

LM1728

Multi Tonal Music Keyboard

Abstract

This design project addresses the complex problem of creating a flexible multi tonal music keyboard interface with a Luminary Micro Stellaris LM3S811 microcontroller. The design uses low cost parts to implement a keyboard that can be programmed to function in 12, 19, 24, and 31 tone music scales. An introduction is provided to multi tonal musical scales, to keyboard circuit interfaces, to digital sound synthesis, to microcontroller programming, to MIDI interfaces, and to microtonal music experiments. A complete three octave prototype instrument, with up to 31 tones per octave, is designed and constructed using the microcontroller. An printed circuit board is designed for interfacing the keyboard to the microcontroller. Micro tonal music is performed on the keyboard for comparison experiments with traditional 12 tone western music. An output control connection is provided for the popular MIDI electronic music instrument interface. The goal is a low cost, easy to construct, experimental music keyboard for use by musicians, theorists, and engineers to investigate alternative music scales and keyboard layouts beyond the traditional western music scale defined with twelve tones per octave using the familiar seven white keys and five black keys found on modern electronic music synthesizers and organs for the last 400 years.

Instrument Design

This multi tonal instrument is constructed from three low cost mini keyboards sold as children's toy synthesizers. The mini keys are combined in various ways to support 12, 19, 24, and 31 tone scales. By using mini keyboards with $\frac{3}{4}$ inch wide white keys and $\frac{1}{2}$ inch wide black keys, a three octave instrument is created that fits within a 19 inch wide, 12 inch deep, and 6 inch high space providing 95 reachable key surfaces. These mini keyboards are available for under \$10 each when on sale. A variety of keyboards were examined including some with full size keys. However, the large keys make the fingering reach difficult for an octave in the 31 tone scale.



Low Cost Mini Keyboard Tree Octave Instrument Front and Rear Inside



Keyboards Mounted on Wood Stands: Top 21 White Keys Bottom 37 Keys

The three keyboards were removed from their plastic cases and mountings. They were then mounted onto wooden strips for prototype layout evaluation for key location and reach in three dimensions. The final design choice was a 37 key on top, a 21 key in the middle overlapped by the top board, and a 37 key on the bottom. This provides a flexible arrangement that can be programmed for 12, 19, 24, and 31 tone scales that are very playable by a performer.



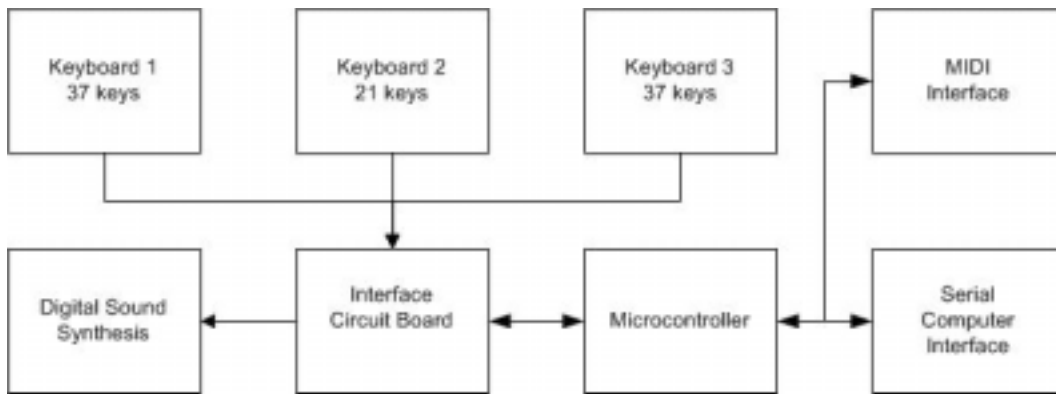
Three Tier Keyboard Prototype Front and Side View with 95 Keys



Possible Instrument Configurations: A(37, 37, 21), B(37, 21, 37), and C(21, 37, 37)

Keyboard Interface

The interface of the keyboards to the Luminary Micro Stellaris LM3S811 microcontroller was helped by the inclusion of a circuit diagram in the owner's manual from the Chinese keyboard manufacturer. I don't know of any American company that includes a circuit diagram in low end consumer products. The following is an overall interface design for the instrument. Three keyboards are connected by a custom interface circuit board. The microcontroller controls the keyboards, the sound synthesis, the computer serial communication, and the MIDI output.

