ASIX®

FORTE

FORTE_APP01 – Popis forte.dll

Aplikační poznámka

ASIX s.r.o.
Staropramenná 4
150 00 Praha 5 - Smíchov

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.

Obsah

1 forte.dll

1.1 Úvod

Funkce implementované ve forte.dll umožňují na jednotlivých pinech programátoru FORTE nastavovat logické úrovně a číst jejich stav, takto lze vytvářet různé komunikační protokoly.

Kromě funkcí umožňujících ovládání jednotlivých pinů knihovna obsahuje také funkce připravené pro komunikaci po sběrnicích SPI, I^2C a 1-Wire, funkce umožňující ovládat napájecí a programovací napětí programátoru a funkce, kterými lze číst velikost napájecího napětí a stav tlačítka GO na programátoru.

1.2 Značení pinů programátoru

Jednotlivé piny jsou značeny stejně jako na krabičce programátoru.

Pin	Typ	Popis
P	I/O, VPP	logický vstup/výstup nebo výstup VPP
VDD	PWR	napájení vstup/výstup
GND	PWR	signálová zem
D, C, I, L, T, S, R	I/O	log. vstup/výstup

Tab.1: Vlastnosti pinů

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

1.3 Způsob práce s programátorem

Příkazy se z podstaty USB vykonávají ve frontě. Ve stejném pořadí, v jakém byly do fronty příkazy (funkce *Q...*) zadávány, se vyčítají i jejich odpovědi.

Vrácená data je možné číst buď blokujícím způsobem funkcí *AGetBlocking* nebo neblokujícím způsobem funkcí *AGet*. Pro neblokující čtení většího množství odpovědí najednou jsou k dispozici také funkce *AGetBlock* a *AGetStatus*.

Čekání na odpověď od každého jednotlivého volání, například *QGetPins*, by práci velmi zpomalilo. V případě, že pro pokračování není potřeba znát předchozí odpověď, je vhodné na odpovědi čekat neblokujícím způsobem a vyčítat data až když jsou k dispozici. V mezičase je možné volat další funkce.

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

Funkce, které jsou jen výstupní se vloží do fronty a vykonají se až když na ně přijde řada. Odpověď ale pošlou už když jsou vloženy do fronty. Týká se to i funkcí pro čekání.

Funkce, které něco čtou ze vstupu nebo měří, odpoví až když se skutečně vykonají.

1.4 Seznam funkcí

```
void __stdcall QOpenProg(int sn);
void __stdcall QCloseProg(void);
void __stdcall QSetActiveLED(int led);
void __stdcall QPoweronVdd(int delayus, int Volt
age_mV);
void __stdcall QPoweroffVdd(void);
void __stdcall QPoweronVpp(int Voltage_mV);
```

```
void __stdcall QPoweroffVpp(void);
void __stdcall QDelay(int delayus);
void __stdcall QDelay_ns(int delayns);
void __stdcall QSetPullUpDowns(int pullupdowns);

void __stdcall QCheckGoButton(void);
void __stdcall QCheckSupplyVoltage(void);
void __stdcall QSetGPIOAnswer(bool answer);
void __stdcall QSetPins(int pins);
void __stdcall QGetPins(void);
void __stdcall QShiftByte(int databyte, int mode
);
void __stdcall QShiftByte_OutIn(int databyte, in
t mode, int InputPin);
void __stdcall QShiftBytes(int *buf, int mode, i
nt count);
void __stdcall QShiftBytes_In(int mode, int Inpu
tPin, int count);
void __stdcall QShiftBytes_OutIn(int *buf, int m
ode, int InputPin, int Count);
void __stdcall QShiftBits(int data, int mode, in
t bits_count);
void __stdcall QShiftBits_OutIn(int data, int mo
de, int InputPin, int bits_count);
void __stdcall QSetShiftSpeed(int speed);
void __stdcall Q1WireInit(void);
void __stdcall Q1WireWriteByte(int data, int str
ong_pullup_time_us);
void __stdcall Q1WireReadByte(void);
void __stdcall QI2CStart(bool UseIntPullUps);
void __stdcall QI2CStop(void);
void __stdcall QI2CWriteByte(int databyte);
void __stdcall QI2CReadByte(bool ACK);
void __stdcall QI2CSetSpeed(int speed);
bool __stdcall AGet(int *answer);
int __stdcall AGetBlocking(void);
bool __stdcall AGetBlock(int *buf, int count, in
t *count_returned);
bool __stdcall AGetStatus(int *NumberOfAnswers);

void __stdcall AGetProgList(int *sn_list, int co
unt, int *count_returned);
void __stdcall AClearFatalError(void);
```

1.5 Popis funkcí

1.5.1 QOpenProg

Funkce zkusí otevřít FORTE. Pokud je parametr **sn** rovný - 1, otevře jedno FORTE 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 programátoru FORTE A6041234, pak **sn** by mělo být 0x1234 nebo 0x041234. Sériové číslo definují poslední čtyři nebo šest znaků v hexa tvaru.

