PLO2 setting & intro to USB interface operation

The USB controller responds to a series of commands that comprise binary data like this, which is sent as a single USB frame (one output command from the PC)

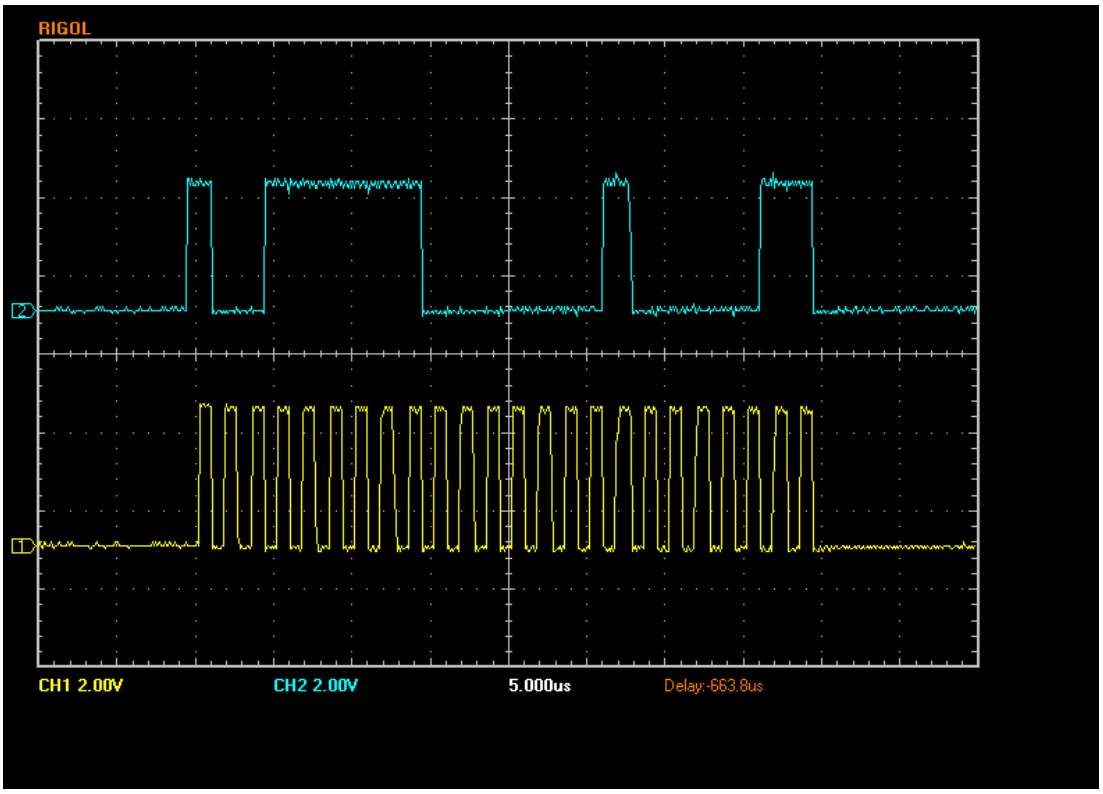
A1 05 00 08 09 08 09 08

All such strings are hex by the way before you ask – with 0x or similar decoration omitted

