Flash Card Audio Player for Head End Unit

By Lindsay Meek

1. ABSTRACT

This project uses an LPC213x to implement a simple digital audio player, capable of playing music tracks from a removable flash card. The player is controlled using the component-bus interface from a car audio head-end unit, and provides a line out stereo signal, suitable for plugging into the auxiliary input connector of the head-end unit.

2. INTRODUCTION

The storage capacity and cost of miniature flash memory cards have improved greatly in recently years, resulted in these cards becoming a popular method for transporting large amounts of data.

This project makes use of ubiquitous multimedia cards (MMC) to transport lossless audio tracks, which can be played back inside a car from the head-end unit. This can be a simple and convenient method for quickly creating albums and listening to them without needing to burn a CD.

Most head-end units come with the ability to control an external CD changer using a serial communications bus, and this bus was adopted to communicate with flash card player. The flash card player emulates the CD changer by organizing the tracks into subdirectories, and makes each directory appear as a CD to the head-end unit.

The flash card player receives commands from the head-end unit such as next track, next CD, play, stop etc, and uses these commands to sequence its own internal playback. The playback mechanism uses a high-speed sampling interrupt feeding a DAC to generate the audio signal, and this is fed back to the head-end unit for amplification to the car speakers.

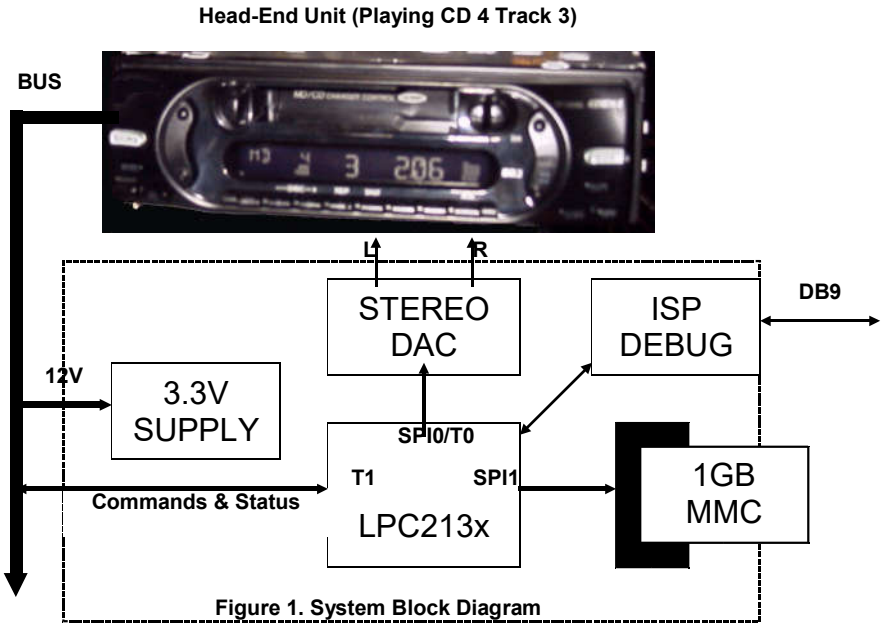


Figure 1. System Block Diagram

3. DESIGN DETAIL

3.1 Resource Requirements

TOOLCHAIN	GCC within Keil μVision
RAM	11K
ROM	24K
CPU CLOCK	60 MHz
UARTS	1 (For In-circuit programming and debugging)
SPI	2
I^2C	1
TIMERS	2
INPUT CAPTURES	1
GPIO USED	15 (Include Peripherals)
MODES	Thumb (Background), ARM (IRQs)
INTERRUPTS	2
TARGETS	LPC2132, LPC2134, LPC2136, LPC2138 LPC2131 with a smaller DAC sample buffer

3.2 System Architecture

As shown in figure 1, the system consisted of the flash player connected to the head-end unit using the bus interface cable. This bus interface cable provided both a power and command interface to the flash player, and the player generated audio signals suitable for connecting to the auxiliary input of the head-end unit.

