AT91SAM Internet Radio

Features

- **Stand-alone**
- **Software MP3 Decoder**
- **Low Memory Footprint**
- **LCD Display**

1. Introduction

Compared to traditional radio broadcasting, audio streaming over the Internet has a major advantage: immediate access to an unlimited number of radio stations with global coverage. On the other hand, the desktop computers used as receivers do not offer the same kind of simple handling as traditional radio receivers. The solution presented here combines the advantages of both. For its implementation, we exclusively used OpenSource tools and components. The solution presented here is based on Atmel AT91 ARM® Thumb®-based microcontrollers: A minimal solution is based on the AT91SAM7X-EK, while the AT91SAM9260-EK is used for a more powerful variant.

Except the MP3 decoder, all source code is provided under BSD style license and may be used for commercial or non-commercial, open or closed source applications free of charge. The source code of the MP3 decoder is published under the OSI (Open Source Initiative) certified RealNetworks® Public Source License (RPSL).

An add-on board named Calypso is presented, which can be mounted on the AT91SAM9260-EK and the AT91SAM7X-EK. It mainly contains a TLV320AIC23B stereo audio codec, an LCD and three push buttons. All CAD files are available in the appendices.

AT91 ARM Thumb Microcontrollers

Application Note

6318A–ATARM–28-May-07

2. Embedded Internet Radio Overview

The problems which are to be expected during the implementation of an Embedded Internet Radio are explained by means of the data flow from the recording station to the listener at the radio receiver (see Figure 2-1).

In almost all cases, audio data is available in analog form and needs to be converted to its digital representation. An analog-to-digital converter (ADC) transfers analog audio to pulse-code modulated (PCM) data. The necessity of audio compression for Internet streaming soon becomes clear when doing a few calculations.

For music re-production in CD quality, 16-bit samples are used at a rate of 44.1 kHz. This results in a bit rate of

- Mono: 44,100 Hz x 16 bits = 705,600 bit/s
- Stereo: 44,100 Hz x 16 bits x 2 = 1,411,200 bit/s

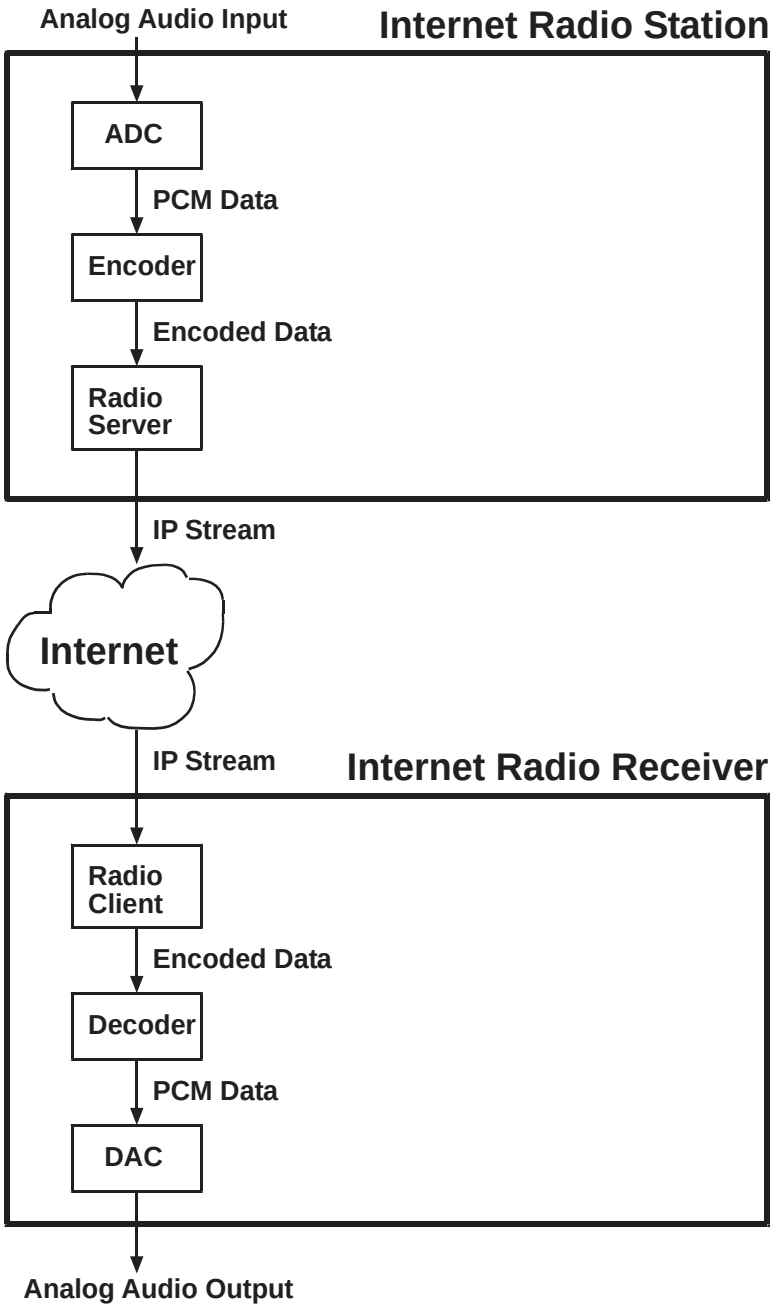
With MP3 encoding it is possible to reduce this rate by a factor of more than 10 without bothering or even noticeable reduction of quality.

Today, realtime encoding still requires CPU performance, which is not always available, specially when high quality reproduction is demanded. Luckily, the encoded data can be created in advance and stored on digital media for later streaming.

At the client side (radio receiver), the digitized and compressed data must be extracted from the IP stream and routed through a decoder for decompression. In contrast with encoding, the MP3 decoding process requires far less computing power. It uses low cost, low power embedded systems as radio receivers.

Finally, the decoded PCM data is transmitted to a digital-to-analog converter (DAC), which reconstructs the analog audio signal. This signal can be used to drive an amplifier with attached headphones or loudspeakers.

Figure 2-1. Internet Radio Data Flow



2.1 Software for Radio Stations

Several products are available to set up and run a radio station. They differ in many aspects, some of which have an impact on our radio receiver as well.

2.1.1 Icecast

Maintained by the Xiph.Org Foundation, a non-profit organization, this solution is quite similar to, but not fully compatible with the SHOUTcast server. Ogg Vorbis is the preferred codec, but MP3 and other codecs may be used as well.

2.1.2 QuickTime

Apple®'s Open Source Server is called Darwin. It supports MP3 coding and HTTP as well as RTSP.

2.1.3 Real Networks

This company offers different levels of streaming solutions for all major platforms (including Mac OS®, Linux®/Unix® and Windows®), starting from free low end servers up to professional solutions with the ability to feed some thousand clients. Several IP protocols are supported, including HTTP and RTSP. The codec is named RealAudio®.

2.1.4 SHOUTcast

Nullsoft's free Winamp-based streaming solution is based on HTTP. MP3 as well as AAC codecs are used.

2.1.5 Windows Media

This closed standard has been developed by Microsoft® and competes against MP3, RealAudio, QuickTime and other codecs and streaming solutions.

2.1.6 Selected Solution

Real Networks and Windows Media, which may have the largest market share, are partly or fully closed standards and therefore difficult to use in Open Source Projects.

SHOUTcast and its alternative, Icecast, are an open standards and use the popular MP3 codec, of which several Open Source implementations have been published. Since 1997 a large number of Internet radio stations offer SHOUTcast streaming. Its main disadvantage is the TCP based HTTP protocol, which has not been designed for real time streaming. Another problem is scalability, because HTTP doesn't support broadcasting or multicasting. Each client opens its own connection to the server and the server has to transmit a separate stream to each client.

On the other hand, compared to RTSP based solutions, SHOUTcast is easy to implement and available on almost all networks, even if located behind firewalls and proxies. Thus, MP3 streaming via SHOUTcast had been selected for this project.

2.2 Expected Problems

2.2.1 Streaming Difficulties

As stated above, the TCP protocol used by SHOUTcast is not suited for real time streaming. If packets are lost, re-transmission is requested and no more data is passed to the application until the lost packet finally arrives.

However, when listening to radio stations, a delay of some seconds between the arrival of the data and its playback is acceptable. This delay can be used for generous data buffering at the client side to compensate re-transmission delays.

Due to the lack of an external memory bus, the AT91SAM7X is limited to its internal RAM, which in turn limits the maximum transfer rate on unreliable connections.

2.2.2 Decoding Difficulties

MP3 decoding is a lot easier than encoding, but still a challenging task for microcontrollers. A fixed point implementation with processor specific optimizations helps.

The Helix™ MP3 decoder, which had been chosen for this project, offers the following features:

- Pure 32-bit fixed-point implementation
- High-quality C reference code for porting to new platforms
- Optimized for ARM processors
- MPEG1 layer 3 - sampling frequencies: 48 kHz, 44.1 kHz, 32 kHz
- MPEG2 layer 3 - sampling frequencies: 24 kHz, 22.05 kHz, 16 kHz
- MPEG2.5 layer 3 - sampling frequencies: 12 kHz, 11.025 kHz, 8 kHz
- Supports constant bit rate, variable bit rate, and free bit rate modes
- Supports mono and all stereo modes (normal stereo, joint stereo, dual-mono)

2.2.3 Converter Difficulties

The required efforts for feeding the DAC should not be underestimated. As a big relief regarding CPU performance, both microcontrollers used here offer I2S hardware with double buffered PDC support.

3. Implementation Details

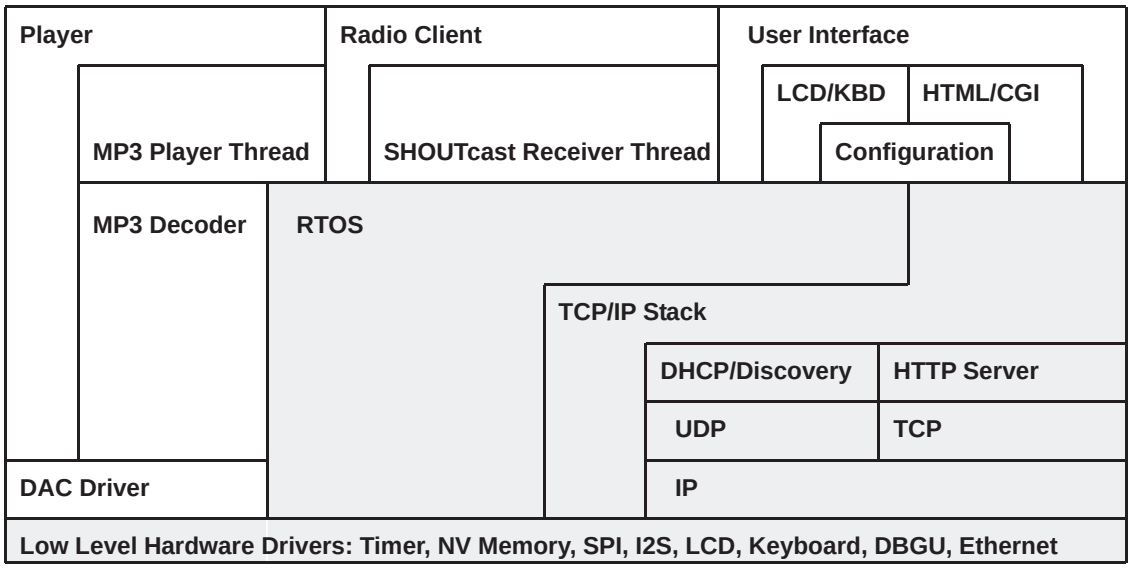
The application specific part of the software is divided into three main parts:

User Interface
Accepts user commands via CGI or push buttons and displays status information on LCD or HTML pages.

Radio Client
Connects to a selected radio station and fills a receiver buffer with MP3 encoded data.

Player
Passes MP3 data from receiver buffer to an MP3 software decoder and transfers PCM data to an I2S driver.

