Developing Audio Applications on a StrongArm platform

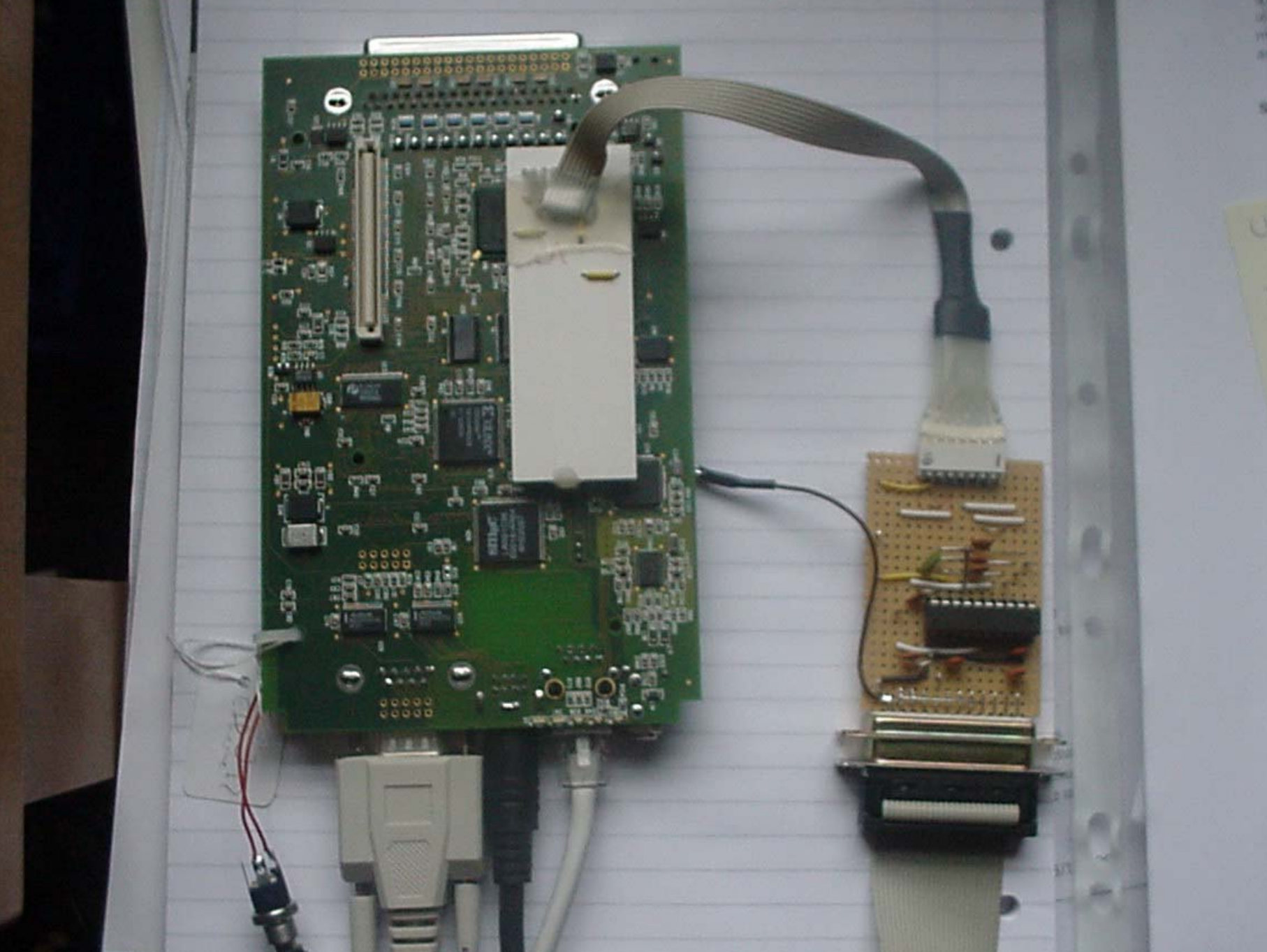
Craig Duffy
Bristol UWE, UK
craig.duffy@uwe.ac.uk