The command interface chosen for this project was unilink [1], which was convenient as there was sufficient documentation on it freely available on the web, and cables & head-end units were readily available.

The digital audio playback system consisted of a multimedia card interface and FAT file system, which streamed uncompressed ‘WAV’ format audio files at high speed to the DAC, where the analog audio signals were generated for the head-end unit. The first prototype used the internal DAC of the LPC213x, whereas the second one used a dedicated high-resolution stereo DAC.

3.3 Hardware Architecture

All of the interfaces on the player were based around the serial bus peripherals. The slower SPI0 peripheral on the LPC213x was allocated to the multimedia interface, allowing the faster 16-bit SPI1 peripheral to be allocated to the time-critical sample data stream going to the stereo DAC.

The head-end protocol also ideally required an SPI peripheral, however the LPC213x was one short, requiring an interrupt-driven bit-banging method to be used.

3.3.1 Flash Card Reader

The multimedia card interface was configured into SPI mode [2], which makes it appear as a standard SPI peripheral with the SPI configuration of CPHA=1 and CPOL=1.

In order to reduce read-latency during audio playback, a maximum baud rate of 2.5Mbits/s was used. The LPC213x had the ability to go faster, however the MMC connector wiring on the first prototype prevented the faster rates from working reliably.

The PCB track wiring on the second prototype should allow for an even faster data rate, however 2.5Mbits/s was found to be sufficient to stream stereo WAV files at 44100 samples/s without gaps.

3.3.2 Digital to Analog Conversion

The audio signal generation was performed by the internal 10-bit mono DAC on the first prototype, and by an external 16-bit stereo DAC on the second prototype.

The stereo DAC chosen was a TLV320DAC23, as it supported multiple sample data formats including the synchronous serial frame format, which is native to the LPC213x. Unfortunately, it turned out the LPC213x only supported 16-bits per frame, whereas for correct stereo playback 32-bits per frame was required.

This required the use of left-justified mode as shown below, with the LPC213x generating the LRCIN signal using a toggling output pin every 2nd sample.

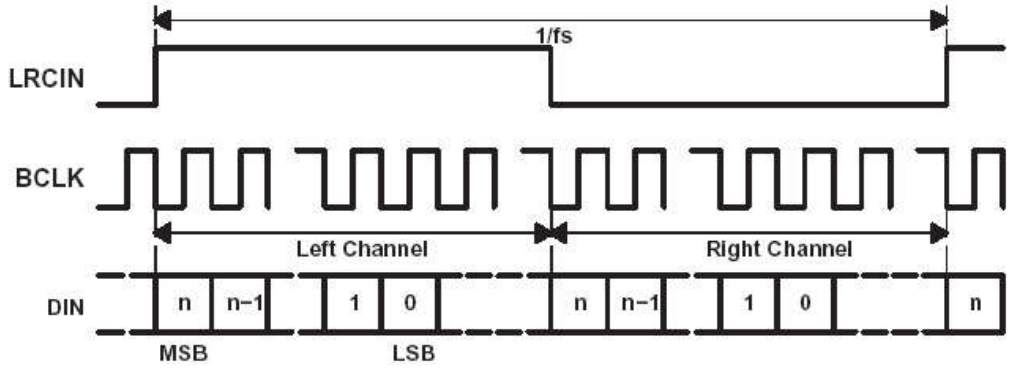


Figure 2. Left-justified audio data format

In addition to the audio sample data, the DAC used a standard I^2C interface for controlling its operation. The I^2C0 peripheral on the LPC213x was used to allow the software to set up the DAC to generate a line out signal, based on 16-bit left-justified sample data.

3.3.3 Head-End Communications Bus

The communications bus from the head-end consisted of a 62kb/s SPI-like protocol with additional signaling lines [3]. This document did not describe the electrical interface in great detail, however upon further investigation it was found that the signals were all 5V TTL compatible.