Definice funkce:

void __stdcall QOpenProg(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

OPEN_BADDRIVERVERSION - špatná verze USB ovladače

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

Příklad:

```
QOpenProg(0x041234); // otevře FORTE SN A6041234
```

1.5.2 QCloseProg

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

Definice funkce:

void __stdcall QCloseProg(void);

Návratové hodnoty:

CLOSE_OK

CLOSE_CANNOTCLOSE - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

1.5.3 QSetActiveLED

Rozsvítí nebo zhasne nebo nastaví blikání ACTIVE LED na programátoru FORTE.

Definice funkce:

void __stdcall QSetActiveLED(int led);

Parametr:

led - Proměnná definuje požadovaný stav ACTIVE LED, viz konstanty.

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

Příklad:

Chci-li rozsvítit ACTIVE LED programátoru žlutě, zavolám funkci *QSetActiveLED(LED_ACT_Y)*, pro její zhasnutí funkci *QSetActiveLED(LED_ACT_OFF)*.

1.5.4 QPoweronVdd

Zapne na pin VDD napětí z programátoru, 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 doba stanovená parametrem delayus.

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 1,2 - 5,5 V. Logické úrovně na datových pinech odpovídají velikosti napětí na VDD.

Při VDD nižším než 1,8 V lze programátorem komunikovat jen omezenou rychlostí.

Definice funkce:

void __stdcall QPoweronVdd(int delayus, int Voltage_mV);

Parametry:

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

Voltage_mV - Velikost napetí z programátoru v mV. Napětí může být v rozmezí 1,2 - 5,5 V.

Návratové hodnoty:

POWERON_OK - Podařilo se zapnout interní napájení.

POWERON_OCURR - Byl detekován nadproud, napětí bylo vypnuto.

POWERON_WRONG_LEVEL - Byla zadána špatná velikost požadovaného napětí.

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

Příklad:

Chci-li zapnout interní napájení 3,3 V na pin VDD a zkontrolovat nadproud po 10 ms, zavolám funkci

```
QPoweronVdd(10000, 3300);
```

1.5.5 QPoweroffVdd

Vypne napětí z programátoru na pinu VDD.

Definice funkce:

void __stdcall QPoweroffVdd(void);

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

1.5.6 QPoweronVpp

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

Definice funkce:

void __stdcall QPoweronVpp(int Voltage_mV);

Parametr:

Voltage_mV - Velikost programovacího napětí v mV. Napětí může být v rozmezí 6,5 - 17 V.

Návratové hodnoty:

VPP_OK - Programovací napětí bylo zapnuto.

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

VPP_WRONG_LEVEL - Byla zadána špatná velikost požadovaného napětí.

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

1.5.7 QPoweroffVpp

Vypne programovací napětí na pinu P.

Definice funkce:

void __stdcall QPoweroffVpp(void);

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

1.5.8 QDelay

Čeká stanovenou dobu.

Definice funkce:

void __stdcall QDelay(int delayus);

Parametr:

delayus - Doba čekání v µs.

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Funkcí *QSetGPIOAnswer* lze vypnout odpověď, v tom případě funkce vrací pouze NOT_OPENED, pokud

programátor není otevřený.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

Příklad:

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

```
QDelay(7000);
```

1.5.9 QDelay_ns

Čeká stanovenou dobu. Granularita časovače je 16,67 ns, zadaná hodnota je zaokrouhlena na nejbližší vyšší násobek 16,67 ns.

V signálech se může objevit delší prodleva, způsobená zpožděním příkazů na USB.

Definice funkce:

void __stdcall QDelay_ns(int delayns);

Parametr:

delayns - Doba čekání v ns.

