Application Note

SCREENKEYS

APP1002 INTERFACING SCREENKEYS TO A MICROCONTROLLER

THIS APPLICATION NOTE DESCRIBES HOW TO IMPLEMENT A FAST BUT SIMPLE INTERFACE BETWEEN A MICROCONTROLLER AND FOUR SCREENKEYS. THE DESIGN IS BASED ON AN ATMEL 89C55WD MICROCONTROLLER USING ONLY ONBOARD MEMORY.

Objective

An effective method for implementing a ScreenKey interface is required. This should use a minimal number of IC components and provide a simple programming interface.

Summary

This Application Note describes a useful real-life method for interfacing a microcontroller with multiple ScreenKeys. It requires only three additional chips and does not need a dedicated or software generated clock to drive the ScreenKeys.

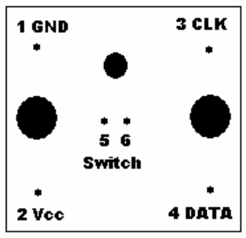
The software requirements to drive the ScreenKeys are greatly simplified by the hardware design and enable very fast ScreenKey display refresh.

The note offers a schematic design and downloadable firmware source code. The firmware can be compiled using the free SDCC GNU public licence compiler.

Hardware Design

ScreenKeys use four pin connections for LCD and LED backlighting control and two pins for the switch:

1. Gnd
2. Vcc (+5V)
3. CLK
4. DATA
5. Switch
6. Switch



Solder side view

The two switch pins implement an electrically isolated single-pole momentary action switch circuit.

Serial Data Generation

Data to the ScreenKey is serially transmitted (DATA) synchronised to a user supplied clock signal (CLK).

The format of the DATA signal is 12 bits transmitted as start bit followed by data (low-bit first) , parity bit and two stop bits:

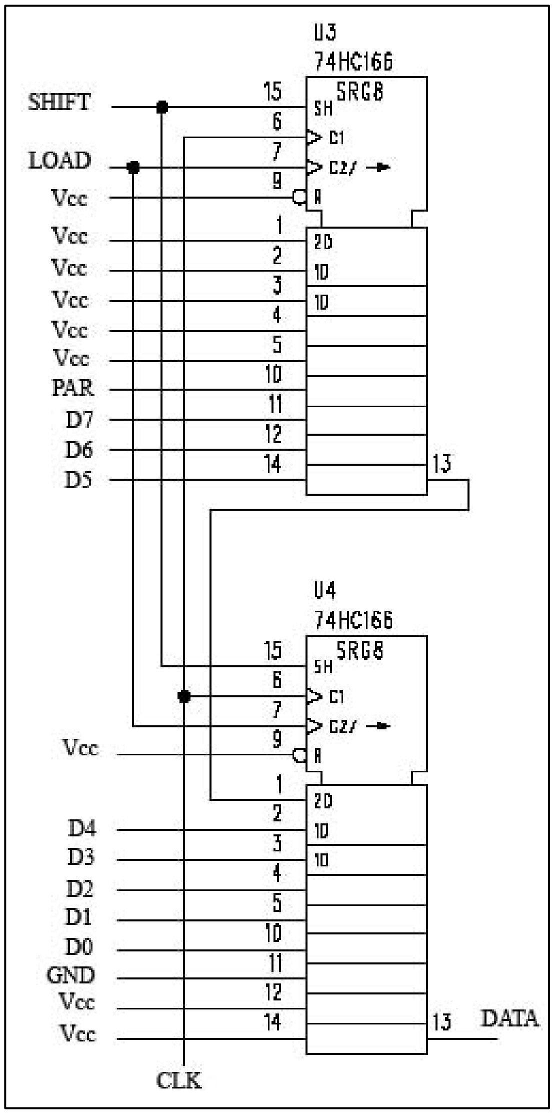
Start Bit	low
Data Bits (d0-d7)	low/high
Parity Bit	low/high
Stop Bits (2 bits)	high

It is possible to generate this data stream serially in software and to coordinate its transmission with a software generated clock. However, such an approach is unnecessarily resource intensive. The speed at which the ScreenKey can be refreshed with new data is restricted by the processing speed of the CPU and what other tasks it may be engaged with.

