Shift Register Expansion Of XMOS XS1 Devices

REV 1.0

2011/05/11

1 Introduction

This document provides useful information on using shift registers to expand the I/O capabilities of XMOS XS1 devices.

XMOS XS1 devices have a large array of I/O capabilities, but in certain applications more (fairly low-speed) I/O ports are required than are available on the device. Although it is possible to step up to a larger XS1 device with either more xCORE Tiles or more pins, this is not always possible or desirable due to cost constraints. Situations where this is likely include driving arrays of LEDs or reading arrays of buttons.

A solution to this issue is to use shift registers to serialize or deserialize the input or output respectively. This means a few pins on the XS1 device can be used to connect to potentially hundreds of inputs or outputs at relatively low cost.

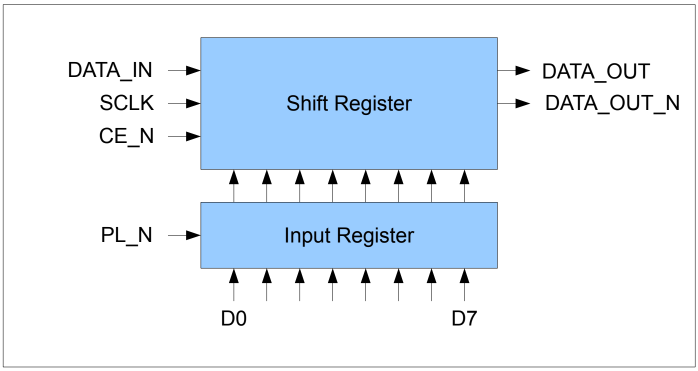
For further details about individual XS1 devices refer to the relevant device datasheet.

2 Shift Registers For Input

To capture input from multiple pins and send it into an XS1 device, a parallel in / serial out shift register can be used. Typically these are 8-bits long, such as the 74HC165, which is available from a variety of manufacturers and costs ~$0.085 in low volume (1000 off quantity from Digi-Key).

Then device has the following signals:

- $DATA\_IN$ (DS or SI) - the serial data input to the shift register.
- $SCLK$ (CP or CK) - the serial clock to clock the data through the shift register.
- CE_N ($\overline{CE}$ or CK_INH) - the chip enable (active low) signal.
- PL_N ($\overline{PL}$ or S/$\overline{L}$) - loads the contents of the input register into the shift register.
- $DATA\_OUT$ ($DATA\_OUT$ or QH) - the data output from the shift register.
- $DATA\_OUT\_N$ ($\overline{DATA\_OUT}$ or $\overline{Q}$H) - the inverted data output from the shift register.
- $D[7:0]$ - the 8-bits of parallel data input.



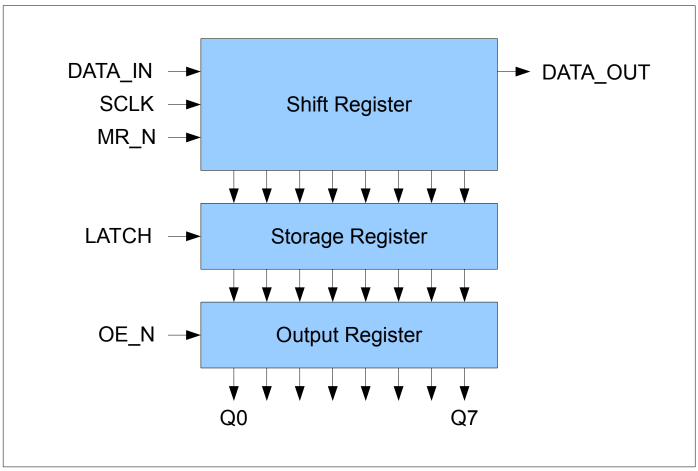
The diagram above details the internal structure of a parallel in / serial out shift register. It can be seen that there is an input register, which on the rising edge of PL_N latches the values of $D[7:0]$ into the shift register. The shift register then serially shifts the data out of $DATA\_OUT$, one bit at a time, on each clock input rising edge. At the same time data is shifted from $DATA\_IN$ into the register. This allows multiple devices to be chained together to form one long shift register. The CE_N enables the shift register to commence shifting, allowing the interface pins to be shared with another interface.

3 Shift Registers For Output

To output data to multiple pins using an XS1 device a serial in / parallel out shift register is used. Typically these are 8bits long such as the 74HC595, which is available from a variety of manufacturers and costs $\tilde{\$}0.086$ in low volume (1000 off quantity from Digi-Key).

Then device has the following signals:

- $DATA\_IN$ (DS or SER) - the serial data input to the shift register.
- $SCLK$ (SH_CP or SCK) - the serial clock to clock the data through the shift register.
- MR_N ($\overline{MR}$ or $\overline{SCLR}$) - master reset to wipe the output of the device.
- $LATCH$ (ST_CP or RCK) - latches the contents of the shift register into the storage register.
- OE_N ($\overline{OE}$ or $\overline{G}$) - enable the output to the pins, otherwise the pins are tri-stated.
- $DATA\_OUT$ ($Q7'$) - the data output from the shift register to chain multiple devices together.
- $Q[7:0]$ - the 8-bits of parallel data output.

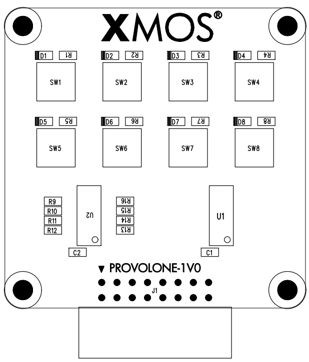


The diagram above details the internal structure of a serial in / parallel out shift register. The shift register shifts the data in from $DATA\_IN$, one bit at a time, on each clock input rising edge. At the same time the data currently in the shift register gets shifted out of $DATA\_OUT$. This allows multiple devices to be chained together to form one long shift register. The MR_N is the master reset signal, which resets all of the shift registers. On the rising edge of the $LATCH$ signal the contents of the shift register are latched from the shift register into the output register. When OE_N (active low) is low, the output register is enabled and the data is available on $Q[7:0]$. Otherwise, when OE_N is high, the output pins are tri-stated into a high impedance state.