Figure 3-1. Software Modules



3.1 Nut/OS

Nut/OS is a modular operating system, which means that there is no big kernel block containing all available functions. Instead, the whole system is packed in libraries. When linked to the application, only the referenced parts of the operating system are included. This way it is well suited for embedded devices with limited memory.

The non preemptive kernel is kept intentionally simple and provides a few basic services only.

Cooperative Multithreading
In opposite to preemptive kernels, this simplifies resource sharing, but requires responsible application or interrupt code to maintain realtime behaviour.

Event Queues
Priority ordered queues, to which threads or interrupts may post events, which in turn wakes up threads waiting on these queues.

Timer Management
Thread may give up CPU control for a specified time or specify a time out value while waiting for an event.

Dynamic Memory Management
The heap manager uses a best fit, address ordered algorithm to keep the free-list as unfragmented as possible.

The kernel's main purpose is to provide a framework for an easy implementation of the included TCP/IP stack. Initially implemented on 8-bit AVR microcontrollers, it had been recently ported to 32-bit MCUs based on ARM7™ and ARM9™ architectures.

The grey shaded parts in Figure 3-1 are provided by Nut/OS libraries. There is currently no DAC driver available and therefore one has been included in the application code. Further, the selected MP3 decoder is published under a different license (RPSL), which is not compatible to the BSD license of the system libraries. Thus, it is packed separately.

3.2 SHOUTcast Streaming

The SHOUTcast protocol is based on a modified version of HTTP.

The client opens a TCP connection to the server and sends a standard HTTP request.

The server responds with „ICY 200 OK“ if the connection is accepted, followed by one or more name-value pairs, separated by carriage return / linefeed characters. Names are separated from their values by colons.

An empty line (two pairs of carriage return / linefeed characters) indicates the end of the header and is immediately followed by a continuous stream of MP3 data.

Client Request	Server Response
`GET / HTTP/1.0`	

	`ICY 200 OK`
	`icy-notice1:This stream requires Winamp`
	`icy-notice2:SHOUTcast Distributed Network`
	`icy-name:FREQUENCE3-www.frequence3.fr`
	`icy-genre:Top 40 Dance Pop Rock`
	`icy-url:http://www.frequence3.fr`
	`content-type:audio/mpeg`
	`icy-pub:1`
	`icy-metaint:32768`
	`icy-br:192`

- **icy-notice1**

General information, which often contains HTML tags. Additional information lines may be sent by using icy-notice2, icy-notice3 etc.

- **icy-name**

Information about this channel, typically its name followed by additional informations.

- **icy-genre**

Genre of the channel.

- **icy-url**

URL of the station's homepage.

- **content-type**

Mime media type of this stream, usually audio/mpeg

- **icy-pub**

Indicates if the stream is public (1) or private (0).

- **icy-metaint**

Metadata interval. Specifies the number of bytes after which a metadata block is embedded into the stream. Typical values are 8192, 16000 or 32768. If no metadata is sent by this channel, the value is either 0 or this header item is missing.

- **icy-br**

Bit rate of the stream in kbit/s. However, not always reliable.

3.2.1 SHOUTcast Metadata

The optional metadata is embedded in the stream at fixed intervals given by the icy-metaint header. The receiver is responsible for stripping this data before the stream is passed to the MP3 decoder.

MP3 Frame n	Metadata	MP3 Frame n cont'd	MP3 Frame n+1	MP3 Frame n+2	Metadata	MP3 Frame n+2 cont'd

Each metadata block begins with a single byte value. It specifies the number of 16-byte segments of metadata that follow the length byte.

Length	Information	Padding

- **Length**

A one byte value. Multiplied by 16 it specifies the total length of the metadata excluding the length byte. May contain zero, in which case the previously sent meta information is still valid.

- **Information**

Contains a variable number of name-value pairs separated by semicolons. Names are separated from its value by an equal sign and values are surrounded by single quotes.

- **Padding**

As the length is given in multiples of 16, unused bytes are padded with zeros.

The information part of a typical metadata block may look like this:

```
StreamTitle='Robbie Williams - Lovelight';StreamUrl='';
```

- **StreamTitle**

Stream contents, like the name and interpreter of a song currently played or the name of the radio moderator currently speaking etc.

- **StreamUrl**

Web site, which is related to the stream contents.

After the metadata had been stripped from the bit stream, the remaining MP3 data is passed to the MP3 decoder for further processing.

3.3 MP3 Decoding

Decoding an MP3 stream is a complex task. This chapter does not describe all details, but presents an overview of the required steps.

MP3 streams are segmented into frames, each of which contains a 32-bit header. In a first step, the decoder must locate a frame header in order to become synchronized.

Table 3-1. Frame Layout

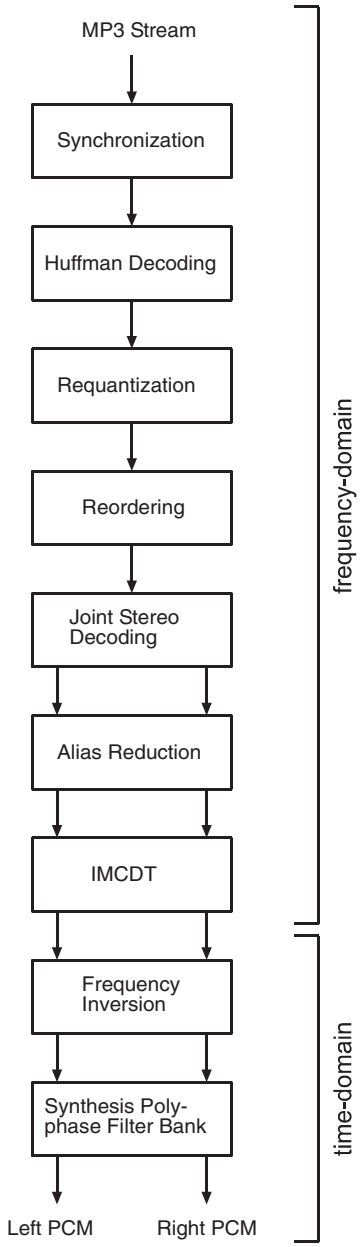
Header	CRC	Side Info	Main Data	Ancillary Data

Huffman decoding, requantization and reordering convert the MP3 bit stream into 32 sub bands with 18 frequency lines each.

If the stream contains a stereo signal, then the two channels are separated in the next step. All further processing has to be performed on both channels.

Antialiasing, IMDCT and filter bank transform the stream from the frequency domain into the time domain.

Figure 3-2. MP3 Decoding



3.3.1 Synchronization

This part searches the incoming bit stream for an 11-bit synchronization word. This enables the decoder to tune in at any position of the transmitted stream. Fortunately, MP3 frames always start at byte boundaries. However, the synchronization word may appear in other parts of a frame, resulting in one or more synchronization errors before the receiver is finally locked.

3.3.2 Huffman Decoding

The Huffman decoder uses the side info part of the MP3 frame to transform the compressed data into scale factors and symbols representing the 576 frequency lines of the 32 sub bands

3.3.3 Requantization

The scaled and quantized frequency lines from the Huffman decoder are used by this part to rescale the symbols into non-scaled frequency lines.

3.3.4 Reordering

Samples close in frequency often get similar values. In order to increase the efficiency of the Huffman compression, the encoder sorts specific frequency lines within sub bands This part of the decoder restores the original order.

3.3.5 Stereo Decoding

In most cases music data contains some redundancy between left and right channel. Thus, compressing the sum and the difference of both channels is more efficient than compressing each channel separately.

This step splits the encoded stereo signal into a separate left and right channel.

3.3.6 Antialiasing

Due to non-ideal bandpass filtering in the encoder, alias effects are inevitable. This part corrects this by merging frequencies with butterfly calculations.

3.3.7 Inverse Modified Discrete Cosine Transformation

The task of the IMDCT is to map the antialiased frequency lines to 32 polyphase filter sub bands and return 18 time domain samples for each of the 32 sub bands

3.3.8 Frequency Inversion

This part of the decoder compensates frequency inversion of the following filter bank by multiplying every odd time sample of every odd sub band with –1.

3.3.9 Synthesis Polyphase Filter Bank

In a final step the 32 sub bands of 18 time domain samples are transformed to 18 blocks, each of which contains 32 PCM samples.

3.4 TLV320AIC23B Driver

As stated earlier, Nut/OS does not provide any built-in support for the TLV320AIC23B Audio Codec. Therefore, a code module named tlv320dac.c had been added to the application code. It contains routines to read and write the DAC registers via I2C as well as routines to transmit the PCM samples from the MP3 decoder via I2S. The latter is described here in more detail.

Since the MP3 decoder transforms frame after frame at various speed, it is not able to produce a continues flow of data, like it is required by the DAC for continuous audio reproduction. Thus,

PCM data needs to be buffered in some way. To provide this, the DAC driver uses an array (pcm_bufq) and two indices to create a circular buffer. One index (bwr_idx) points to the array position that receives the next PCM data block from the decoder, while the other index (brd_idx) points to the position, from which the next data is transmitted to the DAC. The buffer is considered empty, if both indices are pointing at the same position, and considered full, if an increment of the write index would reach the index of the read position.

```
typedef struct _PCM_BUFFER {
    u_short *wbf_dat;
    int wbf_siz;
    int wbf_len;
} PCM_BUFFER;

static PCM_BUFFER pcm_bufq[SAMPLE_BUFFERS];
static volatile u_int brd_idx;
static u_int bwr_idx;
```

In fact, the array does not contain the PCM data itself, but pointers to separately allocated buffers filled with PCM samples.

The application constantly receives MP3 data, calls the decoder and passes the decoded PCM data to the DAC driver using Tlv320DacWrite(). This routine fills the circular buffer or, if the buffer is full, calls NutEventWait(), which is provided by the Nut/OS kernel. The first parameter specifies the queue to wait on, while the second parameter specifies a timeout value in milliseconds. The function returns zero, if an event was posted to this queue and the waiting thread has been woken up before the timeout elapsed. The idea is to install an interrupt handler that triggers on PDC transmit empty, updates the PDC registers for the next buffer transfer and posts an event to this queue.

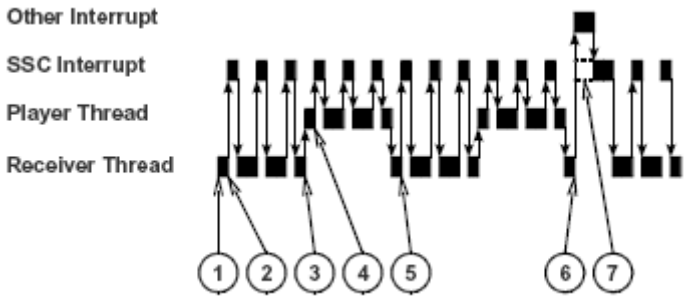
```
static HANDLE i2s_que;

int Tlv320DacWrite(void *buf, int len)
{
    u_int idx;

    idx = bwr_idx + 1;
    if (idx >= SAMPLE_BUFFERS) {
        idx = 0;
    }

    while (idx == brd_idx) {
        if (NutEventWait(&i2s_que, 100)) {
            Tlv320DacStart();
        }
    }

    if (pcm_bufq[idx].wbf_siz < len) {
        if (pcm_bufq[idx].wbf_siz) {
            free(pcm_bufq[idx].wbf_dat);
            pcm_bufq[idx].wbf_siz = 0;
        }
        pcm_bufq[idx].wbf_dat = malloc(len * 2);
```

```
            if (pcm_bufq[idx].wbf_dat == NULL) {
                /* Out of memory. */
                return -1;
            }
            pcm_bufq[idx].wbf_siz = len;
        }

        memcpy(pcm_bufq[idx].wbf_dat, buf, len * 2);
        pcm_bufq[idx].wbf_len = len;
        bwr_idx = idx;

        return 0;
    }
```

3.4.1 Interrupt Handling

Interrupt handling is a key feature of the AT91SAM series. While the CPU is busy with several tasks like MP3 decoding and TCP protocol handling, the DAC demands constant feeding of PCM data. Remember, that stereo audio data, which had been sampled at 44.1 kHz, results in a transfer rate of about 1.4 Mbit per second. In other words, the DAC requests a new 16-bit value roughly every 11.4 microsecond. This strict requirement will significantly reduce the overall performance of many CPU cores. Another critical factor is interrupt latency time. Any delay, even of a few nanoseconds only, will result in audible distortions.