A much simpler and more effective implementation is to write this 12-bit data word in parallel and to use hardware to serially shift the data out synchronised to the clock (CLK) signal. This can be easily achieved using two parallel-to-serial converters in series (74HC166).

In the example opposite, U3 serial out (pin 13) is shifted directly into U4. U4's serial out feeds the DATA line to the ScreenKey. This means that we actually clock 16 bits per word. Our 12-it word is book-ended by HIGHs (equivalent to STOP bits. Serial data is shifted out by the CLK signal which is also fed directly to the key.

In this example, parallel data is written as with an initial two high bits (U4 – 14,12) then the low START bit (U4 – 11). The 8-bit data byte is next with lsb first (U4 – 10,5,4,3,2 and U3 – 14,12,11). The data bits are followed by the PARITY bit (U3 – 10) and then 2 high STOP bits (U3 – 5,4). Highs are forced onto the remainder of the 16 bits (equivalent to additional STOP bits).



To use this hardware, prepare the data word (8 data bits and appropriate parity – see ScreenKey datasheet for parity usage) and set these inputs. Load this word into the shift register by toggling SHIFT to low and back to high. Enable this data to be serially shifted into the ScreenKey by setting LOAD to low.

Important Note:
You must give sufficient time for the data to be clocked out serially before loading new data into the registers.

CLK Signal

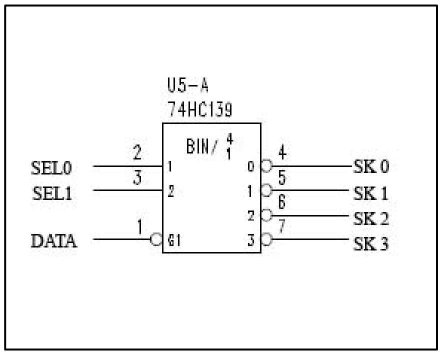
The ScreenKey datasheet specifies that the CLK signal can be anywhere between 50KHz and 4MHz. With a microcontroller circuit there is usually some CPU related clock signal that can be tapped to generate this CLK signal. For example, with an 8051-based CPU, the ALE (Address Latch Enable) signal can be used (some gearing mechanism can be employed to bring the CPU crystal frequency into the allowed CLK frequency range if necessary).

For hardware in this application note, we use a 22.1184MHz crystal. Measurement of the ALE signal shows that this produces a suitable CLK signal of approximately 4MHz. With some simple buffering this signal can be used to feed the ScreenKey CLK signal and to clock out the serial data from the parallel-to-serial converters.

In this example hardware, we use a spare 2-4 Line Decoder (74HC139) to provide the necessary buffering.

Driving Multiple ScreenKeys

Driving more than one ScreenKey is simply a matter of enabling the DATA line to be routed separately to each of the desired number of ScreenKeys.



For this application note, we will implement a design to control four ScreenKeys. A simple 2-4 line decoder will enable us to selectively route the DATA line from the parallel-to-serial converters to one of four individual outputs. The 74HC139 provides two such devices in one package.

There is no need to selectively apply the CLK signal to each ScreenKey. The CLK signal may be applied continuously to all ScreenKeys at the same time. Standard line buffering techniques should be employed if more than four ScreenKeys are to be driven from the same CLK line.

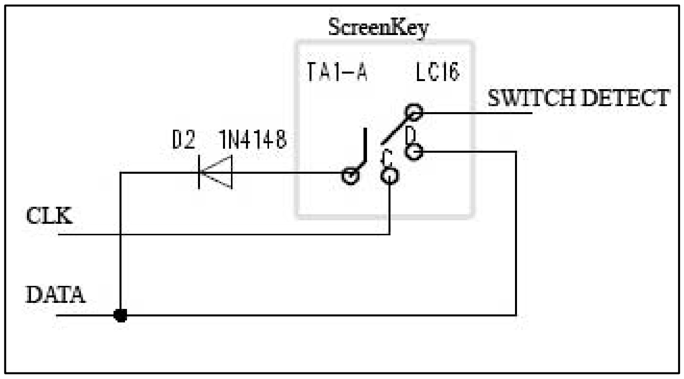
Reading Switch Presses

The ScreenKey switch mechanism is an electrically isolated circuit and operates as a single-pole normally open momentary action.