4 Example Shift Register Board

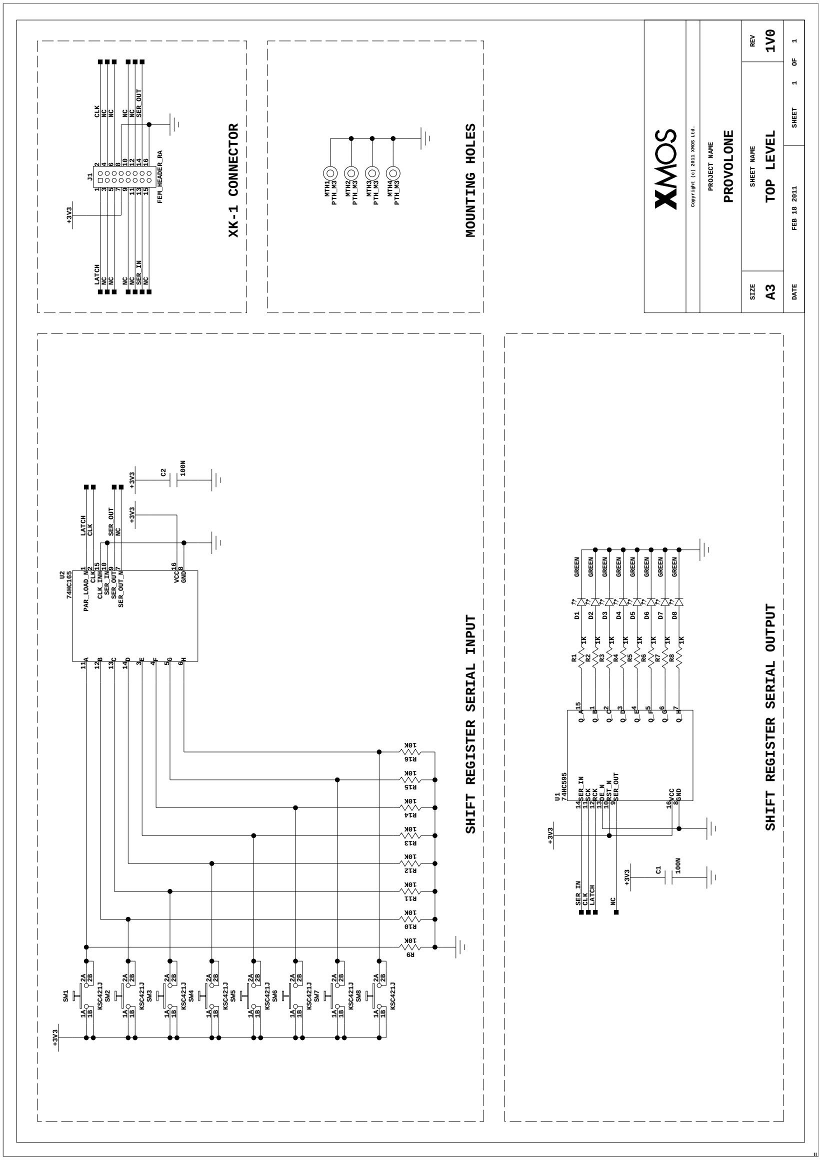
To demonstrate the use of shift registers for both input and output, a board has been designed that uses 4 x 1-bit ports to interface to 2 shift registers. One is for input and one is for output, connected to 8 surface-mount push-button switches and 8 LEDs respectively.

The board is designed to plug onto an I/O expansion bank on an XK-1/XK-1A or XCard format board. It gets 3V3 power and ground from the board which it is plugged into.



The next section provides detailed schematics for this board.

5 Schematics For Example Shift Register Board



6 Software For Example Shift Register Board

Example software to loop the button input back out to the LEDs on the board is shown below. It is written in XC and consists of two threads. One thread simply reflects any data sent to (to loop the buttons back to the LEDs) and the other interfaces to the buttons and LEDs.

There are many different ways of writing the interface code in XC. This one has been designed to use only one timer and plain input and output operations. Such scheme is useful in typical applications, where some or all of the signals are parts of multi-bit ports (and serialization or port conditions cannot be used). This is because it may have to be put into a constrained application where there are not many resources available.

```
#include <xs1.h>
#include <platform.h>
#include <print.h>
#include <stdlib.h>

// Ports for the serial shift registers
on stdcore[0]: out port p_latch = XS1_PORT_1A;
on stdcore[0]: out port p_clk = XS1_PORT_1B;
on stdcore[0]: out port p_mosi = XS1_PORT_1C;
on stdcore[0]: in port p_miso = XS1_PORT_1D;

// Run the clock at 1MHz (2 x 0.5us delays (50 reference clock tick))
#define DELAY 50

// Thread to reflect the data send to it, to loop back the threads and buttons.
void test_reflector ( chanend c_leds, chanend c_buttons )
{
  char my_data;

  // Loop forever
  while ( 1 )
  {
    // Get the value for the buttons
    c_buttons :> my_data;

    // Send it out to the LEDs
    c_leds <: my_data;
  }
}

// Test thread to interface with LEDs and buttons
void test_leds_buttons ( chanend c_leds, chanend c_buttons )
{
  char         led_val = 0, button_val, tmp_but;
  unsigned int i, time, loop_time;
  timer        t;

  // Get the initial timer value
  t :> loop_time;

  // Setup the initial output values
  p_latch <: 1;
  p_clk <: 0;
  p_mosi <: 0;

  // Loop forever
  while ( 1 )
  {
    select
    {
      // Receive a new value for the LEDs over a channel
      case c_leds :> led_val:
        break;