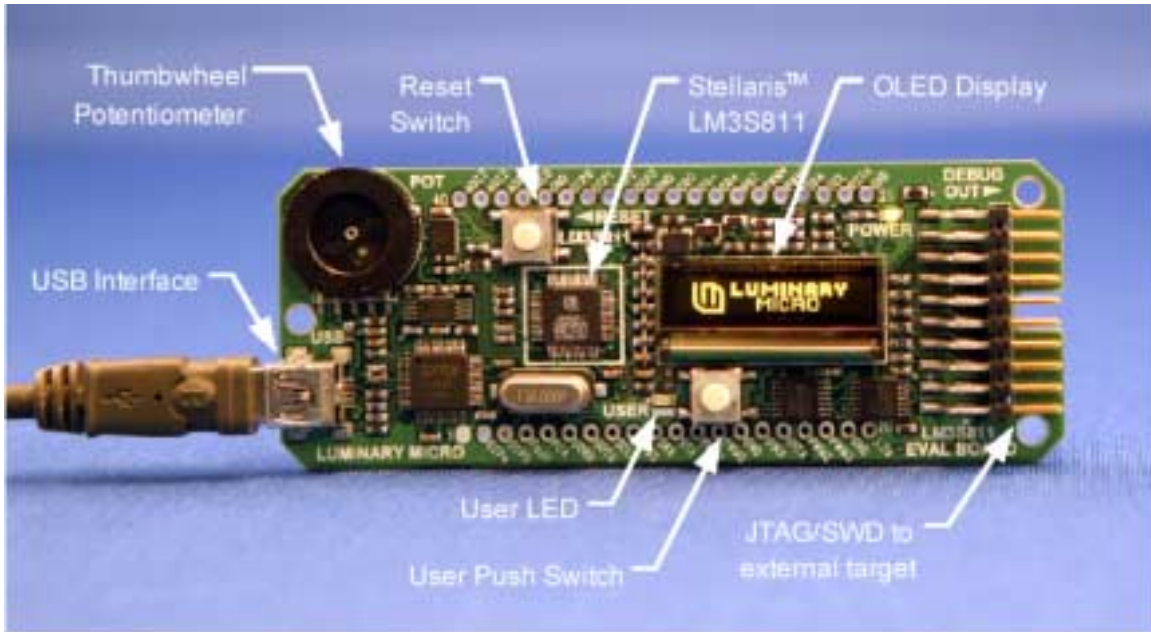


Keyboard Component Interfaces

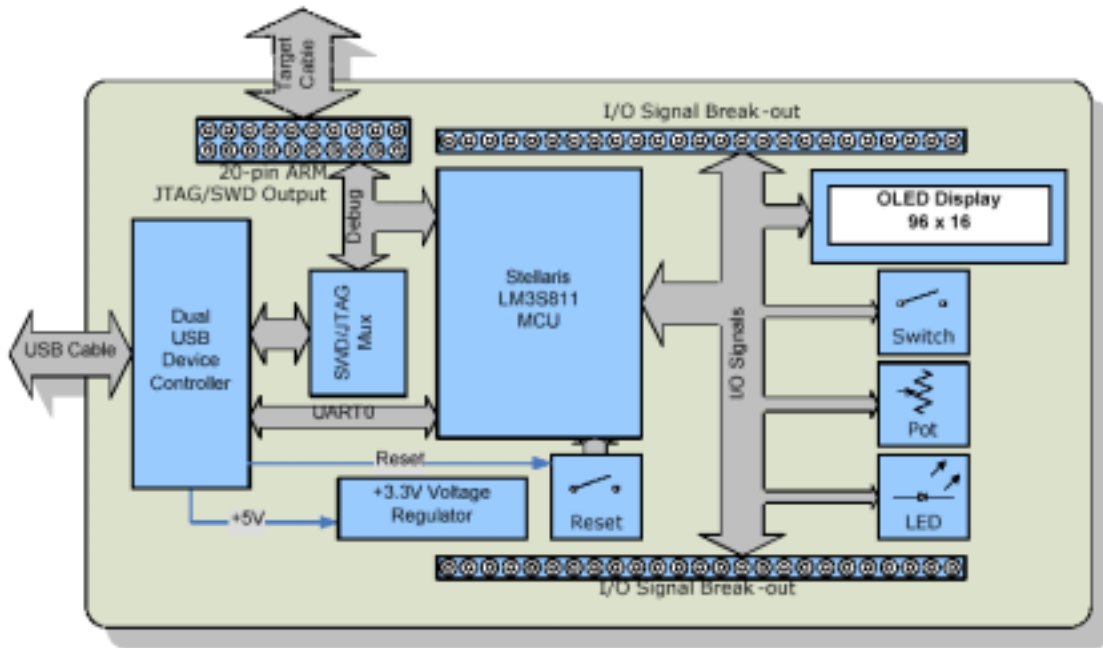
The parallel music keyboards are read continuously at high speed by using a dual parallel port interface array and the 50 MHz clock of the LM3S811. The three key boards have a total of 111 switches to be read but at most 95 are used in the various programmable scales. Each single keyboard is embedded in an array of 6 by 8 lines shown below. By cycling through 6 binary values on the top right lines and reading the eight outputs from the bottom right lines in parallel, the state of each of the 37 key switches can be determined as either open or closed. For anyone who is counting, a maximum of 144 keys can be interfaced with this implementation. For three keyboards, a total of 18 write and read processes can access the state of all of the keys.