Any standard approach to detecting key switches can be employed. Usually this involves setting one side of the switch to a certain state and reading the other side of the switch to see if the signal is being passed across the switch. This requires some form of key scanning to alternately apply this test to each key individually.

This approach is implemented here making use of the ScreenKey selector (2-4 line decoder) described in the section above.

The arrangement in the diagram opposite shows a simple way to detect the switch state. One side of the switch is connected to the DATA line via a fast switching diode (1N4148). The other side is connected to a microcontroller IO line. The MCU IO line is set high and then a series of lows are put onto the DATA line. If the switch is pressed, the MCU IO line will read back as low. If not pressed, the MCU line will still read back as high.

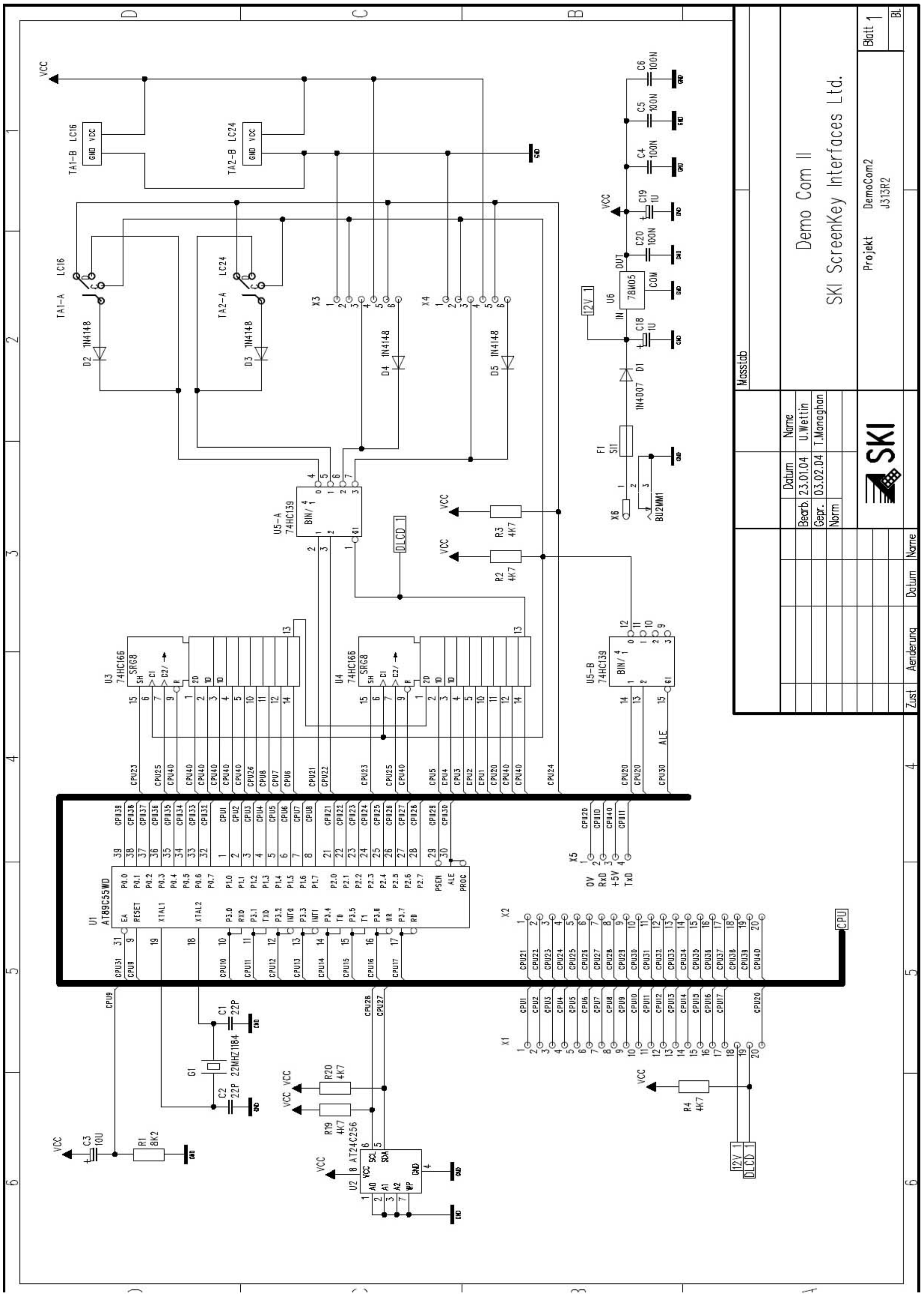


Complete Working Schematic (DemoComII)