```

```
      // At 100Hz sample the buttons and output the value to the LEDs
      case t when timerafter(loop_time + 100000) :> loop_time:

        // Copy over the loop_time value, so the timing can use it
        time = loop_time;

        // Initialize the button value.
        button_val = 0;

        // Place a rising edge on the latch signal
        // This clocks the previously loaded LED data out and captures the button data
        p_latch <: 0;
        t when timerafter(time + DELAY) :> time;
        p_latch <: 1;
        t when timerafter(time + DELAY) :> time;

        // Cycle through 8 bits
        for ( i = 0; i < 8; i++ )
        {
          // Output the data, MSB bit first.
          p_mosi <: (char) (led_val >> i);

          // Wait for half a bit time
          t when timerafter(time + DELAY) :> time;

          // Set the clock high
          p_clk <: 1;

          // Get the current bit from the shift register
          p_miso :> tmp_but;

          // Add the bit to the button value, LSB bit first.
          button_val = button_val + (tmp_but << i);

          // Wait for half a bit time
          t when timerafter(time + DELAY) :> time;

          // Set the clock low
          p_clk <: 0;
        }

        // Send the button value out
        c_buttons <: button_val;

        break;
    }
  }
}

// Program entry point
int main()
{
  chan c_leds, c_buttons;

  par
  {
    // XCore 0
    on stdcore[0] : test_reflector( c_leds, c_buttons );
    on stdcore[0] : test_leds_buttons( c_leds, c_buttons );
  }

  return 0;
}
```

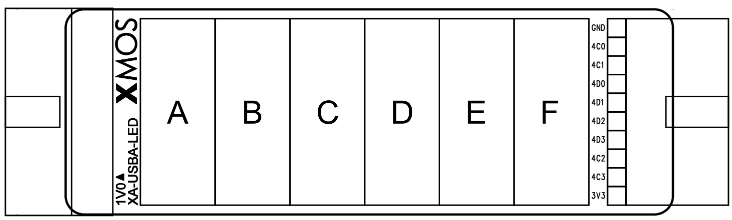
7 XA-USBA-LED Add-On Board

An add-on board had been designed which uses the general purpose input/output expansion header (J19) on the XS1-L2 USB audio board (XR-USB-AUDIO-2.0-MC) to demonstrate the interface to 6 LED bars for level meters. The boards are design to be chained - seven linked together perform satisfactorily. This makes a total length of 42 x LED bars, each being driven by an 8-bit 74VHC595 shift register, making a total shift register length of 336 bits.

To demonstrate the interfacing using a single 4-bit port rather than 1-bit ports, all of the signals are connected to a single 4-bit port with the signal mappings below.

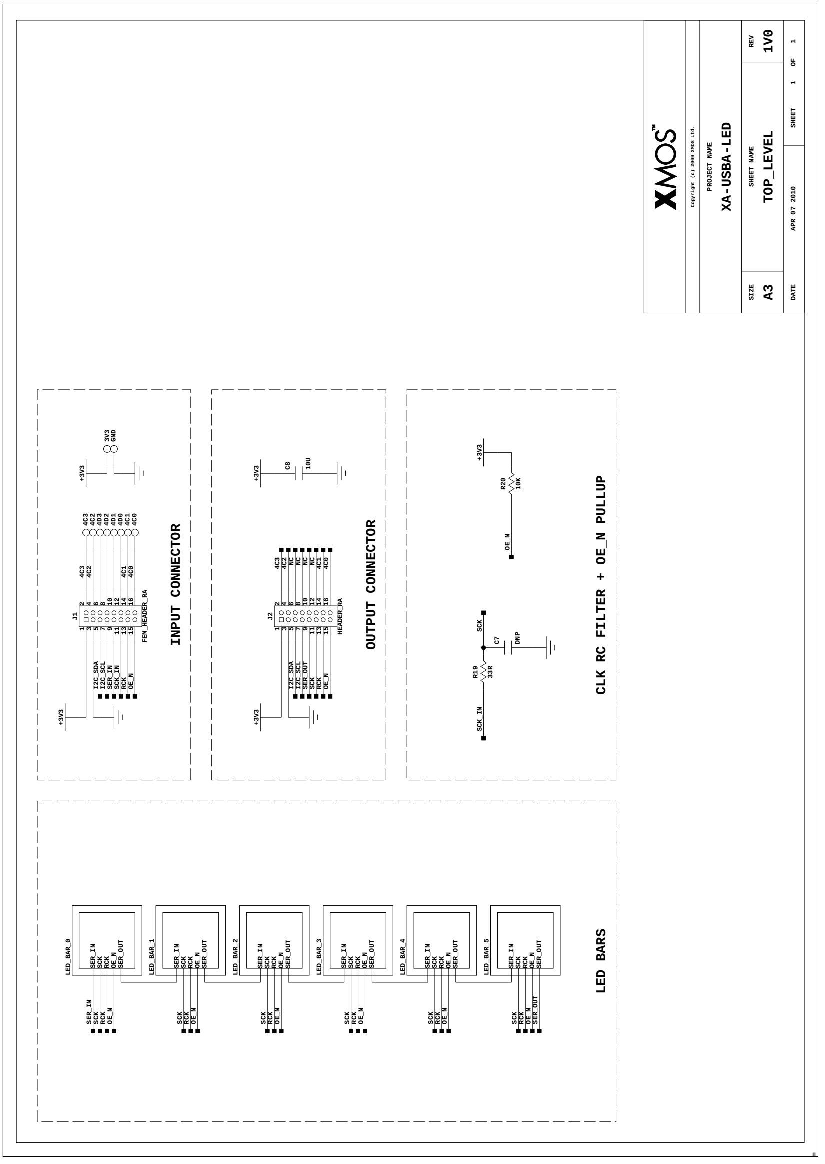
- 4F0 = *SER_IN*
- 4F1 = *CLK*
- 4F2 = *RCK*
- 4F3 = *OE_N*