Microcontroller Electronics

The Luminary Micro Stellaris LM3S811 microcontroller interfaces to the keyboards, the PC, a MIDI connector, the controller LCD, and the digital sound synthesizer. This interface is implemented on a PCB that connects each peripheral to the microcontroller bus. The following is the evaluation board layout for the processor supplied from the manufacturer.



The functions of the evaluation board are shown in the following block diagram. The I/O signal break out is used to interface to the keyboard inputs, sound output, and MIDI interface.



The I/O breakout pins are listed in the following diagram. Only the available unused pins were selected for the instrument interface to avoid conflicts. No JTAG pins are used.

Pad No.	Description	Pad No.	Description
1	BLANK	40	ADC3
2	PC7/CCP4	39	ADC2
3	PB5/CCP5	38	ADC1
4	PD6/Fault	37	ADC0 ^a
5	PC4 ^a	36	GND
6	PA0/U0Rx ^a	35	PD4/CCP0
7	PA1/U0Tx ^a	34	PC5/CCP1
8	PA2/SSIClk	33	PD5/CCP2
9	PA3/SSIFss	32	PC6/CCP3
10	PA4/SSIRx	31	GND
11	PA4/SSITx	30	PD7/C0O
12	PD1/PWM1	29	PB4/C0-
13	PD0/PWM0	28	PB6/C0+
14	GND	27	PB7 ^b
15	PD2/U1Rx	26	PE0/PWM4
16	PD3/U1Tx	25	PE1/PWM5
17	PB0/PWM2	24	PB3/I2CSDA ^a
18	PB1/PWM3	23	PB2/I2CSCL ^a
19	GND	22	$\overline{\text{RESET}}$
20	+3.3V	21	GND

a. Indicates an I/O line that is used by EVB hardware.

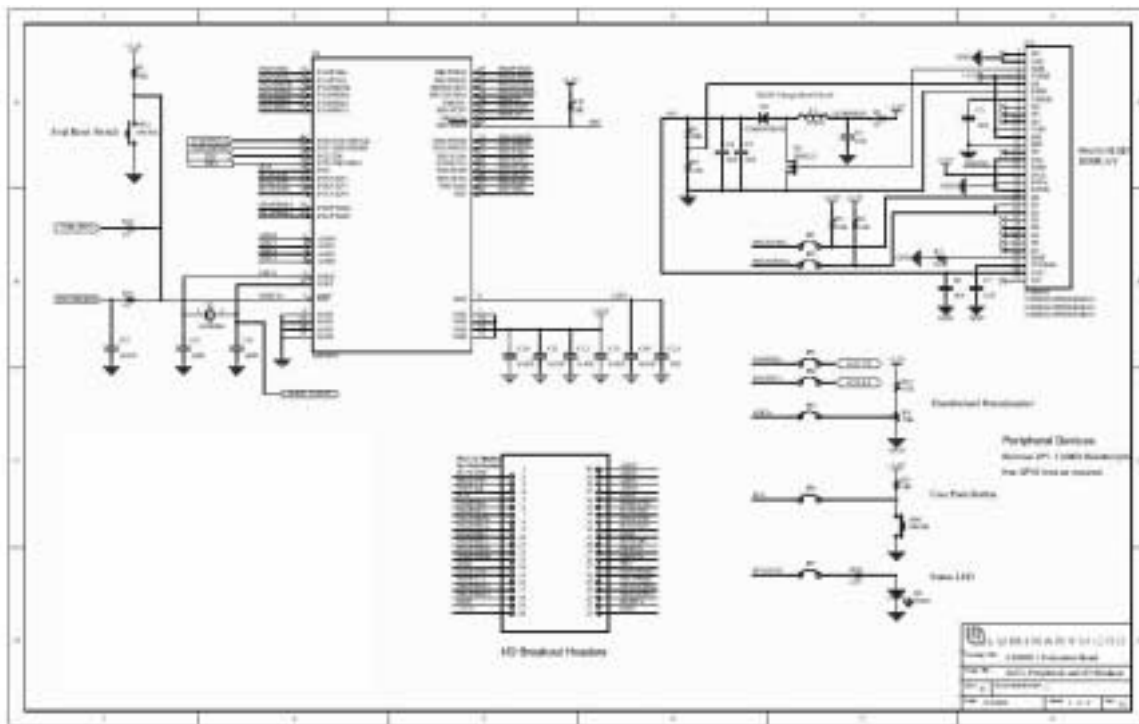
b. PB7 should not be used as a GPIO.