The above features can be combined into a fully working hardware design that can control four ScreenKeys. A populated and fully working pcb of this design is available from SKI, called DemoComII. DemoComII is shipped with two ScreenKeys connected and two connections to allow an additional two ScreenKeys to be connected externally. The full schematic is shown on the next page.

The design uses an Atmel 89C55WD microcontroller with 20kbytes on board flash code memory and 256 bytes internal RAM. There is no need for any additional external memory.

Key position 0 has an 32x16 resolution ScreenKey and key position 1 has a 36x24 resolution ScreenKey. This is only important from the perspective of the firmware. Key positions 2 and 3 are connected optional external attachments.


Demo Com II
SKI ScreenKey Interfaces Ltd.
Projekt DemoCom2 J313R2
Blatt 1
Bearb. 23.01.04 U.Wettin
Gepr. 03.02.04 T.Monaghan
Norm
Datum Name
Masstab
Zust Änderung Datum Name
U1 AT89C55WD
U2 AT24C256
U3 74HC166
U4 74HC166
U5-A 74HC139
U5-B 74HC139
U6 78M05
TA1-A LC16
TA1-B LC16
TA2-A LC24
TA2-B LC24
X1
X2
X3
X4
X5
X6 BU2MM1
F1 511
D1 1N4007
D2 1N4148
D3 1N4148
D4 1N4148
D5 1N4148
G1 22MHZ1184
C1 22P
C2 22P
C3 10U
C4 100N
C5 100N
C6 100N
C18 1U
C19 1U
C20 100N
R1 8K2
R2 4K7
R3 4K7
R4 4K7
R19 4K7
R20 4K7
VCC
GND
12V 1
DLCD 1
CPU
ALE
0V RxD +5V TxD

Firmware Design

This section describes software routines to control the ScreenKeys in the DemoComII schematic. A fully working implementation can be downloaded from www.ScreenKeys.com in "Application Notes". This may be compiled using a free gnu public licence 'C' compiler called SDCC (see sdcc.sourceforge.net).

Key Selection

CPU pins P2.0 and P2.1 are used to control the 2-4 line decoder (U5-A) to select which key is being controlled. This decoder selects the Screenkey to receive the serial DATA out from the parallel-to-serial converters.

```
void SwSelect(char SKey){
   SKey = SKey & 0x03 ;

   switch(SKey){
   case 0 :
      P20 = 0;
      P21 = 0;
      break;
   case 1 :
      P20 = 1;
      P21 = 0;
      break;
   case 2 :
      P20 = 0;
      P21 = 1;
      break;
   case 3 :
      P20 = 1;
      P21 = 1;
      break;
   }
}
```

Writing Commands and Data to a ScreenKey

To send commands or data to a ScreenKey, the data word is constructed and written into the parallel-to-serial converters. This is then shifted out in serial by the CLK signal.