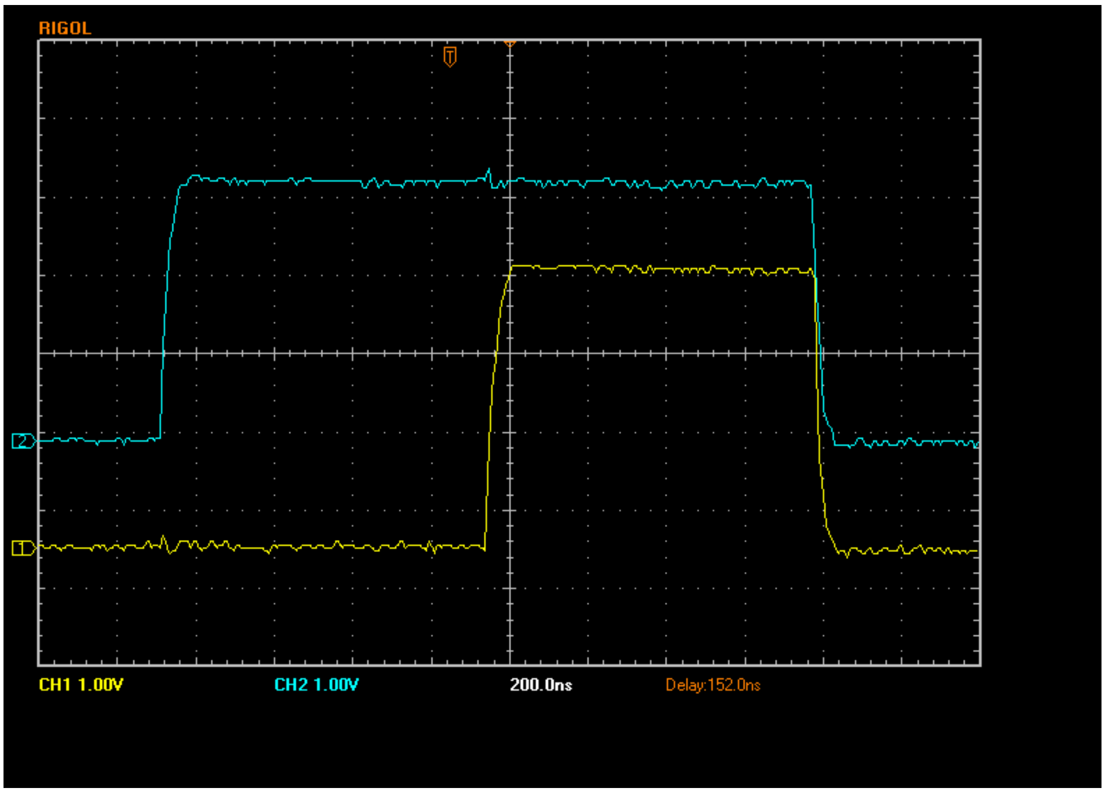
This comprises a command byte (A1) that instructs the USB controller to set port B, a byte count (5), a clock byte that we will come back to (00) and 5 bytes of data – arbitrarily 08 09 08 09 08
This will cause the controller to send the 5 data bytes in sequence to port B which will set bit 3 then pulse bit 0 twice.
Hence to set PLO2, we send a string like this to the USB controller.

From within the basic program then, we can send such a command to the USB controller and request it to set PLO2. The result is something like this (blue is data. Yellow is clock)

This is PLO2 being set with 24 clocked bits into the AFD4112. The USB chip takes 39 usec to clock the data into the chip



Zooming in on the first 2 bits…the clock duty cycle is 800 nsec low / 800 high



However, we are sending way too much data as we are sending two bytes for each clocked line.

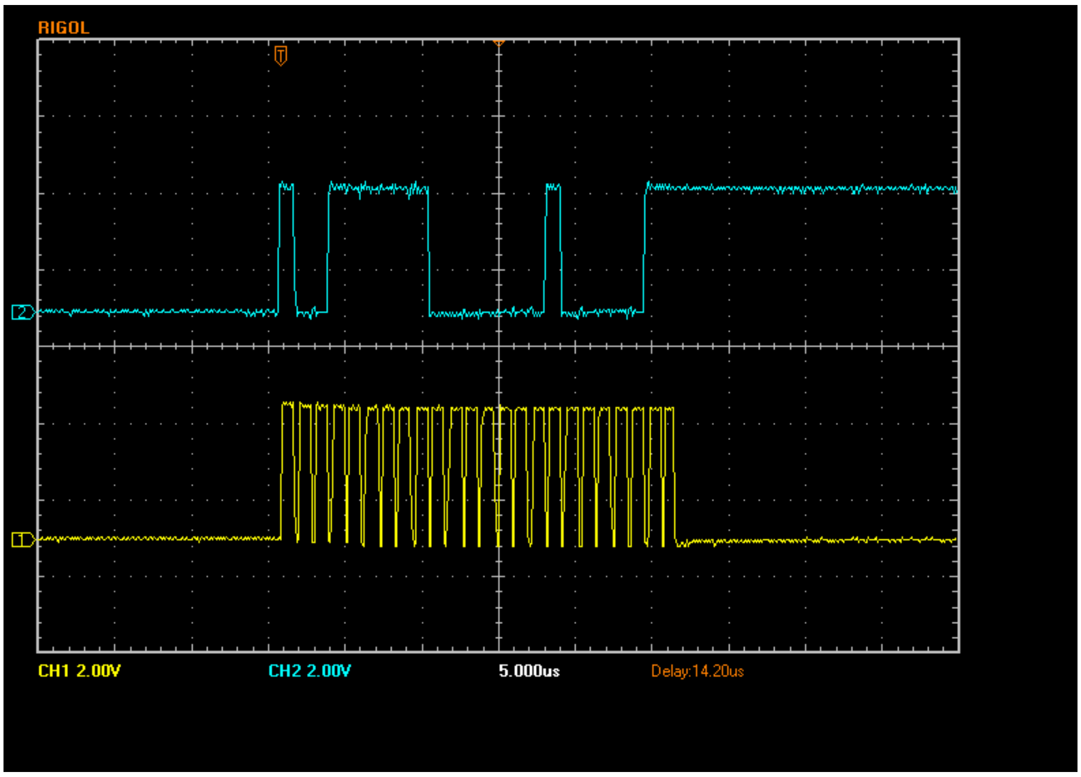
The ‘00’ byte in the above example modifies the command.

