Zedboard Audio

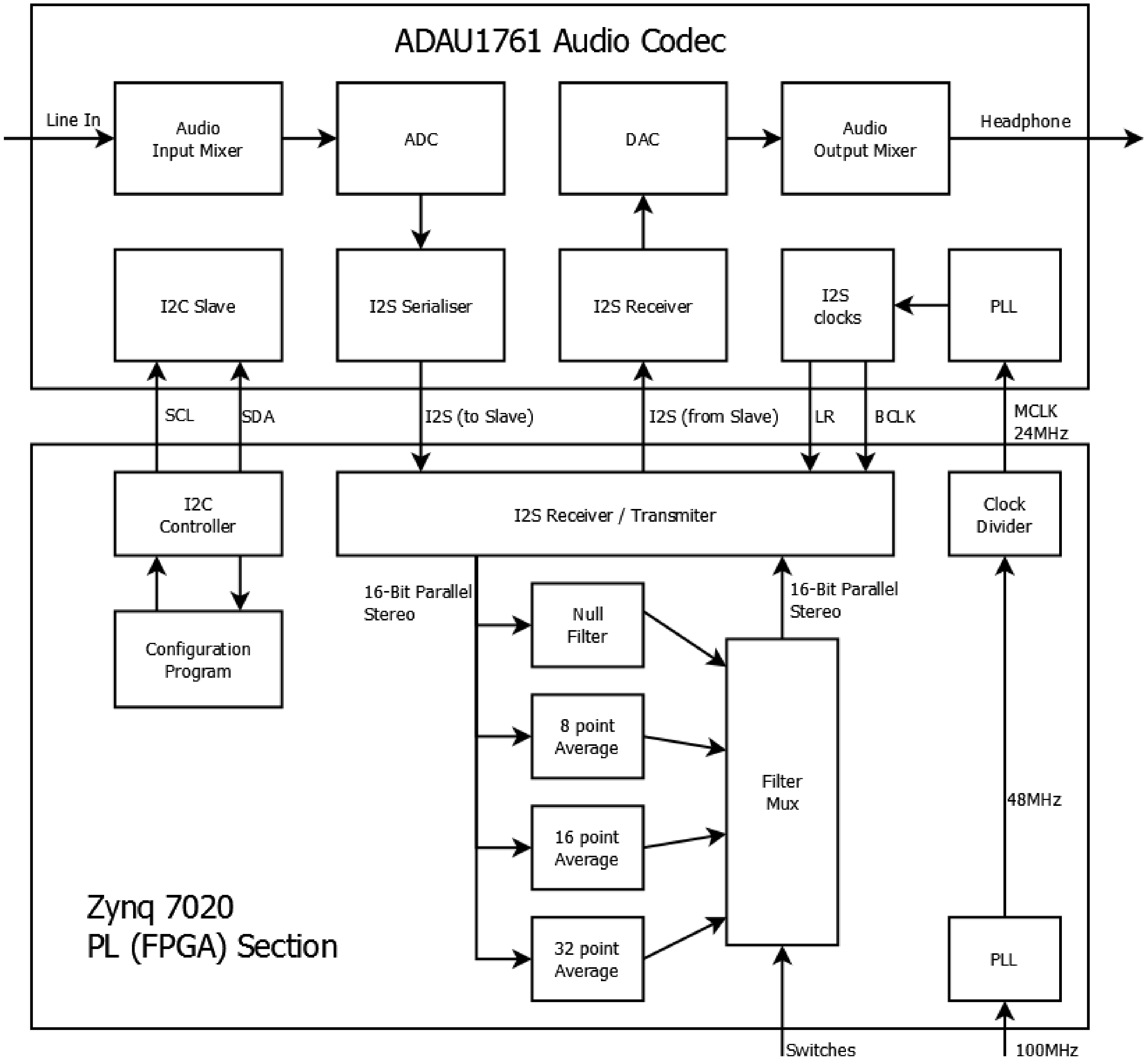
From Hamsterworks Wiki!

This FPGA Project was completed during Jan 2014.

I've had lots of emails about how to use the Audio side of the ZedBoard without using the full Linux stack, so I though I'ld give it a crack.

