetPins((PINS_HI<<PINS_D_BIT)|
(PINS_LO<<PINS_C_BIT));
```

1.5.4 QGetPins

V odpovědi pošle logické úrovně, které PRESTO vidí na pinech D, L, I a P. Pin C nelze číst. Viz konstanty pro AGet().

void __stdcall QGetPins(void);

Návratové hodnoty:

GETPINS_CODE + hodnoty pinů

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkce AGet() nebo AGetBlocking().

Příklad:

Chci-li přečíst stav na pinu I, zavolám funkci *QGetPins()* a následně přečtu vrácená data funkcí *AGet()*. Funkce *AGet()* vrátí např. hodnotu 0x40B. V této hodnotě jsou vráceny stavy všech vstupních pinů, proto vyfiltruji jen stav pro pin I konstantou **GETPINS_PINI**, zde na I je log. 1.

```
if (AGet(data))
 { if ((data & GETPINS_PINI)==GETPINS_PINI )
     {  //on the I pin there is log. 1}
    else {//on the I pin there is log. 0}
 }
```

1.5.5 QDelay

Čeká stanovenou dobu. Granularita časovače je 170,66 µs (12 MHz/2048), zadaná hodnota je zaokrouhlena na nejbližší vyšší násobek 170,66 µs.

void __stdcall QDelay(int delayus);

Parametr:

delayus - Doba čekání v µs.

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkce AGet() nebo AGetBlocking().

Příklad 1:

Chci-li v signálech udělat prodlevu 7 ms, zavolám funkci

```
QDelay(7000);
```

Příklad 2:

Při zavolání *QDelay(5)*, je hodnota zaokrouhlena nahoru a programátor udělá prodlevu 170.66 µs.

1.5.6 QPoweronVdd

Zapne na pin VDD napětí z USB, tj. cca 5 V, čeká stanovenou dobu a zkontroluje, zda proud není větší než 100 mA. Pokud ano, napětí vypne. Napětí při zkratu nebude přítomno výrazně déle než je v parametru stanovená doba. V odpovědi funkce vrátí hodnotu podle výsledku této operace. Přestože odpověď přijde až za cca 20 ms, napětí už je spolehlivě vypnuto, toto je ošetřeno na straně HW. Dobu je třeba volit s rozvahou, dlouhá nastavená doba je při chybě zapojení nebezpečná pro obvody programátoru. V případě, že není zapnuté interní napájení z programátoru, je možné použít externí napětí v rozsahu 2,5 - 5 V. Logické úrovně na datových pinech odpovídají velikosti napětí na VDD.

void __stdcall QPoweronVdd(int delayus);

Parametr:

delayus - Doba v µs po které bude provedena kontrola nadproudu.

Návratové hodnoty:

POWERON_OK - Podařilo se zapnout interní napájení.

POWERON_OCURR - Byl detekován nadproud, napětí bylo vypnuto.

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkce AGet() nebo AGetBlocking().

Příklad:

Chci-li zapnout interní napájení na pin VDD a zkontrolovat nadproud po 10 ms, zavolám funkci