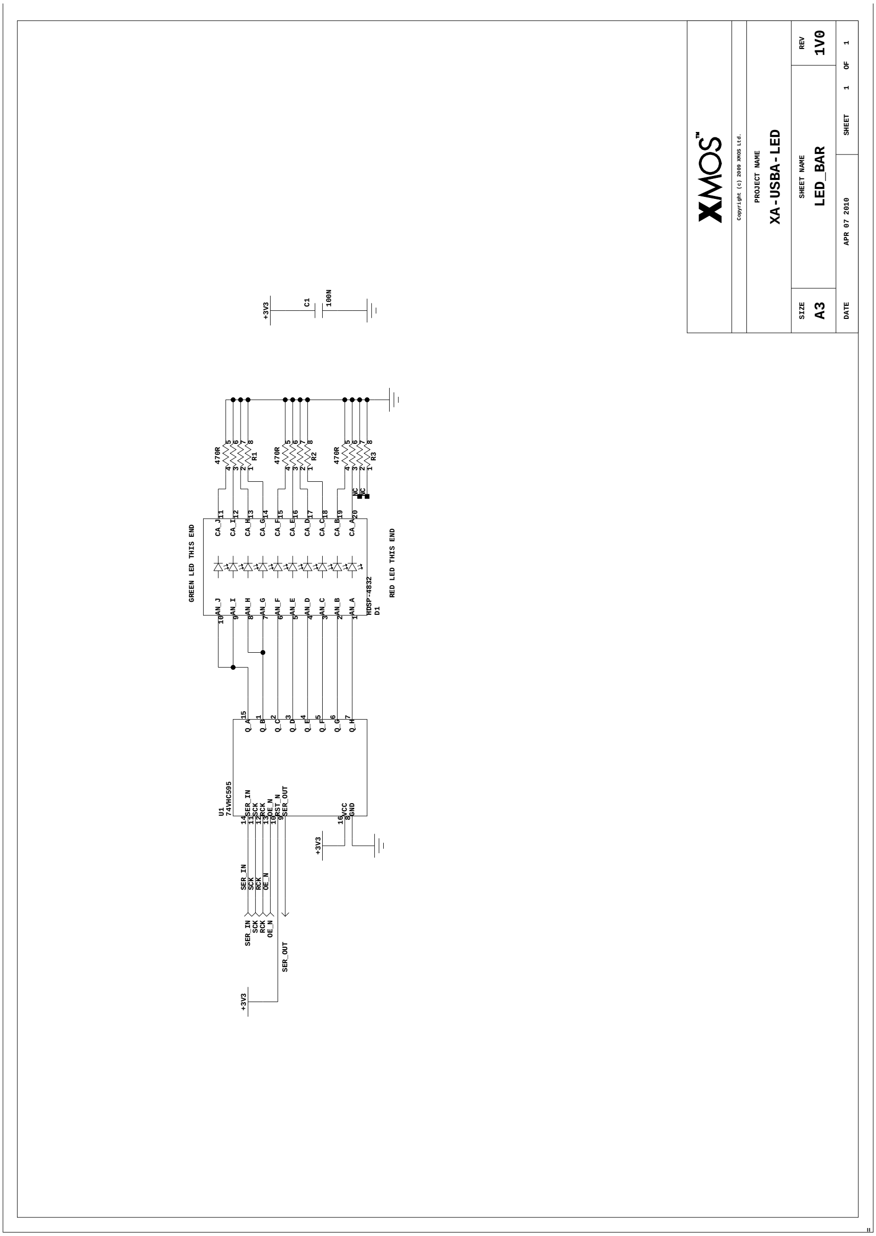
Very High Speed CMOS (VHC) series devices are used to allow the writing of data very quickly to the LED bars, whilst minimizing processing time.