Figure 3-3. SSC Interrupts without PDC

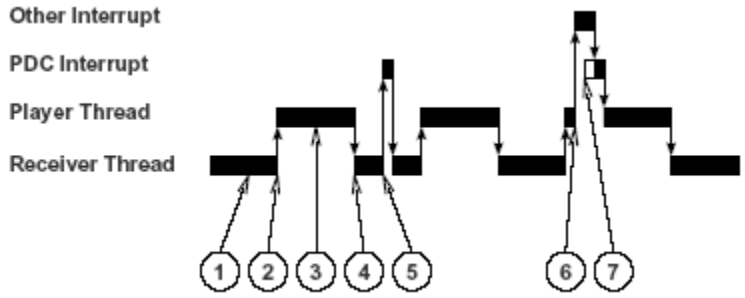


1. Receiver thread receives MP3 data from radio station and fills MP3 buffer.
2. SSC interrupt handling required for each DAC sample, slowing down the receiver thread.
3. Receiver thread informs player thread about new MP3 data.
4. Player thread decodes MP3 data and fills PCM buffer, slowed down by SSC interrupts.
5. Receiver thread running again after player thread finished decoding.
6. Another interrupt occurs, like system timer, Ethernet etc.
7. Critical delay of SSC interrupt handling.

One solution for reducing interrupt latency would be interrupt nesting. But still the overall system performance will suffer from the high SSC interrupt rate. The PDC hardware of the AT91SAM devices removes this burden from the software. While one buffer of PCM data is transferred to the DAC without CPU intervention, a second buffer is already waiting. As soon as the first buffer has been completely sent, the PDC will automatically switch to the second buffer and trigger an

interrupt. While the second buffer is transmitted, there is a lot of time left to write the next buffer address into the PDC register.

Figure 3-4. SSC Interrupts with PDC Enabled



1. Receiver thread receives MP3 data from radio station and fills MP3 buffer. PDC hardware transfers data to the SSC without CPU intervention.
2. Receiver thread informs player thread about new MP3 data.
3. Player thread decodes MP3 data and fills a PCM buffer, while another buffer is concurrently transferred to the SSC by the PDC hardware.
4. Receiver thread running again after player thread finished decoding.
5. PDC completely transferred one buffer and automatically switches to the next buffer, which had been previously filled by the player thread. Finally an interrupt is initiated to set up the PDC for the next buffer switch.
6. Another interrupt occurs, like system timer, Ethernet etc.
7. Another buffer had been transferred and the PDC switches to the next buffer, which had been set in 5. Thus, SSC transfer continues and interrupt latency is not critical.

Nut/OS hides all hardware specific implementation of interrupt handling by providing a set of API calls, most notably NutRegisterIrqHandler(). The DAC driver calls this routine during initialization to register an interrupt handler.

```
static int Tlv320I2sInit(u_int rate)
{
    /* Register SSC interrupt handler and enable SSC interrupt. */
    NutRegisterIrqHandler(&sig_SSC, I2sInterrupt, 0);

    Tlv320I2sDisable();
    Tlv320I2sEnable(rate);
    NutIrqEnable(&sig_SSC);

    return 0;
}
```

The interrupt handler itself is quite simple. It calls a function to update the PDC registers of the SSC and posts an event to a Nut/OS queue.

```
static void I2sInterrupt(void *arg)
{
    I2sPdcFill();
    NutEventPostFromIrq(&i2s_que);
}
```

Here is the routine, which is called by the interrupt handler to update the PDC interface of the SSC:

```
    static void I2sPdcFill(void)
    {
        if (brd_idx != bwr_idx) {
            if (inr(SSC_TNCR) == 0) {
                if (++brd_idx >= SAMPLE_BUFFERS) {
                    brd_idx = 0;
                }
                outr(SSC_TNPR, (u_int) pcm_bufq[brd_idx].wbf_dat);
                outr(SSC_TNCR, pcm_bufq[brd_idx].wbf_len);
            }
        }
}
```

3.5 TCP Streaming

The radio's network interface for MP3 streaming is divided into three modules:

Station Handling
Allows selection of a radio station from a pre-configured list as well as connecting to and disconnecting from a selected station.

Receiver Framework
Controls and monitors a specific protocol plug-in.

SHOUTcast Plug-In
All real work is done here in a background thread, which receives data from the established TCP connection, strips optional metadata and fills the remaining MP3 data into the receiver buffer. A utility routine is provided to parse the HTTP header response collected by the station handler during connection.

The framework / plug-in structure allows to add different protocols like Icecast easily.

Details of the TCP/IP Protocol are handled internally by the operating system. Nut/OS offers a high level interface based on C studio routines such as printf and scanf to transfer data to and from the network. But for maximum performance, native calls like NutTcpReceive and NutTcpSend are used in our radio application.

3.6 Audio Player

Similar to the TCP streaming implementation, the audio player offers a framework / plug-in structure to allow integration of additional decoders like AAC. However, the DAC driver calls are hard coded in the MP3 Player Plug-In. Using a different DAC hardware does not only require to implement a new driver, but also to rewrite the player plug-in.

The two parts of the player are:

Player Framework
Controls and monitors a specific player plug-in.

MP3 Plug-In
A background thread moves MP3 data from the receiver buffer to a large MP3 buffer. If the latter is sufficiently filled, the thread calls the MP3 decoder and optionally a resampler. The resulting

PCM data is passed to the DAC driver. If the MP3 buffer runs out of data, DAC feeding is temporarily stopped until enough MP3 data has been collected again.

3.7 User Interface

Two interfaces are provided:

LCD With Push Buttons
Enables the user to control the audio output volume or to select another radio station. The LCD shows the name of the currently connected station and, if provided by the stream, the name and interpreter of a song currently played. Texts which do not fit in the 16 columns of the LCD are continuously scrolled.

HTTP Server
This module enables the user to retrieve dynamically created HTML pages from the radio, which provides more detailed status information than the LCD interface. Fixed content, like images and the main page, is stored in program memory, using the very simple Nut/OS UROM file system.

3.8 Configuration

While Nut/OS uses the serial DataFlash® on the AT91SAM evaluation board as non-volatile memory to save its basic IP Network Configuration, the radio application's configuration is hard coded. However, most settings can be easily modified by changing preprocessor macros. Some of these macros are conditionally set, depending on whether the application is built for the AT91SAM7X256 or the AT91SAM9260 evaluation kit.

3.8.1 AT91SAM7X Memory Considerations

The evaluation boards, which are directly supported by this application, present two extremes. While the AT91SAM9 board offers far more RAM than our radio would ever be able to consume, the 64 Kbytes RAM of the AT91SAM7X256 are actually less than the minimum.

Compared to all other modules, the Helix MP3 decoder is quite memory hungry. It consumes about 23 Kbytes heap memory and about 1 Kbyte stack space. The internal AT91SAM7X Ethernet MAC's receive and transmit buffers are allocated from internal RAM as well. The Nut/OS driver's default is 4 Kbytes for the receiver and about 3 Kbytes used by two transmit buffers. RAM usage of the modular operating system mainly depends on which parts are actually referenced by the application code. For TCP streaming applications about 16 Kbytes are expected. The application code itself, not counting any data buffers, requires less than 4 Kbytes of RAM for static variables and at least about the same amount of stack space.

In total, this would leave 10 Kbytes for TCP, MP3 and PCM buffers, which is far less than what would be nearly sufficient for fluently playing MP3 streams via TCP from an Internet host. A short latency of just one second of a 192 Kbit stream needs 24 Kbytes buffer capacity. In reality, much larger gaps of several seconds are quite common.

Using the AT91SAM7X based radio with streaming servers in local networks only would solve all these problems at once, because network latencies are in the range of milliseconds. Further, the AT91SAM7X512, which had been announced at the time of this writing, offers twice the RAM capacity and looks like the perfect candidate.

Anyway, after excluding some non-essential parts from the application, limiting audio sample and MP3 transfer rates and fine tuning buffer and stack space, we can finally get the Internet radio running nicely on the AT91SAM7X256.

The following limitations apply to the AT91SAM7X version:

Excellent feed required
The radio server as well as the Internet connection need to offer good performance and reliability. Intermediate streaming gaps of about 1 second are acceptable. This allows to reduce the size of the internal Ethernet and TCP buffers to a minimum.

Low sample rate
Limiting the audio sample rate to 8 kHz stereo and 16 kHz mono allows to run the radio with very small buffers

Fixed sample rate
A fixed sample rate allows to remove the Hermite Resampler, which would otherwise require 1 to 4 Kbytes of additional buffer space, depending on the output rate.

No HTTP Server
The LCD and push button interface is sufficient to run the radio. Removing the HTTP Server frees about 4 Kbytes RAM and significantly reduces the TCP buffer for outgoing data. It further allows to remove the UROM file system. Unlike the 8-bit AVR implementation, for which it had been designed initially, the 32-bit ARM version is not yet able to directly read the file contents from flash memory. Thus, the complete file system of about 7 Kbytes needs to be copied to RAM during runtime initialization.

No network discovery support
The discovery feature of Nut/OS, which allows to detect all Nut/OS based devices in a local network had been disabled.

8 radio stations only
About 1 Kbyte of RAM can be saved with this reduction.

The remaining RAM had been carefully distributed among buffers and thread stacks by trial and error. Though, as no reliable method exists to determine the maximum stack space required by each thread under all conditions, an additional safety margin had to be added.

3.8.2 Network Settings

The radio uses DHCP to retrieve its IP configuration. However, a unique Ethernet MAC address is required and is hard coded in webradio.c.

```
#define MY_MAC { 0x00, 0x06, 0x98, 0x30, 0x00, 0x35 }
```

In case DHCP is not available, the radio uses fixed IP settings, which are also hard coded in webradio.c. Most probably these need to be modified for your network.

```
#define MY_IPADDR "192.168.192.35"
#define MY_IPMASK "255.255.255.0"
#define MY_IPGATE "192.168.192.1"
```

While TCP transfers data via continuous streams, the underlying IP transfers packets, so called datagrams. Thus, TCP divides the data into segments before passing it to IP and re-assembles segments received from the IP layer. On the way from the source to the destination node, IP datagrams may get further fragmented and it should be noted that Nut/OS is not able to handle IP fragments. The maximum segment size is adjustable and a value of 536 is most probably guarantees that no fragmentation of IP packets occur. For best performance, the segment size may be increased up to 1460. The actual value is defined in config.h.

```
#define MAX_TCPSEG_SIZE 536
```

Experience has shown that in some high traffic local networks a segment size of 536 gives better performance than the maximum of 1460.