This allowed them to be connected directly to the LPC213x 5V tolerant I/O pins, eliminating additional voltage level conversion circuitry. In addition, only ‘bus clock’, ‘bus data’ and ‘bus reset’ were required to implement the bus interface, allowing the ‘bus on’ and ‘ready’ lines to be ignored.

A lack of additional SPI peripherals on the LPC213x required the communications interface to be implemented using a software-based bit-banging approach. This was assisted by Timer1’s edge-triggered capture interrupt, which intercepted the first bus clock edge in a byte. In addition the compare function of Timer1 was used to detect gaps between packets.

The bit-banging approach relied on the 1ms delay between bytes to perform the additional protocol processing.

3.3.4 Control Power Supply

The control power supply made use of the 12V rail supplied on the bus interface cable from the head-end unit. This was regulated down to 3.3V using a LM1086 linear regulator. The current consumption of the various devices and regulator dissipation was estimated to be

MMC	50 mA
LPC2138	40 mA
TLV320DAC23	15 mA
TOTAL	95 mA
Pd	~0.9W
Temperature Rise	~72 C (80C/W TO-220)
Max. Ambient	~40 C

When operating in a car, it is recommended that a ~20C/W heatsink be fitted to the regulator.

3.4 Software Architecture

The software system architecture can be broadly divided into three concurrent processes. In the background at the lowest priority, a thread managed the track playback and file system I/O. In the foreground at the highest priority, a periodic interrupt loaded the DAC with sample data at the designated sampling rate. Also in the foreground at a reduced priority, was an interrupt responsible for handling incoming commands from the head-end and sending appropriate responses if necessary.

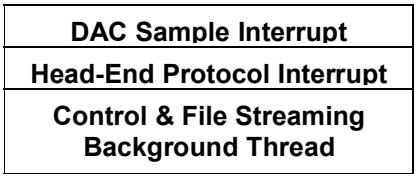


Figure 3. Relative Priority of Concurrent Software Processes

3.4.1 Digital To Analog Sample Interrupt

The DAC sampling interrupt was the highest priority task in the system, and was responsible for loading the DAC with the sample data. It used Timer0 configured as a resetting counter, with the period set to the sampling period for mono playback, and half the sampling period for stereo playback. This generated an interrupt via VIC vector 0, giving it the highest possible hardware priority.

The interrupt service routine was responsible for loading the DAC with 16-bit signed sample values from a sample buffer, using the SPI1 peripheral operating in master mode with a 16-bit word length.

The sample buffer was double buffered to provide glitch-free playback, meaning there was a second buffer that was being loaded by the background file streaming thread whilst the DAC interrupt was streaming from its buffer.

When the DAC interrupt had completed streaming one buffer, it automatically swapped over to the other sample buffer, informing the background thread to start filling the completed buffer with new samples.

In the case of stereo playback, the DAC interrupt ran at twice the sampling frequency, and generated the toggling left/right signal for the external DAC.

In addition to feeding the DAC with sample data, the DAC interrupt also kept track of elapsed time. It did this by incrementing a ticker after each sample buffer finished, which was used to calculate elapsed time based on the sampling rate and the buffer size.

3.4.2 Head End Protocol Interrupt

The head-end protocol interface consisted of a SPI-like slave [1], with a single bi-directional data line, with a 1ms gap between bytes, and a 50ms+ gap between packets.

As no SPI peripheral was available on the LPC213x to provide the low-level byte interface, a bit-banging software approach was used. This operated using Timer1, with a combination of edge capture on the bus clock line and periodic counters.

To detect the gap between packets from the head-end, and resynchronize the receiver state machine, a periodic 0.5ms interrupt was used. If no clock edges were captured during the previous period, then a counter was incremented and used to detect the gap if the counter exceeded a threshold value corresponding to the gap. If any edges were captured, then the counter was reset and the process continued until the edges stopped.

The byte receiver used the edge capture as an interrupt trigger to synchronize with the first clock transition of a new byte. This activated the interrupt service routine, which then used a tight loop to receive the eight SPI bits at 62kb/s. The received byte was passed onto a state machine for packet disassembly and command decoding.