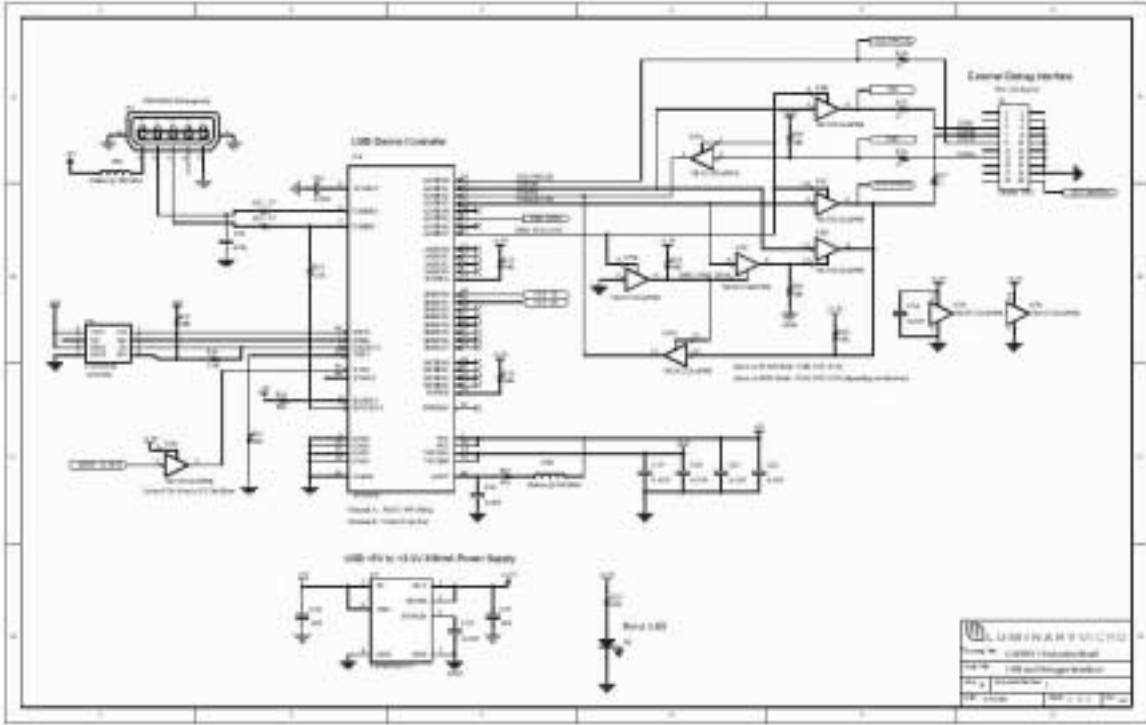
Schematics are provided in the LM3S811 documentation along with electrical specifications for designing the interface board. The on board hardware has several components that use seven of the MCU pins as given in the following table. The I2C interface to the OLED display is implemented with the first two pins. This digital display is 96 by 16 bits and is a perfect match for the 96 keys on the instrument to dynamically display key notation and scale selection information. The next two pins implement the virtual serial port through the USB interface to a PC COM port. The fifth pin is for an analog wheel input useful as a music controller. The sixth

pin is a user pushbutton input to select instrument options. The seventh pin is an LED to indicate instrument states.

