Procyon MP3

A Hard-Disk Based MP3 Player

Pascal Stang
EE281
12/13/00

1. Project Introduction

Procyon MP3 is a hardware MPEG 1 Layer 3 (MP3) audio player which supports user-controlled playback of MP3 files stored on a typical IDE hard disk connected to the device. With the emergence of MP3 as a popular standard for storing, playing, and exchanging music, droves of electronics and computer companies have introduced both portable and desktop MP3 player devices into the consumer market. The rise of MP3 has also produced dozens of personal engineering projects like this one. However, where companies must directly cater to their conceived or pre-conceived notions of the consumer market, hobbyists and engineers are free to explore MP3 player solutions that many companies would balk at producing and selling.

This MP3 player fits into precisely this situation. Most consumer electronics companies have focused on the portable audio market where several of MP3's inherent traits become strong selling points. Typical portable MP3 players store their audio content in non-volatile Flash memory. Devices that play from Flash can be hyper-efficient in power consumption, miniscule in size, and practically shock-proof and skip-proof since there are no moving parts in the playback system. They are, however, severely limited in playback time due to the expense of the Flash memory itself. Typical players selling for $250 are able to store only sixty minutes, or about a CD's worth, of music. Many music enthusiasts don't see the benefits of these tiny players when they still have to lug around they're current CD collection or laptop to keep the player full of new music. To address this issue, Procyon MP3 maintains an decent battery life and acceptable shock-resistance while using a standard laptop IDE hard-disk as the MP3 storage medium. For the same price as a Flash-based player, Procyon MP3 can offer about 53 *hours* of playback time before repeating a track. Spending an extra $50 can double this playback time.

Naturally, there are drawbacks. Procyon MP3 weighs about 14oz with batteries and must not be subjected to shocks higher than 5 to 8G's while playing. Battery life with a set of NiMH prismatic cells and 4MB of buffering SRAM is about eight hours. Nonetheless, for many, the low cost and high capacity outweigh the limitations. As of October 2000, Creative Labs was the first major electronics company to begin selling a hard disk based MP3 player under the model name of "Jukebox". The player has similar features to Procyon MP3 and retails for $599 ($499 street price) with a 6.4GB hard disk. http://www.nomadworld.com/products/jukebox/

2. Hardware Description

Procyon MP3 has six distinct subsystems. Figure 1 is a system block diagram.