Some networks, specifically in larger organizations, may only allow to establish TCP connections via a proxy server. In this case add the following preprocessor macros in config.h. Of course, the specified IP address and port number differ among networks.

```
#define MY_PROXY_IP   "192.168.192.2"
#define MY_PROXY_PORT 80
```

The TCP receive buffer size is usually the same value as the so called TCP window size. It specifies the number of bytes that the sender transmits before expecting an acknowledge response from the receiver. Most likely our MP3 stream uses completely filled segments and it is a good idea to specify the TCP window size an integer multiple of the segment size. For the AT91SAM9260, 60 times the segment size is used, while for the limited memory of the AT91SAM7X a factor of 3 is specified. Again this definition is found in config.h.

```
#ifndef MAX_TCPBUF_SIZE
#if defined(AT91SAM9260_EK)
#define MAX_TCPBUF_SIZE (60 * MAX_TCPSEG_SIZE)
#else
#define MAX_TCPBUF_SIZE (3 * MAX_TCPSEG_SIZE)
#endif
#endif
```

You may be aware that TCP is a connection oriented protocol. This means that a connection has to be established before data can be transferred. By default, TCP intentionally ignores temporarily broken connections. For example, removing the network cable does not force disconnecting established connections. This often results in some difficulties for the application. Luckily, in our case we expect a continuous stream of MP3 data and simply setting a receiver timeout of some seconds is sufficient to detect broken connections. The following macro can be found in config.h and specifies the receiver timeout in milliseconds.

```
#define MAX_TCPRCV_WAIT     5000
```

3.8.3 MP3 Player Settings

Two settings specify the amount of MP3 data buffered, used to bypass TCP stream latencies. One is the total buffer size and the radio starts playing after it is filled to two third. When streaming low bit rates, this might take too long and a second parameter in config.h limits the maximum time in seconds to wait for the buffer being filled.

```
#ifndef MAX_WAIT_MP3BUF_FILLED
#define MAX_WAIT_MP3BUF_FILLED  10
#endif

#ifndef MP3_BUFSIZ
#if defined (AT91SAM9260_EK)
#define MP3_BUFSIZ          1048576
#else
#define MP3_BUFSIZ          (4 * MAINBUF_SIZE)
#endif
```

```
#endif
```

The DAC on the radio extension board can be configured for different sample rates when driven by the on-board 12 Mhz crystal. Furthermore, a jumper on the extension board is provided to alternatively feed the clock input of the DAC chip by the PCK1 output of the AT91SAM9260 or the PCK2 output of the AT91SAM7X. Anyway, the radio application doesn't support DAC clock re-configuration but uses the Hermite software resampler to generate a fixed output sample rate. Although not specifically stated, the resampler seems to work with downsampling only. Thus, the specified output rate is also the maximum sampling rate that the radio is able to play.

```
#ifndef DAC_OUTPUT_RATE
#if defined (AT91SAM9260_EK)
#define DAC_OUTPUT_RATE     44100
#else
#define DAC_OUTPUT_RATE     8000
#endif
#endif
```

The DAC driver adds an additional buffering level for the AT91SAM's SSC device. The time the MP3 decoder needs to decode an MP3 frame is not constant. At high sample rates the decoder may not always be able to keep up with the critical timing requirements of the DAC. The PCM buffer of the DAC driver decouples both processes. On the other hand, decoded samples require much more buffer space than encoded MP3 data for the same amount of audio playing time and the PCM buffer in the DAC driver should not be used to substitute MP3 buffer space. The following macros are defined in tlv320dac.c.

```
#ifndef SAMPLE_BUFFERS
#if defined (AT91SAM9260_EK)
#define SAMPLE_BUFFERS  32
#else
#define SAMPLE_BUFFERS  3
#endif
#endif
```

3.8.4 List of Radio Stations

The list of radio stations the user may choose from is hard coded in config.c, which is sufficient for demonstration purposes. New stations can be easily added by inserting additional calls to ConfigStation. The second parameter is used as a symbolic name and presented to the user when selecting a new station to connect to. Large radio stations offer several servers and if the connection to the first one in the list fails, the radio will automatically switch to the next entry with the same symbolic name.

```
radionics_rstation = ConfigStation(MAXNUM_STATIONS, "FREQUENCE3",
"88.191.22.199:8000");
ConfigStation(MAXNUM_STATIONS, "FREQUENCE3", "194.158.114.68:8000");
ConfigStation(MAXNUM_STATIONS, "FREQUENCE3", "194.158.114.67:8000");
ConfigStation(MAXNUM_STATIONS, "FREQUENCE3", "193.251.154.243:8000");
...
ConfigStation(MAXNUM_STATIONS, "SKY.FM Classic", "128.177.3.80:4078");
ConfigStation(MAXNUM_STATIONS, "SKY.FM Classic", "66.90.113.135:8010");
```

New stations can be found at SHOUTcast website. You may either use the Winamp radio client and view the file information for the URLs of the radio station or directly download the playlist file to you local disk, which may then be viewed with a simple text editor like Notepad. Although Nut/OS in general provides DNS queries, our application expects IP addresses only, not host names.

When starting the radio, it will first connect to the station of which the return code of the Config-Station call is stored in radio.rc_rstation.

3.8.5 Miscellaneous Setting

The LCD scrolling rate is specified in userif.c and given in milliseconds.

```
#ifndef UI_REFRESH_RATE
#define UI_REFRESH_RATE 300
#endif
```

4. Running the Binaries

The archive in the appendix contains ready-to-run binaries for the AT91SAM9260-EK and the AT91SAM7X256-EK. If your local network provides DHCP and allows direct access to the Internet, the binaries should work fine for a first go.

4.1 Software Installation

Download and run the Windows installer executable ethernut-4.3.3.exe (or any later version) from www.ethernut.de and follow the instructions provided by the installation wizard. It is not yet required to run the Nut/OS Configurator. By default, all files will be installed in C:\ethernut-4.3.3. If you choose a different installation directory, the following descriptions need to be adjusted accordingly.

When done with the installation, unpack the radio application archive (samir-1.0.2-2.zip, available in the appendix) into the Nut/OS installation directory. The archive contains several extensions to the standard Nut/OS package. Right now we are interested in the pre-build binaries only.

nut\bin\arm7tdmi\webradio.bin
Binary of the SAM7X build, to be added to the bin\arm7tdmi directory within the Nut/OS source tree, C:\ethernut-4.3.3\nut\bin\arm7tdmi by default.

nut\bin\arm9\webradio.bin
Binary of the SAM9260 build, to be added to the bin\arm9 directory within the Nut/OS source tree, C:\ethernut-4.3.3\nut\bin\arm9 by default.

nut\bin\arm9\Web_Radio_dataflash_at91sam9260ek.bin
DataFlash bootloader for the AT91SAM9260-EK.

nut\bin\arm9\Web_Radio_nandflash_at91sam9260ek.bin
NAND Flash bootloader for the AT91SAM9260-EK.

4.2 Running on the AT91SAM9260-EK

Attach the Calypso radio extension board to the SAM9260 evaluation board, plugging the double row header socket of the Calypso Board on the PIO B header of the AT91SAM9260-EK. Make sure that all 40 pins are connected.

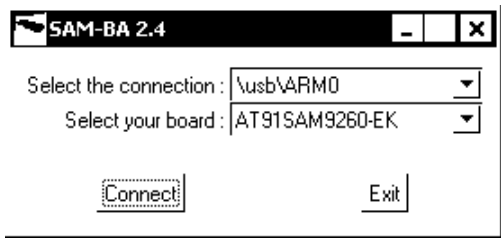
Connect the AT91SAM9260-EK to your local Ethernet, to your PC's USB port and serial port (using the DBGU connector) and to the power supply. Also plug in a headphone to the connector on the Calypso board marked OUTPUT.

Start any terminal emulator (HyperTerminal or TeraTerm, if available) and set the serial port to 115200,8,n,1, all handshakes disabled. When switching on the power supply, the back-lit LCD will go on, but nothing will be displayed yet. You should see the following output from the SAM-BA boot loader in the terminal emulator window:

```
RomBOOT
>
```

Start the SAM-BA v2.4 utility on the PC, select the connection \usb\ARM0 and the board AT91SAM9260-EK and click the connect button.

Figure 4-1. SAM-BA Connection Dialog

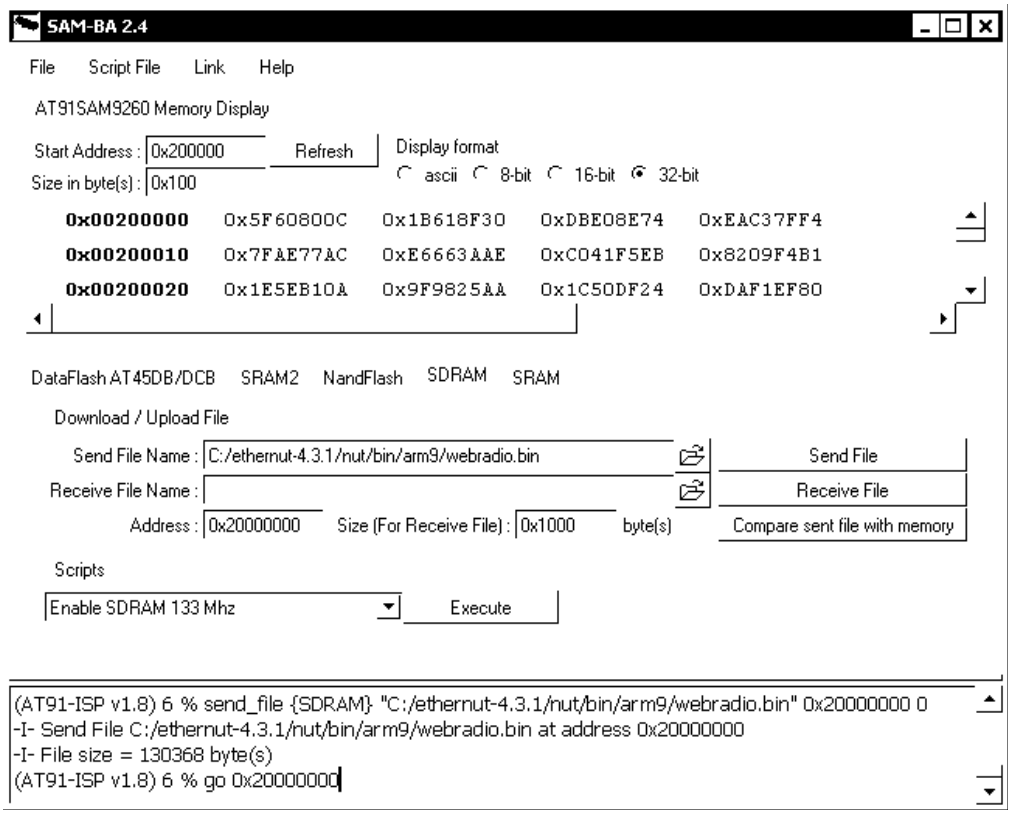


Click on the DataFlash tab, select the script "Enable Dataflash on CS1" and click on the Execute button.

Next select the script "Send Boot File" and again click on the Execute button. A file selection dialog will appear. Navigate to the nut/bin/arm9 directory and open the file Web_Radio_dataflash_at91sam9260ek.bin. The bootloader file will be transferred to the board's non-volatile DataFlash.

Next press the browse button to choose a file to send to the board. Again a file selection dialog pops up. This time open the file webradio.bin. Back in the SAM-BA main window enter 0x8000 in the Address field and finally press the button Send File. The radio application binary will be transferred to DataFlash location 0x8000. You may now exit the SAM-BA tool.

Figure 4-2. SAM-BA SAM9260 DataFlash Programming



Pressing the reset button on the target board will invoke the internal ROM bootloader. However, instead of waiting for a SAM-BA connection, it will now detect the boot file in the DataFlash, load it into the internal RAM of the SAM9260 CPU and start it. In turn the DataFlash bootloader will initialize some board specific hardware, most notably the SDRAM. In a last step 512 kBytes are transferred from serial DataFlash location 0x8000 to memory location 0x20000000 (start of SDRAM) and executed.