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Funkcí *QSetGPIOAnswer* lze vypnout odpověď, v tom případě funkce vrací pouze NOT_OPENED, pokud programátor není otevřený.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

Příklad:

Chci-li v signálech udělat prodlevu minimálně 33 ns, zavolám funkci

```
QDelay_ns(33);
```

Programátor udělá prodlevu 16,67*2=33,34 ns.

1.5.10 QSetPullUpDowns

Připojí/odpojí pull-up nebo pull-down rezistory 2k4 na vybraných datových pinech programátoru. Ve výchozím stavu rezistory nejsou připojeny.

Definice funkce:

void __stdcall QSetPullUpDowns(int pullupdowns);

Parametr:

pullupdowns - Proměnná specifikuje, které rezistory se na datové piny připojí, viz konstanty.

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

Příklad:

Chci-li připojit pull-up rezistor na pin D a pull-down na pin L, zavolám funkci

```
QSetPullUpDowns((PULLUP<<D_PULL)|
(PULLDOWN<<L_PULL));
```

1.5.11 QCheckGoButton

Zkontroluje tlačítko na programátoru a v odpovědi pošle jeho stav.

Definice funkce:

void __stdcall QCheckGoButton(void);

Návratové hodnoty:

GO_BUTTON_NOT_PRESSED

GO_BUTTON_PRESSED

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

Příklad:

Chci-li zjistit, zda je stištěno tlačítko, zavolám funkci *QCheckGoButton*, a následně pokud funkce *AGet(data)* vrátí 0x90001, je tlačítko stištěno.

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

1.5.12 QCheckSupplyVoltage

V odpovědi pošle kód, odpovídající napětí změřenému na pinu VDD.

Definice funkce:

void __stdcall QCheckSupplyVoltage(void);

Návratové hodnoty:

SUPPLY_VOLTAGE_CODE + změřené napětí ve V x10, např. 33 znamená 3,3 V

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

Příklad: Chci-li zkontrolovat napětí na VDD, zavolám funkci *QCheckSupplyVoltage* a následně přečtu výsledek funkcí *AGet*. *AGet* vrátí např. 0x7001B, kde 0x1B je 10x velikost napětí v hexadecimálním tvaru, tedy 27 dekadicky, změřené napětí je tedy 2,7 V.

1.5.13 QSetGPIOAnswer

Funkce umožňuje vypnout nebo zapnout odpovědi od výstupních funkcí. Po otevření programátoru je vždy odpověď zapnutá. V případě vypnutí odpovědi funkce vždy vrací NOT_OPENED, pokud jsou zavolány a programátor není otevřený.

Pokud nastane nějaká závažná chyba, je jí nahrazena odpověď volané funkce. V případě, že jsou odpovědi pro výstupní funkce vypnuté, závažné chyby jsou vraceny jen po volání vstupních funkcí.

Funkce ovlivňuje odpovědi funkcí *QDelay*, *QDelay_ns*, *QSetPins*, *QShiftByte*, *QShiftBytes*, *QShiftBits*, *QSetShiftSpeed*, *Q1WireWriteByte*, *QI2CStart*, *QI2CStop*, *QI2CWriteByte* a *QI2CSetSpeed*.

Definice funkce:

void __stdcall QSetGPIOAnswer(bool answer);

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

1.5.14 QSetPins

Nastaví výstupní piny programátoru podle konstant. Vždy se současně nastavují dvojice pinů D a C, I a L, P a R, S a T. Pokud je definovaný jen jeden pin ze dvojice, stav druhého pinu se nastaví podle uložené hodnoty.

Stav ostatních nedefinovaných pinů se nemění.

Definice funkce:

void __stdcall QSetPins(int pins);

Parametr:

pins - Definuje požadované hodnoty na pinech programátoru, viz konstanty.

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Funkcí *QSetGPIOAnswer* lze vypnout odpověď, v tom případě funkce vrací pouze NOT_OPENED, pokud programátor není otevřený.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

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));
```

1.5.15 QGetPins

V odpovědi pošle logické úrovně, které FORTE vidí na pinech. Viz konstanty pro *QGetPins*.

Definice funkce:

void __stdcall QGetPins(void);

Návratové hodnoty:

GETPINS_CODE + hodnoty pinů

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

Příklad:

Chci-li přečíst stav na pinu I, zavolám funkci *QGetPins* a následně přečtu vrácená data funkcí *AGet*. Funkce *AGet* vrátí např. hodnotu 0x4000C. V této hodnotě jsou vráceny stavy všech 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}
 }
```

1.5.16 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.

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 *QSetShiftSpeed*.

