PRESTO

PRESTO_APP01 – presto.dll description

Application Note

Address:	ASIX s.r.o. Staropramenna 4 150 00 Prague Czech Republic
E-Mail:	sales@asix.net (sales inquiries, ordering) support@asix.net (technical support)
WWW:	tools.asix.net (development tools) www.asix.net (company website)
Tel.:	+420-257 312 378
Fax:	+420-257 329 116

Contents

1. Introduction....3
2. Programmer pins marking....3
3. How to work with the programmer....3
4. List of the functions....4
5. Functions description....4
5.1 QOpenPresto()....4
5.2 QClosePresto()....5
5.3 QSetPins()....5
5.4 QGetPins()....5
5.5 QDelay()....6
5.6 QPoweronVdd()....6
5.7 QPoweroffVdd()....7
5.8 QSetActiveLED()....7
5.9 QShiftByte()....7
5.10 QShiftByte_OutIn()....8
5.11 QCheckSupplyVoltage()....8
5.12 QPoweronVpp13V()....9
5.13 QPoweroffVpp13V()....9
5.14 QSetDPullup()....9
5.15 QCheckGoButton()....9
5.16 QSetPrestoSpeed()....10
5.17 AGet()....10
5.18 AGetBlocking()....11
5.19 AClearFatalError()....11
6. Answers....12
7. Constants....13
7.1 QSetPins() constants....13
7.2 QShiftByte()/QShiftByte_OutIn() constants....13
7.3 QSetPrestoSpeed() constants....13
8. Fatal errors....13
9. I2C warning....14
10. Apendix A - Return values of the functions....14
11. Document history....15

1. Introduction

Functions implemented in the presto.dll enable setting or reading of logical values at single pins of the PRESTO programmer. Various communication protocols can be implemented this way. *QSetPins()* function enables output pins control. *QGetPins()* function enables input pins reading. *QSendByte()* function enables a fast SPI Byte on the data and clock pins to be sent. If also reading is required, then the *QSendByte_OutIn()* can be used. Then there are also functions for the programmer features settings, for supply and programming voltages control and functions for reading of the returned values.

The library can be used with all PRESTO programmers, it does not depend on the version of the programmer.

2. Programmer pins marking

In this document the programmer pins names are simplified, it is better lucid. The simplified marking is described in the table below.

Pin name	Marking	Function	Note
P1 - VPP (13V)	P	I/O, 13 V	logical values or programming voltage
P2		key	
P3 - VDD	VDD	supply	output 5 V or input for external supply voltage
P4 - GND	GND	supply	
P5 - DATA/MOSI	D	I/O	fast data output via QShiftByte() function
P6 - CLOCK	C	O	fast clock output via QShiftByte() function
P7 - MISO	I	I	
P8 - LVP	L	I/O	

Sense: I - input pin, O - output pin, I/O - input and output pin, 13 V - programming voltage

3. How to work with the programmer

Instructions are executed in a queue what corresponds with the USB way of work. Waiting for every answer, e.g. from *QGetPins()*, would slow down the work dramatically.

At some of the instructions it is advisable to wait for their answer before continuing, it is for example *QOpenPresto()*. The cycle **instruction → PRESTO → answer** takes from several milliseconds to tens of milliseconds.

The order of reading answers corresponds with the order of the instructions (*Q...()* functions) given. Returned data can be read either not blocking way via *AGet()* or blocking way via *AGetBlocking()* function.

4. List of the functions

```
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 AClearFatalError(void);
void __stdcall QShiftByte(int databyte, int mode);
void __stdcall QShiftByte_OutIn(int databyte, int 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);
```

5. Functions description

5.1 QOpenPresto()

The function tries to open a PRESTO. If the sn variable is -1 the function opens one PRESTO regardless its serial number. In other cases the sn means the PRESTO serial number. If the PRESTO serial number is A6016789, the sn should be 0x6789.

void __stdcall QOpenPresto(int sn);

Parameter:

sn - The serial number of the programmer.

Return values:

OPEN_OK	- successfully opened
OPEN_NOTFOUND	- programmer not found
OPEN_CANNOTOPEN	- cannot open the programmer
OPEN_ALREADYOPEN	- programmer is already open

Return values are returned via AGet() or AGetBlocking() functions.

Example:

QOpenPresto(0x6789); // open PRESTO SN 6789

5.2 QClosePresto()

It closes the PRESTO and switches output voltages off.

void __stdcall QClosePresto(void);

Return values:

CLOSE_OK
CLOSE_CANNOTCLOSE - The programmer has not been opened.
Return values are returned via AGet() or AGetBlocking() functions.