Now our application code is running and should produce some output on the terminal emulator window.

```
SAM Internet Radio 1.0.2 - Nut/OS 4.3.3.0
66959703 bytes free
CPU Clock   : 90316800
Master Clock: 90316800
Register eth0...OK
Configure eth0...OK
IP Addr: 192.168.192.205
IP Mask: 255.255.255.0
IP Gate: 192.168.192.1
Start Responder...OK
Trying station entry 19
[CNCT 88.191.22.199:8000][RIDLE][PIDLE][CNCTD]GET / HTTP/1.0

Host: 88.191.22.199
```

```
User-Agent: WinampMPEG/2.7
Accept: */*
Icy-MetaData: 1
Connection: close
ICY 200 OK
icy-notice1:<BR>This stream requires <a
href="http://www.winamp.com/">Winamp</a><BR>
icy-notice2:SHOUTcast Distributed Network Audio Server/Linux v1.9.5<BR>
icy-name:FREQUENCE3 - www.frequence3.fr - No ads ! It's only HITS live from
Paris France !
ench Webradio
icy-genre:Top 40 Dance Pop Rock
icy-url:http://www.frequence3.fr
content-type:audio/mpeg
icy-pub:1
icy-metaint:32768
icy-br:192

Connected: 65876311 bytes free
[RSTOP][RSTART][REVT1][RRUN][PSTOP][PSTART][PEVT1][PRUN]
Meta="StreamTitle='Stereophonics - Dakota';StreamUrl='';"

Buffering.....................................................................
..............
.....................................................................................
..............
.....................................................................................
..............
.....................................................................................
..............
............................ 65854031 bytes free
```

After initializing the network interface, a connection to one of the pre-configured Internet radio stations is established. When enough MP3 data had been collected after a few seconds, you should be able to listen to the music on the headphones.

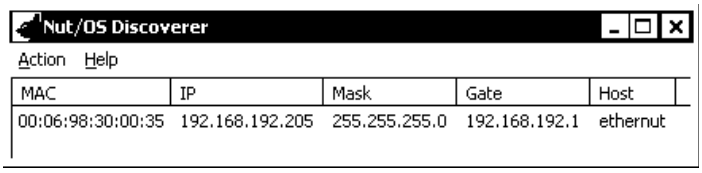
4.3 Solving Problems

If something went wrong, you may end up with dead target board. As explained earlier, the ROM bootloader will not listen to SAM-BA connections after a boot file had been stored in the DataFlash. Luckily the bootloader is able to remove itself from the DataFlash. Simply hold BP4 while pressing reset button BP1 on the AT91SAM9260-EK and wait a few seconds. When pressing reset again, you should be able to get a new SAM-BA connection.

You may alternatively try to boot from NAND Flash. In this case select the NAND Flash tab in the SAM-BA main window, execute the scripts "NandFlash Init" and "NandFlash Send Boot File" to upload Web_Radio_nandflash_at91sam9260ek.bin. Then send the file webradio.bin to NAND Flash address 0x20000.

If your network doesn't offer DHCP, the application will probably start with incorrect network settings and fail to connect the station. In this case run the Nut/OS Discoverer from the Windows Start Menu. This helpful utility had been installed with the Ethernut package.

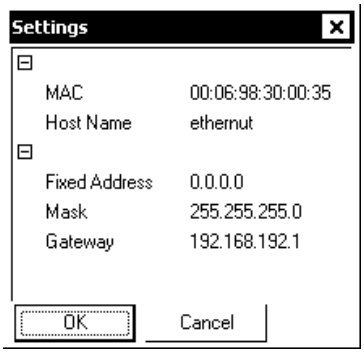
Figure 4-3. Nut/OS Discoverer Main Window



Selecting Scan from the main menu will broadcast a query for Nut/OS nodes to the local network and the Internet Radio should respond with its current IP settings.

Double click on the entry in the Discoverer's main window. This pops up a new window, which allows you to edit the current settings. You need to specify a fixed IP address (must be unique within your network) and the IP address of the gateway (Internet Access Router). Possibly the network mask needs to be adjusted also. Just in case you are running more than one radio, make sure that each gets a unique Ethernet MAC address.

Figure 4-4. Nut/OS Discoverer Settings Dialog



Click the OK button when done and confirm the following Update message box by pressing Yes. The new settings will be stored in the last sector of the serial DataFlash and will be loaded on the next reboot. The radio application should now come up with the proper IP configuration.

If you are behind a firewall which requires to use a HTTP Proxy, the only solution right now is to hard code this and possibly other configurations in the source code. Details were explained in chapter 2 and chapter 4 shows how to build a new binary from the sources.

4.4 Trying the User Interface

While the application is running, the LCD will show some status informations, like the firmware version or the current buffer contents. As soon as the buffer is filled and the player starts playing the stream, the name of the station appears on the first line while the second line shows the stream title fetched from the embedded metadata, if available.

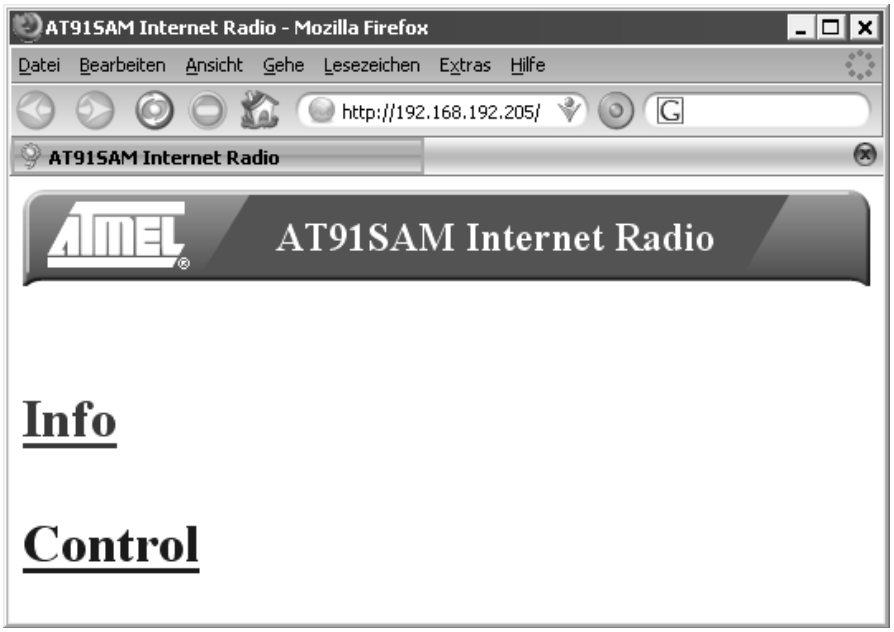
Pressing DOWN will lower the audio volume, pressing UP will increase it.

When pressing SELECT, the display will change to station selection mode, where buttons UP and DOWN can be used to walk through the list of radio stations. Pressing SELECT again will disconnect the radio from the current stream and establish a new connection to the selected station.

The radio application also provides a very simple web interface. Use any web browser to connect to the target board, by entering http:// followed by the IP address of the target board as a

URL. The current IP address is displayed in the terminal emulator during application start or can be retrieved with the Nut/OS Discoverer utility.

Figure 4-5. Internet Radio Web Interface



The Info link loads a self refreshing page with various status informations, while the Control link provides a simple demo, which shows how to control the radio via its web interface. This may provide a good starting point for your own enhancements.

4.5 Running on the AT91SAM7X256-EK

This works similar to the procedure described in the previous chapter. However, the evaluation board is delivered with a pre-loaded demo application. You must erase the SAM7 Flash Memory and reset the non-volatile flag bits to enable the SAM-BA boot loader. To do this, simply short circuit the jumper marked ERASE on the SAM7 board and power it up by plugging in the USB connector.

After a few seconds unplug the board from the USB again and put the ERASE jumper back to its original position. Plug the single row header socket of the Calypso Board on row C of the triple row header of the AT91SAM7X-EK. Make sure that all 32 pins are connected.

Connect the AT91SAM7X256-EK to your local Ethernet and to your PC's serial port (using the DBGU connector). Also plug in a headphone to the connector on the Calypso board marked OUTPUT.

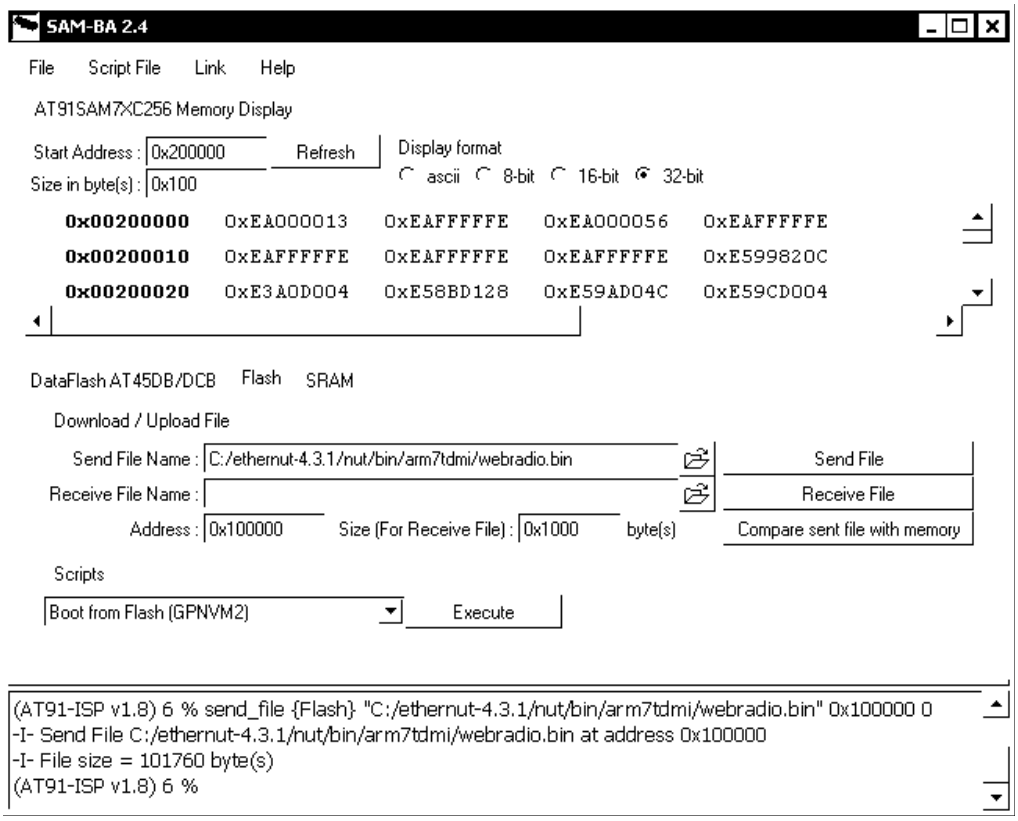
Start any terminal emulator (HyperTerminal or TeraTerm, if available) and set the serial port to 115200,8,n,1, all handshakes disabled.

When applying power to the AT91SAM7X-EK Board via the USB connector, the back-lit LCD will go on, but nothing will be displayed yet.

Start the SAM-BA v2.4 utility on the PC, select the connection \usb\ARM0 and the board AT91SAM7X256-EK and click the connect button.

Click on the Flash tab and press the browse button to choose a file to send to the board. Navigate to the nut/bin/arm7tdmi directory and select the file webradio.bin. Next press the button Send File.

Figure 4-6. SAM-BA SAM7X Flash Upload



Finally select the script "Boot from Flash (GPNVM2)" and click on the Execute button. This will disable the SAM-BA boot loader and instead activate the radio application when pressing the reset button. Note the hole on the Calypso board, which allows to access this most important push button.

The remaining steps are similar to those described in the previous chapter for the SAM9260, except that the SAM7X implementation lacks the web interface.

5. Building from Source Code

This chapter explains how to build the binaries from the source code. It assumed that you already installed ethernut-4.3.3.exe.

