KOMENDY API dla ATB-USBasp ver 4.2

LibUSB commands I2C

USBASP_I2C_SLA_CHECK	= 100
USBASP_FUNC_I2C_BITRATE	= 101
USBASP_FUNC_I2C_TX_SHORT	= 102
USBASP_FUNC_I2C_TX_MORE	= 103
USBASP_FUNC_I2C_RX_SHORT	= 104
USBASP_FUNC_I2C_RX_MORE	= 105
USBASP_I2C_START	= 120
USBASP_I2C_STOP	= 121
USBASP_I2C_TX_BYTE	= 122
USBASP_I2C_RX_BYTE	= 123
USBASP_I2C_BITBANG_ON	= 124
USBASP_I2C_BITBANG_OFF	= 125
USBASP_I2C_SDA_HI	= 126
USBASP_I2C_SDA_LO	= 127
USBASP_I2C_SCL_HI	= 128
USBASP_I2C_SCL_LO	= 129

LibUSB commands SPI

USBASP_SPI_CONNECT	= 150
USBASP_SPI_DISCONNECT	= 151
USBASP_SPI_CS	= 152
USBASP_SPI_DC_A0_LT	= 153
USBASP_SPI_RST	= 154
USBASP_SPI_TX_BYTE	= 155
USBASP_SPI_TXRX_BYTE	= 156
USBASP_SPI_TX_BUF	= 157
USBASP_SPI_SW_XMIT_N_BITS	= 158

Offset	Field	Size	Value	Description
0	bmRequestType	1	Bit-Map	**D7 Data Phase Transfer Direction** 0 = Host to Device 1 = Device to Host **D6..5 Type** 0 = Standard 1 = Class 2 = Vendor 3 = Reserved **D4..0 Recipient** 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4..31 = Reserved

Jeśli ZAPISUJEMY DANE DO urządzenia USB to `bRequestType` = **0x40** *(Write)*

Jeśli ODCZYTUJEMY DANE Z urządzenia USB to `bRequestType` = **0xC0** *(Read)*

USBASP_I2C_SLA_CHECK - Sprawdzenie czy układ o zadanym adresie I2C działa na magistrali.

Argumenty:

```
bRequestType = 0xC0;                // odczyt z USB
bRequest = USBASP_I2C_SLA_CHECK;    // komenda API
wValue = $A2;                       // adres układu na I2C
wIndex = 0;                         // nie istotne (dowolna liczba)
wLength = 254;                      // ilość bajtów na odpowiedź
```

Rezultat:

Jeśli układ odpowiada – to w buforze otrzymamy string ”**OK**”

Jeśli układ nie odpowiada – to w otrzymamy będzie string ”**ERROR**”

USBASP_FUNC_I2C_BITRATE – Ustawienie prędkości (**BitRate** w kHz) na magistrali I2C.

Poprawne prędkości od 5 kHz do 800 kHz

Argumenty:

```
bRequestType = 0xC0;                // odczyt z USB
bRequest = USBASP_FUNC_I2C_BITRATE; // komenda API
wValue = 400;                       // prędkość w kHz
wIndex = 0;                         // nie istotne (dowolna liczba)
wLength = 254;                      // ilość bajtów na odpowiedź
```

Rezultat:

Jeśli operacja się powiodła – to w buforze otrzymamy string ”**OK**”

USBASP_FUNC_I2C_TX_SHORT – Zapis jednego bajta (**B**) do układu o adresie I2C (**X**) do jego rejestru pod adresem (**Y**).

Argumenty:

```
bRequestType = 0x40;                // ZAPIS do USB
bRequest = USBASP_FUNC_I2C_TX_SHORT;     // komenda API
wValue = $A2;                       // adres układu na I2C (X)

wIndex = (LOW BYTE) = 2;            // adres rejestru (Y) w układzie
wIndex = (HIGH BYTE) = 73;          // bajt (B) do zapisu do układu

wLength = 1;                        // ilość bajtów do zapisu
```

Rezultat:

Brak

USBASP_FUNC_I2C_TX_MORE – Zapis dużej ilości bajtów do układu o adresie I2C (**X**) począwszy od rejestru pod adresem (**Y**). Za pomocą argumentu (**Z**) „Auto Increment" który może przyjmować wartości 0 lub 1, określamy czy podczas zapisu:

- adres ma być inkrementowany przez programator (=1) ponieważ układ docelowy (np. RTC PCF8583) sam nie posiada wbudowanego trybu auto inkrementacji
- czy też adres ma pozostawać stały ponieważ autoinkrementacja realizowana jest przez układ docelowy (np. pamięci EEPROM)

Ilość bajtów do zapisu może wynieść maksymalnie 16534 bajty. Ilość (**BUF**) przekazujemy za pomocą wskaźnika do bufora, który zostanie przekazany do funkcji typu: `usb_control_msg()` / `ControlTransfer()`. Nazwa funkcji zależy od implementacji biblioteki DLL w konkretnym środowisku programistycznym.