Contents

The design



Connections to the codec

The Zedboard has an ADAU1761 chip on it, with the following connections to the FPGA:

Pin Name	Use	Direction (from FPGA's frame of reference)	Zynq Pin	ADAU1761 Pin
AC-ADR0	I2C Address Bit 0/SPI Latch Signal	Input	AB1	3
AC-ADR1	I2C Address Bit 1/SPI Data	Input	Y5	30
AC-MCLK	Master Clock	Input	AB2	2
AC-GPIO2	Digital Audio Bit Clock	Input (CODEC as master) / Output (FPGA as master)	AA6	28
AC-GPIO3	Digital Audio Left-Right Clock	Input (CODEC as master) / Output (FPGA as master)	Y6	29
AC-GPIO0	Digital Audio Serial-Data DAC	Input	Y8	27
AC-GPIO1	Digital Audio Serial Data ADC	Output	AA7	26

AC-SDA	I2C Serial Data interface	Bidirtectional	AB5	31
AC-SCK	I2C Serial Data interface	Output	AB4	32

WARNING

The design cuts straight between filters, so you will hear loud switching noise at full volume. So take you ear-buds out when playing with the switches

IMPORTANT NOTES

- On the schematics Line In is connected to the LAUX and RAUX on the Codec. The Microphone input is connected to LIN and RIN! This makes setting the mixers correctly somewhat confusing.
- The SLEWPD bit in P65 (0x40F9) is the Codec slew digital clock engine enable. When powered down (which is the default) the analog playback path volume controls are disabled and stay set to their current state.
- The codec is 24-bit stereo, but I treat it as if it is a 16-bit part.

Clocking

The internal logic of the ADAU1761 needs a core clock of 1024 times the sampling frequency. For 48kHz audio this work out to be 49.152MHz. To help with implementing this the codec chip has an PLL that can be used to generate a core clock from a external reference (MCLK) which is between 8MHz and 27MHz.

For this design I'm going to go with MCLK of 24MHz. To generate this 24MHz clock I'll need to take the PL clock of 100MHz, and then within the FPGA use that to generate 48MHz for the FPGA design to use. I'll divide this by two and send it out to the Codec.

By setting the MS bit of R15 to '1' the ADAU1761 can be told to act as master, so it will supply the I2S_CLK and I2S_LR signals back to the design running in the FPGA.

The pin that receives the I2S_CLK signal is collected to is not one of the Zynq's preferred clock inputs, I'm oversampling it at 48kHz, and then look for the edges where I need to process the I2S_Data signals. This has the benefit of keeping the FPGA logic all in one clocking domain.

The DSP filters

I wanted a simple to implement but functional design. To do this I've added three moving sum filters, of lengths 8, 16 and 32 samples. This gives filters that have critical frequencies of 6KHz, 3kHz and 1.5kHz. The active filter is selected using SW0 and SW1 on the zedboard, and the gain of the filter is removed using bit-shifts in the top level design.

Power up and register setup

Here is what I see is needed to get sound to appear on the line outs - it would pay to verify this yourself by checking against the datasheet.

(from page 24 through 28 of the datasheet)

- Apply a 24MHz clock
- Wait a while, for the FPGA clock to settle.
- Set up the PLL, by writing the following values to registers
- Wait until bit 1 on the last byte of register 2 is '1', indicating that the CODEC's PLL is locked. I don't do this, I just wait a while.
- Enable the core by setting reg[0] to 0x0F
- Wait a short while for the core to power up.

At this point the core is powered up and all the other configuration registers can be set

- Set the codec to be the I2S master
- Setting up the mixers
- Configure the DAC
- Configure the ADC
- Set up the serial signal routing
- Power up the various parts of the cores

At this point if the correct the codec should be transmitting I2S frames, and if I2S data is sent then sound will appear.