```
QPoweronVdd(10000);
```

1.5.7 QPoweroffVdd

Vypne napětí na pinu VDD.

void __stdcall QpoweroffVdd(void);

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkce AGet() nebo AGetBlocking().

1.5.8 QSetActiveLED

Rozsvítí / zhasne ACTIVE LED na PRESTU.

void __stdcall QSetActiveLED(bool led);

Parametr:

led - Pokud je proměnná True, LED se rozsvítí, pokud je False, LED se zhasne.

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkce AGet() nebo AGetBlocking().

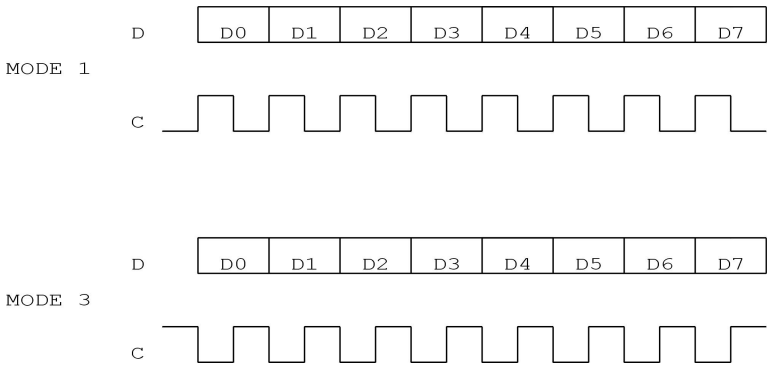
Příklad:

Chci-li rozsvítit ACTIVE LED na programátoru, zavolám funkci *QSetActiveLED(true)*, pro její zhasnutí funkci *QSetActiveLED(false)*.

1.5.9 QShiftByte

Na pinu D odešle 1 Byte specifikovaný proměnnou **databyte** a současně na pinu C generuje hodinový signál. Proměnná **mode** specifikuje mód podle specifikace SPI, podle kterého budou data odeslána, možné jsou módy 1 a 3, ostatní módy funkce nepodporuje, je možné je vytvořit ručně v kombinaci s použitím funkce *QSetPins()*, což bude ale pomalejší. V případě, že je zvolen mód, který neodpovídá současné logické úrovni na pinu C, je tento pin nejprve nastaven do potřebného stavu. Tedy např. pokud je před zavoláním této funkce na C log. 0 a funkce je zavolána s parametrem **mode**=3, C se nejprve nastaví do log. 1 a potom se teprve odešle **databyte**.

Zadaná data jsou posílána počínaje LSB, frekvence hodin odpovídá hodnotě nastavené funkcí *QSetPrestoSpeed()*. Funkce *QShiftByte()* umožňuje rychlejší generování signálů než při použití *QSetPins()*.



Obr. 2: Popis SPI módů

void __stdcall QShiftByte(int databyte, int mode);

Parametry:

databyte - Proměnná pro data, která se mají odeslat.

mode - Proměnná definuje SPI mód, podle zvoleného módu může mít hodnotu 1 nebo 3.

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkce AGet() nebo AGetBlocking().

Příklad:

Chci-li poslat Byte 0x3A v SPI módu 1, zavolám funkci

```
QShiftByte(0x3A, SHIFT_MODE1);
```

1.5.10 QShiftByte_OutIn

Funkce stejně jako *QShiftByte()* vygeneruje signály C a D podle zadaných parametrů, navíc ale ještě současně čte data z pinu zvoleného proměnnou **InputPin**. Viz konstanty pro volbu vstupního pinu. V případě, že je jako vstupní pin zvolen pin D, je před operací nastaven do třetího stavu a data z něj jsou jen čtena.

void __stdcall QShiftByte_OutIn(int databyte, int mode, int InputPin);

Parametry:

databyte - Proměnná pro data, která se mají odeslat.

mode - Proměnná definuje SPI mód, podle zvoleného módu může mít hodnotu 1 nebo 3. V mode 1 jsou data čtena se sestupnou hranou hodin, v mode 3 s náběžnou.

InputPin - Podle hodnoty proměnné je zvolen vstupní pin.

Návratové hodnoty:

SHIFT_BYTE_OUTIN_CODE + přečtená data

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkce AGet() nebo AGetBlocking().

Příklad:

Chci-li poslat Byte 0x4C v SPI módu 3 a současně číst příchozí data na pinu I, zavolám funkci

```
QShiftByte_OutIn(0x4C, SHIFT_MODE3, SHIFT_OUTIN_
PINI);
```

1.5.11 QCheckSupplyVoltage

V odpovědi pošle kód odpovídající napětí, které programátor vidí na pinu VDD. Viz konstanty odpovídající zjištěným napětím.

void __stdcall QCheckSupplyVoltage(void);

Návratové hodnoty:

SUPPLY_VOLTAGE_CODE + SUPPLY_VOLTAGE_xV

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkce AGet() nebo AGetBlocking().

Příklad: Chci-li zkontrolovat napětí na VDD, zavolám funkci *QCheckSupplyVoltage()* a následně přečtu výsledek funkcí *AGet()*. *AGet()* vrátí např. 0x701 z čehož je zřejmé, že na VDD je napětí > 2 V a < 5 V.