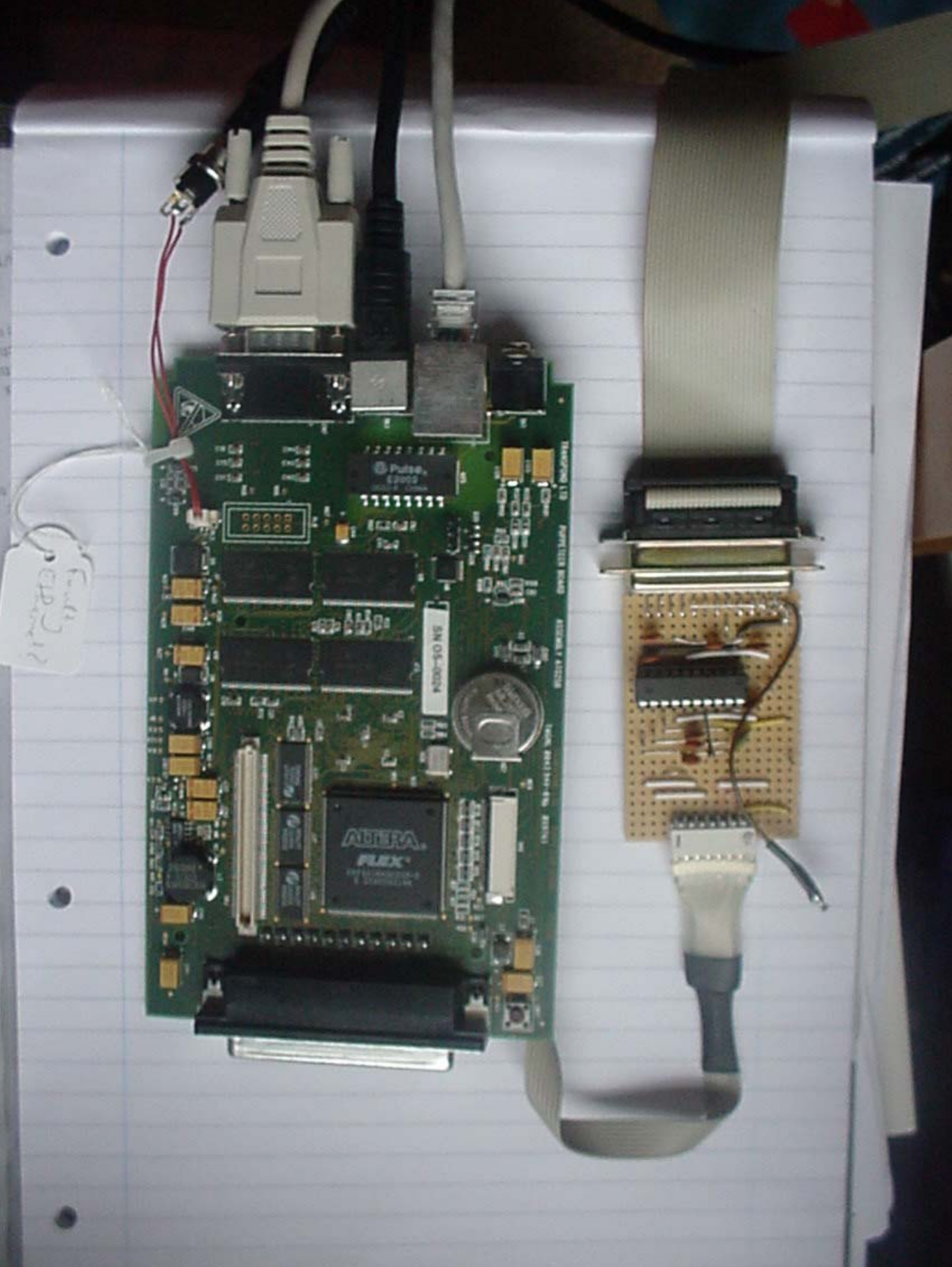
Background

- We use an embedded SA1110 board as part of our teaching of embedded systems at Bristol
  - In the 2nd year the students port a kernel and build some applications
  - In the final year we wanted to get our students building a full Internet radio system
    - This would involve them, in groups, developing a ‘portable’ radio player, a local controller and a central server
- However we first needed to figure out how to do it!

The Puppeteer Board

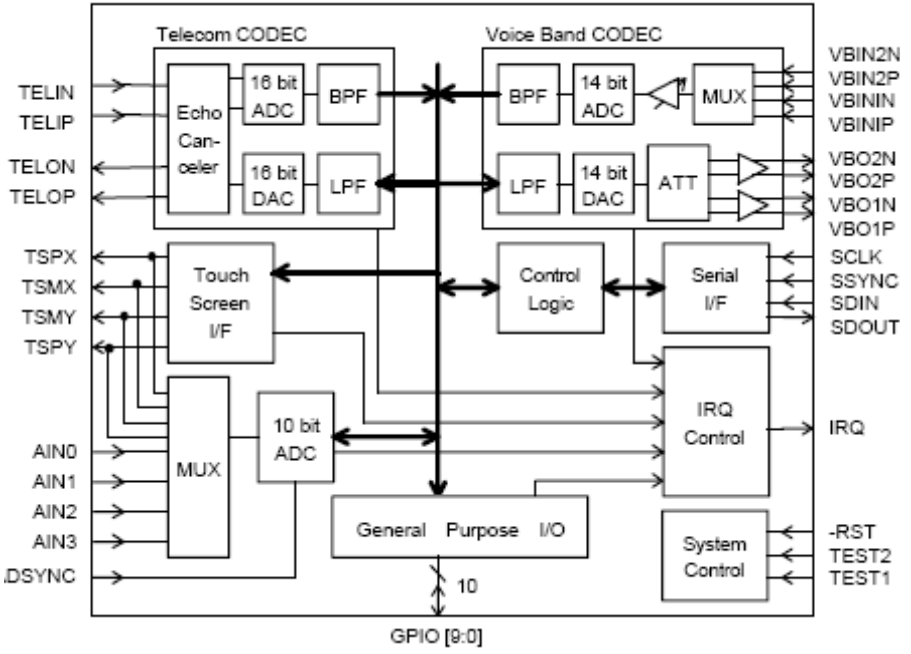
- The specification for the board is given below
  - 206 MHz SA1110 CPU
  - 8 Mbytes of Intel 28F320C3 Advanced+ Bootblock flash
  - 32 Mbytes of DRAM
  - SMCS LAN91c96 Ethernet
  - Toshiba TC35143BF AFE
  - Altera FLEX 6000 FPGA
  - Xilinx XCR3064A CPLD
  - I^2C EEPROM and RTC
    - Atmel AT24C02AN EEPROM
    - EPSON RTC-8563SA/JE
  - Ported Uboot and Linux 2.6.20 onto the board