Here is the source for the I3C2 script that is used to configure the codec:

```
restart:
    ; Wait a while, for the FPGA clock to settle.
        DELAY 32768 ; a while

    ; Set up the PLL, by first turing it off
        ;   reg[0] <= 0x0E
        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; subaddress MSB
        WRITE   0x00
        WRITE   0x0E
        STOP

    ; then write the configuration values
        ; reg[2] <= 0x02 0x71 0x01 0x00 0x1b 0x01  (it requires 6 bytes written to it)
        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; subaddress MSB
        WRITE   0x02
        WRITE   0x00
        WRITE   0x7D
        WRITE   0x00
        WRITE   0x0C
        WRITE   0x23
        WRITE   0x01
        STOP

    ; Wait until bit 1 on the last byte of register 2  is '1',
    ; indicating that the CODEC's
    ;
    ; I cheat and wait 32,768 I2C clock cycles (about 64ms)
        DELAY   32768 ; a while

    ; Enable the core by setting reg[0] to 0x0F
        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; subaddress MSB
        WRITE   0x0
        WRITE   0xF
        STOP

    ; Wait a short while for the core to power up.
        DELAY   32768 ; a while

; At this point the core is powered up and all the other registers can be set.

    ; Become the I2S master.
        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; subaddress MSB
        WRITE   0x15 ; R15 Serial Port 0
        WRITE   0x1
        STOP


   ; Setting all the input mixers
        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; Reg[4] - Record mixer left control 0
        WRITE   0x0A
        WRITE   0x01 ; Enable the mixer
        STOP

        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; Reg[5] - Record mixer left control 1
        WRITE   0x0B
        WRITE   0x05 ; MX1AUXG = 0db
        STOP

        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; Reg[6] - Record mixer right control 0
        WRITE   0x0C
        WRITE   0x01 ; Just enable the mixer
        STOP

        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; Reg[7] - Record mixer right control 1
        WRITE   0x0D
        WRITE   0x05 ; Record mixer right Control 1 - MX2AUXG = 0db
        STOP

    ; Setting all the output mixers
        ; reg[0x1C] <= 0x2D   Playback mixer left 0
        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; subaddress MSB
        WRITE   0x1C
        WRITE   0x21
        STOP

        ; reg[0x1E] <= 0x4E   Playback Mixer right 0
        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; subaddress MSB
        WRITE   0x1E
        WRITE   0x41
        STOP

        ; reg[0x23] <= 0xE7   Playback Headphone volume left
        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; subaddress MSB
        WRITE   0x23
        WRITE   0xE7
        STOP

        ; reg[0x24] <= 0xE7   Playback Headphone volume right
        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; subaddress MSB
        WRITE   0x24
        WRITE   0xE7
        STOP

        ; reg[0x25] <= 0xE7   Playback Line out volume left
        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; subaddress MSB
        WRITE   0x25
        WRITE   0xE7
        STOP

        ; reg[0x26] <= 0xE7   Playback Line out volume right
        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; subaddress MSB
        WRITE   0x26
        WRITE   0xE7
        STOP


     ; Set up the ADC
        ; reg[0x19] <= 0x???
        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; subaddress MSB
        WRITE   0x19
        WRITE   0x03
        STOP

     ; Set up the DAC
        ; reg[0x29] <= 0x???
        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; subaddress MSB
        WRITE   0x29
        WRITE   0x03
        STOP

        ; reg[0x2A] <= 0x03
        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; subaddress MSB
        WRITE   0x2A
        WRITE   0x03
        STOP

     ; Set up the signal routing
        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; R58 - Serial Input Route Control
        WRITE   0xF2
        WRITE   0x01
        STOP
    ; Power up the various parts of the cores
        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; R59 - Serial Output Route Control
        WRITE   0xF3
        WRITE   0x01
        STOP

    ; Power up the various parts of the cores
        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; Reg[65] - Clock enable 0
        WRITE   0xF9
        WRITE   0x7F ; Power up all the modules
        STOP

        WRITE   0x76 ; dev_write_address
        WRITE   0x40 ; Reg[66] - Clock enable 1
        WRITE   0xFA
        WRITE   0x03 ; Enable the two clocks
        STOP

done:
        DELAY 32768
        JUMP done
```

Source project

This is the zip file for the complete project:

File:Zedboard dsp base project.zip

And here is the same project upgraded to 24-bit end-to-end.

File:Zedboard dsp base project 24bit.zip

Enhanced version

I've updated the project to route the switches via the I3C2 controller, allowing it to turn the audio mute on before switching filters. Most of the annoying loud pops when changing filters has gone. Much better.

The updated I3C2 program is in the project, as a text file.

File:Zedboard dsp base project 24bit v0.2.zip

Retrieved from "http://hamsterworks.co.nz/mediawiki/index.php/Zedboard_Audio"

- This page was last modified on 24 January 2014, at 08:33.