```
if (AGet(data))
{if (data == SUPPLY_VOLTAGE_CODE | SUPPLY_VOLTAG
E_5V)
  {// na pinu VDD je 5V}
 else if (data == SUPPLY_VOLTAGE_CODE | SUPPLY_V
OLTAGE_2V)
  {// na pinu VDD je >2V}
 else if (data == SUPPLY_VOLTAGE_CODE | SUPPLY_V
OLTAGE_0V)
  {// na pinu VDD je 0V}
}
```

1.5.12 QPoweronVpp13V

Funkce zapne programovací napětí 13 V na pinu VPP programátoru. Pokud programátor po zapnutí 13 V detekuje na VPP nadproud, je toto napětí opět vypnuto. V odpovědi pošle informaci zda zapnutí VPP proběhlo v pořádku.

void __stdcall QPoweronVpp13V(void);

Návratové hodnoty:

VPP_OK - Programovací napětí bylo zapnuto.

VPP_OCURR - Byl detekován nadproud, programovací napětí bylo opět vypnuto.

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkce AGet() nebo AGetBlocking().

1.5.13 QPoweroffVpp13V

Vypne programovací napětí 13 V na pinu VPP.

void __stdcall QPoweroffVpp13V(void);

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkce AGet() nebo AGetBlocking().

1.5.14 QSetDPullup

Připojí/odpojí pull-up rezistor 2k2 na pinu D. Ve výchozím stavu tento rezistor není připojený.

void __stdcall QSetDPullup(bool dpullup_on);

Parametr:

dpullup_on - Proměnná specifikuje, zda se připojí (dpullup_on=True) nebo odpojí (dpullup_on=False) pull-up na D.

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkce AGet() nebo AGetBlocking().

1.5.15 QCheckGoButton

Zkontroluje tlačítko na programátoru a v odpovědi pošle jeho stav. Viz specifikaci odpovědí.

void __stdcall QCheckGoButton(void);

Návratové hodnoty:

GO_BUTTON_NOT_PRESSED

GO_BUTTON_PRESSED

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkce AGet() nebo AGetBlocking().

Příklad:

Chci-li zjistit, zda je stištěno tlačítko, zavolám funkci *QCheckGoButton()*, a následně pokud funkce *AGet(data)* vrátí 0x901, je tlačítko stištěno.

```
if (data==GO_BUTTON_PRESSED)
{// tlačítko je stištěno}
```

1.5.16 QSetPrestoSpeed

Nastaví rychlost hodin na pinu C. Toto nastavení ovlivňuje rychlost signálů generovaných funkcemi QShiftByte() a QShiftByte_OutIn(), ale i QSetPins(). Viz specifikaci konstant.

void __stdcall QSetPrestoSpeed(int speed);

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkce AGet() nebo AGetBlocking().

Parametr:

speed - Definuje rychlost hodin.

Příklad:

Chci-li nastavit rychlost hodin pro funkce *QShiftByte* na 750 kHz, zavolám funkci *QSetPrestoSpeed(PRESTO_CLK4);*

1.5.17 AGet

Vrátí bool podle toho, jestli je nebo není k dispozici nějaká odpověď, případně vrátí v parametru i hodnotu odpovědi.

bool __stdcall AGet(int *answer);

Návratové hodnoty:

Funkce vrátí True, pokud jsou k dispozici vrácená data, False v opačném případě.

answer - Vrácená hodnota odpovědi.

Příklad:

Chci-li zjistit, zda už přišla odpověď od programátoru a jaká, otestuji to funkcí