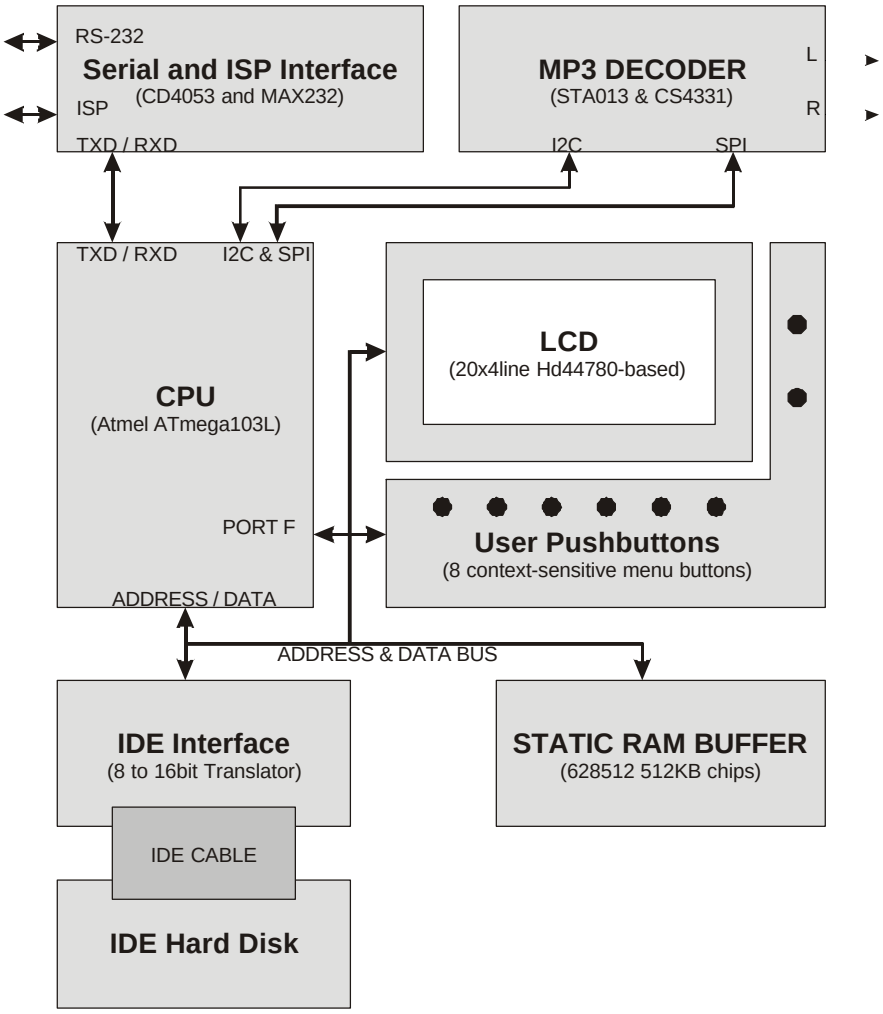


Figure 1 - System Block Diagram of Procyon MP3

2.1. CPU: Central Control

Procyon MP3's CPU is an Atmel Atmega103L. This low-power RISC microcontroller from Atmel operates at 3.3V and offers a UART interface, SPI interface, multiple timers, five 8-bit I/O ports, A/D functions, 4KB of RAM, 512 bytes of EEPROM, and 128KB of Flash program memory. The CPU handles the overall operation of Procyon MP3 and must do everything from interact with the user to operate the hard disk and MP3 decoder. For speed and expandability, all peripherals, with the exception of the user pushbuttons and I^2C MP3 decoder interface, are mapped into the processor's external memory space.

2.2. User interface: Display and Pushbuttons

Procyon MP3's user interface is simple. A character-based (20 char x 4 line) LCD provides status and playback information to the user, while the user may control the player from eight context-sensitive pushbuttons (six along the bottom of the LCD, two on the right side). A ninth button serves as the power on/off control. While the LCD interface is straight-forward, notice that a few NANDs (U12) must be used to

convert the CPUs independent RD and WR lines into a combined R/W and Enable clock for the LCD. *Note* The pushbutton interface has not yet been implemented, but will connect to the CPU's port F, one button per port pin.*

2.3. RAM Memory: System Storage and Audio Buffering

While the CPU system does provide 4KB of built-in RAM for internal variables and program control, more is needed since Procyon MP3 buffers MP3 files into RAM and then shuts down the hard disk to save power. To serve this purpose, a series of 4Mbit (512KB) SRAMs (U13, U14, …) are mapped into the CPU's high address space from 0x8000 to 0xFFFF. (The lower address space is reserved for peripherals and internal RAM.) The entire extended memory can be accessed though this 32K window by selecting the desired page address and latching that into the memory-mapped page latch, U15.

2.4. ATA/IDE Interface and Hard Disk

The IDE interface memory-maps the ten standard IDE registers, through which IDE transaction are made, into the CPU system's address space. While it's true that using I/O ports and firmware to operate the IDE bus would require less external hardware, memory-mapping allows for the fastest possible access to the disk for a given CPU clock speed. This is an important consideration when we intend to quickly buffer MP3s to RAM and then shut down the hard disk. The faster we can read the disk, the more power we can save.

To memory-map the IDE interface, one major problem had to be solved. The IDE interface is inherently a 16-bit bus, while the CPU system has only an 8-bit bus. The obvious solution is to map the low-order byte (D0-D7) of the IDE bus into one address bank and the high-order byte (D8-15) into another address bank, not unlike the way 16-bit integers are stored into 8-bit RAM. The not-so-obvious portion of this problem is that hard disk registers are not random access like bytes in an 8-bit SRAM; the entire 16-bit word must be written in one transaction!

The solution implemented in Procyon MP3 was to add some carefully designed hardware. Latches U7 and U8, along with control logic U9 and U10, form a bi-directional high-byte latch for transactions with the IDE bus. By wiring the latches into a two-byte "ring" buffer, with the high-order byte on one side of the ring and the low-order on the other, we effectively create what can be viewed as a bridge between the high and low order byte-wide bus. Using the bridge, we can transfer one byte to or from the high-order IDE bus lines. This circuit can also be viewed as a single byte dual-port RAM. On one side, the CPU has the ability to read and write the RAM via it's byte-wide bus. On the other side, when the CPU reads or writes the low-order byte or the bus, the IDE disk reads or writes it's high-order byte to the dual-port RAM at the same time. To illustrate, here is an example of a 16-bit write to the IDE bus:

1. CPU writes high-order byte to latches (U8 is used in the case of writing).
2. CPU writes the low-order byte to the IDE bus.

3. During the low-order write cycle in step #2, the output-enable of latch U8 is brought low, causing the high-order byte written in step 1 to be asserted on the IDE bus D8-D15 lines.
4. 16-bit write finished when step 2 (low-order byte access) completes.
5. Return to step 1 for next 16-bit write.