Argumenty:

```
bRequestType = 0x40;              // ZAPIS do USB
bRequest = USBASP_FUNC_I2C_TX_MORE;     // komenda API

wValue = (LOW BYTE) = $A2;        // adres układu na I2C (X)
wValue = (HIGH BYTE) = 1;         // Auto Increment address (Z)

wIndex = 120;                     // adres rejestru (Y) w układzie
wLength = 750;                    // ilość bajtów do zapisu (BUF)
```

Rezultat:

Brak

USBASP_FUNC_I2C_RX_SHORT – Odczyt niewielkiej ilości bajtów (max 254) (**BUF**) z urządzenia na magistrali I2C o adresie (**X**) z jego rejestru o adresie (**Y**).

Argumenty:

```
bRequestType = 0xC0;              // ODCZYT z USB
bRequest = USBASP_FUNC_I2C_RX_SHORT;    // komenda API
wValue = $A2;                     // adres układu na I2C (X)
wIndex = 2;                       // adres rejestru (Y) w układzie
wLength = 167;                    // ilość bajtów (BUF) do odczytu
```

Rezultat:

Odebrane bajty zostaną umieszczone w buforze którego wskaźnik zostanie przekazany w funkcji typu: `usb_control_msg()` / `ControlTransfer()`. Nazwa funkcji zależy od implementacji biblioteki DLL w konkretnym środowisku programistycznym.

USBASP_I2C_START – Wygenerowanie sekwencji START na magistrali I2C.

Argumenty:

```
bRequestType = 0x40;              // zapis do USB
bRequest = USBASP_I2C_START;      // komenda API
wValue = 0;                       // nie istotne (dowolna liczba)
wIndex = 0;                       // nie istotne (dowolna liczba)
wLength = 0;                      // zero
```

Rezultat:

Brak

USBASP_I2C_STOP - Wygenerowanie sekwencji STOP na magistrali I2C.

Argumenty:

```
bRequestType = 0x40;              // zapis do USB
bRequest = USBASP_I2C_STOP;       // komenda API
wValue = 0;                       // nie istotne (dowolna liczba)
wIndex = 0;                       // nie istotne (dowolna liczba)
wLength = 0;                      // zero
```

Rezultat:

Brak

USBASP_I2C_TX_BYTE – Wysłanie pojedynczego bajta (**B**) do urządzenia I2C.

Argumenty:

```
bRequestType = 0x40;              // zapis do USB
bRequest = USBASP_I2C_TX_BYTE;    // komenda API
wValue = 56;                      // bajt (B) do wysłania
wIndex = 0;                       // nie istotne (dowolna liczba)
wLength = 0;                      // JEDEN
```

Rezultat:

Pierwszy bajt w buforze odbiorczym będzie statusem informującym, czy układ docelowy odpowiada na magistrali I2C. ZERO – brak układu, JEDEN – układ odpowiada.

USBASP_I2C_RX_BYTE – Odebranie pojedynczego bajta (**B**) z urządzenia I2C.

Argumenty:

```
bRequestType = 0xC0;              // odczyt z USB
bRequest = USBASP_I2C_RX_BYTE;    // komenda API
wValue = 0;                       // 0-NO ACK, 1-ACK
wIndex = 0;                       // nie istotne (dowolna liczba)
wLength = 1;                      // JEDEN
```

Rezultat:

Odebrany bajt pojawi się w buforze odbiorczym jako pierwszy element bufora.

USBASP_I2C_BITBANG_ON – Przełączenie pinów magistrali I2C (SDA i SCL) w tryb pracy portów IO, jako wyjścia, aby można było wystawiać na każdy z nich stan wysoki lub niski komendami API. Po przełączeniu w tryb wyjściowy na pinach panuje stan wysoki. Wydajność prądowa wyjścia w stanie wysokim (SOURCE) jest praktycznie żadna, ponieważ wyjście jest typu Open Drain z podciągnięciem do VCC za pomocą rezystora 10K. W stanie niskim wydajność prądowa jest taka sama jak dla pinu mikrokontrolera czyli ok 30-40mA

Argumenty:

```
bRequestType = 0x40;              // zapis do USB
bRequest = USBASP_I2C_BITBANG_ON; // komenda API
wValue = 0;                       // nie istotne (dowolna liczba)
wIndex = 0;                       // nie istotne (dowolna liczba)
wLength = 0;                      // zero
```

Rezultat:

Brak

USBASP_I2C_BITBANG_OFF – Przywrócenie trybu pracy pinów magistrali I2C do pracy domyślnie jako wejścia.