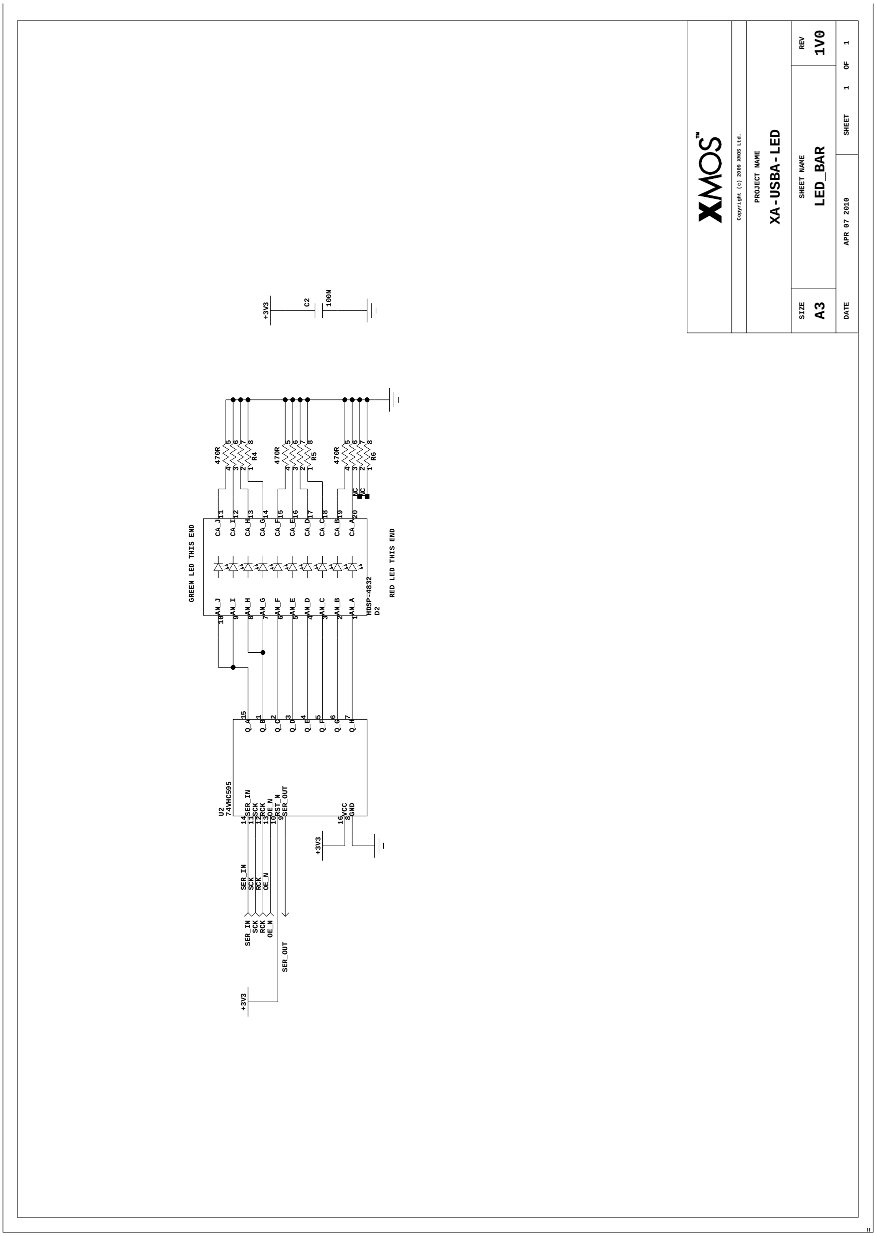


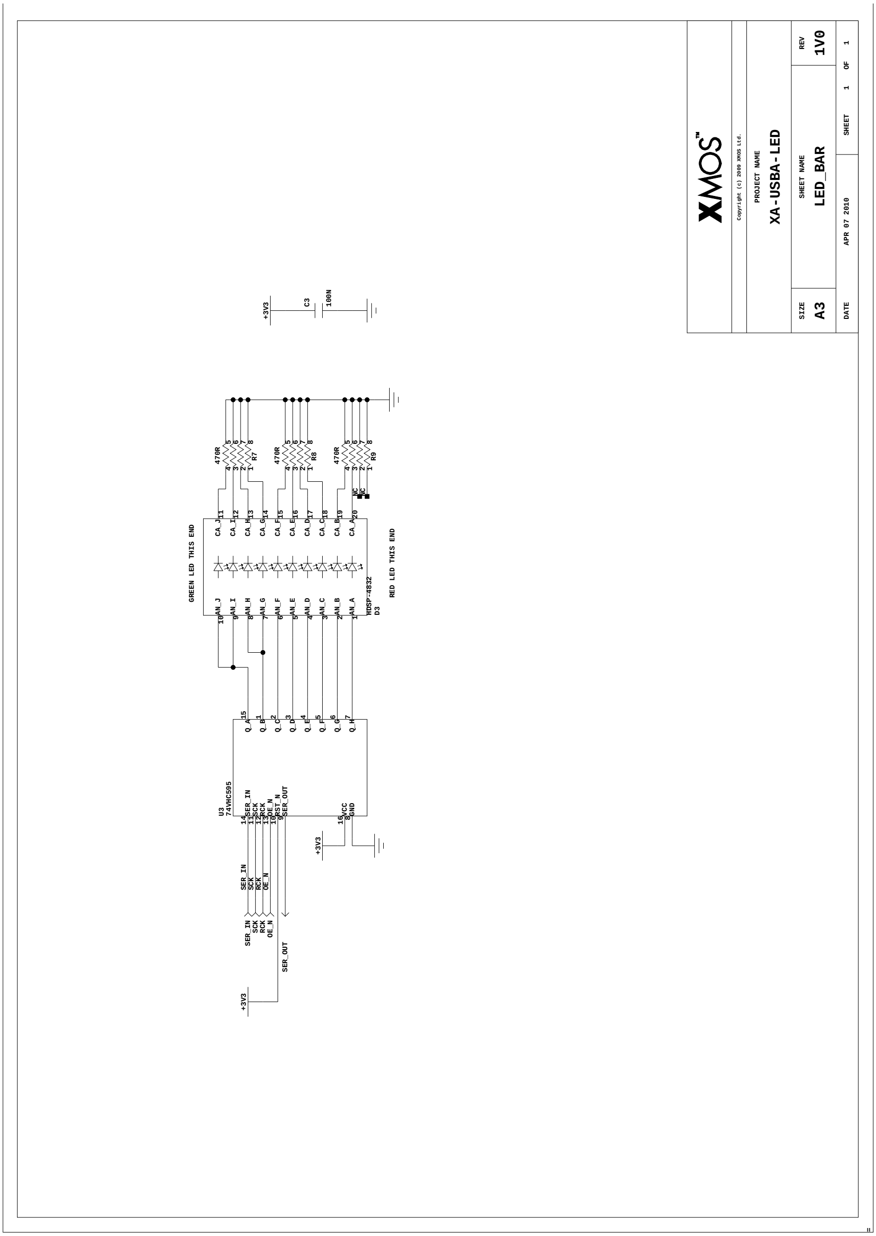
The next section provides detailed schematics for this board.