By creatively using the latch line of U7 and U8 as a WR and output-enable line as a RD, we were able to create this bridge or dual-port RAM functionality in a minimum of discrete logic parts. The logic gates U9 and U10 simply transmit the CPU's RD and WR lines to the proper device (IDE bus or latches) with the proper inversion (active-high or active-low) based on the memory address being accessed.

Impressively, with a CPU clock speed of 4MHz, disk-to-RAM transfers in excess of 350KBytes per second were achieved. *As a side-note to engineers designing hardware for the IDE bus, consider yourselves warned that ALL seven ground lines on the IDE interface are critical and must be connected. I wasted nearly a week of development time tracking down a bug which had the bizarre manifestation of missing 0xFFFF words in the data stream I was reading from the disk. In the end, the cause turned out to be nothing more than my ill-fated decision to connect only four out of the seven grounds on the IDE bus. Think about it, 0xFFFF goes missing…*

2.5. MP3 Decoder

The MP3 decoder subsystem of Procyon MP3 is based on the STA013 MP3 audio decoder made by STMicroelectronics and the CS4331 18-bit stereo DAC from Cirrus Logic. While the support circuit for the STA013 and CS4331 (U18 and U19) is indeed simple, enormous complexity is carefully hidden underneath the hood of this specialized DSP MP3 audio decoder.

The decoder requires only six I/O lines for a complete interface to the CPU system. Two lines establish an I^2C interface through which the decoder can be initialized and controlled. Control is accomplished by reading and writing some 80 I^2C registers inside the STA013 allowing a bewildering quantity of options ranging from reading the current MP3 bitrate and encoding parameters to setting the digital volume and bass/treble levels. Three I/O lines are dedicated to supplying the MP3 audio via an I^2S (serial clocked bitstream) interface. Of the these three lines, two are the serial data and clock, while one is the Demand output which indicates when the STA013 is ready for more MP3 data. The Demand line roughly reflects the status of the decoders internal MP3 buffer. Finally, the sixth line is a hardware reset which is not entirely necessary to implement but is useful for returning a device of this complexity to a known state.

One poorly documented part of the STA013 is the decoder's need for a firmware configuration to be uploaded to the device at before using it. The configuration takes the form of a long series of writes to reserved I^2C registers within the device. In full, the configuration amounts to about 4K of data. It's not clear whether the configuration is a firmware patch on a faulty decoding routine, or a high-level code to setup the decoding engine. Interestingly, STMicroelectronics appears to offer other firmware configurations for other audio formats such as PCM audio.

2.6. Power Supply

The final subsystem in the Procyon MP3 player is the power supply. The power supply must efficiently convert variable battery voltages into a steady 5V and 3.3V supply rail to operate the CPU, LCD, IDE interface, IDE hard disk, and MP3 decoder. The supply should also have a shutdown feature and micropower regulator allowing the processor to turn off the MP3 player, put itself into hibernation, and wait for the user to perform an on-button click to revive the player.

The Maxim MAX710 is a high-efficiency switching regulator made for portable applications and offers a 1.8-11V input range with selectable 3.3/5V output and shutdown. Although the Procyon MP3 power supply is still under development, dual MAX710 form the currently favored solution.

3. Software Description

The Procyon MP3 software is written entirely in C and is compiled for the Atmel AVR ATmega103 using a variant of the GCC compiler, AVRGCC. Although all C functions are global, the software is divided into files and groups of functions by purpose. Figure 2 is a hierarchical block diagram of the Procyon MP3 software. The diagram shows the usage and control paths of the software.