Definice funkce:

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 0, 1, 2 nebo 3.

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

WRONG_INPUT - Špatně zadané parametry.

Funkcí *QSetGPIOAnswer* lze vypnout odpověď, v tom případě funkce vrací pouze NOT_OPENED, pokud programátor není otevřený.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

Příklad:

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

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

1.5.17 QShiftByte_OutIn

Funkce, stejně jako *QShiftByte*, vygeneruje signály C a D podle zadaných parametrů, navíc 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.

Definice funkce:

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 0, 1, 2 nebo 3.

InputPin - Podle hodnoty proměnné je zvolen vstupní pin.

Návratové hodnoty:

SHIFT_BYTE_OUTIN_CODE + přečtená data

NOT_OPENED - Nebyl otevřený programátor.

WRONG_INPUT - Špatně zadané parametry.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

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);
```

1.5.18 QShiftBytes

Na pinech D a C odešle více Bytů dat, stejně jako funkce *QShiftByte*, která ale odesílá jen jeden Byte.

Definice funkce:

void __stdcall QShiftBytes(int *buf, int mode, int count);

Parametry:

buf - Pole dat, která se mají odeslat.

mode - Proměnná definuje SPI mód, podle zvoleného módu může mít hodnotu 0, 1, 2 nebo 3.

count - Počet Bytů, které mají být odeslány.

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

WRONG_INPUT - Špatně zadané parametry.

Funkcí *QSetGPIOAnswer* lze vypnout odpověď, v tom případě funkce vrací pouze NOT_OPENED, pokud programátor není otevřený.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

1.5.19 QShiftBytes_In

Funkce, stejně jako *QShiftByte_OutIn*, čte data ze zvoleného vstupního pinu a současně generuje hodinový signál na pinu C, ale umožňuje číst více Bytů na jedno volání, což může být vhodné např. při čtení velkých SPI pamětí.

Funkce je jen vstupní, neumožňuje odesílání dat.

V případě, že je jako vstupní pin zvolen pin D, je před operací nastaven do třetího stavu.

Definice funkce:

void __stdcall QShiftBytes_In(int mode, int InputPin, int count);

Parametry:

mode - Proměnná definuje SPI mód, podle zvoleného módu může mít hodnotu 0, 1, 2 nebo 3.

InputPin - Podle hodnoty proměnné je zvolen vstupní pin.

count - Proměnná určuje kolik Bytů má být přečteno. Funkcí je možné číst maximálně 512 Bytů.

Návratové hodnoty:

SHIFT_BYTE_OUTIN_CODE + přečtená data - hodnota je vrácena pro každý čtený Byte

NOT_OPENED - Nebyl otevřený programátor.

WRONG_INPUT - Špatně zadané parametry.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

Příklad:

Chci-li přečíst 100 Bytů v SPI módu 0 na pinu I, zavolám funkci

```
QShiftBytes_In(SHIFT_MODE0, SHIFT_OUTIN_PINI, 100);
```

1.5.20 QShiftBytes_OutIn

Na pinech D a C odešle více Bytů dat a současně čte ze vstupu, stejně jako funkce *QShiftByte_OutIn*, která ale odesílá jen jeden Byte.

Definice funkce:

void __stdcall QShiftBytes_OutIn(int *buf, int mode, int InputPin, int Count);

Parametry:

buf - Pole dat, která se mají odeslat.

mode - Proměnná definuje SPI mód, podle zvoleného módu může mít hodnotu 0, 1, 2 nebo 3.

InputPin - Podle hodnoty proměnné je zvolen vstupní pin.

count - Počet Bytů, které mají být odeslány.

Návratové hodnoty:

SHIFT_BYTE_OUTIN_CODE + přečtená data - hodnota je vrácena pro každý čtený Byte

OK

NOT_OPENED - Nebyl otevřený programátor.

WRONG_INPUT - Špatně zadané parametry.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

1.5.21 QShiftBits

Odešle zvolený počet bitů na pinech D a C, stejným způsobem, jakým funkce *QShiftByte* posílá Byty.

Možnost neposílat data po celých Bytech se může hodit pro některé protokoly.

Zadaná data jsou posílána počínaje LSB, frekvence hodin odpovídá hodnotě nastavené funkcí *QSetShiftSpeed*.

Definice funkce:

void __stdcall QShiftBits(int data, int mode, int bits_count);

Parametry:

data - Proměnná pro data, která se mají odeslat.

mode - Proměnná definuje SPI mód, podle zvoleného módu může mít hodnotu 0, 1, 2 nebo 3.

bits_count - Určuje počet bitů, které se mají odeslat. Je možné nastavit 1 až 16 bitů.

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

WRONG_INPUT - Špatně zadané parametry.

Funkcí *QSetGPIOAnswer* lze vypnout odpověď, v tom případě funkce vrací pouze NOT_OPENED, pokud programátor není otevřený.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

Příklad:

Chci-li poslat 2 bity 0, 1, v SPI módu 0, zavolám funkci