If the controller is sent this command

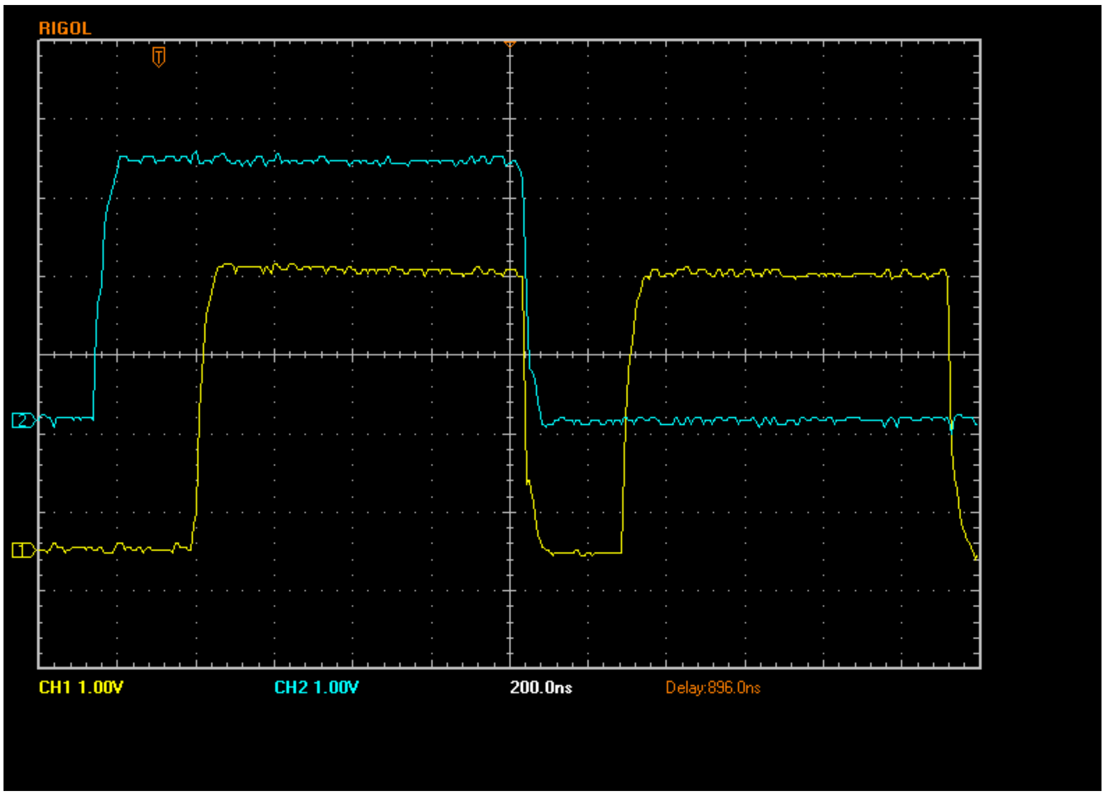
A1 02 01 08 08

Then it will automatically pulse the clock high for each data byte giving the same result (but more quickly as well as with less data to send)

The result is this; takes about 26 usec to clock the data in



Clock duty cycle is just over 800 ns high / 200 ns low



The coding in the USB controller chip is this

```
if( clock )
{
      do
      {
            IOB = EXTAUTODAT1;
            IOB |= clock;
      } while( --i );
      IOB &= ~clock;
}
else
      do
      {
            IOB = EXTAUTODAT1;
      } while( --i );
```

Where ‘clock’ is the byte set to 01 in the above example, IOB is the port B register, i is the byte count and EXTAUTODAT1 is an automatic pointer used to step through the data.

There are several commands to the A1 example above.