5.3 QSetPins()

The function sets the output pins in accordance with the constants.
Attention: if there are more pin changes in one request, first the C and D pins are set together and then the L and the P. So it is not possible to make edges on for example the L and C pins simultaneously, but it is possible on the D and the C. This can be utilized for serial communications.

void __stdcall QSetPins(int pins);

Parameter:

pins - The variable defines required values on the programmer pins.

Example:
To set the D pin to log.1 and the C pin to log.0 and if the other pins state is to be unchanged, call function

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

5.4 QGetPins()

The function sends back the values that the programmer sees on the D, L, I and P pins, the C pin cannot be read. See the *AGet()* answers constants.

void __stdcall QGetPins(void);

Return values:

GETPINS_CODE + values of pins
Return values are returned via AGet() or AGetBlocking() functions.

Example:
To read the I pin state, call the *QGetPins()* function and then read returned data using *AGet()* function. The *AGet()* function for example returns 0x40B value. In this value all the input pins values are returned, so the I pin state must be filtered with GETPINS_PINI constant, in our example the value which has been read on the I pin is 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}
 }
```

5.5 QDelay()

The function waits for specified time. The timer granularity is 170.66 µs (12 MHz/2048), the specified value is rounded to the nearest higher multiple of the 170.66 µs.

void __stdcall QDelay(int delayus);

Parameter:
delayus - Waiting time in µs.

Example 1:
To do a delay of 7 ms in the signals, call function

QDelay(7000);

Example 2:
On QDelay(5), the value is rounded up and the programmer does a delay of 170.66 µs.

5.6 QPoweronVdd()

The function switches on 5 V from USB on the VDD pin, then it waits for specified time and checks whether the current is less than 100 mA. If the current is higher, the programmer switches the voltage off. If there is a short circuit on the VDD pin, the supply voltage will not be present for much longer time than the specified time is. The function returns a value in accordance with the result of the operation. Although the result is returned in about 20 ms, the voltage is already switched off, this is solved in the HW. It is recommended to choose the time carefully, because long specified time is dangerous for the programmer circuits if there is an error in connections.
If the internal supply voltage from the programmer is switched off, an external supply voltage 2.5 to 5 V may be connected to the programmer. The data pins logical levels come up to the VDD supply voltage.

void __stdcall QPoweronVdd(int delayus);

Parameter: delayus - Time in µs after what the overcurrent will be checked.

Return values:
POWERON_OK - The supply voltage has been switched on successfully.
POWERON_OCURR - Overcurrent had been detected, supply voltage was switched off.
Return values are returned via AGet() or AGetBlocking() functions.

Example:
To switch on the internal supply voltage on the VDD pin and to check the overcurrent after 10 ms, call function

QPoweronVdd(10000);

5.7 QPoweroffVdd()

The function switches the VDD supply voltage off.

void __stdcall QPoweroffVdd(void);

5.8 QSetActiveLED()

The function switches on / off the ACTIVE LED on the PRESTO.

void __stdcall QSetActiveLED(bool led);

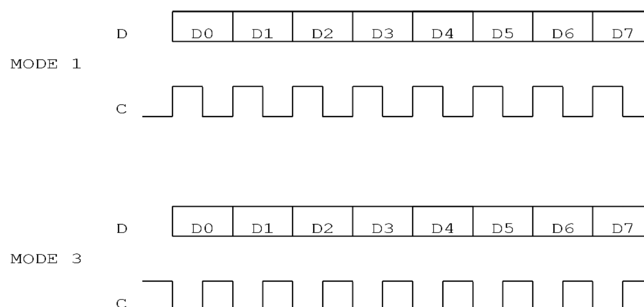
Parameter:

led - If the variable is True, the LED will switch on, if it is False, the LED will switch off.

Example:
To switch on the programmer ACTIVE LED, call QSetActiveLED(true) function, to switch it off, call QSetActiveLED(false).

5.9 QShiftByte()

The function sends a Byte on the D pin and generates clock signal on the C pin. The Byte value is specified by the **databyte** variable. The **mode** variable specifies a mode in accordance with the SPI definition. Only modes 1 and 3 are supported, the other modes can be done manually in combination with *QSetPins()* function usage, but this will be much slower. In case the user selects a mode that does not correspond with the current logic level on the C pin, the C logic level is first set to the required state. For example if there is log.0 on the C pin and **mode**=1, the C will first change to log. 1 and then the **databyte** will be sent.
The LSB is sent first, the clock signal is generated in accordance with the value set using *QSetPrestoSpeed()*. The *QShiftByte()* function generates signals faster than if the *QSetPins()* is used.



Drawing 1: SPI modes description

void __stdcall QShiftByte(int databyte, int mode);

Parameters:

databyte - A variable for data to be sent.
mode - A variable defining SPI mode, its value may be 1 or 3 in accordance with the mode.

Example:
To send a 0x3A Byte in the SPI mode 1, call function

QShiftByte(0x3A, SHIFT_MODE1);

5.10 QShiftByte_OutIn()

The function generates the C and D signals in accordance with specified parameters as *QShiftByte()* function do, but in addition it also reads data from the chosen pin at the same time. The input pin can be chosen with **InputPin** variable value. See constants defining possible values of the **InputPin** variable. If the D pin is chosen as input, it is first set to the high impedance state and the programmer only reads.

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

Parameters:

databyte - The variable for the data to be sent.
mode - The variable defines SPI mode, its value may be 1 or 3 in accordance with the mode. In mode 1 the data are read on rising edge, in mode 3 on falling edge.
InputPin - The input pin is chosen in accordance with this variable.

Return value:

SHIFT_BYTE_OUTIN_CODE + read data
Return values are returned via AGet() or AGetBlocking() functions.

Example:
To send a 0x4C Byte in SPI mode 3 and at the same time to read input data on the I pin, call function

QShiftByte_OutIn(0x4C, SHIFT_MODE3, SHIFT_OUTIN_PINI);

5.11 QCheckSupplyVoltage()

In its answer the function sends a code corresponding with the supply voltage measured on the VDD pin of the programmer. See constants defining possible returned values.

void __stdcall QCheckSupplyVoltage(void);

Return values:

SUPPLY_VOLTAGE_CODE + SUPPLY_VOLTAGE_xV
Return values are returned via AGet() or AGetBlocking() functions.

Example:
To check the VDD supply voltage value, call QCheckSupplyVoltage() function and then read the result with AGet() function. The AGet() will return for example 0x701, it means that on the VDD there is supply voltage > 2 V and < 5 V.