5.1 Application and Decoder Source Code

You may already have unpacked the radio application archive (samir-1.0.2-2.zip, see appendix) into the Nut/OS installation directory. The following list completes the description of the archive's contents:

nut\hxmp3
Source code of the Helix MP3 Decoder including the Hermite Resampler, to be added to the Nut/OS source tree, C:\ethernut-4.3.3\nut by default. Note that this software is distributed under a different license.

nut\include\hxmp3
The header files of the Helix MP3 Decoder and the Hermite Resampler, to be added to the include directory of Nut/OS source tree. Again, this comes with a different license.

nut\conf
Prepared configuration files for the AT91SAM7X and AT91SAM9 radio application. Copy these files to the conf directory of your Nut/OS source tree, C:\ethernut-4.3.3\nut\conf by default.

nutbld-sam9260-ram\hxmp3
Contains a prepared Makefile for building the Helix MP3 Decoder Library. This is part of the Nut/OS build tree for the AT91SAM9260, C:\ethernut-4.3.3\nutbld-sam9260-ram.

nutbld-sam7x-radio\hxmp3
Contains the Makefile for building the MP3 decoder for the AT91SAM7X. This is part of the build tree for the AT91SAM7X, C:\ethernut-4.3.3\nutbld-sam7x-radio.

nutbld-sam9260-ram\lib
Pre-build decoder library. You may use this one in case you don't succeed in building your own from the source code.

nutbld-sam7x-radio\lib
Pre-build decoder library for the AT91SAM7X.

nutapp-sam9260-ram\webradio
Source code of the radio application. This is part of the application tree for the AT91SAM9260, C:\ethernut-4.3.3\nutapp-sam9260-ram by default.

nutapp-sam7x-radio\webradio
Source code of the radio application. This is part of the application tree for the AT91SAM7X, C:\ethernut-4.3.3\nutapp-sam7x-radio.

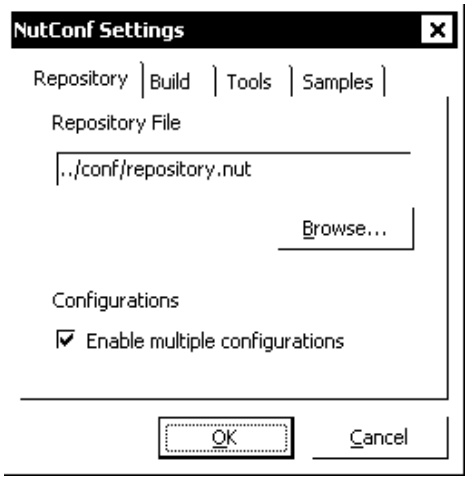
5.2 Tools Installation and Configuration

We need a C compiler and associated tools to create the binaries from the source code. Almost any toolchain distribution based on the free GNU Compiler Collection should do. We recommend YAGARTO, a free collection of tools, including the GNU Compiler for ARM and the Eclipse™ IDE. It is available for download at www.yagarto.de. Please follow the instructions provided with this package. Refer to the Atmel application note GNU-based Software Development on AT91SAM Microcontrollers, lit. no. 6310.

When done, start the Nut/OS Configurator from the Windows Start Menu. The Configurator first presents a file selection dialog, from which you can choose a pre-configuration for a specific board. Select at91sam9260-ek-radio.conf when building for the AT91SAM9260 or at91sam7x-ek-radio.conf for the AT91SAM7X target.

When the configuration has been loaded, choose Edit -> Settings from the main menu. A tabbed dialog box with four pages appears. On the first page titled Repository you need to specify the path of the repository file. The repository is a set of scripts, which lists all Nut/OS modules and their optional settings. Note that the Configurator is available for different platforms and prefers slashes to separate directory paths, not Windows-like backslashes.

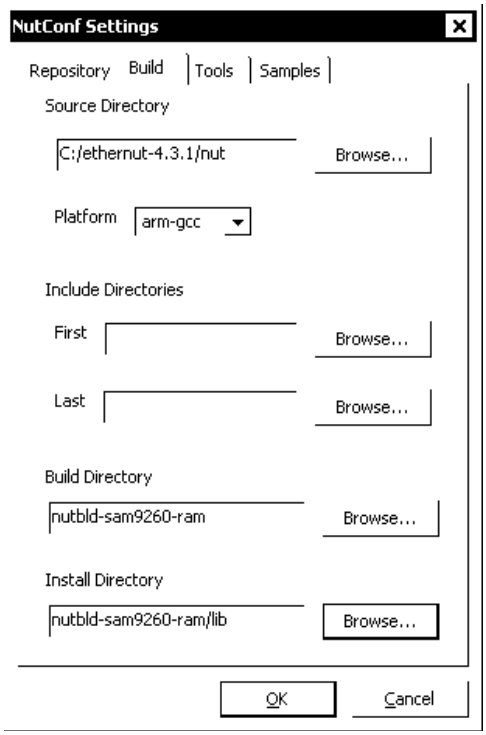
Figure 5-1. Nut/OS Configurator Settings Page 1



The settings on the second page specify the locations of the Nut/OS source code, header files, build directory and the final destination of the newly build libraries. Note that the Install Directory should be located within the Build Directory. Enter nutbld-sam9260-ram/ and nutbld-sam9260-ram/lib/ for the AT91SAM9260 or nutbld-sam7x-radio/ and nutbld-sam7x-radio/lib/ for the AT91SAM7X.

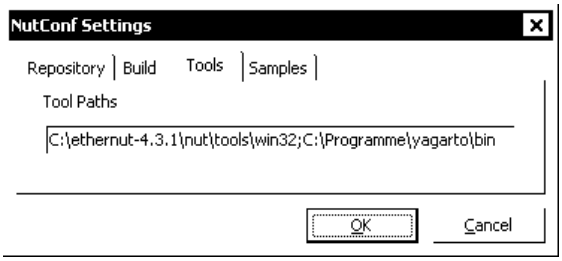
Select arm-gcc from the Platform drop-down list.

Figure 5-2. Nut/OS Configurator Settings Page 2



On the third page the build environment has to be specified. In contrary to the rule stated above, backslashes must be used here. It is very important that the Nut/OS tools directory comes first, followed by the path to the YAGARTO compiler binaries.

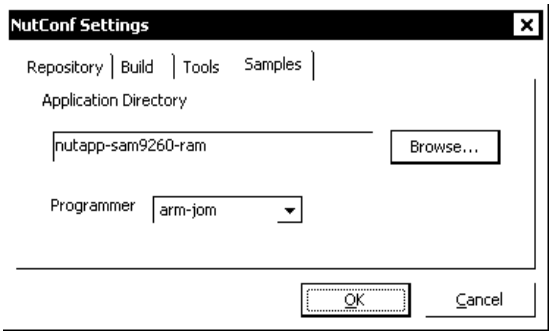
Figure 5-3. Nut/OS Configurator Settings Page 3



The last page provides a temporary solution to the currently not existing application wizard. The Configurator simply creates the directory specified on this page if it doesn't exist and copies the Nut/OS samples to this location. Enter nutapp-sam9260-ram for the AT91SAM9260 or nutapp-sam7x-radio and for the AT91SAM7X.

The programmer selection can be ignored. We use the SAM-BA tool instead.

Figure 5-4. Nut/OS Configurator Settings Page 4



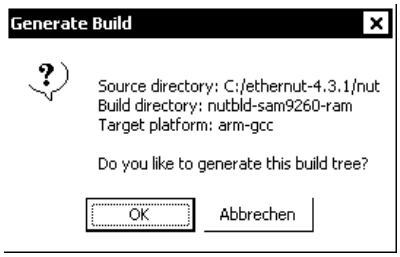
Click OK to confirm your settings.

5.3 Building the Nut/OS Libraries

After having modified the Configurator Settings in the previous chapter, it is recommended to reload the configuration file by selecting File -> Open from the Configurator's main menu. Again, select at91sam9260-ek-radio.conf for the AT91SAM9260 or at91sam7x-ek-radio.conf for the AT91SAM7X target.

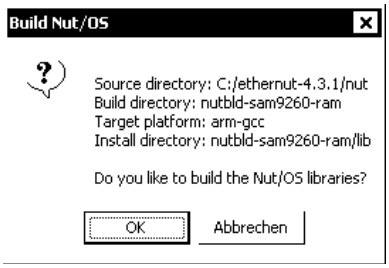
Now select Build -> Build Nut/OS from the main menu. In a first step this updates our build tree.

Figure 5-5. Nut/OS Build Tree Update



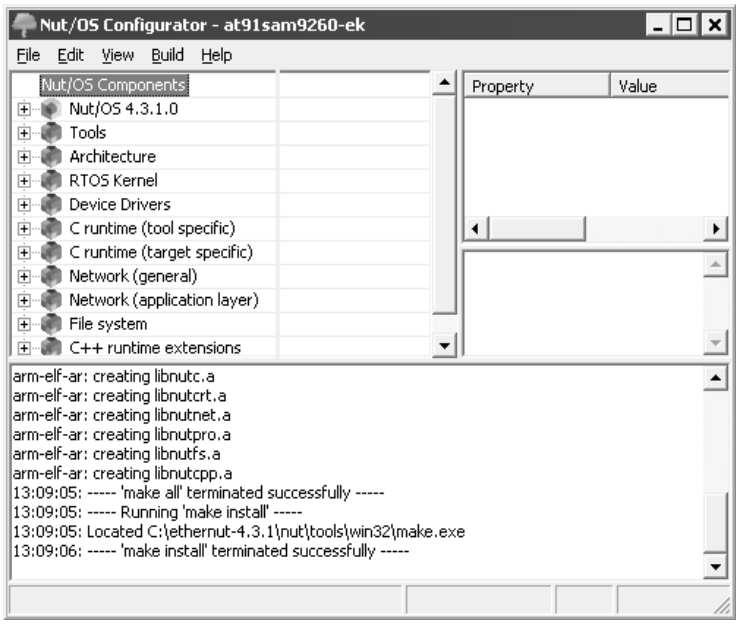
In a second step, the Nut/OS libraries are built.

Figure 5-6. Nut/OS Build



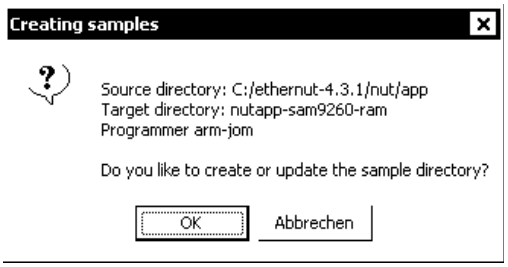
This step takes some time. The result is displayed in the lower text window of the Nut/OS Configurator.

Figure 5-7. Nut/OS Build Result



Finally select Build -> Create Sample Directory from the Configurator's main menu to update the application tree.

Figure 5-8. Nut/OS Application Tree Update



5.4 Building the Software MP3 Decoder Library

Open a command line window and change to the Helix Decoder subdirectory within your build tree. This directory contains just a Makefile, which compiles the Helix Source Code located in the Nut/OS source tree and copies the resulting library in the lib/ subdirectory of this build tree when entering 'make install'. However, make sure your PATH environment is correctly set. The following command sequence can be used when building for the AT91SAM9260 target with YAGARTO.

```
cd C:\ethernut-4.3.3\nutbld-sam9260-ram\hxmp3
SET PATH=C:\ethernut-4.3.3\nut\tools\win32;C:\Programme\yagarto\bin;%PATH%
make install
```