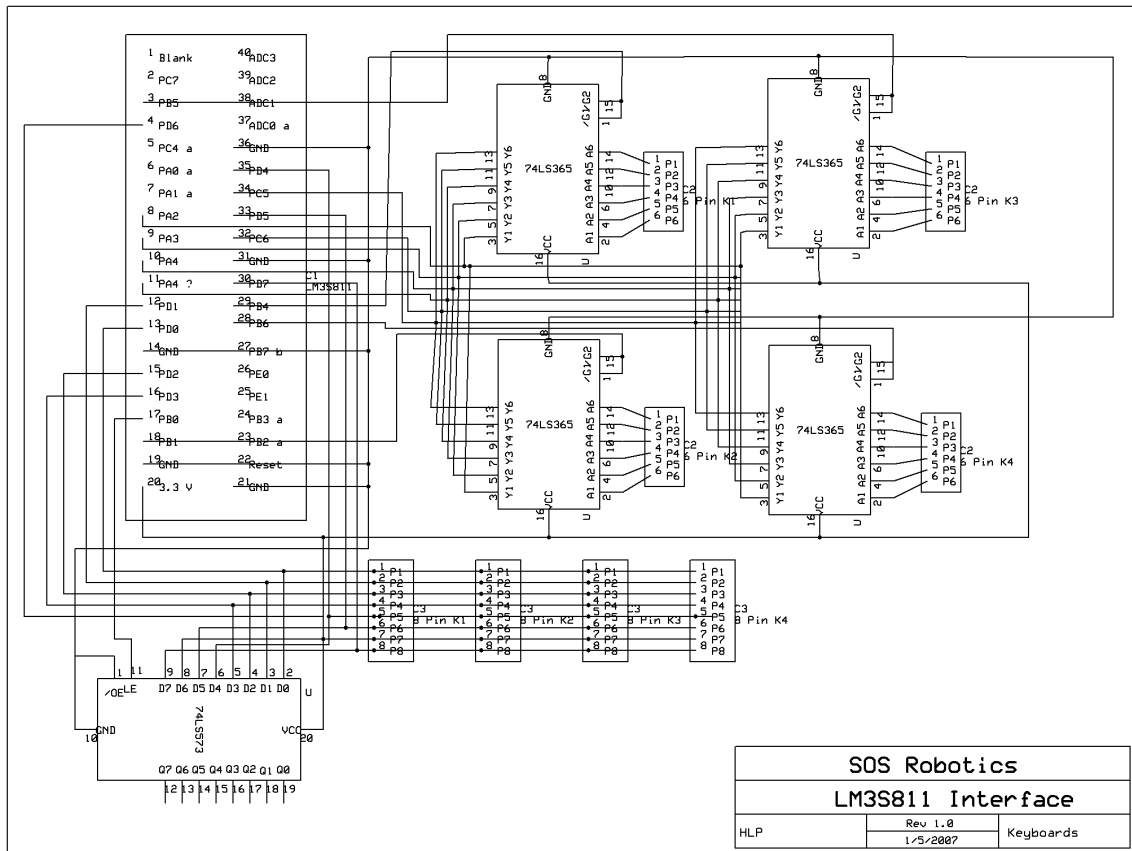
MCU Pin	EVB Function
Pin 33 PB2/I2CSCL	I2C SCL to Display
Pin 34 PB3/I2CSDA	I2C SDA to Display
Pin 17 PA0/U0Rx	VCP Receive
Pin 18 PA1/U0Tx	VCP Transmit
Pin 1 ADC0	ADC Input from Thumbwheel Potentiometer
Pin 30 GPIO PB7	User Push Switch Input
Pin 29 GPIO PC4	User LED output

The remaining pins are available for the instrument interface and are allocated as shown in the following schematic for the interface PCB. The following two schematics are for the LM3S811 evaluation kit used in the instrument.