```
if (AGet(data))
{if (data == SUPPLY_VOLTAGE_CODE | SUPPLY_VOLTAGE_5V)
  {// on the VDD pin there is 5V}
 else if (data == SUPPLY_VOLTAGE_CODE | SUPPLY_VOLTAGE_2V)
  {// on the VDD pin there is voltage >2V}
 else if (data == SUPPLY_VOLTAGE_CODE | SUPPLY_VOLTAGE_0V)
  {// on the VDD pin there is 0V}
}
```

5.12 QPoweronVpp13V()

The function switches on the 13 V programming voltage on the VPP pin of the programmer. If the overcurrent is detected on the VPP pin after the voltage is switched on, it is switched off. The function sends the operation result as answer.

void __stdcall QPoweronVpp13V(void);

Return values:

VPP_OK - Programming voltage has been switched on successfully.
VPP_OCURR - Overcurrent detected, supply voltage was switched off again.

Return values are returned via AGet() or AGetBlocking() functions.

5.13 QPoweroffVpp13V()

The function switches off the 13 V programming voltage on the VPP pin.

void __stdcall QPoweroffVpp13V(void);

5.14 QSetDPullup()

The function connects/disconnects the D pin 2k2 pull-up resistor. In the default state the resistor is disconnected.

void __stdcall QSetDPullup(bool dpullup_on);

Parameter:

dpullup_on - The variable specifies if the pull-up resistor will be connected to (dpullup_on=True) or disconnected (dpullup_on=False) from the D pin.

5.15 QCheckGoButton()

The function checks the programmer button and sends its state as result. See constants defining possible returned values.

void __stdcall QCheckGoButton(void);

Return values:

GO_BUTTON_NOT_PRESSED
GO_BUTTON_PRESSED

Return values are returned via AGet() or AGetBlocking() functions.

Example:
To find out if the programmer button has been pressed, call QCheckGoButton() function and then if the AGet(data) function returns 0x901, the button has been pressed.

```
if (data==GO_BUTTON_PRESSED)
{// the button is pressed}
```

5.16 QSetPrestoSpeed()

The function sets the maximal clock frequency on the C pin. This setting affects the speed of the signals generated with *QShifByte()*, *QShiftByte_OutIn()* and *QSetPins()*. See the constants definition.