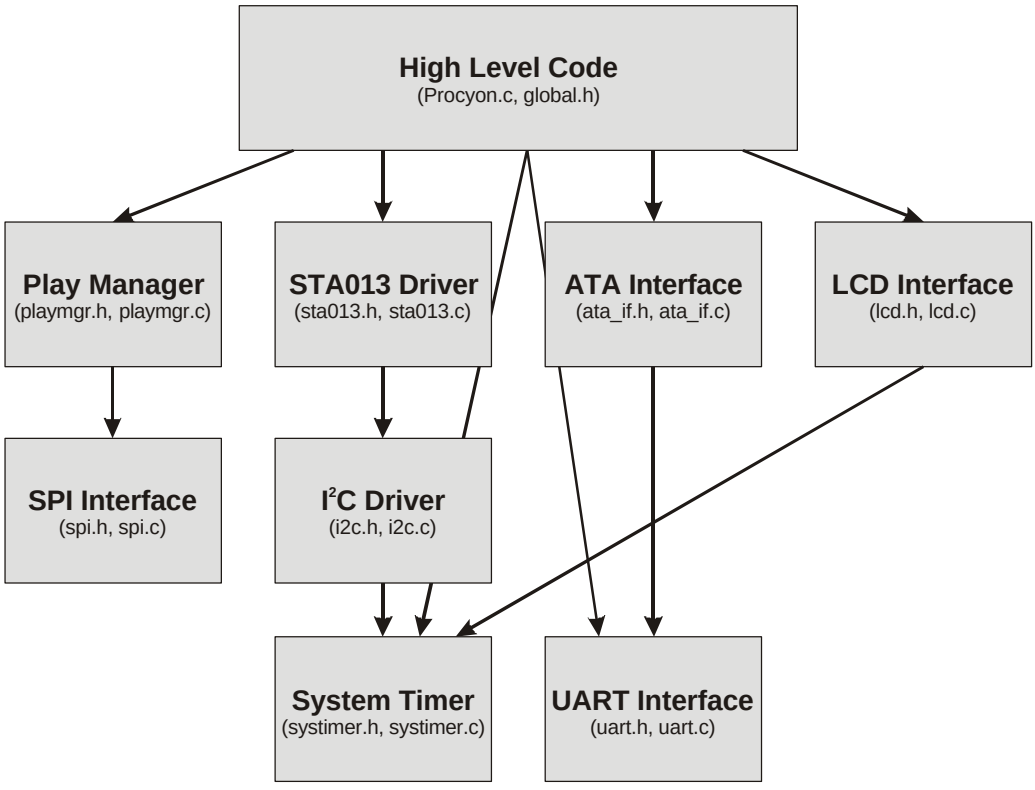


Figure 2 - Procyon MP3 Software Hierarchy

3.1. High-level code (Procyon.c)

The high-level code currently contains most of the user interface and general player initialization and management. At turn-on, all interfaces and devices are initialized and then control is passed to a terminal command line prompt available

through the device serial port. Since a stand-alone pushbutton user interface is not yet complete, nearly all the functions of the player are available from this command line prompt. It's not necessary to list the available commands here since on-line help is available by pressing "?" at the command prompt. Also implemented in the high level code is a command-line based "disk explorer" which allows verification of the IDE interface and media, and various test modes of the MP3 decoder.

3.2. ATA/IDE Interface (ata_if.h, ata_if.c)

The IDE driver code provides a high-level interface for the disk and offers higher layers a uniform disk access mode regardless of hardware implementation, disk type, size, or features. Higher layers need only execute ataInit() and ataDriveInit() after which all disk access can be performed using the LBA model via the commands ataReadSector(…) and ataWriteSector(…). There is no need to keep track of heads or tracks, the entire disk appears as one huge linearly adresssable array of 512 byte sectors.

3.3. I^2C and SPI Interface (i2c.h, i2c.c, spi.h, spi.c)

The I^2C and SPI interface driver code provides higher layers with simple access methods to the interface. In the case of each driver, an initialization routine (spiInit() and i2c_init()) must be called before using the library. The user functions for I2C are simply i2c_send(…) and i2c_receive(…). For SPI, only spiSendByte(…) is needed.

3.4. LCD Interface (lcd.h, lcd.c)

The LCD interface code provides high-level access to all LCD operation abstracting the user from details about how the LCD hardware is actually implemented. In addition to the lcdInit() command, a host of functions are available including cursor positioning, string print, char, int, and long print, and a simple printf.

3.5. Playback Manager (playmgr.h, playmgr.c)

Despite its apparent simplicity, the play manager is an important part of the Procyon MP3 player system. The play manager establishes two interrupt service routines and a management structure for the continuous transmission of MP3s buffer memory to the STA013. Once started, the play manager uses the SPI transaction interrupt to continuously send MP3 stream bytes to the decoder until signaled to stop by the STA013 demand line dropping low. A second interrupt, triggered by the rising edge of the demand line connected to INT4, causes the burst SPI transmission to recommence. This process is performed entirely in the background allowing the foreground task to tend to the user interface and keeping the play buffer full of new data.

3.6. STA013 Driver (sta013.h, sta013.c)

The STA013 driver code is a partial abstraction of the decoder access and control methods. The complexity of the decoder control make full abstraction prohibitive, but the basic functions such as sta013Init(), sta013DownloadUpdate(),

sta013StartDecoder(), sta013StopDecoder(), etc are implemented. From there, #define-d addresses are available for all major decoder control registers. These values can be used directly with the sta013ReadReg(…) and sta013WriteReg(…) commands for complete control over the decoder.