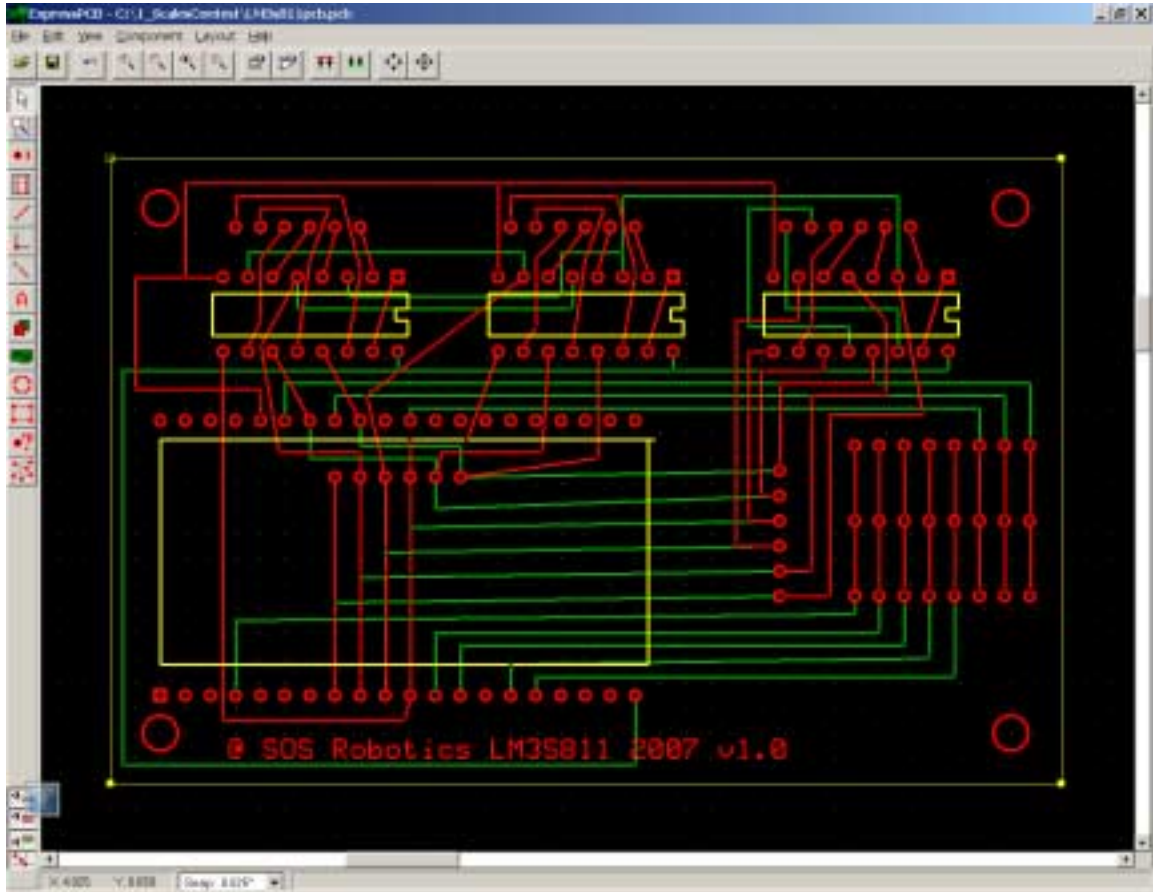


The following schematic is for the keyboard and processor interfaces to the LM3S811.

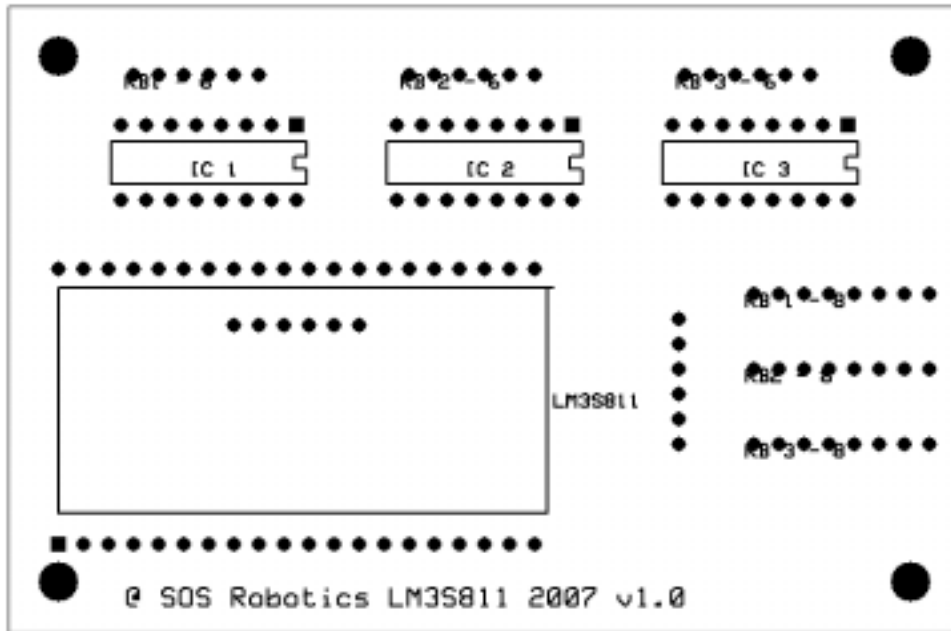


The LM3S811 evaluation kit PCB outputs to the eight by six key switch array of each board with eight common outputs on the eight pins PD0 to PD7. The six inputs are tri state buffered to select the inputs from each keyboard individually using pins PB4, PB5, PB6, and PB1 to input signals into the six pins PA2, PA3, PA4, PA5, PC6, and PC5. Four keyboard interfaces are implemented to allow for future expansion and to support other key inputs. The digital output sound signal is latched by PB0 from pins PD0 to PD7 for input into a digital to analog converter.

