Application Note

APP1001 BASIC SCREENKEY CONTROL

THIS APPLICATION NOTE DESCRIBES A BASIC METHOD FOR INTERFACING AND CONTROLLING SCREENKEYS USING A MICROCONTROLLER WITH NO OTHER COMPONENTS. THIS METHOD IS NOT RECOMMENDED FOR ACTUAL USE AND IS PRESENTED ONLY AS A MEANS FOR UNDERSTANDING HOW SCREENKEYS WORK.

Objective

A basic method for controlling ScreenKeys is required using only a simple microcontroller with no other components. The clock and data signal to the ScreenKey are to be generated in software only as a means to understand how ScreenKeys operate.

Summary

This Application Note describes the basic operation of ScreenKeys and how to implement a very simple interface. The ScreenKey clock and data signals are generated in software as per the ScreenKey datasheet and implemented using a simple microcontroller with no other components.

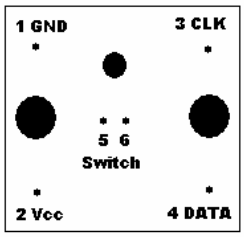
This design and software is presented only for instructional purposes and is not recommended for actual use.

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

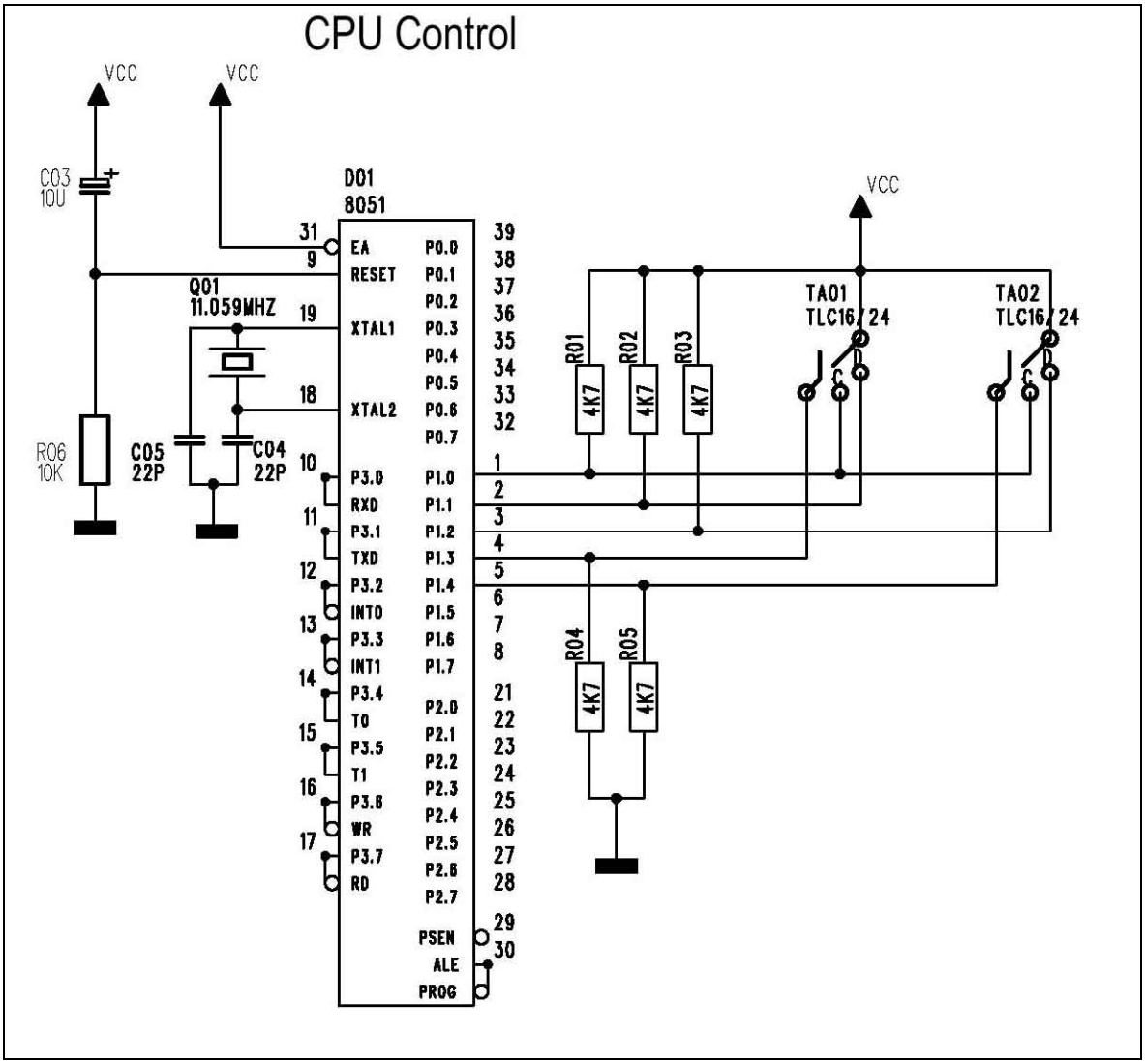


Solder side view

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

Simple ScreenKey Interface

The easiest method for connecting a single ScreenKey is to directly connect one microcontroller output to the ScreenKey DATA pin and another microcontroller output to the ScreenKey CLK pin:



In this example, a standard 8051-based microcontroller is used to directly drive two ScreenKeys. The following 8051 port pins are directly connected to the ScreenKeys:

Port 1.0	CLK line connected to both ScreenKeys
Port 1.1	DATA line to ScreenKey TA01
Port 1.2	DATA line to ScreenKey TA02
Port 1.3	Read switch from ScreenKey TA01
Port 1.4	Read switch from ScreenKey TA02

The CLK and DATA lines should use 4k7 pull-up resistors to +5V supply.

One side of both switch mechanisms on the ScreenKeys should be connected to +5V supply line. The other side of the switch (input to the microcontroller) should be connected to ground via a 4k7 pull-down resistor.

Firmware Design

This section describes software routines to control the ScreenKeys in the example schematic above. 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).

Clock Signal

The CLK signal is generated on the microcontroller output pin P1.0. The ScreenKey datasheet explains that DATA is clocked into the ScreenKey on the falling edge of the CLK signal. Below is a simple routine that toggles P1.0 this line to low and back to high. This routine assumes that P1.0 is normally left set to high.

To use this routine, each data bit should be setup on P1.1 (TA01) or P1.2 (TA02) and then call Clock(1) to register this bit value into the key. Clock() accepts a parameter to allow multiple clock cycles to be executed (clocking the same data bit value on each clock cycle).

```
//*************************************************************
//*************************************************************
//
// FUNCTION NAME  : Clock
//
// INPUTS         : cTick    - No of clock cycles to execute
//
// OUTPUTS        : None
//
//
// DESCRIPTION    : Toggles CLK line
//
//*************************************************************
//*************************************************************
void Clock(unsigned char cTicks)
{
   unsigned char i ;

   for(i = 0 ; i < cTicks ; i++) {
      P10 = 0 ;
      P10 = 1 ;
   }

   return;
}
```

Serial Data Generation

Data to the ScreenKey is serially transmitted (DATA) synchronised to the 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

The parity bit is set depending on the type of data being written to the key.

To send an instruction to a ScreenKey the following structure must be followed:

Start Byte	even parity	(always 0x00)
Command Byte	odd parity	(see datasheet for list of commands)
Data Byte(s)	odd parity	(1 or more data bytes)
End Byte	even parity	(always 0xAA)

Start and end bytes frame the packet being sent to the key. Start and end bytes use even parity while command and data bytes use odd parity.

The following code example describes a routine that structures each byte according to the ScreenKey specification and outputs this on the relevant microcontroller output line.

```
//*************************************************************
//*************************************************************
//
// FUNCTION NAME  : SendByte
//
// INPUTS         : cDataByte    - Data byte to send to the key
//                  cParity      - 0 = start/end byte,
//                                 1 = command/data byte
//                  cKey         - ScreenKey no to send to
//
// OUTPUTS        : None
//
//
// DESCRIPTION    : Formats data byte according to ScreenKey
//                  specification and sends data to the key:
//                         1 x start bit
//                         8 x data bits (lsb first)
//                         1 x parity bit
//                         2 x stop bits
//
//*************************************************************
//*************************************************************
void SendByte(unsigned char cDataByte, unsigned char cParity, unsigned char
cKey)
{
   unsigned char i, cBit;

   //Calculate parity for this databyte type
   // cParity = 0 => even parity (start/end byte)
   //         = 1 => odd parity (command/data byte)
   for (i = 0; i < 8; i++)
       cParity = (cDataByte >> i) ^ cParity;

   cParity = cParity & 0x01;

   //First send 1 start bit
   SetBit(0, cKey)  ;   //Set data up
   Clock(1) ;           //Toggle clock line

   //Now send 8 data bits - lsb first
   for(i = 0 ; i < 8 ; i++) {
      //Get lsb of data
      cBit = ( cDataByte & 0x01) ;

      //Send bit to key
      SetBit(cBit, cKey) ;
      Clock(1) ;
```

```
      //Shift data right to ready next bit
      cDataByte = cDataByte >> 1 ;
   }

   //Send computed parity bit
   SetBit(cParity, cKey) ;
   Clock(1) ;

   //Lastly, send two stop bits
   SetBit(1, cKey) ;
   Clock(1) ;
   SetBit(1, cKey) ;
   Clock(1) ;

   return;
}


//*************************************************************
//*************************************************************
//
// FUNCTION NAME  : SetBit
//
// INPUTS         : cBit     - Data bit to send to the key
//                  Key      - ScreenKey no to send to
//
// OUTPUTS        : None
//
//
// DESCRIPTION    : Sets DATA line to specified key
//
//*************************************************************
//*************************************************************
void SetBit(unsigned char cBit, unsigned char cKey)
{
   //Set P1.1 for TA01 (LC16)
   if(cKey == TA01 ){
      if(cBit == 0 )
         P11 = 0 ;
      else
         P11 = 1 ;
   }

   //Set P1.2 for TA02 (LC24)
   if(cKey == TA02 ){
      if(cBit == 0)
         P12 = 0 ;
      else
         P12 = 1 ;
   }
   return;
}
```