ALTERA
FLEX

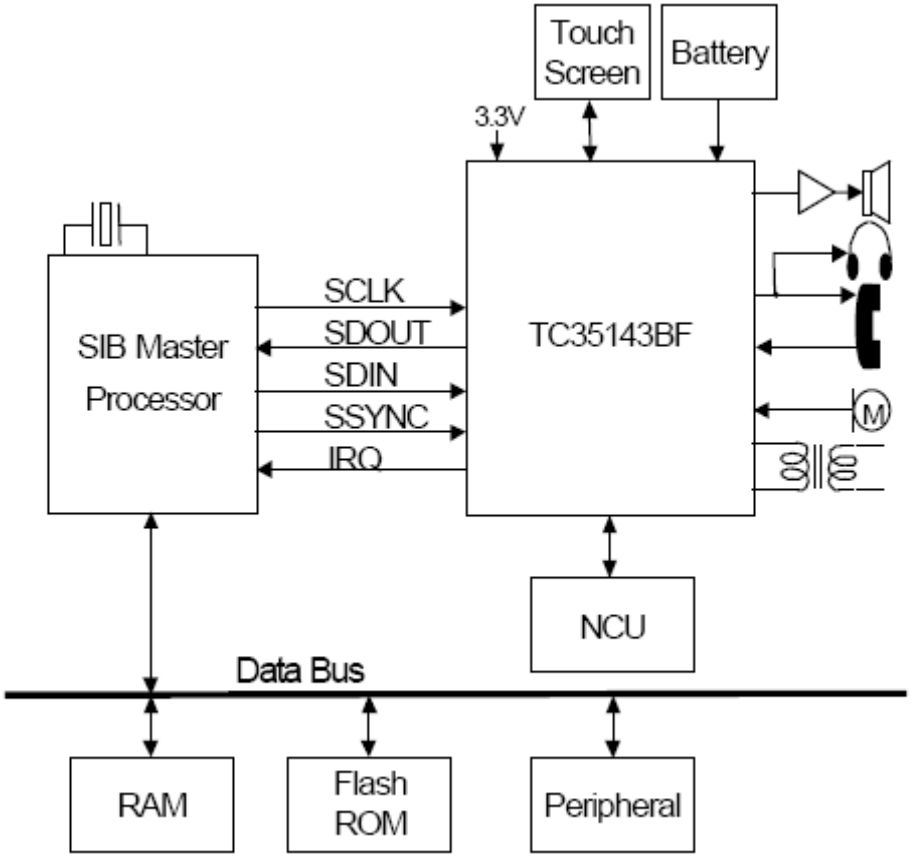
The Toshiba TC35143 AFE

- 16 bit telecom interface
- 14 bit audio interface
  - 2 input and 2 output channels
  - 22.1 kHz, mono
- 10 bit touch screen interface
- 10 bit GPIO interface

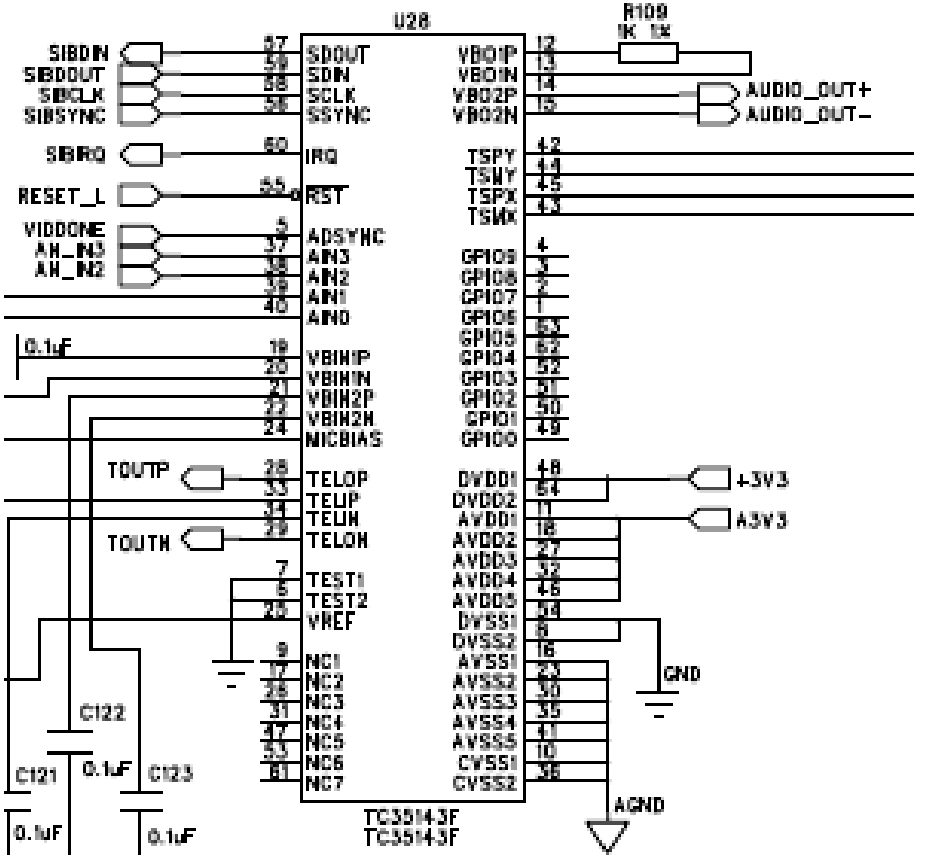


Toshiba SIB interface

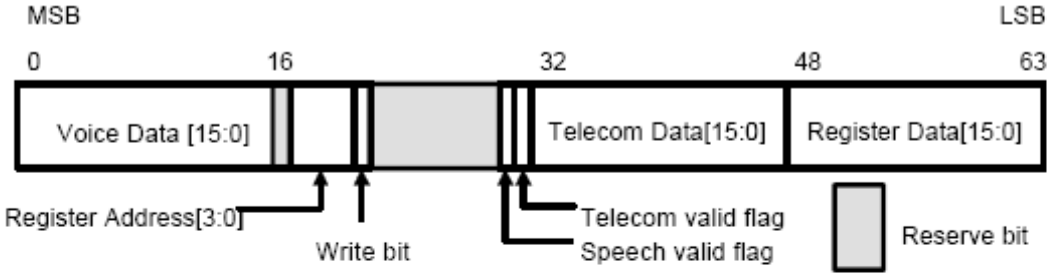
- The codec connects to the CPU through a serial interface bus (SIB)
- This is similar to the JTAG interface
- It runs continuously when the device is enabled