The following is the PCB layout for the components and wiring. It is a two sided board 3.8 inches wide and 2.5 inches high created with the ExpressPCB free schematic capture and PCB layout software.



The following is the silkscreen for the component placement on the PCB.



The finished circuit board carries the LM3S811 evaluation kit microcontroller installed in an extra wide 40 pin DIP.

Conclusion

This design project created a low cost, easy to construct, experimental multi tone music keyboard for use by musicians, theorists, and engineers. The goal is a tool to investigate alternative music scales and to define a keyboard layout that functions beyond the traditional 12 tone layout. The design modifies three standard keyboards that can be programmed to operate in 12, 19, 31, and several other scales. It is a complete three octave prototype instrument that can be performed in real time without any other equipment. The heart of the device is a LM3S811 microcontroller that provides the computational capability for configuring the various scales and controlling the digital keyboards. The instrument has interfaces for a serial data port and for MIDI components.

Appendix – C Program for LM3S811

```

//*****
//
// Modified Example program uart_out.c by SOS, Inc. All Rights Reserved, 2007
// Created to connect LM3S811 to a high speed PC virtual serial port
// Program reads music keyboard and sends keys to PC for processing
//

```

```

//          Set up Clock
//          Init OLCD Display
//          Setup UART
//          SETup GPIO
//          Init Interrupts
//          Set GPIO A0 ans A1
//          Configure UART
//          Enable UART
//          Init GPIO for keyboard
//          Read keys and send to PC
//*****

#include "hw_ints.h"
#include "hw_memmap.h"
#include "hw_types.h"
#include "debug.h"
#include "gpio.h"
#include "interrupt.h"
#include "sysctl.h"
#include "uart.h"
#include "../osram96x16.h"
#include <stdio.h>
#include <string.h>

//*****
//! The example application utilizes the UART to display text. The first UART
//! (connected to the FTDI virtual serial port on the Stellaris LM3S811
//! Evaluation Board) will be configured in 115,200 baud, 8-n-1 mode and
//! continuously display text.
//*****

//*****//
// The string that is to be written to the UART.
//*****
//static const unsigned char g_pucString[] ;

//*****
// Key states, select masks, and port selects
//*****
static unsigned int key_state[144] ;
static unsigned int key_value[144] ;
static unsigned int mask1[6] ;
static unsigned int mask2[6] ;
static unsigned int portx[3] ;

//*****
// The variables that track the data to be transmitted.
//*****
static unsigned char *g_pucBuffer = 0;
static unsigned long g_ulCount = 0;

//*****
// The error routine that is called if the driver library encounters an error.
//*****
#ifdef DEBUG
void

```

```

__error__(char *pcFilename, unsigned long ulLine)
{
}
#endif

//*****//
// The transmit interrupt handler for the UART.
//*****//
void
UARTTxIntHandler(void)
{
    //
    // Loop while there are more characters to send and space available in the
    // hardware FIFO.
    //
    while(g_ulCount && UARTSpaceAvail(UART0_BASE))
    {
        //
        // Send the next character.
        //
        UARTCharNonBlockingPut(UART0_BASE, *g_pucBuffer++);

        //
        // Decrement the count of characters to send.
        //
        g_ulCount--;
    }
}

//*****//
// The UART interrupt handler.
//*****//
void
UARTIntHandler(void)
{
    unsigned long ulStatus;

    //
    // Get the interrupt status.
    //
    ulStatus = UARTIntStatus(UART0_BASE, true);

    //
    // Clear the asserted interrupts.
    //
    UARTIntClear(UART0_BASE, ulStatus);

    //
    // See if the transmit interrupt was asserted.
    //
    if(ulStatus & UART_INT_TX)
    {
        //
        // Handle the transmit interrupt.
        //
        UARTTxIntHandler();
    }
}

