Notes on miniDSP 8x8 v1.4

ANALOGUE AUDIO INPUTS/OUTPUTS	SigmaStudio ADC input	SigmaStudio DAC output
1	6	16
2	7	17
3	2	0
4	3	1
5	4	2
6	5	3
7	0	4
8	1	5

Multipurpose I/O:

MP0/ADC0	100 series resistor	J13/2	J13/1 = 3.3V	J13/3 = Gnd
MP1/ADC1	100 series resistor	J14/2	J14/1 = 3.3V	J14/3 = Gnd
MP2/ADC2	100 series resistor	J15/2	J15/1 = 3.3V	J15/3 = Gnd
MP3/ADC3	100 series resistor	J16/2	J16/1 = 3.3V	J16/3 = Gnd
MP4	10K pull-up resistor to 3.3V	J17/1	J17/2 = 3.3V	
MP5	10K pull-up resistor to 3.3V	J17/3	J17/4 = 3.3V	
MP6	10K pull-up resistor to 3.3V	J17/5	J17/6 = 3.3V	
MP7	10K pull-up resistor to 3.3V	J17/7	J17/8 = 3.3V	
MP8	10K pull-up resistor to 3.3V	J17/9	J17/10 = 3.3V	
MP9	10K pull-up resistor to 3.3V	J17/11	J17/12 = 3.3V	
MP10	10K pull-up resistor to 3.3V	J17/13	J17/14= 3.3V	
MP11	10K pull-up resistor to 3.3V	J17/15	J17/16 = 3.3V	

To hold on-board PIC microcontroller in reset mode connect pins 1 (/MCLR) and 3 (Gnd) of J18 together

5V pick-up point on J19 pin 1.

12v trigger on J25 pin 1. Remove link and supply trigger in or a switch as shown in user manual.

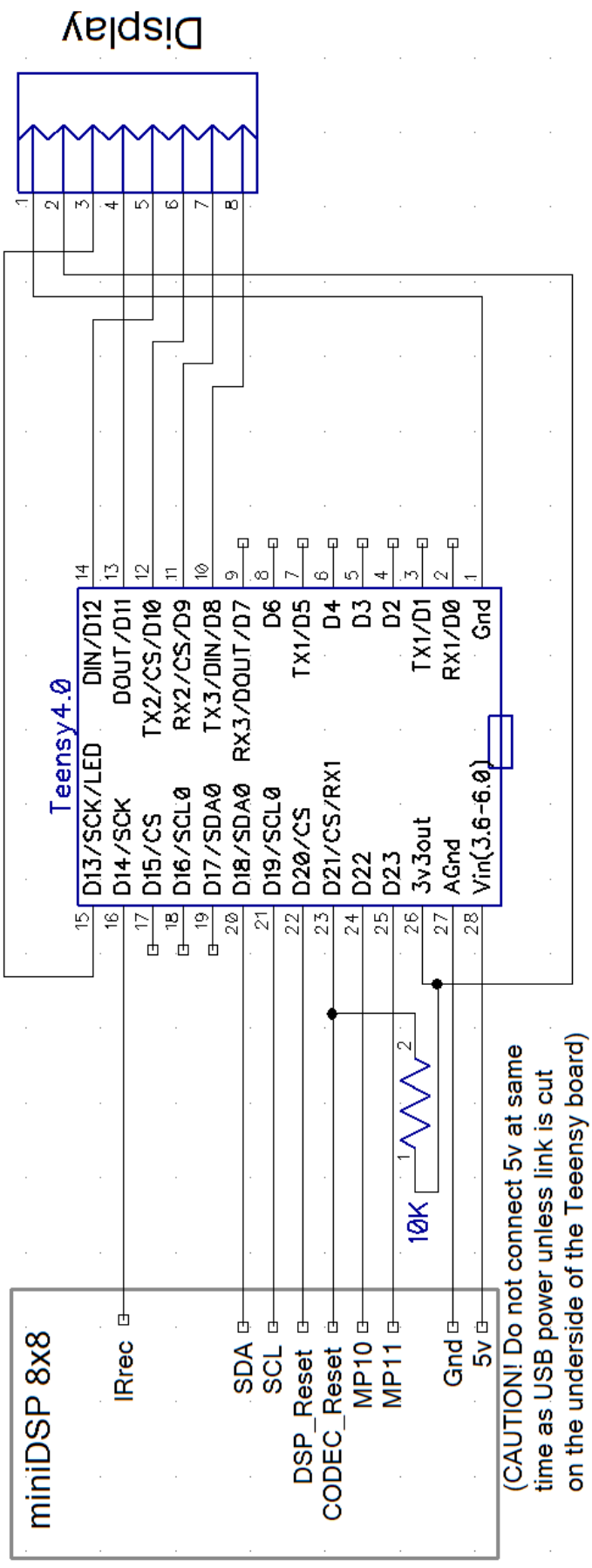
Connections for external programming and setup control:

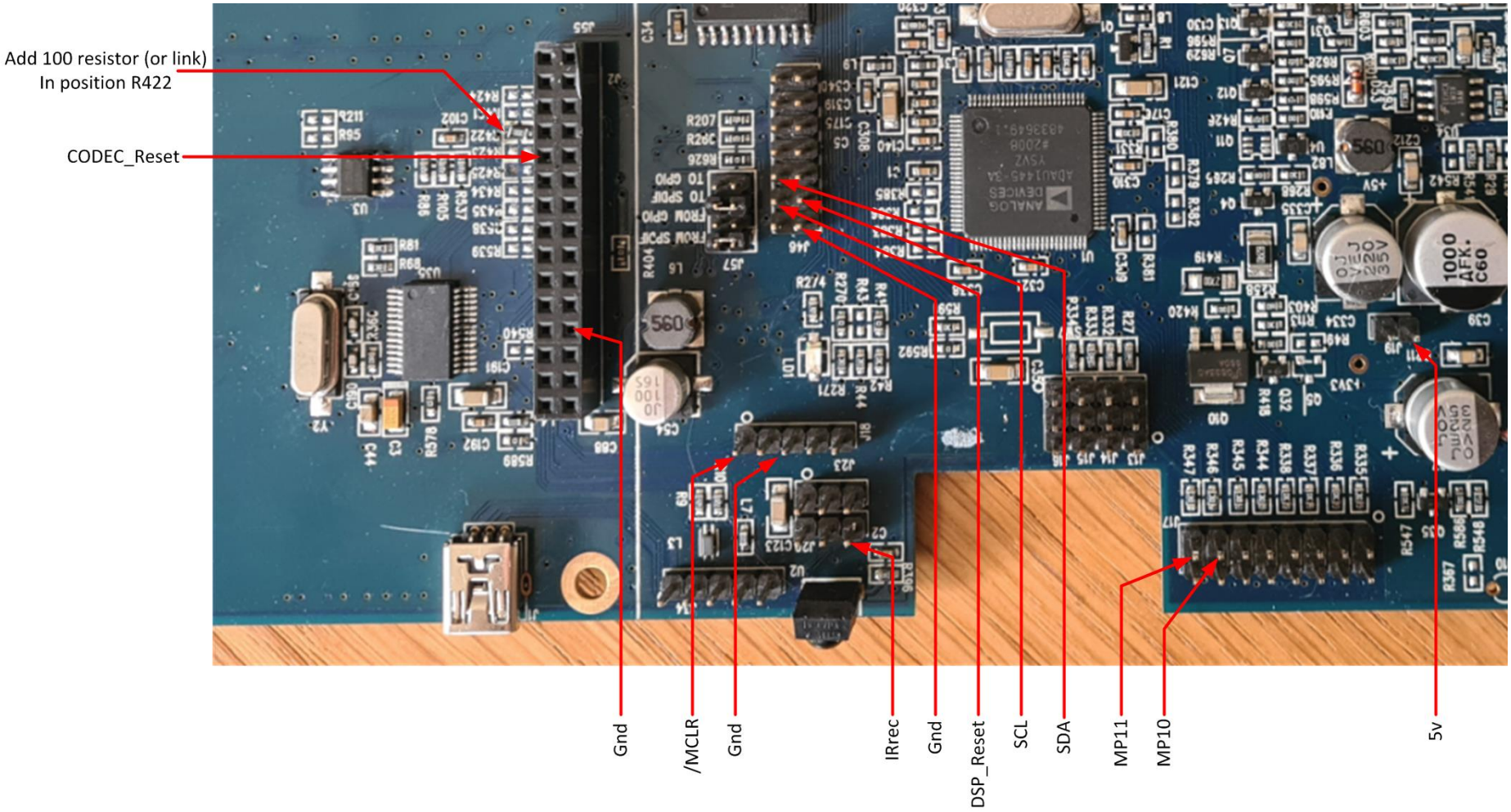
I2C Clock	J46/12	
I2C Data	J46/9	
DSP /reset	J46/11	
Gnd	J46/14	
CODEC /reset	J2/7*	External 10k pullup to 3.3V required