If a command from the head-end required a response from the player, then a byte transmit interrupt was used to send an outgoing response packet. The transmit interrupt worked essentially the same was as the byte receiver interrupt, except the initial clock edge polarity was inverted and a buffer was used to contain the response.

Player commands from the head-end such as play, stop, next etc. were communicated to the background thread by setting a callback function vector with the requested function. The background thread polled the vector and detected a non-null function, at which time it dispatched the required function and cleared the vector again.

One final problem that arose with using an interrupt to process the head-end commands was that it introduced jitter into the high-speed DAC interrupt. To overcome this, the head-end interrupt was modified to allow the DAC interrupt to nest over the top of it using the technique described in [9].

3.4.3 Background Thread

The background thread contained the interface for communicating with the multimedia card interface, and controlling the player.

At the lowest level, the multimedia card interface used the SPI0 peripheral operating in a polled mode, so as to not interfere with the DAC interrupt. An

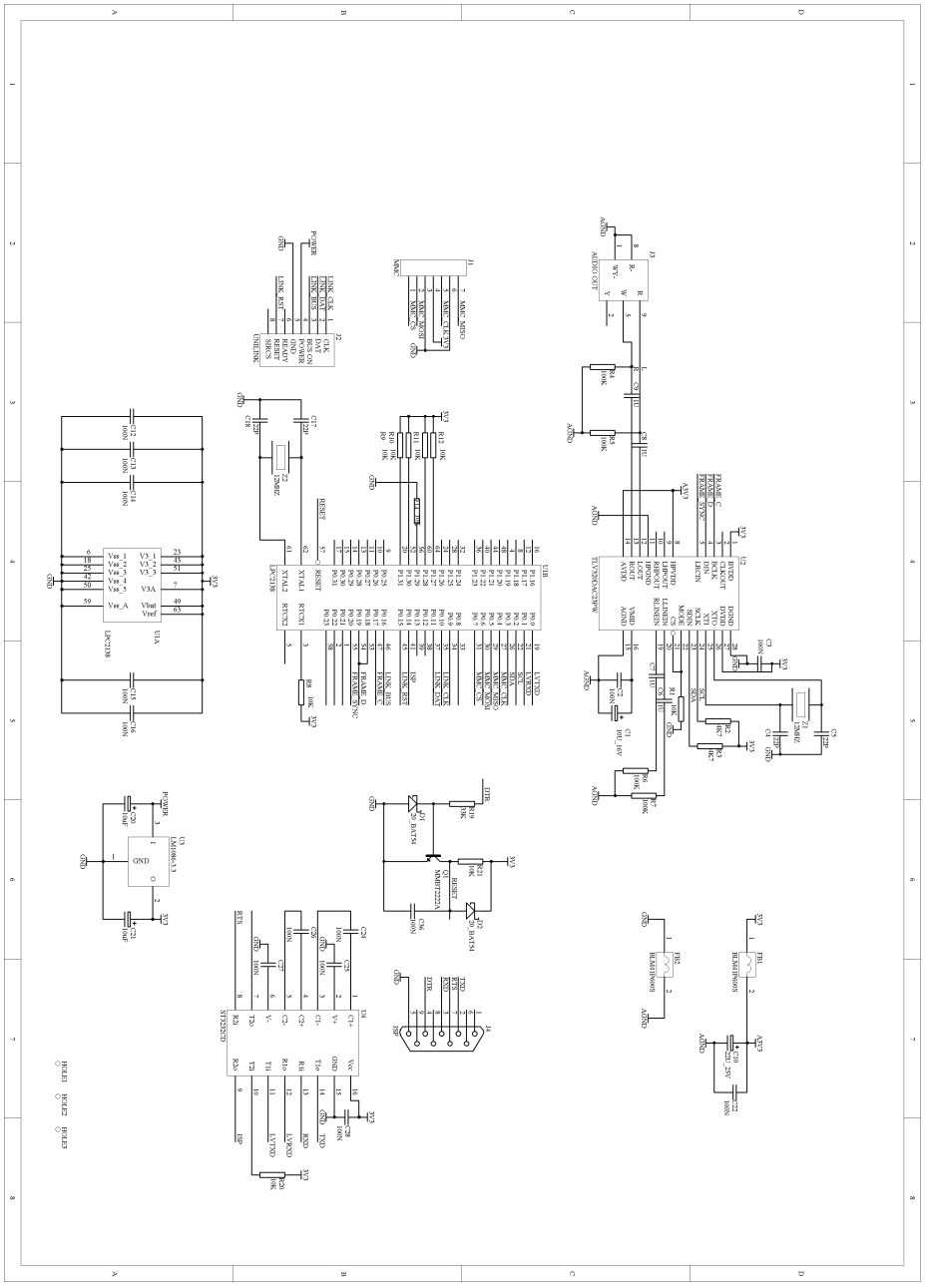
optimized read sector routine was provided to interface to the higher-level file system routines.