The following output is expected:

```
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Werror -Wstrict-prototypes -Wa,-ahlms=mp3dec.lst -DAT91SAM9260_EK -
I../include -I../../nut/include ../../nut/hxmp3/mp3dec.c -o mp3dec.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Werror -Wstrict-prototypes -Wa,-ahlms=mp3tabs.lst -DAT91SAM9260_EK -
I../include -I../../nut/include ../../nut/hxmp3/mp3tabs.c -o mp3tabs.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Werror -Wstrict-prototypes -Wa,-ahlms=buffers.lst -DAT91SAM9260_EK -
I../include -I../../nut/include ../../nut/hxmp3/buffers.c -o buffers.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Werror -Wstrict-prototypes -Wa,-ahlms=bitstream.lst -DAT91SAM9260_EK -
I../include -I../../nut/include ../../nut/hxmp3/bitstream.c -o bitstream.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Werror -Wstrict-prototypes -Wa,-ahlms=dct32.lst -DAT91SAM9260_EK -
I../include -I../../nut/include ../../nut/hxmp3/dct32.c -o dct32.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Werror -Wstrict-prototypes -Wa,-ahlms=dequant.lst -DAT91SAM9260_EK -
I../include -I../../nut/include ../../nut/hxmp3/dequant.c -o dequant.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Werror -Wstrict-prototypes -Wa,-ahlms=dqchan.lst -DAT91SAM9260_EK -
I../include -I../../nut/include ../../nut/hxmp3/dqchan.c -o dqchan.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Werror -Wstrict-prototypes -Wa,-ahlms=huffman.lst -DAT91SAM9260_EK -
I../include -I../../nut/include ../../nut/hxmp3/huffman.c -o huffman.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Werror -Wstrict-prototypes -Wa,-ahlms=hufftabs.lst -DAT91SAM9260_EK -
I../include -I../../nut/include ../../nut/hxmp3/hufftabs.c -o hufftabs.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Werror -Wstrict-prototypes -Wa,-ahlms=imdct.lst -DAT91SAM9260_EK -
I../include -I../../nut/include ../../nut/hxmp3/imdct.c -o imdct.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Werror -Wstrict-prototypes -Wa,-ahlms=scalfact.lst -DAT91SAM9260_EK -
I../include -I../../nut/include ../../nut/hxmp3/scalfact.c -o scalfact.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Werror -Wstrict-prototypes -Wa,-ahlms=stproc.lst -DAT91SAM9260_EK -
I../include -I../../nut/include ../../nut/hxmp3/stproc.c -o stproc.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Werror -Wstrict-prototypes -Wa,-ahlms=subband.lst -DAT91SAM9260_EK -
I../include -I../../nut/include ../../nut/hxmp3/subband.c -o subband.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Werror -Wstrict-prototypes -Wa,-ahlms=trigtabs.lst -DAT91SAM9260_EK -
I../include -I../../nut/include ../../nut/hxmp3/trigtabs.c -o trigtabs.o
arm-elf-gcc -x assembler-with-cpp -c -mcpu=arm9 -Wa,-amhls=asmpoly_gcc.lst
-DAT91SAM9260_EK -I../include -I../../nut/include
../../nut/hxmp3/asmpoly_gcc.S -o asmpoly_gcc.o
arm-elf-ar rsv libhxmp3.a mp3dec.o mp3tabs.o buffers.o bitstream.o dct32.o
dequant.o dqchan.o huffman.o hufftabs.o imdct.o scalfact.o stproc.o
subband.o trigtabs.o asmpoly_gcc.o
arm-elf-ar: creating libhxmp3.a
a - mp3dec.o
a - mp3tabs.o
a - buffers.o
a - bitstream.o
a - dct32.o
a - dequant.o
```

```
a - dqchan.o
a - huffman.o
a - hufftabs.o
a - imdct.o
a - scalfact.o
a - stproc.o
a - subband.o
a - trigtabs.o
a - asmpoly_gcc.o
cp libhxmp3.a ../lib/libhxmp3.a
```

5.5 Building the Internet Radio Application

After having successfully built the decoder library, change to the web radio subdirectory within the Nut/OS application sample directory and again run 'make install'.

```
cd C:\ethernut-4.3.3\nutapp-sam9260-ram\webradio
make install
```

Here's the expected result:

```
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Wstrict-prototypes -fverbose-asm -Wa,-ahlms=webradio.lst -DAT91SAM9260_EK
-IC:/ethernut-4.3.3/nutbld-sam9260-ram/include -IC:/ethernut-
4.3.3/nut/include  webradio.c -o webradio.o
crurom -r -ourom.c html
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Wstrict-prototypes -fverbose-asm -Wa,-ahlms=urom.lst -DAT91SAM9260_EK -
IC:/ethernut-4.3.3/nutbld-sam9260-ram/include -IC:/ethernut-
4.3.3/nut/include  urom.c -o urom.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Wstrict-prototypes -fverbose-asm -Wa,-ahlms=httpserv.lst -DAT91SAM9260_EK
-IC:/ethernut-4.3.3/nutbld-sam9260-ram/include -IC:/ethernut-
4.3.3/nut/include  httpserv.c -o httpserv.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Wstrict-prototypes -fverbose-asm -Wa,-ahlms=config.lst -DAT91SAM9260_EK -
IC:/ethernut-4.3.3/nutbld-sam9260-ram/include -IC:/ethernut-
4.3.3/nut/include  config.c -o config.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Wstrict-prototypes -fverbose-asm -Wa,-ahlms=station.lst -DAT91SAM9260_EK -
IC:/ethernut-4.3.3/nutbld-sam9260-ram/include -IC:/ethernut-
4.3.3/nut/include  station.c -o station.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Wstrict-prototypes -fverbose-asm -Wa,-ahlms=receiver.lst -DAT91SAM9260_EK
-IC:/ethernut-4.3.3/nutbld-sam9260-ram/include -IC:/ethernut-
4.3.3/nut/include  receiver.c -o receiver.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Wstrict-prototypes -fverbose-asm -Wa,-ahlms=shoutcast.lst -DAT91SAM9260_EK
-IC:/ethernut-4.3.3/nutbld-sam9260-ram/include -IC:/ethernut-
4.3.3/nut/include  shoutcast.c -o shoutcast.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Wstrict-prototypes -fverbose-asm -Wa,-ahlms=player.lst -DAT91SAM9260_EK -
IC:/ethernut-4.3.3/nutbld-sam9260-ram/include -IC:/ethernut-
4.3.3/nut/include  player.c -o player.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Wstrict-prototypes -fverbose-asm -Wa,-ahlms=mp3player.lst -DAT91SAM9260_EK
-IC:/ethernut-4.3.3/nutbld-sam9260-ram/include -IC:/ethernut-
4.3.3/nut/include  mp3player.c -o mp3player.o
arm-elf-gcc -c -mcpu=arm9 -Os -mthumb-interwork -fomit-frame-pointer -Wall
-Wstrict-prototypes -fverbose-asm -Wa,-ahlms=tlv320dac.lst -DAT91SAM9260_EK
-IC:/ethernut-4.3.3/nutbld-sam9260-ram/include -IC:/ethernut-
4.3.3/nut/include  tlv320dac.c -o tlv320dac.o
arm-elf-gcc webradio.o urom.o httpserv.o config.o station.o receiver.o
shoutcast.o player.o mp3player.o tlv320dac.o -mcpu=arm9 -nostartfiles -
TC:/ethernut-4.3.3/nut/arch/arm/ldscripts/at91sam9260_ram.ld -Wl,-
Map=webradio.map,--cref,--no-warn-mismatch -LC:/ethernut-4.3.3-rc10/nutbld-
sam9260-ram/lib -Wl,--start-group C:/ethernut-4.3.3/nutbld-sam9260-
```

```
ram/lib/nutinit.o -lnutpro -lnutnet -lnutfs -lnutos -lnutdev -lnutarch -
lnutcrt -lhxmp3  -Wl,--end-group -o webradio.elf
arm-elf-objcopy -O ihex webradio.elf webradio.hex
arm-elf-objcopy -O binary webradio.elf webradio.bin
cp webradio.hex C:/ethernut-4.3.3/nut/bin/arm9/webradio.hex
cp webradio.bin C:/ethernut-4.3.3/nut/bin/arm9/webradio.bin
```

In the last line the resulting binary is copied to bin/arm9 inside the Nut/OS source tree.

6. Calypso Add-on Board

The Calypso add-on is an extension to the AT91SAM9260-EK and the AT91SAM7X-EK from Atmel and provides a TLV320AIC23B stereo audio codec, an LCD, three push buttons and two audio connectors. The Eagle CAD files are available in the appendices.

6.1 Jumper Configuration

Jumper JP1 selects the DAC clock source, either PCK1 output of the AT91SAM9260 or the PCK2 output of the AT91SAM7X when pin 1 and 2 are shortened, or the on-board crystal when pin 2 and 3 are shortened. The latter is the default used with the radio application.