In the DemoComII schematic, the 8-bit data word is output on CPU Port 1 (bits 0 – 7) and the parity bit is output on CPU pin 2.5. Control of the parallel-to-serial converters is managed by port pin P2.2 to load the data into the registers and port pin P2.4 to enable the serial shift out.

```
#define        LCD_DATA         P1
#define        SHIFT_LOAD       P22
#define        CLK_INHIBIT      P24
#define        PARITY           P25

/******************************************************
//  Function: SKDataOut
//            Write data to ScreenKey
//
//  Parameters:
//      SKData = data to be written to ScreenKey
//      Parity = type of parity reqd
//               0 = even parity (even number of 1’s
//               1 = odd parity (odd number od 1’s)
******************************************************/
void SKDataOut(char SKData, char Parity)
{
   char i;

   //Load data into shift registers
   LCD_DATA = SKData;

   //Calculate parity bit
   for (i=0; i<8; i++)
   {
      Parity = (SKData >> i) ^ Parity;
   }

   //Set parity bit on port P2.5
   PARITY = Parit & 0x01;

   //Load data into shift registers
   CLK_INHIBIT = 0;
   SHIFT_LOAD  = 0;
   CLK_INHIBIT = 1;
```

```
        //Start data shift out
        SHIFT_LOAD  = 1;
        CLK_INHIBIT = 0;

        //Use this function to read state of switch closure
        P23 = 1 ;
        SwitchFlag = P23 ;

        //Implement delay to give enough time
        //for data to shift out
        i=0;
        while(i<6)
          i++;
     }
```

Read Switch Status

The section of code in the above function (SKDataOut) that manipulates port pin P2.3 is included as a way of detecting the switch status of the ScreenKey that has just been written to. P2.3 is made high as the CPU output. If the low nibble of the data byte just written is 0, then reading back pin P2.3 as soon as shifting begins will read as high if the switch is not pressed and low if it is pressed.

To use this functionality, each ScreenKey must be individually selected and a suitable data word (e.g. 0x80) written to the key. The result of the check is checked in the variable SwitchFlag set in SKDataOut().

```
     void Scan(void){

        // Scanning is only active when ScanFlag is set (== 1)
        if(ScanFlag == 1){

           //First check key position 0
           SwSelect(0);
           SKDataOut(0x80,0);
           if(SwitchFlag == 0){

              //Switch pressed
              //Do something

           } else {

              //Switch NOT pressed
              //Do something
```

```
            }

            //Check key position 1
            SwSelect(1);
            SKDataOut(0x80,0);
            if(SwitchFlag == 0){

               //Switch pressed
               //Do something

            } else {

               //Switch NOT pressed
               //Do something

            }

            //Disable scanning
            //It must then be forcible truned back on if key
            //switch detection required
            ScanFlag = 0;
         }
      }
```

ScreenKey Initialisation

ScreenKeys must be initialised on power on. The ScreenKey datasheets explain the sequence of commands to be sent. This code assumes there is a 32x16 ScreenKey in position 0 and a 36x24 ScreenKey in position 1.

This initialisation code writes a significantly lower frequency value than that specified in the datasheet. This is a safe decision as the ALE signal is not necessarily consistent and depends on exactly what opcodes are currently being executed. The freq register may be set lower than the actual frequency but should never be set higher. Usually it is safe to use a value for one-quarter the actual frequency. This hardware uses an approximate clock frequency of 4MHz but the keys are intialised for an equivalent value of 1.5MHz.

```
      #define        LC_FREQREG        0xEE
      #define        LC_MUXREG         0xEF

      void LC16Init(void){
         //Select ScreenKey in position 0
         SwSelect(0) ;

         //Init LC16 Freq Reg for 1.5MHz clock
```

```
        SKDataOut(0x00,0);          //Start byte, even parity
        SKDataOut(LC_FREQREG,1);    //Cmd Freq Reg, odd parity
        SKDataOut(0xB0,1);          //Data, odd parity
        SKDataOut(0xAA,0);          //End byte, odd parity

        //Write required LC16 values to MUX Reg
        SKDataOut(0x00,0);
        SKDataOut(LC_MUXREG,1);
        SKDataOut(0x02,1);
        SKDataOut(0x05,1);
        SKDataOut(0xAA,0);
     }


     void LC24Init(void){
        //Select ScreenKey in position 1
        SwSelect(1);

        //Init LC24 Freq Reg for 1.5MHz clock
        SKDataOut(0x00,0);          //Start byte, even parity
        SKDataOut(LC_FREQREG,1);    //Cmd Freq Reg, odd parity
        SKDataOut(0xC0,1);          //Data, odd parity
        SKDataOut(0xAA,0);          //End byte, odd parity

        //Write required LC24 values to MUX Reg
        SKDataOut(0x00,0);
        SKDataOut(LC_MUXREG,1);
        SKDataOut(0x07,1);
        SKDataOut(0x00,1);
        SKDataOut(0xAA,0);
     }
```

Writing Graphics to a ScreenKey

ScreenKeys use bit-mapped graphics. Graphic data begins at location 0x80 in the ScreenKey memory. Each memory bit from location 0x80 onwards is mapped to a particular pixel on the ScreenKey LCD screen.