```
void __stdcall QSetPrestoSpeed(int speed);
```

Parameter:
speed - Defines the programmer clock speed.

Example:
To set the maximal clock frequency for the QShiftByte... functions up to 750 kHz, call function

```
QSetPrestoSpeed(PRESTO_CLK4);
```

5.17 AGet()

The function returns bool value which says whether an answer is available. If the answer is available, its value is returned in the function parameter.

```
bool __stdcall AGet(int *answer);
```

Return values:
The function returns True if the returned data are available, if they are not, it returns False.
answer - Returned answer value.

Example:
To find out if the programmer has answered and what its answer is, test it with function

```
if (AGet(data))
 { // the returned value is available in the data variable}
```

5.18 AGetBlocking()

The function waits until the answer is available and then it returns the answer value.

int __stdcall AGetBlocking(void);

Return value:
The function returns answer value.

Example:
To wait until the programmer answer is available and then to continue, use the AGetBlocking(). This function can be used for example after the QOpenPresto() has been called.

```
QOpenPresto(-1);
if (AGetBlocking()==OPEN_OK)
  {// programmer open OK}
else
  {// programmer open failed}
```

5.19 AClearFatalError()

The function erases fatal error. After the error is erased the PRESTO is closed and it must be opened again. No commands in the queue will be executed and the answers that should have come via *AGet()* or *AGetBlocking()* are lost.

void __stdcall AClearFatalError(void);

6. Answers

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;
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}

7. Constants

7.1 QSetPins() constants

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;
Example:
PINS_D_HI = PINS_HI << PINS_D_BIT;
PINS_D_LO = PINS_LO << PINS_D_BIT;
PINS_D_HIZ = PINS_HIZ << PINS_D_BIT;

7.2 QShiftByte()/QShiftByte_OutIn() constants

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;

7.3 QSetPrestoSpeed() constants

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

8. Fatal errors

None of the above described functions *Q...()* returns fatal errors, they are generated asynchronously. If such an error appears, the *AGet()* and *AGetBlocking()* repeats the one error value until the error is erased with *AClearFatalError()*. After the fatal error is erased, the PRESTO is closed and it must be opened again. Any instructions in the queue will not be executed and the answers that should come via *AGet()* or *AGetBlocking()* are lost.

The fatal errors appear if the current drawn from the VPP (13 V) power supply is higher than 70 mA or the current drawn from the VDD (5 V) power supply is higher than 100 mA or if there is more than about 7 V on the VDD pin.

Attention! If the fatal error is caused by a voltage over 7 V detected on the VDD pin, the fatal error does not save the programmer against its damage. First of all, the programmer must be immediately disconnected from the voltage power supply.

9. I2C warning

The PRESTO cannot read the C (SCL) pin. It cannot work on the bus where there are devices doing WAIT states and cannot be on the bus with another master.

10. Appendix A - Return values of the functions

Function	Return value	Sense
QOpenPresto	OPEN_OK	
	OPEN_NOTFOUND	
	OPEN_CANNOTOPEN	
	OPEN_ALREADYOPEN	
QClosePresto	CLOSE_OK	
	CLOSE_CANNOTCLOSE	The programmer was not open.
QPoweronVdd	POWERON_OK	
	POWERON_OCURR	Higher current than 100 mA was drawn from VDD pin after the **delayus** time. The voltage has been switched off again.
QSetPins	returns nothing	
QGetPins	GETPINS_CODE + pins values	
QDelay	returns nothing	
QSetActiveLED	returns nothing	
QPoweroffVdd	returns nothing	
QShiftByte	returns nothing	
QShiftByte_OutIn	SHIFT_BYTE_OUTIN_CODE + read data	
QCheckSupplyVoltage	SUPPLY_VOLTAGE_CODE + SUPPLY_VOLTAGE_xV	
QPoweronVpp13V	VPP_OK	13 V on the P1 is on.
	VPP_OCURR	Overcurrent was found, 13 V has been switched off again.
QPoweroffVpp13V	returns nothing	
QSetDPullup	returns nothing	
QCheckGoButton	GO_BUTTON_NOT_PRESSED	
	GO_BUTTON_PRESSED	
QSetPrestoSpeed	returns nothing	

11. Document history

Version	Date	Changes
1.0	12-July-2010	Initial version.
1.1	12-October-2011	Fixed Drawing 1, the SPI modes were interchanged.