*Access to the CODEC /reset line is required. Easiest way to do this is to solder a 100 ohm 0805 smt resistor into the empty R422 position which will then allow access to the reset line via J2 pin 7. (Note that J55 is just a two pin header on the end of J2 – looks like a later extension to J2 was required)

Output from the I/R receiver can be picked up on J20 pin 3

Connections for Teensy and LCD Display




Add 100 resistor (or link)
In position R422
CODEC_Reset
Gnd
/MCLR
Gnd
IRrec
Gnd
DSP_Reset
SCL
SDA
MP11
MP10
5v

Software

Required libraries – see sketch file

Uses example project from Analog Devices article: **Interfacing SigmaDSP Processors with a Microcontroller**
https://wiki.analog.com/resources/tools-software/sigmastudio/tutorials/microcontroller?s[]=teensy
Note the following:
The USER_HEADINGS.h file implies support for the ADAU1445
.
.
#define DSP_TYPE_SIGMA200 2 // Sigma200: ADAU176x/ADAU178x/ADAU144x
.
.
but this is not specifically supported in the SigmaStudioFW.h and requires a value change from 5 to 6 in line 87:

.
.
```
#elif DSP_TYPE == DSP_TYPE_SIGMA200
    // Based on ADAU1761
    if (address < 0x0800) {
        return 4;    // Parameter RAM is 4 bytes deep
    } else {
        return 6;    //FV set to 5 in original file for ADAU1761
    }
```
.
.

On line 28 of the USER_SETTINGS.h file set dsp type to SIGMA200:
.
.
```
// Which type of DSP is connected?
#define DSP_TYPE DSP_TYPE_SIGMA200
```

On line 65, comment out #define for DSP_RESET_PIN as now defined in the sketch file

main.cpp is not used in this example project as the relevant functions are now in the sketch file

In the sketch file the two #include lines:

#include <FV1445_miniDSP_8x8_v1_IC_1.h>

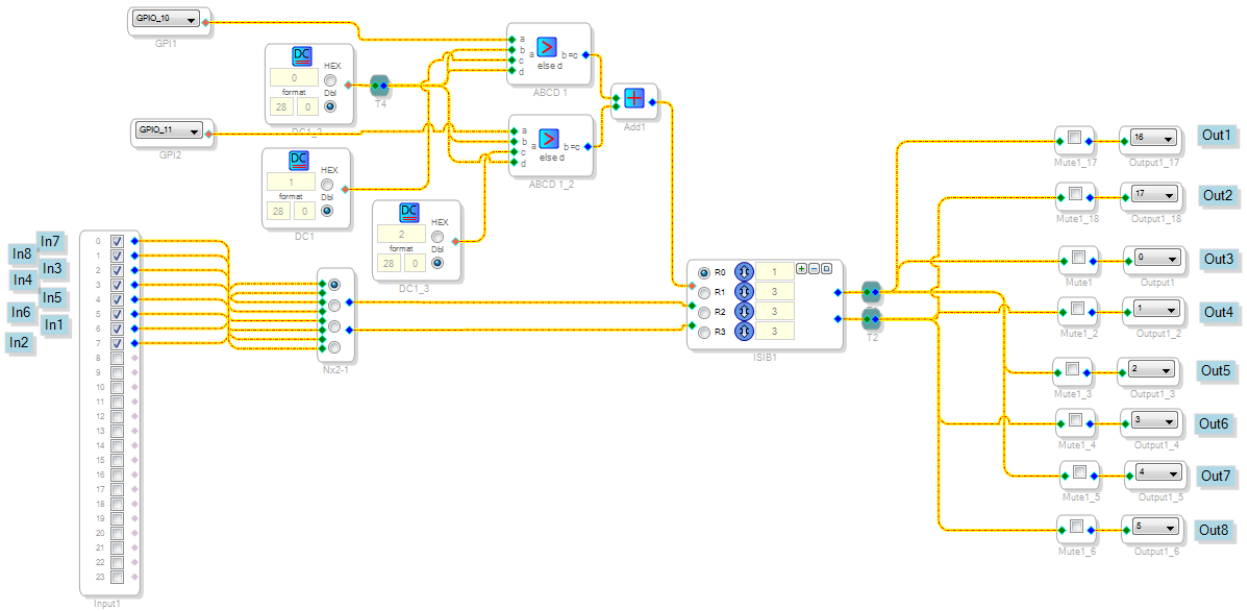
#include <FV1445_miniDSP_8x8_v1_IC_1_PARAM.h>

need to be changed to your SigmaStudio project name. SigmaStudio will add the _IC_1.h and _IC_1_PARAMS.h to your project name.