Argumenty:

```
bRequestType = 0x40;              // zapis do USB
bRequest = USBASP_I2C_BITBANG_OFF; // komenda API
wValue = 0;                       // nie istotne (dowolna liczba)
wIndex = 0;                       // nie istotne (dowolna liczba)
wLength = 0;                      // zero
```

Rezultat:

Brak

USBASP_I2C_SDA_HI – Wystawienie stano WYSOKIEGO na linię SDA (pin nr 9 złącza KANDA)

Argumenty:

```
bRequestType = 0x40;              // zapis do USB
bRequest = USBASP_I2C_SDA_HI;     // komenda API
wValue = 0;                       // nie istotne (dowolna liczba)
wIndex = 0;                       // nie istotne (dowolna liczba)
wLength = 0;                      // zero
```

Rezultat: Brak

USBASP_I2C_SDA_LO – Wystawienie stano NISKIEGO na linię SDA (pin nr 9 złącza KANDA)

Argumenty:

```
bRequestType = 0x40;              // zapis do USB
bRequest = USBASP_I2C_SDA_LO;     // komenda API
wValue = 0;                       // nie istotne (dowolna liczba)
wIndex = 0;                       // nie istotne (dowolna liczba)
wLength = 0;                      // zero
```

Rezultat: Brak

USBASP_I2C_SCL_HI – Wystawienie stano WYSOKIEGO na linię SCL (pin nr 7 złącza KANDA)

Argumenty:

```
bRequestType = 0x40;              // zapis do USB
bRequest = USBASP_I2C_SDA_HI;     // komenda API
wValue = 0;                       // nie istotne (dowolna liczba)
wIndex = 0;                       // nie istotne (dowolna liczba)
wLength = 0;                      // zero
```

Rezultat: Brak

USBASP_I2C_SCL_LO – Wystawienie stano NISKIEGO na linię SCL (pin nr 7 złącza KANDA)

Argumenty:

```
bRequestType = 0x40;              // zapis do USB
bRequest = USBASP_I2C_SDA_LO;     // komenda API
wValue = 0;                       // nie istotne (dowolna liczba)
wIndex = 0;                       // nie istotne (dowolna liczba)
wLength = 0;                      // zero
```

Rezultat: Brak

LibUSB commands SPI

USBASP_SPI_CONNECT – Konfiguracja magistrali SPI. Możliwe są DWA TRYBY pracy:

1. Sprzętowy – z możliwością regulacji częstotliwości sygnału SCK
2. Programowy – ze stałą prędkością ok 50 kHz

Wybór częstotliwości taktowania oraz trybu pracy dokonuje się poprzez ustawienie 4 młodszych bitów w bajcie (**B**) konfiguracyjnym, które załączają odpowiedni dzielnik częstotliwości **F_CPU = 20MHz**, zgodnie z tabelą niżej.

SCK / TRYB	Wartość bajtu konfiguracyjnego	Częstotliwość SCK
HW SPI - F_CPU / 4	0	5 MHz
HW SPI - F_CPU / 16	1	1,25 MHz
HW SPI - F_CPU / 64	2	312,5 kHz
HW SPI - F_CPU / 128	3	156,25 kHz
HW SPI - F_CPU / 2	4	10 MHz
HW SPI - F_CPU / 8	5	2,5 MHz
HW SPI - F_CPU / 32	6	625 kHz
HW SPI - F_CPU / 64	7	312,5 kHz
SOFTWARE SPI	8	~ 50 kHz

Komenda powoduje załączenie bufora wyjściowego w programatorze.

Argumenty:

```
bRequestType = 0x40;              // zapis do USB
bRequest = USBASP_SPI_CONNECT;    // komenda API
wValue = 5;                       // Bajt (B) konfiguracyjny
wIndex = 0;                       // nie istotne (dowolna liczba)
wLength = 0;                      // zero
```

Rezultat: Brak

USBASP_SPI_DISCONNECT – Wyłączenie sprzętowego modułu SPI i bufora wyjściowego

Argumenty:

```
bRequestType = 0x40;              // zapis do USB
bRequest = USBASP_SPI_DISCONNECT; // komenda API
wValue = 0;                       // Bajt (B) konfiguracyjny
wIndex = 0;                       // nie istotne (dowolna liczba)
wLength = 0;                      // zero
```

Rezultat: Brak

USBASP_SPI_CS – Sterownie stanem logicznym pinu **CS** (Chip Select). Bajt (**B**) konfiguracyjny może przyjąć dwie wartości: **1** – stan **WYSOKI**, **0** – stan **NISKI** (pin nr 5 w złączu KANDA)