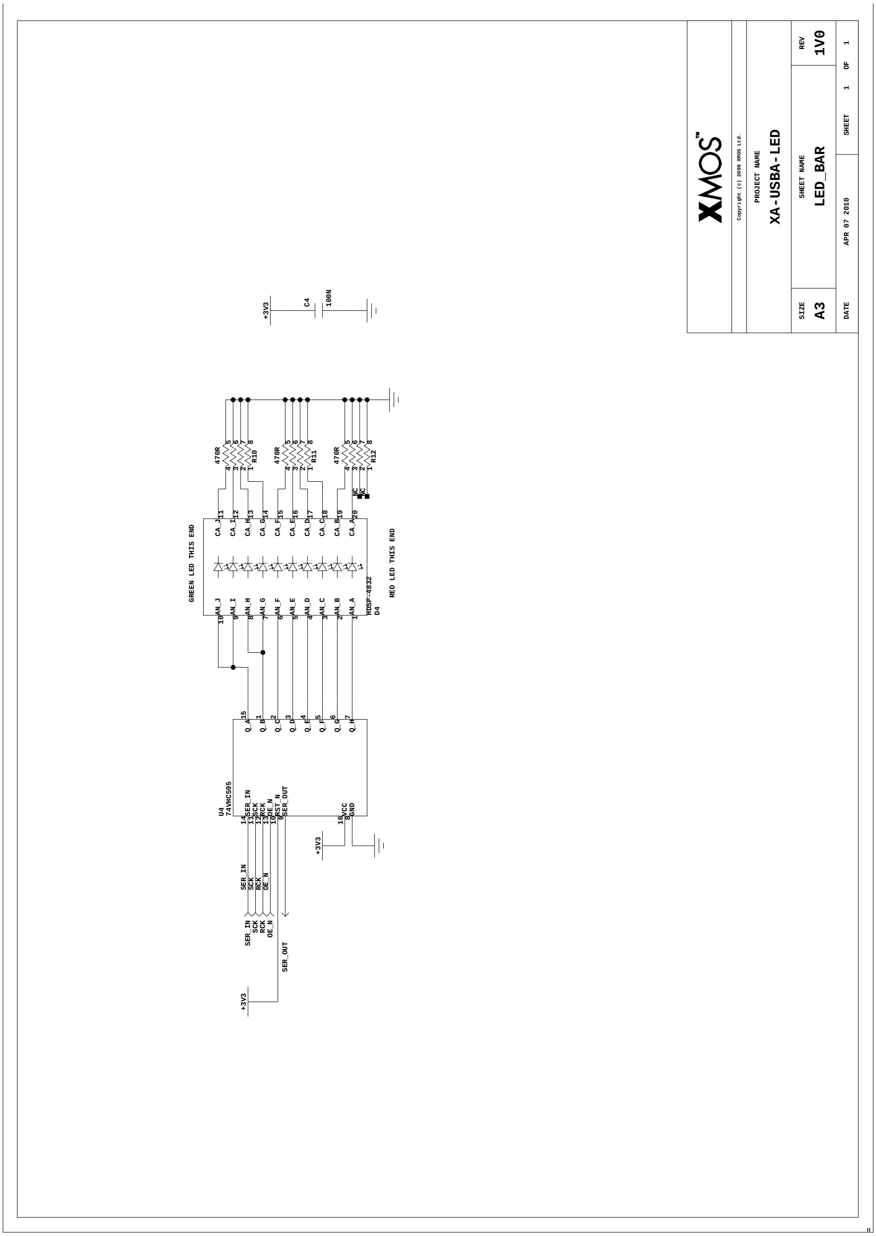
8 Schematics For XA-USBA-LED Add-On Board