I created a SigmaStudioFiles folder in the Arduino libraries folder for the above files to be saved in. From the example SigmaStudio project, copy the SigmaStudioFW.h and USER_SETTINGS.h files into this library folder and make the above file edits.

When creating a file set in SigmaStudio for a project (Action|Export System Files), point the saving folder to this library folder.

The example project is a simple one with an input switch to select one of four input pairs. The stereo signal is routed through a tilt filter and then sent out to all 8 outputs in stereo pairs:



This demonstrates two interface types to the DSP with the Teensy. Firstly a two wire hardware link via the DSP MP10 and MP11 pins to select tilt and secondly an input switching function driven via I2C interface. This latter function is a bit complicated because it is first required to know which address to write to for where the relevant cell resides and this will change with a redesign of the SigmaStudio project and will then need a matching update in the Teensy sketch. The way I have done it so far is to take the cell name from the SigmaStudio schematic, in this case Nx2-1 and look it up in the *_IC_1_PARAM.h file where I found the address defined:

```
/* Module Nx2-1 - Stereo Switch Nx2*/
#define MOD_NX2_1_COUNT                    1
#define MOD_NX2_1_DEVICE                   "IC1"
#define MOD_NX2_1_STEREOSWSLEW_ADDR        15
#define MOD_NX2_1_STEREOSWSLEW_FIXPT       0x00000000
#define MOD_NX2_1_STEREOSWSLEW_VALUE       SIGMASTUDIOTYPE_INTEGER_CONVERT(0)
#define MOD_NX2_1_STEREOSWSLEW_TYPE        SIGMASTUDIOTYPE_INTEGER
```

In my sketch I defined a constant to pick up this address at compile time:

#define INPUT_SWITCH_CELL MOD_NX2_1_STEREOSWSLEW_ADDR

Providing the switch is not deleted or renamed, other changes to the schematic should not change the entry, just the cell address.

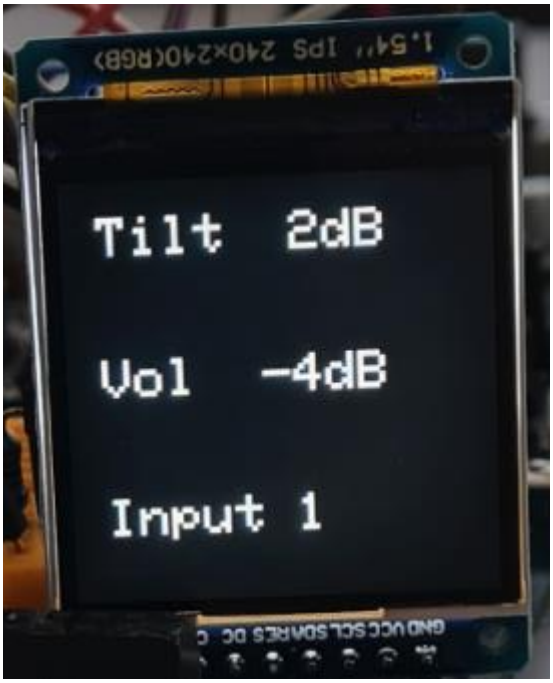
The third example feature in the test project is a write to the DAC for volume. Could have added a volume cell to the SIgmaStudio project instead, but thought this would make for a better demo of using the board.

Note the following for the remote control:

The project was created using 10 buttons on a remote I have here which uses NEC code. You will need to edit the sketch to match whatever 10 commands you wish to use with the test project. The sketch uses the IRremote library and codes for NEC, SONY, RC5, RC6 should be supported. In the function pollIR comment in/out for the relevant remote protocol type. Note that NEC uses a special repeat, some others just repeat the full code so not sure if my sketch will react to volume up/down continuous press.

To find the remote codes for your transmitter, get the project up and running in the Teensy, open the IDE serial window and you should see a code printed out for each button press. Edit the sketch #defines for the 10 buttons.

The display used in the example project is a 1.54” RGB 240x240 module using the ST7789 driver. For reasonable update of values with minimal flicker, I just wrote a background box over the old values and then wrote the new values. The area to do the blanking was determined on a try it and see (with a red box to test) basis, so not really the way to go for a proper project but ok for this demo.



The SigmaStudio project is attached to the forum post alongside this file as FV1445_miniDSP_8x8_v1.zip and contains the SigmaStudio project file and the Arduino code.

Start a new project in the Arduino IDE and copy the code from the file Arduino_code.txt into the sketch. Change the project name in the two include lines and redefine the remote codes as mentioned above.