ScreenKey Initialisation

Before use, each ScreenKey must be initialised according to the ScreenKey specification. The following ScreenKey registers **must** be configured correctly for the key to operate properly:

MUX Register	Address 0xEF – 0xF0	Controls internal freq divider Must be set according to datasheet: 36x24 -> 0x07 / 0x00 32x16 -> 0x02 / 0x05
FREQ Register	Address 0xEF – 0xF0	LCD freq divider for supplied CLK freq Must be set according to table in datasheet.

Before use, each ScreenKey must be initialised according to the ScreenKey specification. The following ScreenKey registers **must** be configured correctly for the key to operate properly:

```
//*************************************************************
//*************************************************************
//
// FUNCTION NAME  : SetKeyInit
//
// INPUTS         : cKey - No of switch to init
//
// OUTPUTS        : None
//
// DESCRIPTION    : Initialises ScreenKey as per datasheet
//                  Key 0 is LC16, Key 1 is LC24
//
//*************************************************************
//*************************************************************
void SetKeyInit(unsigned char cKey)
{
   Byte(0x00,0,cKey) ;          //Start byte
   Byte(0xEF,1,cKey) ;          //MUX register address
   if ( cKey == 0 ){
      Byte(0x02, 1,cKey) ;      //32x16 MUX values
      Byte(0x05, 1,cKey) ;
   }
   else {
      Byte(0x07, 1,cKey) ;      //32x16 MUX values
      Byte(0x00, 1,cKey) ;
   }
   Byte(0xAA,0,cKey) ;          //End byte

   Byte(0x00,0,cKey) ;          //Start byte
   Byte(0xEE,1,cKey) ;          //FREQ register address
   Byte(01,1,cKey) ;            //Slow clock as software generated
   Byte(0xAA,0,cKey) ;          //End byte

   return;
}
```

Set ScreenKey Backlight Colour

The ScreenKey colour register is at address 0xED. This register controls the LED backlighting according to the following tables (from ScreenKey datasheets).

RG / LC Series Products:

		B7	B6	B5	B4	B3	B2	B1	B0
		Brightness Specification				*Colour Selection*			
LED		red left	red right	green left	green right	red left	red right	green left	green right
Function	0	dark	dark	dark	dark	off	off	off	off
	1	bright	bright	bright	bright	on	on	on	on

RGB Series Products:

		B7	B6	B5	B4	B3	B2	B1	B0
		Brightness Specification				*Colour Selection*			
LED		reserved	green	red	blue	reserved	green	red	blue
Function	0		dark	dark	dark		off	off	off
	1		bright	bright	bright		on	on	on

The ScreenKey datasheets list the colours that can be created for each ScreenKey type, e.g. 0x22 creates bright blue when written to an RGB colour register. Composed colours are created by mixing the base colours, e.g. red and green on the RG/LC series creates orange (0xFF = bright orange, 0x0F = dark orange).

```
//*************************************************************
//*************************************************************
//
// FUNCTION NAME  : SetKeyColor
//
// INPUTS         : cKey - Switch to set colour
//                  cCol - Colour to set
//
// OUTPUTS        : None
//
// DESCRIPTION    : Writes to ScreenKey COLOUR register
//
//*************************************************************
//*************************************************************
void SetKeyColor(unsigned char cKey, unsigned char cCol )
{
   Byte(START_BYTE,0,cKey) ;    //Start byte
   Byte(LC_COLREG,1,cKey) ;     //COLOUR register address
   Byte(cCol, 1,cKey) ;         //Write colour value
   Byte(END_BYTE,0,cKey) ;      //End byte
   return;
}
```

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.