```
QShiftByte(0x02, SHIFT_MODE0,2);
```

1.5.22 QShiftBits_OutIn

Odešle zvolený počet bitů na pinech D a C a současně čte na zvoleném pinu, stejným způsobem, jakým funkce *QShiftByte_OutIn* posílá Byty.

Možnost neposílat data po celých Bytech se může hodit pro některé protokoly.

V případě, že je jako vstupní pin zvolen pin D, je před operací nastaven do třetího stavu a data z něj jsou jen čtena.

Zadaná data jsou posílána počínaje LSB, frekvence hodin odpovídá hodnotě nastavené funkcí *QSetShiftSpeed*.

Definice funkce:

void __stdcall QShiftBits_OutIn(int data, int mode, int InputPin, int bits_count);

Parametry:

data - Proměnná pro data, která se mají odeslat.

mode - Proměnná definuje SPI mód, podle zvoleného módu může mít hodnotu 0, 1, 2 nebo 3.

InputPin - Podle hodnoty proměnné je zvolen vstupní pin.

bits_count - Určuje počet bitů, které se mají odeslat. Je možné nastavit 1 až 16 bitů.

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

WRONG_INPUT - Špatně zadané parametry.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

Příklad:

Chci-li poslat 4 bity 0xA, v SPI módu 3 a současně číst na pinu I, zavolám funkci

```
QShiftBits_OutIn(0x0A, SHIFT_MODE3, SHIFT_OUTIN_
PINI, 4);
```

1.5.23 QSetShiftSpeed

Nastaví rychlost hodin na pinu C pro *QShift...* funkce.

Definice funkce:

void __stdcall QSetShiftSpeed(int speed);

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

WRONG_INPUT - Špatně zadané parametry.

Funkcí *QSetGPIOAnswer* lze vypnout odpověď, v tom případě funkce vrací pouze NOT_OPENED, pokud programátor není otevřený.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

Parametr:

speed - Definuje rychlost hodin. Viz specifikaci konstant.

Příklad:

Chci-li nastavit rychlost hodin pro funkce *QShift...* na 1 MHz, zavolám funkci

```
QSetShiftSpeed(SHIFT_CLK_1000kHz);
```

1.5.24 Q1WireInit

Provede inicializaci na 1-Wire sběrnici, udělá reset puls a přečte presence puls od součástky.

Funkce pro 1-Wire sběrnici komunikují na pinu P programátoru. Na sběrnici je potřeba připojit pull-up rezistor podle specifikace.

Definice funkce:

void __stdcall Q1WireInit(void);

Návratové hodnoty:

_1WIRE_PRESENT - Součástka odpověděla, na sběrnici byla čtena 0.

_1WIRE_NOT_PRESENT - Součástka neodpověděla, na sběrnici byla čtena 1.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

1.5.25 Q1WireWriteByte

Na 1-Wire sběrnici odešle jeden Byte. Pokud je zvolena nenulová doba pro strong pull-up, po tuto dobu programátor připne na sběrnici log.1.

Připojení strong pull-up rezistoru je realizováno jako připojení sběrnice na log.1, z pinu nesmí být odebírán větší proud, než je definováno ve specifikacích programátoru v jeho manuálu.

Definice funkce:

void __stdcall Q1WireWriteByte(int data, int strong_pullup_time_us);

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Funkcí *QSetGPIOAnswer* lze vypnout odpověď, v tom případě funkce vrací pouze NOT_OPENED, pokud programátor není otevřený.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

Parametr:

data - Definuje data, která mají být odeslána. Data jsou odesílána LSB first.

strong_pullup_time_us - Definuje dobu, po kterou má být po odeslání Bytu na sběrnici zapnutý strong pull-up (log.1).

Příklad:

Chci-li odeslat Byte 0xCC a strong pull-up nepoužívat, zavolám funkci