Puppeteer SA1110 SIB interface



SIB packet format



Frame Bit	SDIN Field Definition	SDOUT Field Definition
[0:15]	Voice DAC Data (16 bits) ; Bit[0]=MSB	Voice ADC Data (16 bits) ; Bit[0]=MSB
[17:20]	Register Address (4 bits) ; Bit[17]=MSB	Register Address (4 bits) ; Bit[17]=MSB
[21]	Write Bit (Write=1)	Must write to 0
[30]	Voice Valid DAC Sample Flag	Voice Valid ADC Flag
[31]	Telecom Valid DAC Sample Flag	Telecom Valid ADC Flag
[32:47]	Telecom DAC Data (16 bits) ; Bit[32]=MSB	Telecom ADC Data (16 bits) ; Bit[32]=MSB
[48:63]	Register Data (16 bits) ; Bit[48]=MSB	Register Data (16 bits) ; Bit[48]=MSB

SA1110 Codec SIB interface

- Actually the SIB packet is twice as large as in the previous slides
  - 128 bits instead of 64
- Toshiba claims that the second SIB packet is for addressing other codec devices
- Intel claims the second packet is for providing stereo
- It doesn’t seem to work
  - It does however waste a lot of data

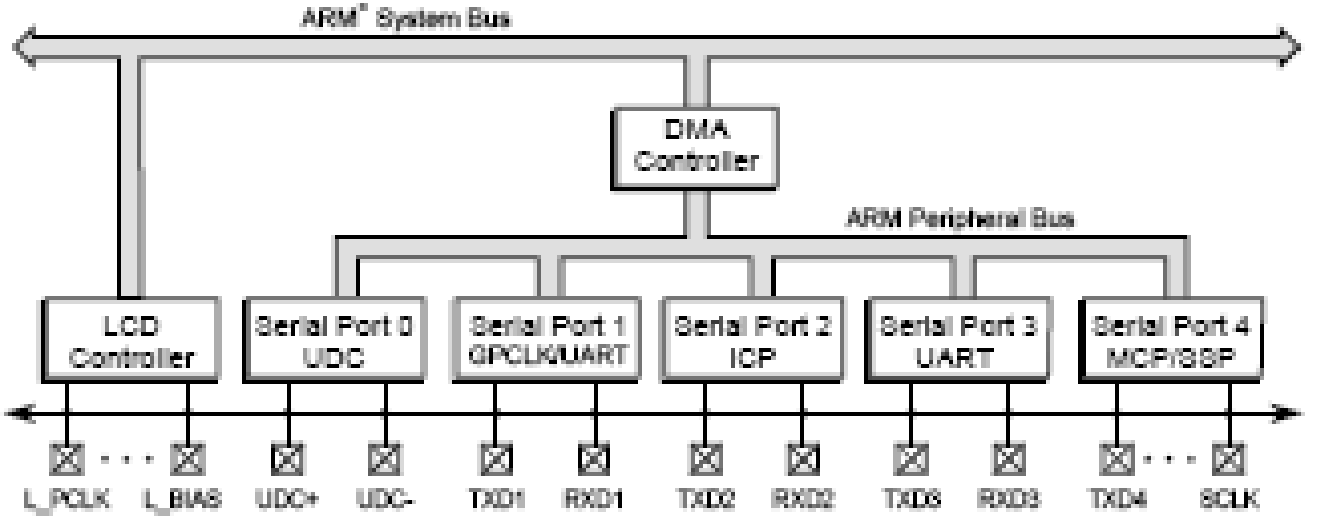
Codec register set

Address	Name	Description
0	IODAT	I/O Port Data – read status – write control
1	IODIR	I/O Port Direction
2	RISINT	Rising Edge Interrupt
3	FALINT	Falling Edge Interrupt
4	INTCLR	Interrupt Clear/Status
5	TELA	Telecom Control/Status A
6	TELB	Telecom Control/Status B
7	VOICEA	Audio Control/Status A
8	VOICEB	Audio Control/Status B
9	TSC	Touch Screen Control/Status
10-0xA	ADC	10 bit Analogue/Digital Control/Status
11-0xB	ADCDAT	10 bit Analogue/Digital – write only
12-0xC	DATE	Shows development date – read only
13-0xD	MODE	Control/Status of operation mode
14-0xE	RSV	Reserved
15-0xF	RSV	Reserved

The SA1110 Multimedia Control Port (MCP)

- The interface to the codec on the StrongARM is through the Multimedia Control Port (MCP)
- The port is part of the ARM peripheral bus
- The MCP was designed for the Philips UCB1x00 series of codecs
  - These appear to be identical to the TC35143

The SA1110 Peripheral Control Module.