```

```

    }
}

//*****
// Send a string to the UART.
//*****
void
UARTSend(const unsigned char *pucBuffer, unsigned long ulCount)
{
    //
    // Wait until any previous string has been sent.
    //
    while(g_ulCount)
    {
        //
        // Save the data buffer to be transmitted.
        //
        // g_pucBuffer = pucBuffer;
        g_ulCount = ulCount;

        //
        // Prime the UART FIFO. This is done with interrupts disabled to avoid
        // problems if a transmit interrupt occurs mid way through the priming.
        //
        UARTIntDisable(UART0_BASE, UART_INT_TX);
        UARTTxIntHandler();
        UARTIntEnable(UART0_BASE, UART_INT_TX);
    }

//*****
// This example demonstrates how to send a string of data to the UART.
//*****

//*****

void
init_keys(void)
{
    int i ;

    // init keys for gpio

    // D port as inputs
    GPIODirModeSet(GPIO_PORTD_BASE, (GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 |
    GPIO_PIN_3
    | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7), GPIO_DIR_MODE_IN);

    // C port as outputs
    GPIODirModeSet(GPIO_PORTC_BASE, (GPIO_PIN_5 | GPIO_PIN_6 ),
    GPIO_DIR_MODE_OUT);

    // B port as outputs
    GPIODirModeSet(GPIO_PORTB_BASE, ( GPIO_PIN_1 | GPIO_PIN_4 | GPIO_PIN_5 |
    GPIO_PIN_6), GPIO_DIR_MODE_OUT);
}

```

```

// A port as outputs
GPIODirModeSet(GPIO_PORTA_BASE, ( GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 |
GPIO_PIN_5), GPIO_DIR_MODE_OUT);

// key states
for(i=1;i<144;i++) { key_state[i] = 0 ; key_value[i] = i ; }

//masks and ports
mask1[0] = 0x04 ;
mask1[1] = 0x08 ;
mask1[2] = 0x10 ;
mask1[3] = 0x20 ;
mask1[4] = 0x00 ;
mask1[5] = 0x00 ;

mask2[0] = 0x00 ;
mask2[1] = 0x00 ;
mask2[2] = 0x00 ;
mask2[3] = 0x00 ;
mask2[4] = 0x20 ;
mask2[5] = 0x40 ;

portx[0] = 0x10 ;
portx[1] = 0x20 ;
portx[2] = 0x40 ;

}

//*****

void
read_keys(void)
{
int i, j, p, m ;
int keyin ;
int keynum ;
//char str[10] ;

keynum = 0 ;

//read 144 key array values 3 * 6 * 8
for(p=0;p<3;p++)
{
//select ports with PB4, PB5, PB6
GPIOPinWrite(GPIO_PORTB_BASE, ( GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6), portx[p]);

for(m=0;m<6;m++)
{

//write data signal to switches PA2, PA3, PA4, PA5, PC6, PC5
GPIOPinWrite(GPIO_PORTA_BASE, ( GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 |
GPIO_PIN_5), mask1[m]);
GPIOPinWrite(GPIO_PORTC_BASE, ( GPIO_PIN_5 | GPIO_PIN_6 ), mask2[m]);

```

```

//read key switches
keyin = GPIOPinRead(GPIO_PORTD_BASE, (GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 |
GPIO_PIN_3
| GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7) );

//update key states
for(i=0;i<8;i++)
{
j = keyin >> i ;
key_state[keynum] = 0 ;
if( j > 0 ) key_state[keynum] = 1 ;
keynum++ ;
}

}

}

// send key states to UART
for(i=0;i<144;i++)
{
if( key_state[i] > 0 )
{
// sprintf(str, "%d ",key_value) ;
// strcat(g_pucString,str) ;
printf("%d %d \n",i,key_value[i] ) ;
}
}

//UARTSend(g_pucString, sizeof(g_pucString) - 1);

}

//*****
int
main(void)
{
//
// Set the clocking to run directly from the crystal.
//
SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
SYSCTL_XTAL_6MHZ);

//
// Initialize the OLED display and write status.
//
OSRAMInit(false);
OSRAMStringDraw("UART out on SER0", 0, 0);
OSRAMStringDraw("115,200, 8-N-1", 6, 1);

//
// Enable the peripherals used by this example.
//
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

```

```

//
// Enable processor interrupts.
//
IntMasterEnable();

//
// Set GPIO A0 and A1 as UART pins.
//
GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

//
// Configure the UART for 115,200, 8-N-1 operation.
//
UARTConfigSet(UART0_BASE, 115200, (UART_CONFIG_WLEN_8 |
                                   UART_CONFIG_STOP_ONE |
                                   UART_CONFIG_PAR_NONE));

//
// Enable the UART interrupt.
//
IntEnable(INT_UART0);
UARTIntEnable(UART0_BASE, UART_INT_TX);

    //
    // init gpio pins
    //
    init_keys();

//
// Loop reading keys and sending out the UART.
//
while(1)
{
    //
    // Send the string to the UART.
    //
    // UARTSend(g_pucString, sizeof(g_pucString) - 1);
    read_keys();
}
}

```