A0 = send data to port A
A1 = send data to port B
A2 = send data to port C
A3 = send data to port D
A4 = send data to port E
A5 = reset all port lines low (needs no data with it)

It is also possible to make the USB controller chip more intelligent by moving more of the logic to it but that would probably detract from experimentation as it would be harder for people to try new architectures or modules

What does the basic program look like?

Here is an example

This is the current liberty basic coding for the above sequence via the SLIM controller module

```
'step 1. Command the PLL without changing the filter bank.
'For PLL1,datavalue=2, for PLL2,datavalue=16, for PLL3,datavalue=8
'following code lines changed in ver113-3c
a=filtbank + N23*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N22*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N21*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N20*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N19*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N18*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N17*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N16*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N15*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N14*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N13*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N12*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N11*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N10*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N9*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N8*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N7*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N6*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N5*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N4*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N3*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N2*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N1*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
a=filtbank + N0*datavalue:out port, a:out control, SELT:out control, contclear:out port, a+1:out control, SELT:out control, contclear
out port, filtbank:out control, SELT:out control, contclear 'leaving lines latched to filter bank
out port, 0
```

and here the USB

```
USBwrbuf$ = "A11801"
a=filtbank + N23*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a
a=filtbank + N22*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
a=filtbank + N21*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a
a=filtbank + N20*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a
a=filtbank + N19*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
a=filtbank + N18*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
a=filtbank + N17*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
a=filtbank + N16*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
```

```
a=filtbank + N15*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
a=filtbank + N14*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
a=filtbank + N13*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
a=filtbank + N12*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
a=filtbank + N11*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
a=filtbank + N10*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
a=filtbank + N9*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
a=filtbank + N8*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
a=filtbank + N7*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
a=filtbank + N6*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
a=filtbank + N5*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
a=filtbank + N4*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
a=filtbank + N3*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
a=filtbank + N2*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
a=filtbank + N1*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
a=filtbank + N0*datavalue:USBwrbuf$ = USBwrbuf$ + ToHex$(a)
if USBdevice <> 0 then CALLDLL #USB, "UsbMSADeviceWriteString", USBdevice as long, USBwrbuf$ as ptr, 27 as short, result as boolean
```

the function ToHex$ just takes a value and returns the hex string as 2 characters (the built in function annoyingly strips leading zeros)

```
function ToHex$(value)
    if value <= 15 then
        ToHex$ = "0" + DECHEX$(value)
    else
        ToHex$ = DECHEX$(value and 255)
    end if
end function
```

The use of hex like this is not very efficient – it would be better to do this as a char array but I'm still struggling to get that coding right. The string could, as Sam pointed out, be simply stored and replayed – such optimisations can follow later. Initially I just want to get things going. At the moment there is a custom DLL that is converting the ascii back into decimal then sending it to the USB controller thus

```
extern "C" __declspec(dllexport) int  UsbMSADeviceWriteString( void *pMSAData, char *data, int message_size )
{
        MSA_TXBUFFER writebuf;
        for( int i=0; i< message_size && i < 254; i++ )
        {
                if( !isxdigit( *data ) )
                        return false;
                unsigned char c = toupper(*data);
                data++;
                c -= '0';
                if( c > 9 )
                        c = c -'A' + '0' + 10;
                if( !isxdigit( *data ) )
                        return false;
                unsigned char c1 =  toupper(*data);
                data++;
                c1 -= '0';
                if( c1 > 9 )
                        c1 = c1 -'A' + '0' + 10;
                c = (c<<4)+c1;
                if( i == 0 )
                        writebuf.set.command_code = c;
                else if( i == 1 )
                        writebuf.set.length = c;
                else
                        writebuf.set.data[i-2] = c;
        }
        MSADevice* MSA = (MSADevice* )pMSAData;
        return MSA->Write(&writebuf, message_size);
}
```

This is a bit crude but there is no point hardening it as the aim is to get rid of the use of hex and just use binary

```
extern "C" __declspec(dllexport) int  UsbMSADeviceWrite( void *pMSAData, MSA_TXBUFFER * writebuf, int
message_size )
{
      MSADevice* MSA = (MSADevice* )pMSAData;
      return MSA->Write(writebuf, message_size);
}
```

So, the basic program links to the DLL and sends it a command such as the above example. The DLL accepts such a command, and sends it on to the USB drivers, that send it to the USB controller where the USB controller chip actions it.

The code for the USB controller chip is downloaded automatically when the interface is plugged in to the PC or on powerup

Some of this will look like gibberish – suspend reality till I get the full code uploaded.

Dave