```
if (AGet(data))
 { // vrácená hodnota je k dispozici v proměnné
data}
```

1.5.18 AGetBlocking

Čeká, dokud nějaká odpověď nepřijde, poté vrátí hodnotu odpovědi.

int __stdcall AGetBlocking(void);

Návratová hodnota:

Funkce vrátí hodnotu odpovědi.

Příklad:

Chci-li počkat, až přijde odpověď od programátoru a až potom pokračovat, použiji *AGetBlocking()*. Tuto funkci může být vhodné použít např. po otevření PRESTA

```
QOpenPresto(-1);
if (AGetBlocking()==OPEN_OK)
  {// programátor otevřen}
else
  {// nepodařilo se otevřít programátor}
```

1.5.19 AGetProgList

V parametru vrátí seznam SN programátorů PRESTO, které jsou k dispozici.

Definice funkce:

void __stdcall AGetProgList(int *sn_list, int count, int *count_returned);

Parametry:

sn_list - Vrací seznam SN dostupných programátorů PRESTO. Vrací SN v délce 24bitů, jako je zobrazováno v programu UP.

count - Požadovaný počet hodnot.

count_returned - V této proměnné je vrácen počet skutečně vrácených hodnot, počet dostupných programátorů.

Návratové hodnoty:

Bez ohledu na závažné chyby vrací seznam dostupných programátorů v **sn_list** a počet vracených SN v **count_returned**.

1.5.20 AClearFatalError

Smaže závažnou chybu. Po smazání závažné chyby je PRESTO zavřeno a je ho potřeba znovu otevřít. Jakékoli příkazy ve frontě již nebudou provedeny a žádné odpovědi, které měly přijít přes *AGet()* nebo *AGetBlocking()* už nikdy nepřijdou.

void __stdcall AClearFatalError(void);

1.6 Odpovědi

```
OPEN_OK = 0x100;
OPEN_NOTFOUND = 0x101;
OPEN_CANNOTOPEN = 0x102;
OPEN_ALREADYOPEN = 0x103;
CLOSE_OK = 0x200;
CLOSE_CANNOTCLOSE = 0x201;
POWERON_OK = 0x300;
POWERON_OCURR = 0x301;
GETPINS_CODE = 0x400;
 GETPINS_PIND = 0x01;
 GETPINS_PINL = 0x02;
 GETPINS_PINP = 0x04;
 GETPINS_PINI = 0x08;
OK = 0x500;
NOT_OPENED = 0x501;
SHIFT_BYTE_OUTIN_CODE = 0x600;
SUPPLY_VOLTAGE_CODE = 0x700;
 SUPPLY_VOLTAGE_0V = 0x00;
 SUPPLY_VOLTAGE_2V = 0x01;
 SUPPLY_VOLTAGE_5V = 0x03;  {1 or 2}
VPP_OK = 0x800;
VPP_OCURR = 0x801;
GO_BUTTON_NOT_PRESSED=0x900;
GO_BUTTON_PRESSED=0x901;

FATAL_OVERCURRENTVDD = 0x01; {or mask}
FATAL_OVERCURRENTVPP = 0x02; {or mask}
FATAL_OVERVOLTAGEVDD = 0x04; {or mask}
```

1.7 Konstanty

1.7.1 QSetPins konstanty

```
PINS_HI = 0x3;
PINS_LO = 0x2;
PINS_HIZ = 0x1;
PINS_D_BIT = 0x0;
PINS_C_BIT = 0x2;
PINS_P_BIT = 0x4;
PINS_L_BIT = 0x6;
```

Příklad:

```
PINS_D_HI = PINS_HI << PINS_D_BIT;
PINS_D_LO = PINS_LO << PINS_D_BIT;
PINS_D_HIZ = PINS_HIZ << PINS_D_BIT;
```

1.7.2 QShiftByte/QShiftByte_OutIn konstanty