The 36x24 resolution ScreenKey and the 32x16 resolution ScreenKey use very different bit-mappings. The particular ScreenKey datasheets explain each mapping in detail.

32x16 resolution ScreenKeys use 64 bytes to map the LCD screen: 16 rows with 4 bytes (32 bits) per row. The lsb of byte 0 corresponds to the top **left** hand pixel on the LCD.

36x24 resolution ScreenKeys use 108 bytes to map the LCD screen: 24 rows with 4.5 bytes (32 + 4 bits) per row. The lsb of byte 0 corresponds to the top **right** hand pixel on the LCD. Starting with byte 4, every 9th byte is shared between rows (low nibble at end of previous row and high nibble on start of next row).

Another application note describes how to manipulate and design graphics for each ScreenKey resolution.

```
code unsigned char LC2436x24[108] = {
    0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x80 ,
    0x03, 0x00, 0x00, 0x00, 0x7C, 0x00, 0x00, 0x00 ,
    0x00, 0x01, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00 ,
    0x00, 0x00, 0x01, 0x00, 0x00, 0x80, 0xC1, 0x01 ,
    0x00, 0x00, 0x28, 0x22, 0x00, 0x00, 0x80, 0x24 ,
    0x00, 0x00, 0x00, 0x88, 0x0C, 0x00, 0x00, 0xC0 ,
    0x0F, 0x01, 0x00, 0x00, 0x08, 0x20, 0x00, 0x00 ,
    0x80, 0xE0, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00 ,
    0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x00, 0x01 ,
    0x00, 0x8E, 0x03, 0x10, 0x40, 0x00, 0x45, 0x00 ,
    0x21, 0x06, 0x50, 0x00, 0x10, 0xF6, 0xE7, 0x9D ,
    0xFF, 0xFF, 0x06, 0x51, 0x00, 0x10, 0x46, 0x10 ,
    0x45, 0xC0, 0x27, 0x00, 0x8E, 0x03, 0x38, 0x00 ,
    0x00, 0x00, 0x00, 0x01 };

code unsigned char LC1632x16[64] = {
    0xFB, 0xFF, 0xFF, 0xFF, 0xF1, 0xFF, 0xFF, 0xFF,
    0xE0, 0xFF, 0xFF, 0xFF, 0xFB, 0xFF, 0xFF, 0xFF,
    0x3B, 0xCE, 0xBF, 0xF3, 0xFB, 0xB5, 0x9F, 0xED,
    0xFB, 0xBD, 0xBF, 0xFD, 0xFB, 0xDE, 0xB6, 0xF1,
    0xFB, 0xED, 0xB9, 0xED, 0xFB, 0xF5, 0xB9, 0xED,
    0x3B, 0x86, 0x16, 0xF3, 0xFB, 0xFF, 0xFF, 0xDF,
    0xFB, 0xFF, 0xFF, 0x9F, 0x00, 0x00, 0x00, 0x00,
    0xFB, 0xFF, 0xFF, 0x9F, 0xFB, 0xFF, 0xFF, 0xDF };

#define        LC_PIXREG         0x80

void LC16Pict(char KeyNr, char *PictData){
   unsigned char i;

   SwSelect(KeyNr);
   SKDataOut(0x00, 0);     //Start Byte
   SKDataOut(LC_PIXREG,1); //Select Pixel Address

   //Write graphic data – 64 bytes * 8 pixels
   for(i=0 ; i<64 ; i=i+1) {
      SKDataOut(PictData[i], 1);
   }
}
```

```
void LC24Pict(char KeyNr, char *PictData){
   unsigned char i;

   SwSelect(KeyNr);
   SKDataOut(0x00, 0);      //Start Byte
   SKDataOut(LC_PIXREG,1);  //Select Pixel Address

   //Write graphic data – 108 bytes * 8 pixels
   for(i=0 ; i<108 ; i=i+1) {
      SKDataOut(PictData[i], 1);
   }
}
```

Set ScreenKey Backlight Colors

There are two series of ScreenKeys with different backlight color selections:

LC Series:	red and green LED backlighting
RGB Series:	red, green and blue backlighting

The respective datasheets explain how each LED can be switched on or off and set to full or half brightness. The datasheets also explain how composite colors can be achieved by mixing the base LEDs in different ways.