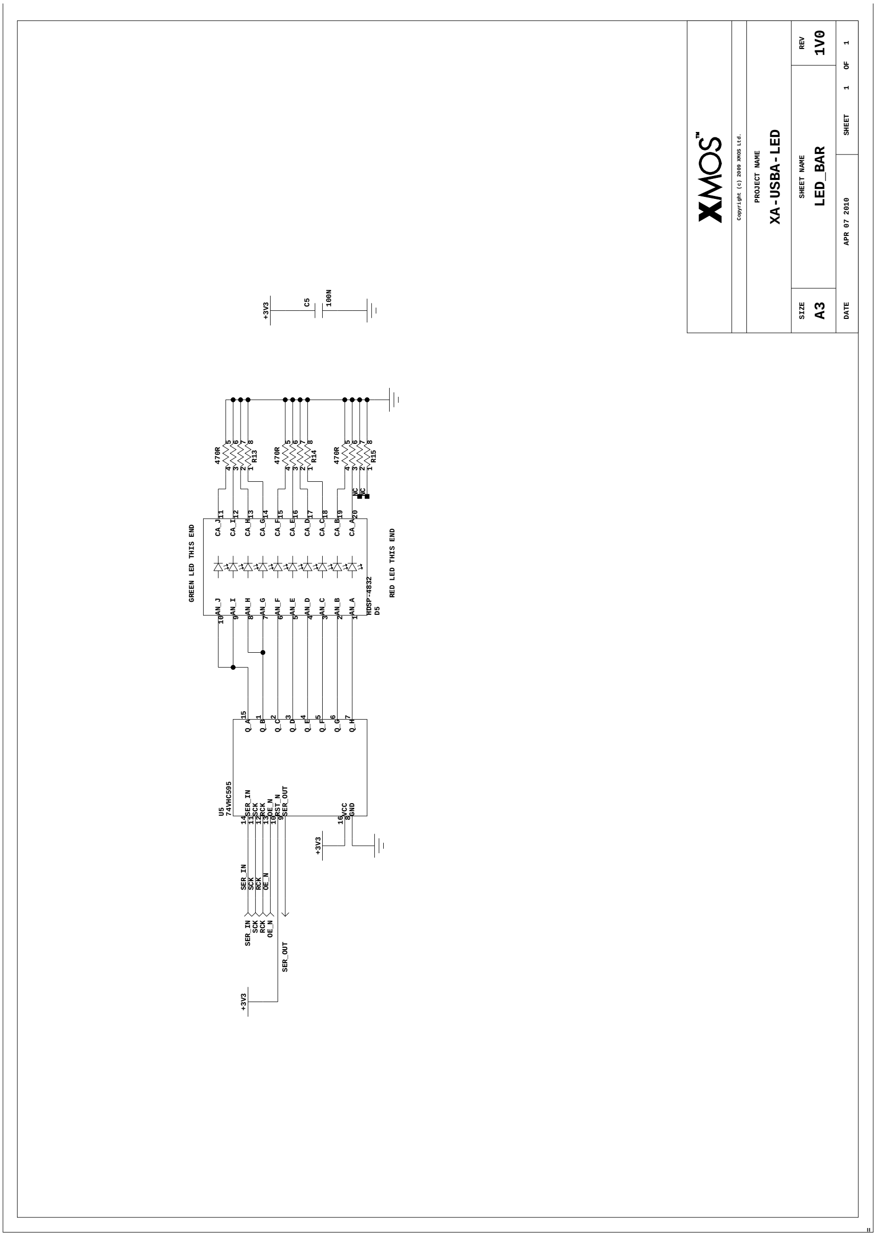


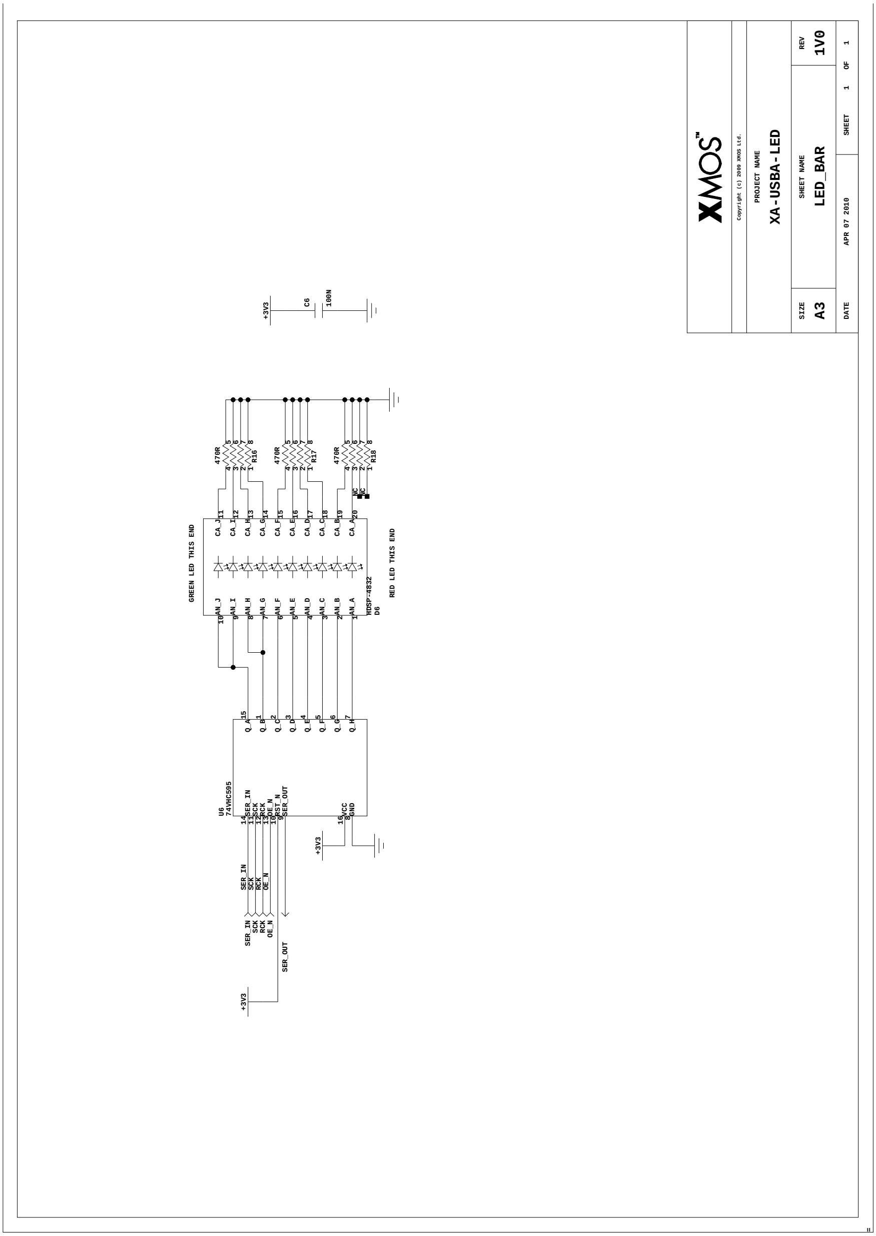

XMOS
Copyright (c) 2009 XMOS Ltd.
PROJECT NAME
XA-USBA-LED
SHEET NAME
LED_BAR
REV
1V0
SIZE
A3
DATE
APR 07 2010
SHEET 1 OF 1
GREEN LED THIS END
RED LED THIS END
HDSP-4832
D1
U1
74VHC595
470R
R1
R2
R3
C1
100N
+3V3
SER_IN
SCK
RCK
OE_N
SER_OUT