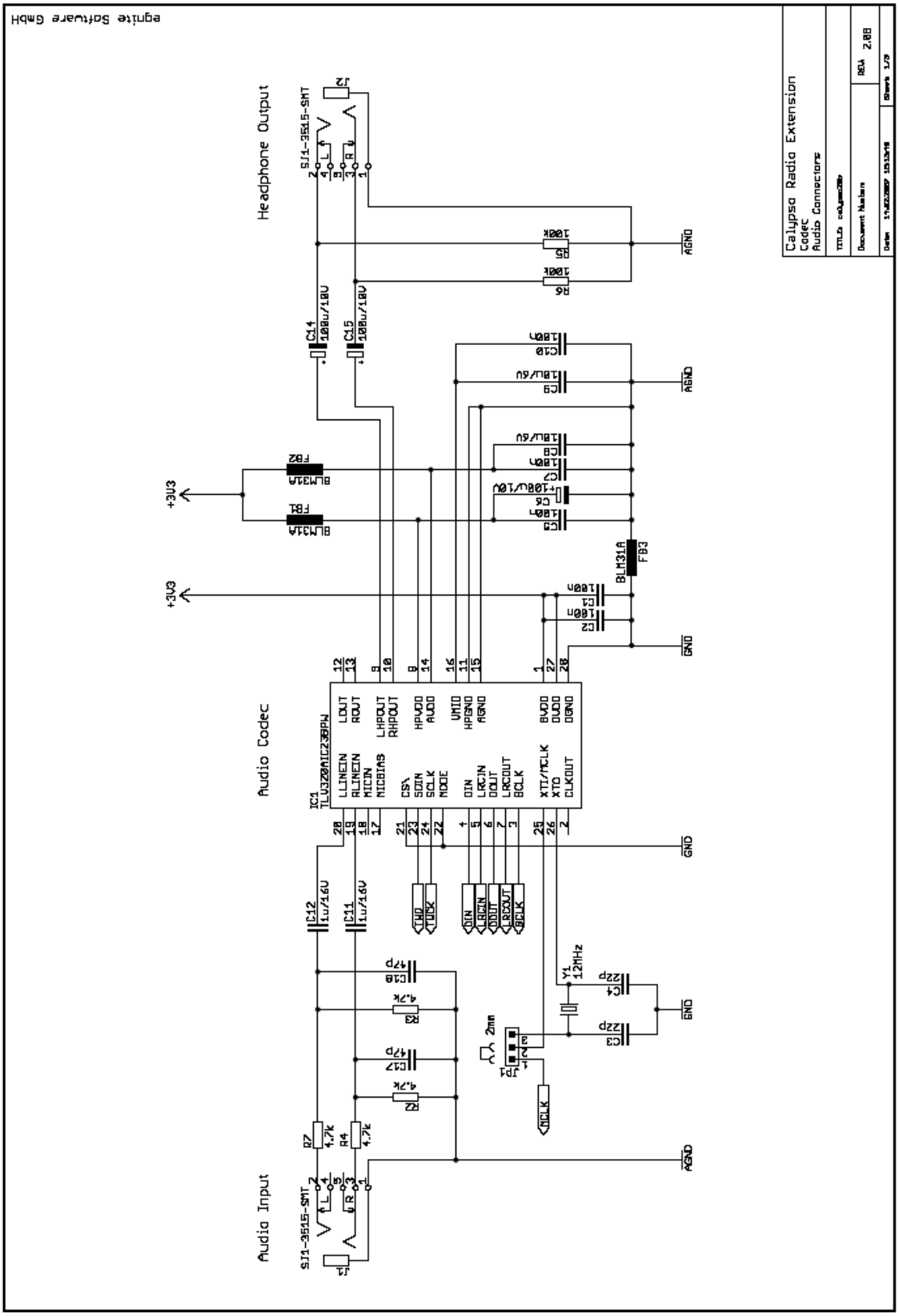
6.2 Connectors

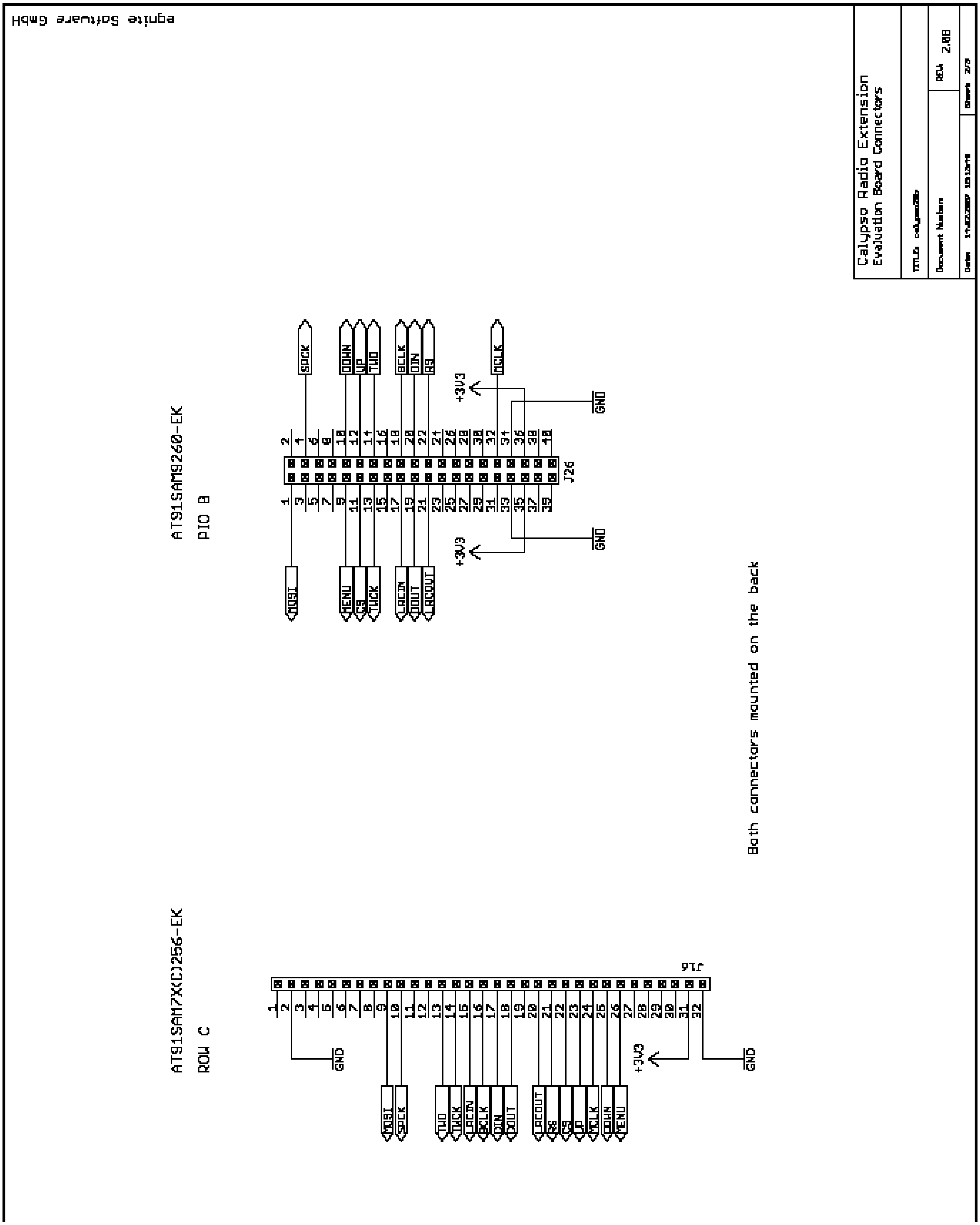
J1, a 2.5 mm jack, provides a stereo audio input, which however is not used in this application. The 2.5 mm audio jack J2 offers stereo output and can directly drive an ear-phone or the line-input of an audio amplifier.

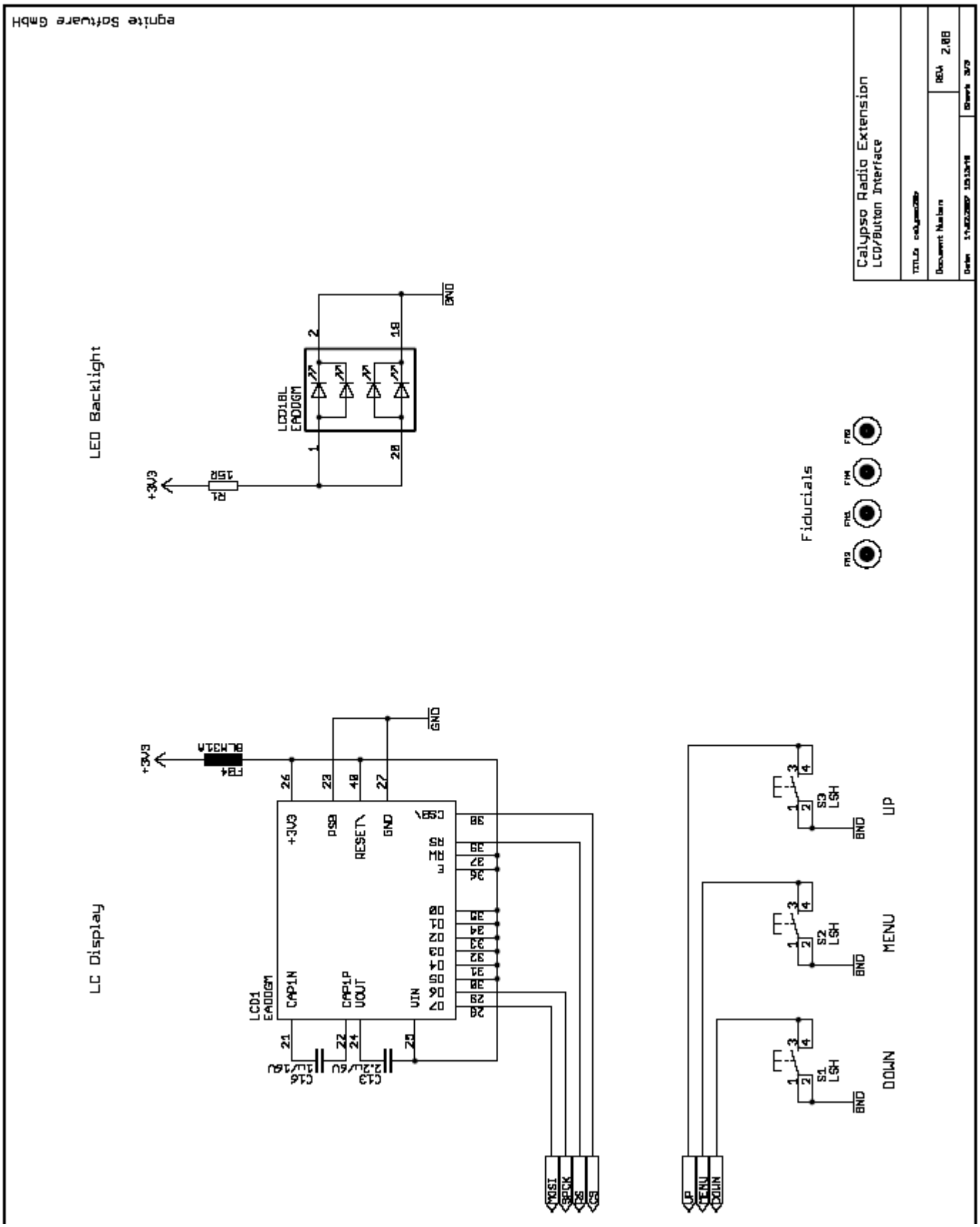
Two header sockets J16 and 126 on the back side are attached to row C of the AT91SAM7X256 expansion connector or to the PIO B connector of the AT91SAM9260 Board, respectively.

Signal Name	**AT91SAM7X Extension**		**AT91SAM9260 Extension**	
LCD MOSI	J16-C9	SPI0_MOSI	J26-1	SPI1_MOSI
LCD SPCK	J16-C10	SPI0_SPCK	J26-4	SPI1_SPCK
LCD RS	J16-C21	PB23	J26-22	PB20
LCD CS	J16-C22	PB24	J26-11	PB11
BUTTON UP	J16-C23	PB29	J26-12	PB10
BUTTON SELECT	J16-C26	PB28	J26-9	PB9
BUTTON DOWN	J16-C25	PB27	J26-10	PB8
DAC SDIN	J16-C13	TWD	J26-14	PB12
DAC SCLK	J16-C14	TWCK	J26-13	PB13
DAC LRCIN	J16-C15	TF	J26-17	TF0
DAC BCLK	J16-C16	TK	J26-18	TK0
DAC DIN	J16-C17	TD	J26-20	TD0
DAC DOUT	J16-C18	RD	J26-19	RD0
DAC LRCOUT	J16-C20	RF	J26-21	RF0
DAC MCLK	J16-C24	PCK2	J26-31	PCK1
+3.3V	J16-C31	+3.3V	J26-35 J26-36	+3.3V
GND	J16-C2 J16-C32	GND	J26-33 J26-34	GND

6.3 Schematics




egnite Software GmbH
Calypso Radio Extension
Evaluation Board Connectors
REV 2.00
AT91SAM9260-EK
PIO B
J26
SPCK
DOWN
UP
TWD
BCLK
DIN
A9
MCLK
+3V3
GND
MOSI
MENU
CS
TWCK
LACIN
DOUT
LACOUT
Both connectors mounted on the back
AT91SAM7X(C)256-EK
ROW C
J16


egnite Software GmbH
Calypso Radio Extension
LCD/Button Interface
LED Backlight
LCD18L
EA DOGM
+3V3
R1
150R
GND
Fiducials
FID3
FID2
FID1
FID0
LC Display
LCD1
EA DOGM
+3V3
FB4
BLM31A
CAP1N
CAP1P
VOUT
VIN
PSB
RESET
GND
C16
1µ/16V
C13
2,2µ/6V
MOSI
SPCK
CS
UP
MENU
DOWN
S3
S2
S1
LSH

7. Appendix

7.1 Downloading the Required Tools

7.1.1 GNU Toolchain

The YAGARTO toolchain is available at

http://www.yagarto.de/

7.1.2 Nut/OS

The latest Nut/OS release can be downloaded at

http://ethernut.de/en/download/

7.2 Source Code and Add-On Hardware

The complete source code of the Internet Radio Application including the ported MP3 decoder is available in the archive. It further contains the Eagle CAD files of the schematic, the bill of material and the layout of the add-on board.

7.3 References

http://www.mikrocontroller.net/articles/ARM_MP3/AAC_Player
ARM MP3/AAC Player based on AT91SAM7S256

Atmel Application Note 6250
Using Open Source Tools for AT91SAM7 Cross Development

http://www.xiph.org/
Icecast homepage.

http://www.shoutcast.com/
SHOUTcast homepage.

8. Revision History

Table 8-1.

Document Reference	Comments	Change Request Ref.
6318A	First issue.	

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-Yvelines
Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Operations

Memory
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

Microcontrollers
2325 Orchard Parkway
San Jose, CA 95131, USA
Tel: 1(408) 441-0311
Fax: 1(408) 436-4314

La Chantrerie
BP 70602
44306 Nantes Cedex 3, France
Tel: (33) 2-40-18-18-18
Fax: (33) 2-40-18-19-60

ASIC/ASSP/Smart Cards
Zone Industrielle
13106 Rousset Cedex, France
Tel: (33) 4-42-53-60-00
Fax: (33) 4-42-53-60-01

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Scottish Enterprise Technology Park
Maxwell Building
East Kilbride G75 0QR, Scotland
Tel: (44) 1355-803-000
Fax: (44) 1355-242-743

RF/Automotive
Theresienstrasse 2
Postfach 3535
74025 Heilbronn, Germany
Tel: (49) 71-31-67-0
Fax: (49) 71-31-67-2340

1150 East Cheyenne Mtn. Blvd.
Colorado Springs, CO 80906, USA
Tel: 1(719) 576-3300
Fax: 1(719) 540-1759

Biometrics
Avenue de Rochepleine
BP 123
38521 Saint-Egreve Cedex, France
Tel: (33) 4-76-58-47-50
Fax: (33) 4-76-58-47-60

Literature Requests
www.atmel.com/literature