Basically, the LED color register is at location 0xED in the ScreenKey memory. The bit control structure at this location differs for RGB and LC series.

LC Series backlight control:

	B7	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**
Colour	Red (Left)	Red (Right)	Green (Left)	Green (Right)	Red (Left)	Red (Left)	Green (Right)	Green (Right)
0/ 1	Dark / Bright	Dark / Bright	Dark / Bright	Dark / Bright	Off / On	Off / On	Off / On	Off / On

LED backlighting in the LC Series ScreenKeys can be individually controlled on both the left and right sides. The lower nibble in the control register (B0-B3) controls whether the particular LED's are on (=1) or off (=0). The upper nibble (B4-B7) specifies the brightness for LEDs that are turned on (1 = full or bright / 0 = half or dark).

RGB Series backlight control:

	B7	B6	B5	B4	B3	B2	B1	B0
Colour	n/a	Green	Red	Blue	n/a	Green	Red	Blue
0/ 1		Dark / Bright	Dark / Bright	Dark / Bright		Off / On	Off / On	Off / On

RGB Series backlighting can be individually controlled for each LED base color. The lower nibble in the control register (B0-B2) controls whether the particular LED's are on (=1) or off (=0). The upper nibble (B4-B6) specifies the brightness for LEDs that are turned on (1 = full or bright / 0 = half or dark). Bits B3 and B7 are not used on the RGB Series.

```
#define        LC_COLREG        0xED

//ScreenKey RGB Colors
#ifdef RGB_TYPE
     #define        OFF                  0x00
     #define        DARKRED              0x02
     #define        BRIGHTRED            0x22
     #define        DARKGREEN            0x04
     #define        BRIGHTGREEN          0x44
     #define        DARKBLUE             0x01
     #define        BRIGHTBLUE           0x11
     #define        YELLOW               0x26
     #define        DARKPURPLE           0x03
     #define        BRIGHTPURPLE         0x33
     #define        PINK                 0x23
     #define        TORQUOISE            0x05
     #define        WHITE                0x27
#endif

//ScreenKey LC Colors
#ifdef LC_TYPE
     #define        OFF                  0x00
     #define        DARKGREEN            0x03
     #define        BRIGHTGREEN          0x33
     #define        DARKRED              0x0C
     #define        BRIGHTRED            0xCC
     #define        DARKORANGE           0x0F
     #define        BRIGHTORANGE         0xFF
     #define        GREENORANGE          0x3F
     #define        REDORANGE            0xCF
#endif
```

```
void SetColor(char KeyNr, char Color){
   SwSelect(KeyNr);
   SKDataOut(0x00, 0);       //Start Byte
   SKDataOut(LC_COLREG,1);   //Select COLOR Register
   SKDataOut(Color, 1);      //Write selected color
}
```

References & Downloads

The hardware design used and described in this Application Note is implemented by SKI in a demonstration/development board called DemoComII. This is available for purchase directly from SKI.

The following downloads related to this Application Note are available from www.ScreenKeys.com:

[1] LC16, LC24, RGB16 and RGB24 Datasheets

[2] DemoComII schematic

[3] ‘C’ source code that implements basic functions to set ScreenKey colors, draw images and to read keypresses

[4] ‘C’ source code for full DemoComII demonstration application

[5] SDCC gnu public licence compiler (sdcc.sourceforge.net) that can be used to compile the above source code to run on the DemoComII hardware.

ScreenKeys Contact Details

SK Interfaces Ltd	Phone: +353 1 8855 075
Unit 11 Keypoint Business Park	Fax: +353 1 8855095
42 Rosemount Park Drive	
Ballycoolin Road	Email:
Dublin 11	info@ScreenKeys.com
Ireland	

www.ScreenKeys.com