XMOS
Copyright (c) 2009 XMOS Ltd.
PROJECT NAME
XA-USBA-LED
SHEET NAME
LED_BAR
SIZE
A3
DATE
APR 07 2010
SHEET 1 OF 1
REV
1V0
+3V3
C2
100N
470R
R4
R5
R6
GREEN LED THIS END
RED LED THIS END
HDSP-4832
D2
U2
74VHC595
SER_IN
SCK
RCK
OE_N
RST_N
SER_OUT
VCC
GND


Copyright (c) 2009 XMOS Ltd.
PROJECT NAME
XA-USBA-LED
SHEET NAME
LED_BAR
SIZE
A3
REV
1V0
DATE
APR 07 2010
SHEET 1 OF 1
+3V3
C3
100N
470R
R7
R8
R9
GREEN LED THIS END
RED LED THIS END
HDSP-4832
D3
U3
74VHC595
SER_IN
SCK
RCK
OE_N
RST_N
SER_OUT
VCC
GND


XMOS
Copyright (c) 2009 XMOS Ltd.
PROJECT NAME
XA-USBA-LED
SHEET NAME
LED_BAR
SIZE
A3
REV
1V0
DATE
APR 07 2010
SHEET 1 OF 1
GREEN LED THIS END
RED LED THIS END
HDSP-4832
D4
U4
74VHC595
C4
100N
+3V3
470R
R10
R11
R12
SER_IN
SCK
RCK
OE_N
SER_OUT


XMOS
Copyright (c) 2009 XMOS Ltd.
PROJECT NAME
XA-USBA-LED
SHEET NAME
LED_BAR
SIZE
A3
REV
1V0
DATE
APR 07 2010
SHEET 1 OF 1
+3V3
C5
100N
470R
R13
R14
R15
GREEN LED THIS END
RED LED THIS END
HDSP-4832
D5
U5
74VHC595
SER_IN
SCK
RCK
OE_N
RST_N
SER_OUT
VCC
GND


XMOS
Copyright (c) 2009 XMOS Ltd.
PROJECT NAME
XA-USBA-LED
SHEET NAME
LED_BAR
SIZE
A3
REV
1V0
DATE
APR 07 2010
SHEET 1 OF 1
GREEN LED THIS END
RED LED THIS END
HDSP-4832
D6
U6
74VHC595
R16
R17
R18
470R
C6
100N
+3V3
SER_IN
SCK
RCK
OE_N
RST_N
SER_OUT
VCC
GND

9 Software For XA-USBA-LED Add-On Board

Example software for a chain of seven XA-USBA-LED add-on boards is shown below. It is written in XC and consists of a single thread, lighting each column on in the chain in turn. The software takes account of the fact that the shift register on each board is effectively byte-reversed relative to the chaining of the boards.

```

// Includes
#include <xs1.h>
#include <platform.h>
#include <print.h>
#include <stdlib.h>


// Ports for the serial shift registers
on stdcore[1]: out port p_led = XS1_PORT_4F;
on stdcore[1]: clock my_clk = XS1_CLKBLK_1;
// 0 = SER_IN
// 1 = CLK
// 2 = RCK
// 3 = OE_N


// Defines for the LED panels
#define NUM_PANELS 7
#define NUM_COLUMNS_PER_PANEL 6
#define TOTAL_COLUMNS (NUM_PANELS * NUM_COLUMNS_PER_PANEL)


// Test thread to interface with LEDs
void test_leds ( void )
{
  char         led_val[TOTAL_COLUMNS], temp = 0;
  signed int     panel, column, row;
  unsigned int   i, loop_time, my_row = 0;
  timer          t;

  // Configure the output to run from a clk blk at 25MHz clock rate
  set_clock_div(my_clk,2);
  configure_out_port(p_led,my_clk,0);
  start_clock(my_clk);

  // Get the initial timer value
  t :> loop_time;

  // Setup the initial output values
  p_led <: 0;

  // Wipe the led_vals
  for ( i = 0; i < TOTAL_COLUMNS; i++ )
  {
    led_val[i] = 0x00;
  }

  // Loop forever
  while ( 1 )
  {
    select
    {
      // At 10Hz update and output the value to the LEDs
      case t when timerafter(loop_time + 10000000) :> loop_time:

        // Loop though all the panels (boards)
        for ( panel = 0; panel < NUM_PANELS; panel++ )
        {
          // Loop through the 6 columns
          for ( column = 5; column > -1; column-- )
          {
            // Loop through each row of the column of 8 bits
            for ( row = 7; row > -1; row-- )
```

```
            {
              // Get the required bit of the LED data, MSB bit first.
              temp = (led_val[(panel * 6) + column] >> row) & 0x1;

              // Place the data onto the port and set the clock high
              p_led <: (temp + 2);

              // Set the clock low
              p_led <: temp;
            }
          }
        }

        // Clock the data into the output registers
        p_led <: 4;

        // Set the current row to 0
        led_val[my_row++] = 0x00;

        // Prevent my_row from oerflowing out of led_val
        if (my_row == TOTAL_COLUMNS)
        {
          my_row = 0;
        }

        // Set the next row to 1
        led_val[my_row] = 0xFF;

        break;
    }
  }
}


// Program entry point
int main()
{
  par
  {
    // XCore 1
    on stdcore[1] : test_leds( );
  }

  return 0;
}
```

10 Related Documents

Information about XMOS technology is primarily available from the XMOS web site; please see http://xmos.com/documentation for the latest documents or click on one of the links below to find out more information.

Document title	Document reference
Programming XC on XMOS Devices	programming-xc-xmos-devices
XK-1 Hardware Manual	xk-1-hardware-manual
USB-AUDIO-2.0-MC Hardware Manual	usb-audio-20-mc-hardware-manual
NXP 74HC165 Datasheet	74hc165-datasheet
NXP 74HC595 Datasheet	74hc595-datasheet

Document History

Date	Release	Comment
2011-05-11	1.0	First release