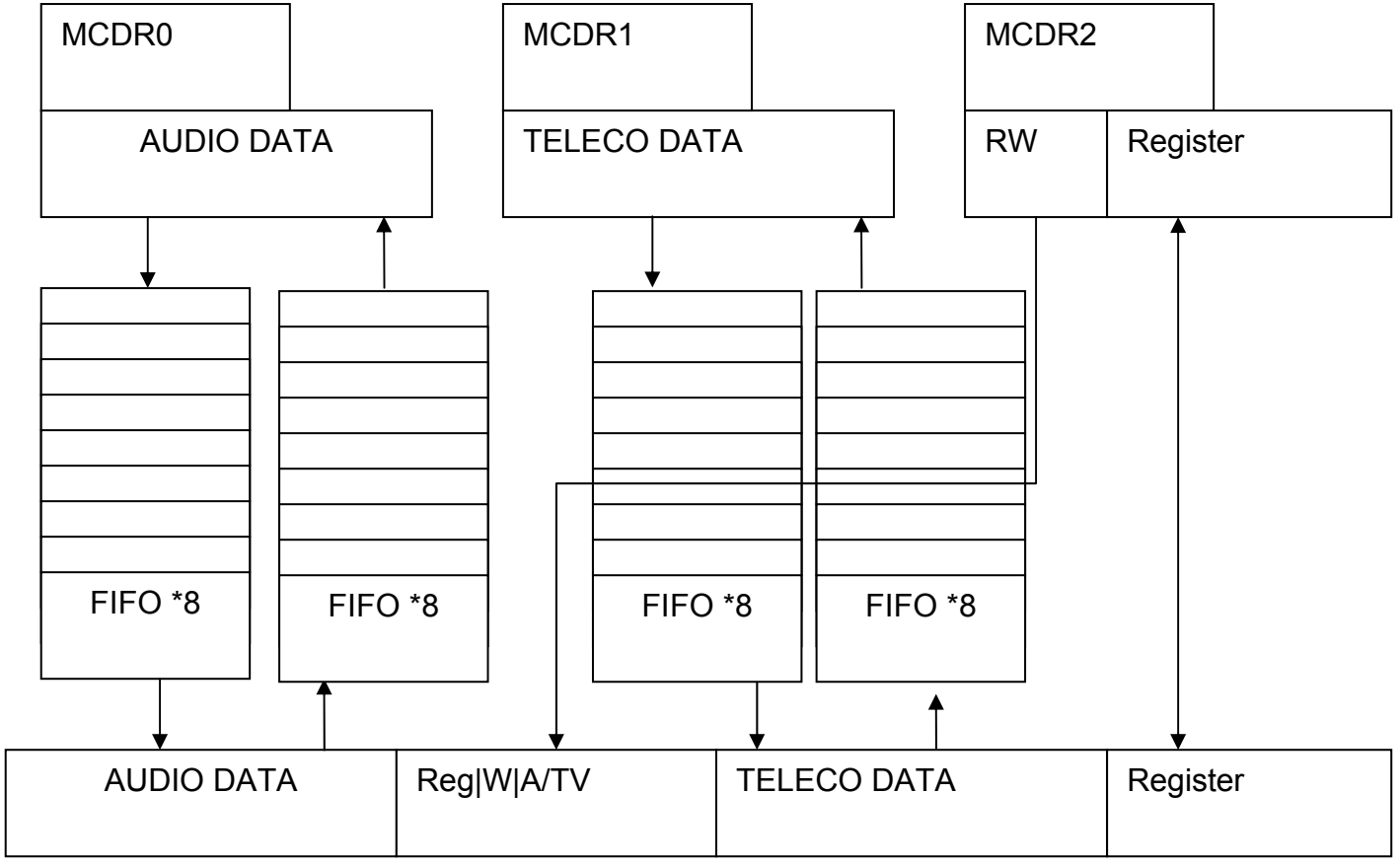
MCP Registers

Address	Name	Description
0h 8006 0000	MCCR0	MCP control register 0
0h 8006 0004	—	Reserved
0h 8006 0008	MCDR0	MCP data register 0
0h 8006 000C	MCDR1	MCP data register 1
0h 8006 0010	MCDR2	MCP data register 2
0h 8006 0014	—	Reserved
0h 8006 0018	MCSR	MCP status register
0h 8006 001C – 0h 8006 005C	—	Reserved

Note: MCCR1 resides within the same address space as the PPC.

0h 9006 0030	MCCR1	MCP control register 1

MCP codec SIB interface



Linux device driver support

- The MCP/Codec device driver is supported in the 2.6 series kernel
- Enabled through Multimedia port drivers in menuconfig
- In 2.6.20 it is, rather oddly, in drivers/mfd
- It is a very basic implementation
  - It only supports some of the touch screen and GPIO functionality
  - There is an assasbet touch screen module

2.6.20 drivers/mfd files

File	Description
mcp-core.c	Generic MCP functions – used for all (?) ARM MCP actions, all are pointers to those in mcp-sa11x0.c
mcp-sa11x0.c	Interface for actual SA11x0 MCP port – deals with the SA11x0 MCP port registers
ucb1x00-core.c	The low level sets of functions and control for interfacing to the Toshiba chip. Most function calls are wrappers to mcp-sa11x0.c
ucb1x00-assabet.c	Basic interfaces for a touch screen driver for the assabet board – not use by out project
ucb1x00.h	Toshiba register bit masks and settings.
mcp.h	mcp driver data structure and ops.