```
SHIFT_OUTIN_PIND = 0x00;  // InputPin value
SHIFT_OUTIN_PINL = 0x01;
SHIFT_OUTIN_PINP = 0x02;
SHIFT_OUTIN_PINI = 0x03;
SHIFT_MODE1=0x01;  //mode value
SHIFT_MODE3=0x03;
```

1.7.3 QSetPrestoSpeed konstanty

```
PRESTO_CLK1=0x00; // 3MHz - default value
PRESTO_CLK2=0x01; // 1.5MHz
PRESTO_CLK4=0x02; // 750kHz
PRESTO_CLK32=0x03; // 93.75kHz
```

1.8 Závažné chyby

Žádná z výše uvedených funkcí *Q...()* nevrací závažné chyby, ty se generují asynchronně. Pokud k takové chybě dojde, *AGet()* a *AGetBlocking()* pořád dokola opakují tuto jednu hodnotu, dokud není chyba smazána pomocí *AClearFatalError()*. Po smazání závažné chyby je PRESTO zavřeno a je ho potřeba znovu otevřít. Jakékoli příkazy ve frontě již nebudou provedeny a žádné odpovědi, které měly přijít přes *AGet()* nebo *AGetBlocking()* už nikdy nepřijdou.

K závažným chybám dojde, pokud je ze zdroje VPP (13 V) odběr více než 70 mA, ze zdroje VDD (5 V) odběr více než 100 mA nebo na pinu VDD je větší napětí než cca 7 V.

Pozor! Pokud je závažná chyba vyvolána tím, že na pinu VDD je napětí větší než 7 V, samotné vyvolání chyby PRESTO nezachrání před zničením. Je především potřeba ho okamžitě odpojit od zdroje.

1.9 Upozornění pro I2C

PRESTO neumí číst pin C – SCL. Nemůže tedy pracovat na sběrnici, na které jsou součástky, které dělají WAIT stavy. PRESTO se nesnese s dalším masterem na sběrnici.

2

Příloha A - Návratové hodnoty funkcí

Funkce	Návratový kód	Poznámka
QOpenPresto	OPEN_OK	
	OPEN_NOTFOUND	
	OPEN_CANNOTOPEN	
	OPEN_ALREADYOPEN	
QClosePresto	CLOSE_OK	
	CLOSE_CANNOTCLOSE	Nebyl otevřený programátor.
QPoweronVdd	POWERON_OK	
	POWERON_OCURR	Po stanovené době **delayus** byl zaznamenán proud větší než 100 mA, napětí se opět vypnulo.
	NOT_OPENED	
QSetPins	OK	
	NOT_OPENED	
QGetPins	GETPINS_CODE + hodnoty pinů	
	NOT_OPENED	
QDelay	OK	
	NOT_OPENED	
QSetActiveLED	OK	
	NOT_OPENED	
QPoweroffVdd	OK	
	NOT_OPENED	
QShiftByte	OK	
	NOT_OPENED	
QShiftByte_OutIn	SHIFT_BYTE_OUTIN_CODE + přečtená data	
	NOT_OPENED	
QCheckSupplyVoltage	SUPPLY_VOLTAGE_CODE + SUPPLY_VOLTAGE_xV	
	NOT_OPENED	
QPoweronVpp13V	VPP_OK	13 V na P1 bylo zapnuto.
	VPP_OCURR	Byl zaznamenán nadproud, 13 V bylo opět vypnuto.
	NOT_OPENED	
QPoweroffVpp13V	OK	
	NOT_OPENED	
QSetDPullup	OK	
	NOT_OPENED	
QCheckGoButton	GO_BUTTON_NOT_PRESSED	
	GO_BUTTON_PRESSED	
	NOT_OPENED	
QSetPrestoSpeed	OK	
	NOT_OPENED	

3

Historie dokumentu

Revize dokumentu	Provedené úpravy
21.10.2014	Dokument vytvořen.
20.5.2015	Opraven popis funkcí QShiftByte a QShiftByte_OutIn.
11.4.2022	Přidán popis funkce AGetProgList.