```
QSetPrestoSpeed(0xCC, 0);
```

1.5.26 Q1WireReadByte

Přečte z 1-Wire sběrnice jeden Byte.

Definice funkce:

void __stdcall Q1WireReadByte(void);

Návratové hodnoty:

_1WIRE + přečtená data

NOT_OPENED - Nebyl otevřený programátor.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

1.5.27 QI2CStart

Funkce udělá na sběrnici start bit.

Funkce pro I²C sběrnici komunikují na pinech D (SDA) a C (SCL) programátoru. Parametrem je buď možné zapnout interní pull-up resitory na obou pinech nebo je nutné mít připojené externí rezistory.

Rychlost komunikace lze nastavit funkcí *QI2CSetSpeed*.

Definice funkce:

void __stdcall QI2CStart(bool UseIntPullUps);

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Funkcí *QSetGPIOAnswer* lze vypnout odpověď, v tom případě funkce vrací pouze NOT_OPENED, pokud programátor není otevřený.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

Parametr:

UseIntPullUps - Pokud je true, zapnou se interní pull-up rezistory. Pokud je false, interní pull-up rezistory zůstanou v takovém stavu, jak byly před zavoláním této funkce.

1.5.28 QI2CStop

Funkce udělá na sběrnici stop bit.

Rychlost komunikace lze nastavit funkcí *QI2CSetSpeed*.

Definice funkce:

void __stdcall QI2CStop(void);

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Funkcí *QSetGPIOAnswer* lze vypnout odpověď, v tom případě funkce vrací pouze NOT_OPENED, pokud programátor není otevřený.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

1.5.29 QI2CWriteByte

Zapíše na I²C sběrnici jeden Byte.

Rychlost komunikace lze nastavit funkcí *QI2CSetSpeed*.

Definice funkce:

void __stdcall QI2CWriteByte(int databyte);

Návratové hodnoty:

NOT_OPENED - Nebyl otevřený programátor.

I2C_ACK - Po odeslání Bytu součástka odpověděla (ACK).

I2C_NACK - Po odeslání Bytu součástka neodpověděla (NO ACK).

Funkcí *QSetGPIOAnswer* lze vypnout odpověď, v tom případě funkce vrací pouze NOT_OPENED, pokud programátor není otevřený.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

Parametr:

databyte - Parametr specifikuje Byte, který bude odeslán na sběrnici. Byte bude odeslán MSB first.

Příklad:

Chci-li odeslat Byte 0xAB, zavolám funkci

```
QI2CWriteByte(0xAB);
```

1.5.30 QI2CReadByte

Přečte z I²C sběrnice jeden Byte.

Rychlost komunikace lze nastavit funkcí *QI2CSetSpeed*.

Definice funkce:

void __stdcall QI2CReadByte(bool ACK);

Návratové hodnoty:

NOT_OPENED - Nebyl otevřený programátor.

I2C_CODE + přečtená data

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

Parametr:

ACK - Pokud je true, programátor po přečtení Bytu pošle ACK, v opačné případě NO ACK.

1.5.31 QI2CSetSpeed

Nastaví rychlost komunikace pro funkce komunikující po I²C sběrnici. Viz specifikaci konstant.

Definice funkce:

void __stdcall QI2CSetSpeed(int speed);

Návratové hodnoty:

OK

NOT_OPENED - Nebyl otevřený programátor.

Funkcí *QSetGPIOAnswer* lze vypnout odpověď, v tom případě funkce vrací pouze NOT_OPENED, pokud programátor není otevřený.

Návratové hodnoty jsou vraceny prostřednictvím funkcí *AGet*, *AGetBlocking* nebo *AGetBlock*.

Parametr:

speed - Specifikuje zvolenou rychlost komunikace. Po otevření programátoru je nastavena frekvence 100 kHz. Viz specifikaci konstant.

Příklad:

Chci-li nastavit pro I²C komunikaci frekvenci 400 kHz, zavolám funkci

```
QI2CSetSpeed(I2C_CLK_400kHz);
```

1.5.32 AGet

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

Definice funkce:

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}
```

1.5.33 AGetBlocking

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

Definice funkce:

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 programátoru.

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

1.5.34 AGetBlock

Přečte požadovaný počet odpovědí. Pokud požadovaný počet odpovědí není k dispozici, přečte tolik odpovědí, kolik jich je k dispozici. Funkce je neblokující.

Tuto funkci je vhodné používat společně s funkcí *AGetStatus*.