Changes to puppeteer.c

- Some changes were required to the board start-up file, puppeteer.c

- ```
  static struct mcp_plat_data puppeteer_mcp_data = {
          .mccr0                  = MCCR0_ADM,
          .sclk_rate    = 11981000,
  };
  ……
  ```
- ……
- ```
  static void __init puppeteer_init(void)
  {
    sa11x0_set_flash_data(&puppeteer_flash_data, puppeteer_flash_resources,
                                            ARRAY_SIZE(puppeteer_flash_resources));
    platform_add_devices(devices, ARRAY_SIZE(devices));
    sa11x0_set_mcp_data(&puppeteer_mcp_data);
  ```

2.6.20 driver functionality

- The driver doesn’t actually do very much
- However in ucb1x00_probe the TC35143 does respond
- And on inserting the bit masks to enable and un-mute to the codec control register we do get hissing through headphones.
- The code is in ucb1x00-core.c

```
static int ucb1x00_probe(struct mcp *mcp)
{
struct ucb1x00 *ucb;
struct ucb1x00_driver *drv;
unsigned int id;
int ret = -ENODEV;
mcp_enable(mcp);
id = mcp_reg_read(mcp, UCB_ID);
if (id != UCB_ID_1200 && id != UCB_ID_1300 && id != UCB_ID_TC35143) {
            printk(KERN_WARNING "UCB1x00 ID not found: %04x\n", id);
            goto err_disable;
            }
.....
.....
id = mcp_reg_read(mcp, UCB_AC_B);
id |= UCB_AC_B_OUT_ENA;
mcp_reg_write(mcp,UCB_AC_B,UCB_AC_B_OUT_ENA);
mcp_reg_write(mcp,UCB_AC_A,0x0d);//set the TC35143 rate to 22.1 kHz
mcp_reg_write(mcp,UCB_MODE,0x400);//set the TC35143 rate to 22.1 kHz
ucb1x00_set_audio_divisor(ucb,0x5e0);//set the MCP  rate to 22.1 kHz
.........
```

A Polled I/O application

- To test out the codec capabilities we created a basic polled driver
- This simply creates a character device and implements the write operation for it
- After some fiddling got reasonable sound from it but
  - It wouldn't function in a multiprocessing environment
    - You could hear the scheduler!
  - It clicked all the time!

ucb1x00-core write operation

```
static ssize_t device_write(struct file *filp, const char __user *buffer, size_t length, loff_t *offse
{
  short *data = NULL;
  short *sample = NULL;

  printk("Audio device write\n");
  if ( (data = kmalloc(length, GFP_KERNEL)) == NULL)
    return -ENOMEM;

  sample = data;

  copy_from_user(data, buffer, length);

  ucb1x00_audio_codec_write(ourmcp, data,length);
  kfree(sample);

  return length;
}
```

mcp-sa11x0.c audio write operation

```
static int mcp_sa11x0_audio_write(struct mcp *mcp, const char *buffer, unsigned int length)
{
  unsigned int counter =0;
  short *data;
  DPRINTK("in mcp_sa11x0_audio_write\n");
  data = buffer;
  while ( counter < length )
    {
      Ser4MCDR0 = (int)*data;
      data++;
      counter +=2;
      while( (Ser4MCSR & MCSR_ANF) == 0 )
                ;  //nothing
    }
  return length;
}
```

A Silicon bug in the SA1110?

- The clicking is claimed to be due to a bug in the SA11x0 silicon
- Apparently calls to the OS timer when the MPC FIFO is empty causes random garbage to be sent which is rendered by the codec, causing a click
  - This happens quite a lot!
- Could test this by writing a standalone, non-OS, application
  - Life is too short
- The DMA driver can solve this by continually writing data (even 0s).

Direct Memory Access (DMA) in the SA1110

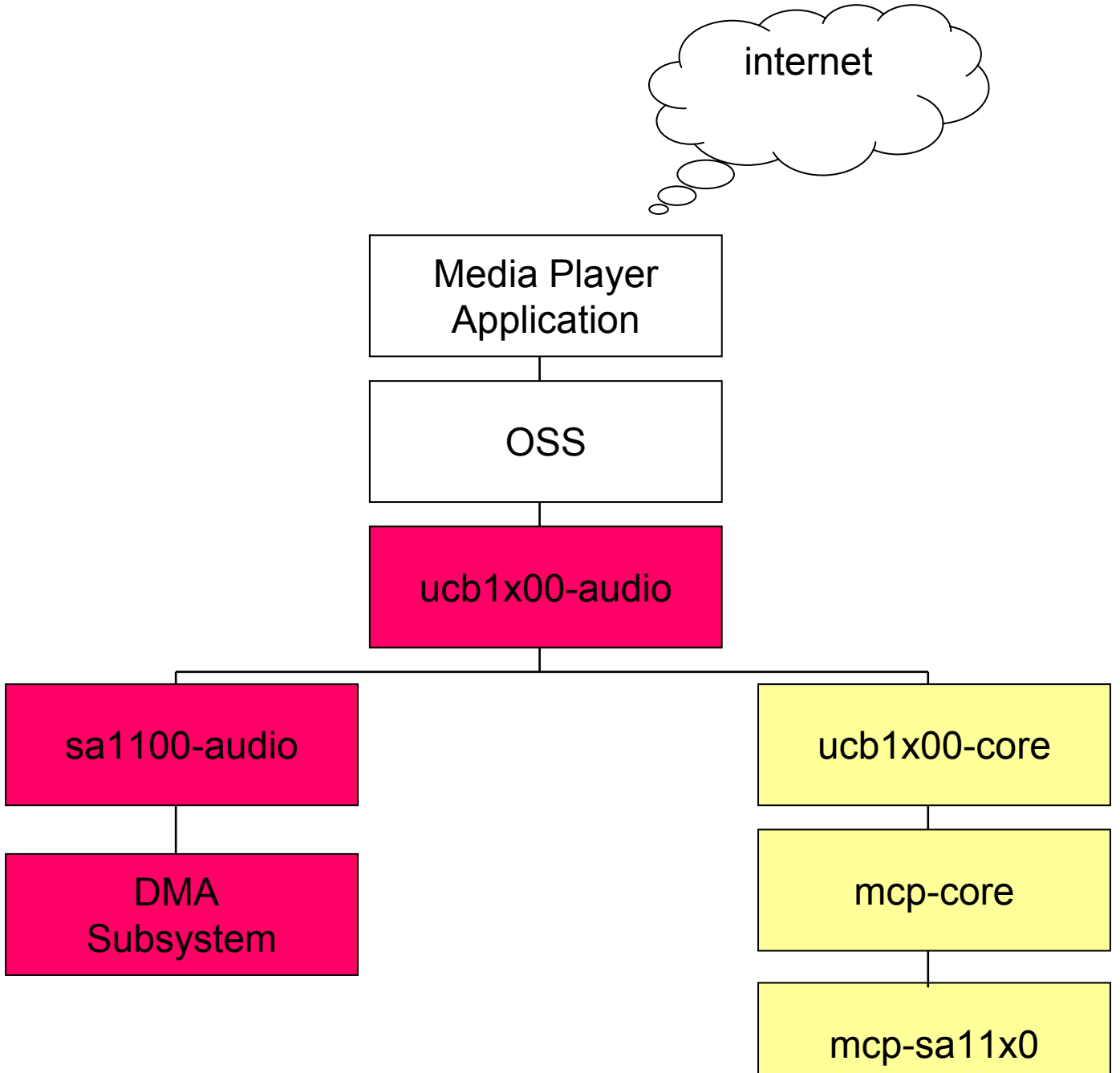
- The StrongARM provides 6 duplex DMA channels
- Each can be programmed to operate with 1 of the peripheral control ports – for example the MCP port
- They can work in polled or interrupt mode