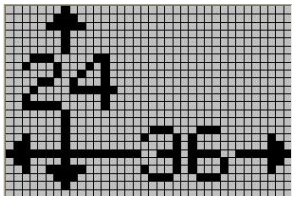
36x24 resolution ScreenKeys use 108 bytes to map the LCD screen: 24 rows with 4.5 bytes (32 + 4 bits) per row. The lsb of byte 0 corresponds to the top **right** hand pixel on the LCD. Starting with byte 4, every 9th byte is shared between rows (low nibble at end of previous row and high nibble on start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

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

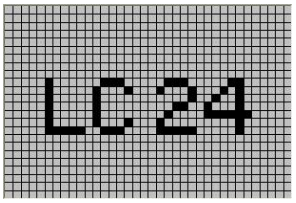
```
//*************************************************************
//*************************************************************
//
// FUNCTION NAME  : SetKeyPix16
//
// INPUTS         : cKey  - Switch to set pixels
//                  pcData - pointer to pixel data
//
// OUTPUTS        : None
//
// DESCRIPTION    : Writes to ScreenKey pixel area (begins at 0x80)
//                  32x16 ScreenKey requires 64 bytes to fully
//                  define all pixels
//
//*************************************************************
//*************************************************************
void SetKeyPix16(unsigned char cKey, unsigned char *pcData)
{
   unsigned char i;

   Byte(START_BYTE,0,cKey) ;            //Start byte
   Byte(LC_PIXREG,0,cKey) ;             //Pixel start address

   for(i = 0 ; i < MAX_SIZE_16 ; i++){
      Byte(pcData[i], 0, cKey) ;        //Write byte to next register address
   }

   Byte(END_BYTE,0,cKey) ;              //End byte

   return;
}
```

The following code examples demonstrate some pixel bitmaps for each ScreenKey resolution:

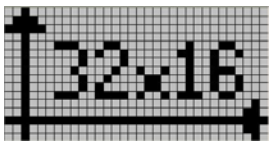
```
//Displays the dimensions of the key (36x24)
code char LC2436x24[108] = {
    0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x80, 0x03,
    0x00, 0x00, 0x00, 0x7C, 0x00, 0x00, 0x00, 0x00, 0x01,
    0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x00, 0x01,
    0x00, 0x00, 0x80, 0xC1, 0x01, 0x00, 0x00, 0x28, 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 };
```



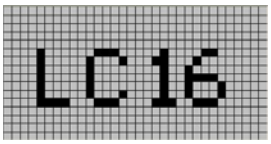

```
//Says "LC24"
code char LC24Text[108] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x0c, 0x0e, 0x0e, 0x04,
    0x40, 0x11, 0x11, 0x41, 0x00, 0x24, 0x01, 0x10, 0x04,
    0x40, 0x64, 0x00, 0x41, 0x00, 0x7e, 0x08, 0x10, 0x04,
    0x40, 0x00, 0x11, 0x41, 0x00, 0x04, 0x1f, 0xce, 0x07,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,};
```




```
//Displays dimensions of the key (32x16)
code char LC1632x16[64] = {
    0x04, 0x00, 0x00, 0x00,
    0x0e, 0x00, 0x00, 0x00,
    0x1f, 0x00, 0x00, 0x00,
    0x04, 0x00, 0x00, 0x00,
    0xc4, 0x31, 0x40, 0x0c,
    0x04, 0x4a, 0x60, 0x12,
    0x04, 0x42, 0x40, 0x02,
    0x04, 0x21, 0x49, 0x0e,
    0x04, 0x12, 0x46, 0x12,
    0x04, 0x0a, 0x46, 0x12,
    0xc4, 0x79, 0xe9, 0x0c,
    0x04, 0x00, 0x00, 0x40,
    0x04, 0x00, 0x00, 0x60,
    0xff, 0xff, 0xff, 0xff,
    0x04, 0x00, 0x00, 0x60,
    0x04, 0x00, 0x00, 0x40 };
```



```
//Says "LC16"
code char LC16Text[64] = {
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x10, 0x38, 0x08, 0x03,
    0x10, 0x44, 0x8c, 0x00,
    0x10, 0x04, 0x48, 0x00,
    0x10, 0x04, 0xc8, 0x03,
    0x10, 0x04, 0x48, 0x04,
    0x10, 0x44, 0x48, 0x04,
    0xf0, 0x38, 0x9c, 0x03,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00,};
```



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.

The approach implemented in this example simply checks the return line from each switch (P1.3 for TA01 and P1.4 for TA04).

References & Downloads

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

[1] LC16, LC24, RGB16 and RGB24 Datasheets

[2] Simple Microcontroller Interface schematic

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

[4] 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
Unit 11 Keypoint Business Park
42 Rosemount Park Drive
Ballycoolin Road
Dublin 11
Ireland

Phone: +353 1 8855 075
Fax: +353 1 8855095

Email:
info@ScreenKeys.com

www.ScreenKeys.com