Definice funkce:

bool __stdcall AGetBlock(int *buf, int count, int *count_returned);

Parametry:

buf - Pole typu Integer, kde budou vráceny odpovědi.

count - Požadovaný počet odpovědí.

count_returned - V této proměnné je vrácen počet skutečně vrácených odpovědí.

Návratové hodnoty:

Funkce vrátí False, pokud nastala závažná chyba. V tom případě v poli vrátí jen jednu hodnotu, tuto chybu.

Pokud závažná chyba nenastala, funkce vrátí True a v poli **buf** vrátí návratové hodnoty a v proměnné **count_returned** vrátí počet těchto hodnot.

Funkce vrátí maximálně 65536 hodnot.

Příklad:

Chci-li počkat, až bude k dispozici 10 odpovědí a následně je najednou přečíst, použiji funkci *AGetBlock*

```
int data[10];
int data_returned;
int i;

i=0;
while (i<10)
 {
  if (!AGetStatus(*i))
    { // Fatal error, exit with error report}
 }
if (!AGetBlock(data, 10, *data_returned))
  { // Fatal error, exit with error report}
```

1.5.35 AGetStatus

V parametru vrátí počet odpovědí, které jsou k dispozici.

Definice funkce:

bool __stdcall AGetStatus(int *NumberOfAnswers);

Parametr:

NumberOfAnswers - Vrací počet odpovědí, které jsou k dispozici.

Návratové hodnoty:

Funkce vrátí False, pokud nastala závažná chyba.

Pokud závažná chyba nenastala, funkce vrátí True a v proměnné **NumberOfAnswers** vrátí počet odpovědí, které jsou k dispozici pro přečtení.

1.5.36 AGetProgList

V parametru vrátí seznam SN programátorů FORTE, 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 FORTE. 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**.

1.5.37 AClearFatalError

Smaže závažnou chybu.

Po smazání závažné chyby je programátor FORTE zavřen a je ho potřeba znovu otevřít. Jakékoli příkazy ve frontě již nebudou provedeny a žádné odpovědi, které měly přijít přes *AGet*, *AGetBlocking* nebo *AGetBlock* už nikdy nepřijdou.

Definice funkce:

```
void __stdcall AClearFatalError(void);
```

1.6 Závažné chyby

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

K závažným chybám dojde, pokud je na zdroji programovacího nebo napájecího napětí detekován nadproud, nebo pokud je na pinu VDD větší napětí než cca 6 V.

Pozor! Pokud je závažná chyba vyvolána tím, že na pinu VDD je napětí větší než 6 V, samotné vyvolání chyby FORTE nezachrání před zničením. Je především potřeba ho okamžitě odpojit od zdroje.

1.7 Odpovědi

```
OPEN_OK = 0x10000;
OPEN_NOTFOUND = 0x10001;
OPEN_CANNOTOPEN = 0x10002;
OPEN_ALREADYOPEN = 0x10003;
OPEN_BADDRIVERVERSION = 0x10004;
CLOSE_OK = 0x20000;
CLOSE_CANNOTCLOSE = 0x20001;
POWERON_OK = 0x30000;
POWERON_OCURR = 0x30001;
POWERON_WRONG_LEVEL = 0x30002;

GETPINS_CODE = 0x40000; //
ored with GETPINS_PINx
      GETPINS_PIND = 0x01;
      GETPINS_PINC = 0x02;
      GETPINS_PINI = 0x04;
      GETPINS_PINL = 0x08;
      GETPINS_PINP = 0x10;
      GETPINS_PINR = 0x20;
      GETPINS_PINS = 0x40;
      GETPINS_PINT = 0x80;
OK = 0x50000;
NOT_OPENED = 0x50001;
WRONG_INPUT = 0x50002;
SHIFT_BYTE_OUTIN_CODE = 0x60000;
SUPPLY_VOLTAGE_CODE = 0x70000;
VPP_OK = 0x80000;
VPP_OCURR = 0x80001;
VPP_WRONG_LEVEL = 0x80002;
GO_BUTTON_NOT_PRESSED=0x90000;
GO_BUTTON_PRESSED=0x90001;
SHIFT_BITS_OUTIN_CODE = 0xA0000;
_1WIRE = 0xB0000;
```

```
_1WIRE_PRESENT = 0xB0100;
_1WIRE_NOT_PRESENT = 0xB0200;
I2C_CODE  = 0xC0000;
I2C_ACK   = 0xC0100;
I2C_NACK  = 0xC0200;

FATAL_OVERCURRENTVDD = 0x01;
FATAL_OVERCURRENTVPP = 0x02;
FATAL_OVERVOLTAGEVDD = 0x04;
FATAL_OTHER = 0x08;
```