The multimedia file system routines were based on a FAT16 file system library [4], as this was the file system of choice for cards larger than 32 MB. This library was optimized for high-speed streaming playback, by modifying the open() and read() functions so that the FAT table was pre-scanned at the start of a track to determine the number of consecutive clusters. This had the effect of reducing the number of times the FAT needed to be rescanned during sequential accesses in the file. In addition, the read() routine was further optimized to reduce the number of memory-to-memory copies required.

Above the file system were the track scanning and streaming algorithms. The track scanners consisted of some simple routines that could count directories, and scan within them for files with a '.WAV' extension. The main streaming routine consisted of a WAV format [5] file player, which interfaced with the DAC interrupt described previously in section 3.4.1.

In addition to the uncompressed WAV format, a number of 2:1 lossless decompression algorithms were also attempted on the player [6,7,8]. It turned out that the CPU load imposed by the DAC interrupt was too great for these decompression algorithms to operate at the required sampling rate. The CPU load should be able to be reduced significantly by better utilizing the FIFO of SPI1, combined with some suitable external logic to interface to the DAC.

4. SCHEMATIC



5. PROJECT PHOTOS

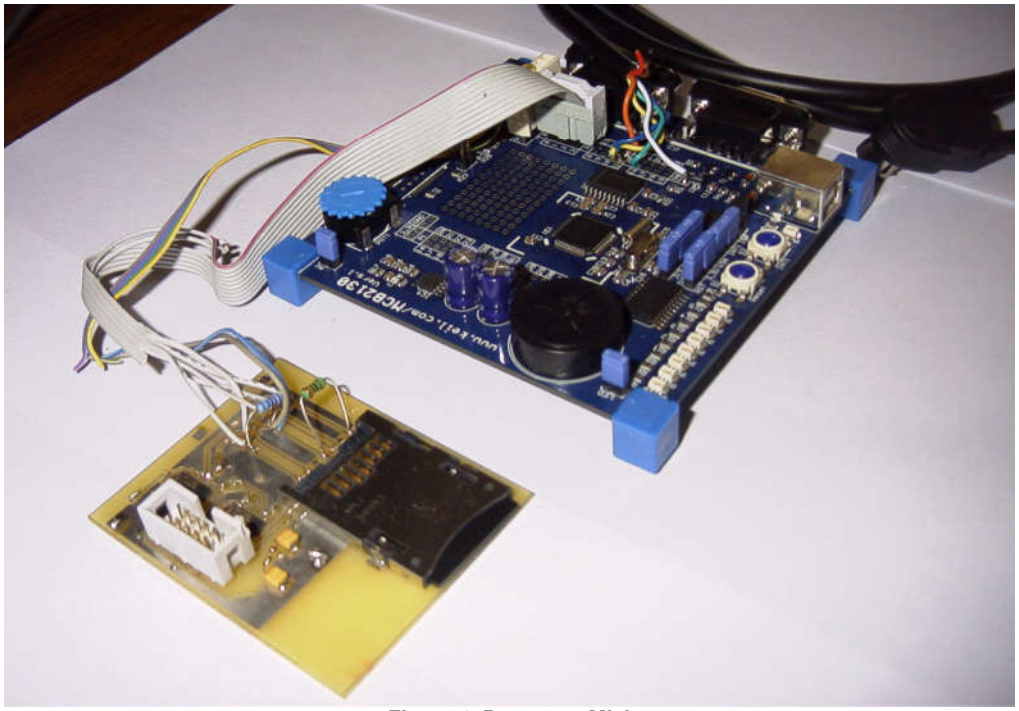


Figure 4. Prototype Mk I
Bottom: MMC. Top: Bus cable.
Audio output is via LPC2138's DAC and EVM onboard audio stage.

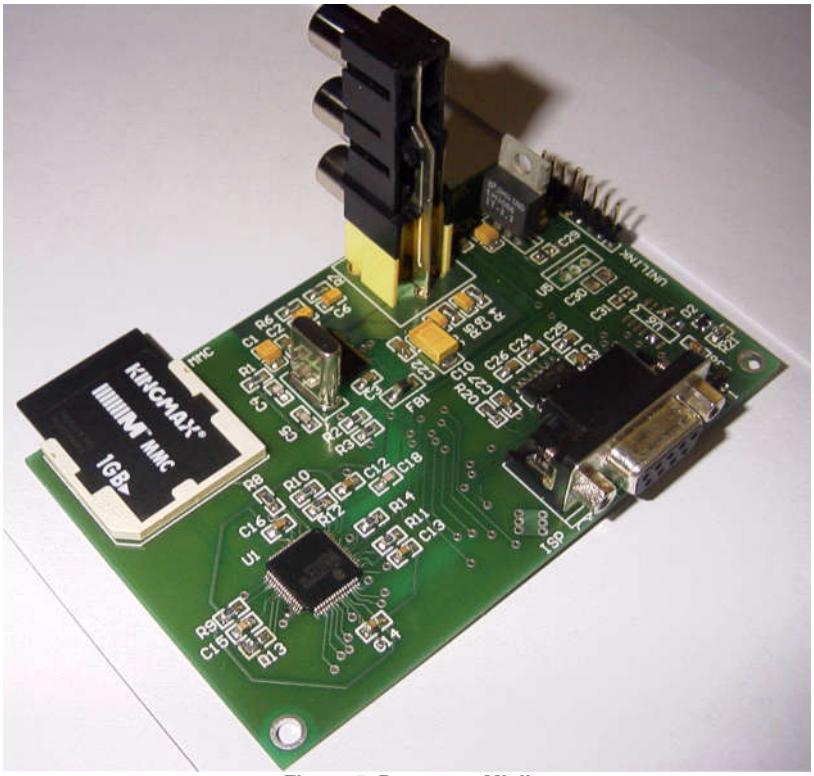


Figure 5. Prototype Mk II
Bottom: ARM and MMC.
Middle: external Stereo DAC, audio-out and ISP header.
Top: Supply and Bus cable connector.

6. REFERENCES

[1] Unilink head-end protocol
http://cleggy.umcus.org/protocol.htm

[2] Multimedia card interface specification
http://www.mmca.org/compliance/buy_spec/MMCA_System_SummaryV41.pdf

[3] GNU head-end protocol interface
http://gnunilink.sourceforge.net/

[4] Apple II IDE/ATA interface
http://s.guillard.free.fr/Apple2IDE/

[5] WAV file format
http://www.sonicspot.com/guide/wavefiles.html

[6] WAVPACK lossless audio codec
http://www.wavpack.com/

[7] TTA lossless audio codec
http://www.true-audio.com/

[8] FLAC lossless audio codec
http://flac.sourceforge.net/

[9] LPC213x Nested Interrupts
http://www.semiconductors.philips.com/acrobat_download/applicationnotes/AN10381_1.pdf