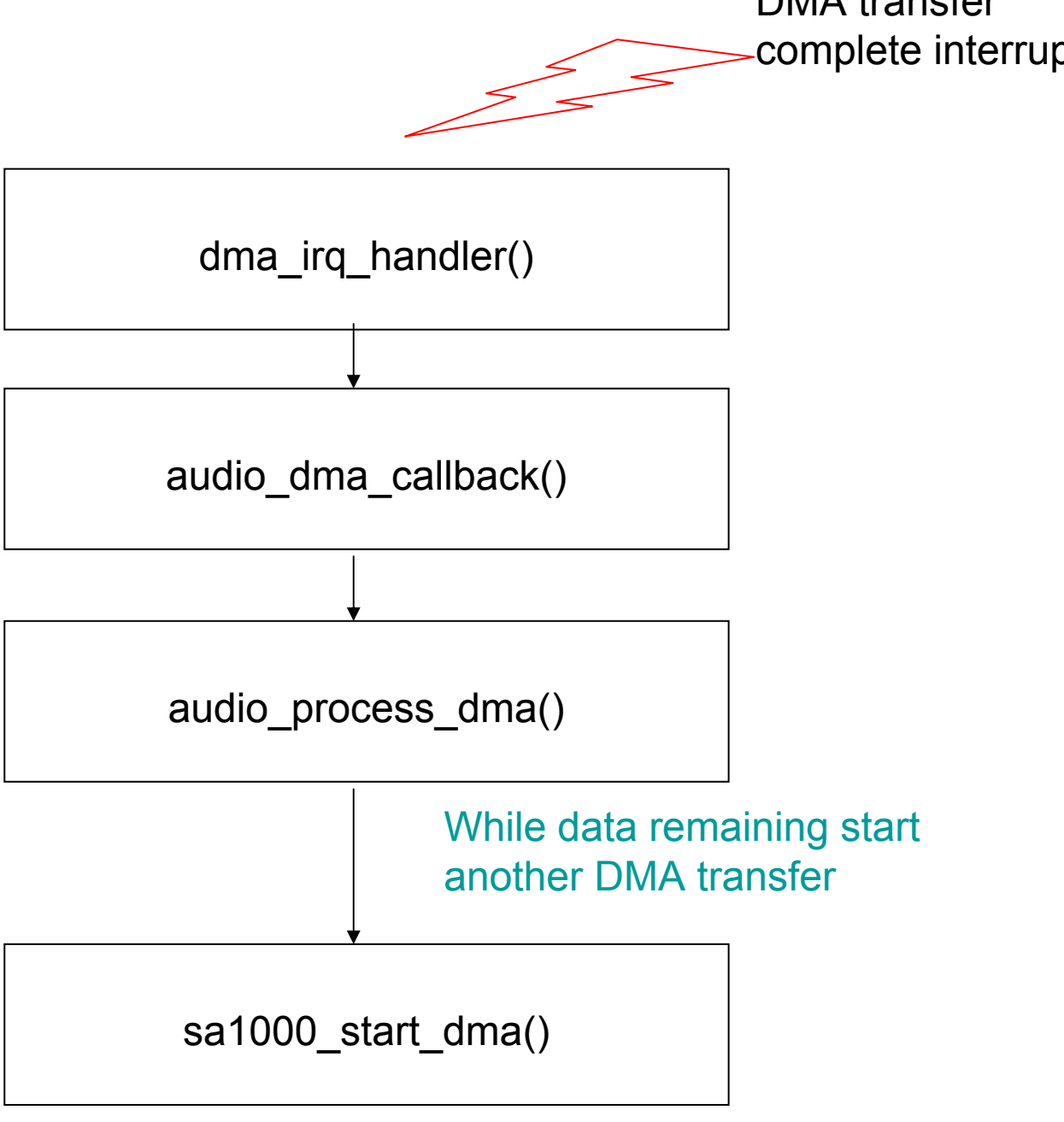
SA1110 DMA registers

Physical Address	Register Name	Symbol
Channel 0 Registers		
0h B000 0000	DMA device address register.	DDAR0
0h B000 0004	DMA control/status register 0. Write 0x7F to set.	DCSR0
0h B000 0008	Write 0x7F to clear.	
0h B000 000C	Read only.	
0h B000 0010	DMA buffer A start address 0.	DBSA0
0h B000 0014	DMA buffer A transfer count 0.	DBTA0
0h B000 0018	DMA buffer B start address 0.	DBSB0
0h B000 001C	DMA buffer B transfer count 0.	DBTB0