1.8 Konstanty

1.8.1 QSetPins konstanty

```
PINS_HIZ = 0x01;
PINS_LO = 0x02;
PINS_HI = 0x03;
PINS_D_BIT = 0x00;
PINS_C_BIT = 0x02;
PINS_I_BIT = 0x04;
PINS_L_BIT = 0x06;
PINS_P_BIT = 0x08;
PINS_R_BIT = 0x0A;
PINS_S_BIT = 0x0C;
PINS_T_BIT = 0x0E;
```

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;
```

1.8.2 QShift... konstanty

```
SHIFT_OUTIN_PIND = 0x00;
SHIFT_OUTIN_PINI = 0x02;
SHIFT_OUTIN_PINL = 0x03;
SHIFT_OUTIN_PINP = 0x04;
SHIFT_OUTIN_PINR = 0x05;
SHIFT_OUTIN_PINS = 0x06;
SHIFT_OUTIN_PINT = 0x07;

SHIFT_MODE0=0x00;
SHIFT_MODE1=0x01;
SHIFT_MODE2=0x02;
SHIFT_MODE3=0x03;
```

1.8.3 QSetShiftSpeed konstanty

```
SHIFT_CLK_15000kHz = 1;
SHIFT_CLK_10000kHz = 2;
SHIFT_CLK_7500kHz = 3;
SHIFT_CLK_6000kHz = 4;
SHIFT_CLK_5000kHz = 5;
SHIFT_CLK_3750kHz = 6;
SHIFT_CLK_3330kHz = 7;
SHIFT_CLK_3000kHz = 8;
SHIFT_CLK_2500kHz = 9;
SHIFT_CLK_2000kHz = 10;
SHIFT_CLK_1500kHz = 11;
SHIFT_CLK_1000kHz = 12;
SHIFT_CLK_750kHz = 13;
SHIFT_CLK_600kHz = 14;
SHIFT_CLK_500kHz = 15;
SHIFT_CLK_400kHz = 16;
SHIFT_CLK_375kHz = 17;
SHIFT_CLK_333kHz = 18;
SHIFT_CLK_300kHz = 19;
SHIFT_CLK_250kHz = 20;
SHIFT_CLK_200kHz = 21;
SHIFT_CLK_150kHz = 22;
SHIFT_CLK_120kHz = 23;
SHIFT_CLK_100kHz = 24;
SHIFT_CLK_75kHz = 25;
SHIFT_CLK_60kHz = 26;
SHIFT_CLK_50kHz = 27;
SHIFT_CLK_40kHz = 28;
SHIFT_CLK_37kHz = 29;
SHIFT_CLK_33kHz = 30;
SHIFT_CLK_30kHz = 31;
```

1.8.4 QSetPullUpDowns konstanty

```
PULLDOWN = 0x01;
PULLUP  =  0x02;
D_PULL = 0x00;
C_PULL = 0x02;
I_PULL = 0x04;
L_PULL = 0x06;
S_PULL = 0x08;
T_PULL = 0x0A;
P_PULL = 0x0C;
R_PULL = 0x0E;
```

1.8.5 QI2CSetSpeed konstanty

```
I2C_CLK_100kHz = 0x00;
I2C_CLK_400kHz = 0x01;
I2C_CLK_1MHz = 0x02;
```

1.8.6 QSetActiveLED konstanty

```
LED_ACT_OFF           = 0x00;
LED_ACT_Y             = 0x01;
LED_ACT_R             = 0x02;
LED_ACT_Y_BLINK       = 0x03;
LED_ACT_R_BLINK       = 0x04;
LED_ACT_Y_FAST_BLINK  = 0x05;
LED_ACT_R_FAST_BLINK  = 0x06;
LED_ACT_YR_FAST_BLINK = 0x07;
```

2

Historie dokumentu

Revize dokumentu	Provedené úpravy
2.4.2015	Dokument vytvořen.
1.2.2017	Oprava definice funkcí.
	Doplněny popisy funkcí a kapitola Způsob práce s programátorem.
	Doplněn popis nových funkcí AGetProgList, QShiftBytes, QShiftBytes_OutIn.