Argumenty:

```
bRequestType = 0x40;              // zapis do USB
bRequest = USBASP_SPI_CS;         // komenda API
wValue = 1;                       // Bajt (B) konfiguracyjny
wIndex = 0;                       // nie istotne (dowolna liczba)
wLength = 0;                      // zero
```

Rezultat: Brak

USBASP_SPI_DC_A0_LT – Sterownie stanem logicznym pinu **DC** (Data/Command). Bajt (**B**) konfiguracyjny może przyjąć dwie wartości: **1** – stan **WYSOKI**, **0** – stan **NISKI** (pin nr 3 w złączu KANDA)

Argumenty:

```
bRequestType = 0x40;              // zapis do USB
bRequest = USBASP_SPI_DC_A0_LT;   // komenda API
wValue = 1;                       // Bajt (B) konfiguracyjny
wIndex = 0;                       // nie istotne (dowolna liczba)
wLength = 0;                      // zero
```

Rezultat: Brak

USBASP_SPI_RST – jeszcze nie zaimplementowane

Argumenty: Brak

Rezultat: Brak

USBASP_SPI_TX_BYTE – wysłanie jednego bajtu (**B**) na magistralę SPI

Argumenty:

```
bRequestType = 0x40;              // zapis do USB
bRequest = USBASP_SPI_TX_BYTE;    // komenda API
wValue = 134;                     // Bajt (B) danych
wIndex = 0;                       // nie istotne (dowolna liczba)
wLength = 0;                      // zero
```

Rezultat: Brak

USBASP_SPI_TXRX_BYTE – wysłanie jednego bajtu (**B**) na magistralę SPI i odbiór jednego bajtu

Argumenty:

```
bRequestType = 0xC0;              // odczyt do USB
bRequest = USBASP_SPI_TXRX_BYTE;  // komenda API
wValue = 134;                     // Bajt (B) danych
wIndex = 0;                       // nie istotne (dowolna liczba)
wLength = 1;                      // jeden
```

Rezultat: W buforze przekazanym do funkcji pojawi się odebrany bajt

---

USBASP_SPI_TX_BUF – Przesłanie dowolnej ilości danych (max 16383 bajty). Dane do wysłania należy umieścić w buforze, którego wskaźnik zostanie przekazany do funkcji i określić ilość bajtów (**Y**) do przesłania z bufora.

Argumenty:

```
bRequestType = 0x40;              // zapis do USB
bRequest = USBASP_SPI_TX_BUF;     // komenda API
wValue = 134;                     // nie istotne (dowolna liczba)
wIndex = 0;                       // nie istotne (dowolna liczba)
wLength = 1272;                   // ilość bajtów (Y) do wysłania
```

Rezultat: Brak

USBASP_SPI_SW_XMIT_N_BITS – Przesłanie na magistralę SPI ramki danych składającej się z 8 do max 16 bitów. Transmisja odbywa się za pomocą programowego SPI z częstotliwością SCK ok 50 kHz. Za pomocą argumentów funkcji można skonfigurować postać ramki. W przypadku ramki o długości 8 bitów na magistralę zostanie przesłany tylko jeden bajt, jeśli zaś ilość bitów będzie większa niż 8 to przesłany zostanie jeden bajt i część kolejnego bajtu. Bajty wysyłane są od MSB (najstarszego bitu) przy czym w przypadku ramki o długości większej niż 8 bitów ale mniejszej niż 16, zawartość słowa zostanie przesunięta w lewo o odpowiednią ilość bitów tak aby wysłać tylko pożądaną długość ramki. W przypadku ramki 16 bitowej przesłane zostaną dwa bajty. Bity wysyłane są podczas narastającego zbocza sygnału SCK. Zawartość ramki 8-16-bitowej (**X**) przesyłamy zawsze za pomocą słowa 16-bitowego umieszczając w młodszym bajcie pierwsze 8 bitów i w starszym bajcie pozostałe bity ramki do przesłania. Rozmiar ramki (**Y**) podajemy w kolejnym argumencie. W ostatnim argumencie zawsze należy podać wartość = 2 ponieważ niezależnie od ilości bitów w ramce, w odpowiedzi zawsze odesłane zostanie całe słowo (dwa bajty). W młodszym bajcie zawsze będzie pierwsze 8 bitów ramki zaś pozostałe bity w starszym bajcie słowa. W buforze zwrotnym pierwszy element to starszy bajt, drugi element to młodszy bajt.

Argumenty:

```
bRequestType = 0xC0;                    // odczyt do USB
bRequest = USBASP_SPI_SW_XMIT_N_BITS;     // komenda API
wValue = 77;                            // Dane ramki (X)
wIndex = 0;                             // Ilość bitów ramki (Y)
wLength = 2;                            // zawsze 2 bajty zwrotne
```

Rezultat: Dwa bajty słowa w buforze zwrotnym (odebranej ramki)