Finding an audio driver

- The Sharp Zauras PDA has a very similar hardware profile to the Puppeteer
- There has been a lot of open source development on the Zauras – some supported by Sharp
  - All of the releases are named after dogs
    - Collie, poddle, tosa
- Lots of good web sites and wikis
- A set of audio drivers with DMA were found at handhelds.org
  - Part of the familiar PDA distribution


internet
Media Player Application
OSS
ucb1x00-audio
sa1100-audio
DMA Subsystem
ucb1x00-core
mcp-core
mcp-sa11x0


DMA transfer
complete interrup
dma_irq_handler()
audio_dma_callback()
audio_process_dma()
While data remaining start
another DMA transfer
sa1000_start_dma()

Fixing the Handheld code

- The handheld code had quite a few problems
- It was mainly intended for telecoms operation
  - This was hard coded in
- It didn't export any of the driver functions
- None of the MCP private data was passed
- The semaphores were back to front or missing in some key routines
- The DMA coherency mask was missing
- The driver classes were different between the stock 2,6 kernel MCP driver and the handhelds one

A word of warning

- When you do this type of development you will probably end up coding late into the night
- You will probably turn up your speakers/headphones to maximum (just in case!)
- You will put the driver into maximum debug mode
- So
  - At 3am you might find that
  - Your music blares out
  - And your screen fills with debug messages after hours of not doing anything!

#cat f3.wav > /dev/dsp0

```
[ 8290.160000] sa1100_audio_attach fmode 14
[ 8290.160000] MCCR0 is 57f2f
[ 8290.160000] sa1100_audio_attach registered
[ 8290.200000] audio_write: count=4096
[ 8290.210000] buf 0: start ffc00000 dma 0xc0090000
[ 8290.210000] buf 1: start ffc02000 dma 0xc0092000
[ 8290.220000] buf 2: start ffc04000 dma 0xc0094000
[ 8290.220000] buf 3: start ffc06000 dma 0xc0096000
[ 8290.230000] buf 4: start ffc08000 dma 0xc0098000
[ 8290.230000] buf 5: start ffc0a000 dma 0xc009a000
[ 8290.240000] buf 6: start ffc0c000 dma 0xc009c000
[ 8290.240000] buf 7: start ffc0e000 dma 0xc009e000
[ 8290.250000] write 4096 to 0
[ 8290.250000] audio_write: return=4096
[ 8290.250000] audio_write: count=4096
[ 8290.260000] write 4096 to 0
[ 8290.260000] audio_process_dma
[ 8290.260000] audio_write: return=4096
[ 8290.270000] audio_write: count=4096
[ 8290.270000] write 4096 to 1
[ 8290.270000] audio_write: return=4096
[ 8290.280000] audio_write: count=4096
[ 8290.280000] write 4096 to 1
[ 8290.280000] audio_process_dma
[ 8290.290000] audio_write: return=4096
[ 8290.290000] audio_write: count=4096
[ 8290.290000] write 4096 to 2
[ 8290.300000] audio_write: return=4096
[ 8290.300000] audio_write: count=4096
[ 8290.300000] write 4096 to 2
[ 8290.310000] audio_process_dma
[ 8290.310000] audio_write: return=4096
[ 8290.310000] audio_write: count=4096
[ 8290.320000] write 4096 to 3
[ 8290.320000] audio_write: return=4096
[ 8290.320000] audio_write: count=4096
[ 8290.330000] write 4096 to 3
```

Codec library support

- The StrongARM doesn’t have a Floating Point Unit (FPU)
- Therefore library support must be provided to make up for the lack of an FPU
- There are lots of audio libraries
- We chose libmad
  - Mpeg Audio Decoder library
    - www.sourceforge.net/projects/mad/

Buildroot, libmad and mpg123

- Buildroot a system for building cross compilers, root file systems, libraries and application comes with an increasing range of apps.
- In recent versions support for both libmad and mpg123 has been included
- This makes for a very easy way of testing some of the board’s audio capabilities

Building an application Mplayer

- If we wanted our students to get a fuller feel for developing a more commercial audio application then we felt Mplayer was a better application
- Mplayer has
  - Loads of codecs/containers
  - -slave option to allow it to be a backend for another local server
  - LIRC Linux Infrared Remote Control

Building Mplayer for ARM

- This is quite a process as it is mainly supported on x86 platforms
- A slightly older version worked
- It required a lot of reconfiguration

- CC=arm-linux-gcc RANLIB=arm-linux-ranlib LD=arm-linux-ld ./configure –disable-x11
- –target=arm-linux –enable-crash-debug –prefix=../bin/ --language=en –disable-tv
- –disable-rtc –disable-vm –disable-freetype –disable-pmg –disable-esd –disable-nas
- –disable-sdl –disable-gl –disable-menucoder –disable-iconv –disable-gif –disable-jpeg
- –disable-xv –disable-dga –enable-menu –enable-static –host-cc=gcc –enable-mad
- –with-extraincdir=$PWD/../bin/include/ –with-extralibdir=$PWD/../bin/lib/
- --disable-largefiles --disable-libavformat –disable-libavcodec

Conclusion

- The board can produce reasonable sounding mono audio and stream data from Internet radio stations
- The project is going to be quite large and should challenge students
- We will look into IR and I2C integration