3.7. System Timer (systimer.h, systimer.c)

The system timer code is simply a interface to timer-based and "NOP" based delay routines for general use throughout the player software. Future implementations may include a real-time clock as well.

3.8. UART Interface (uart.h, uart.c)

Much like the LCD interface code, the uart code provides a general initialization, uartInit(), and basic stdio type access function to the uart, uartPrintStr(…), uartPrintf(…), etc. Unlike the LCD, the uart is an asynchronous, bi-directional, and time-dependent interface. Thus the uart interface code also provides an interrupt-driven transmit and receive buffer to keep the processor free from having to worry about polling.

3.9. Global Includes (global.h)

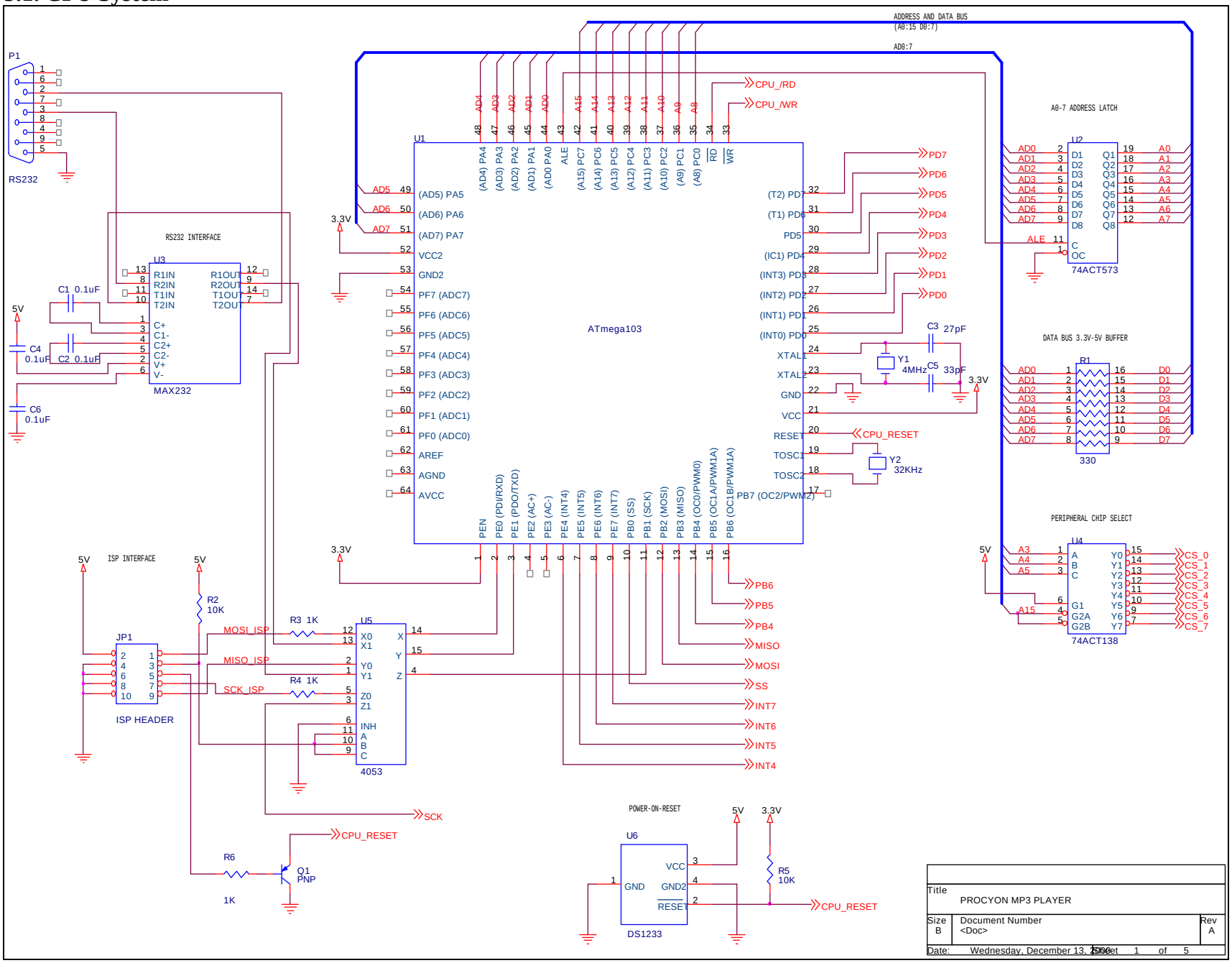
The global include file contains items valid to the entire player. Contained here are globally relevant typedefs, #defines, the peripheral memory map, and global variables.

4. Source Listing

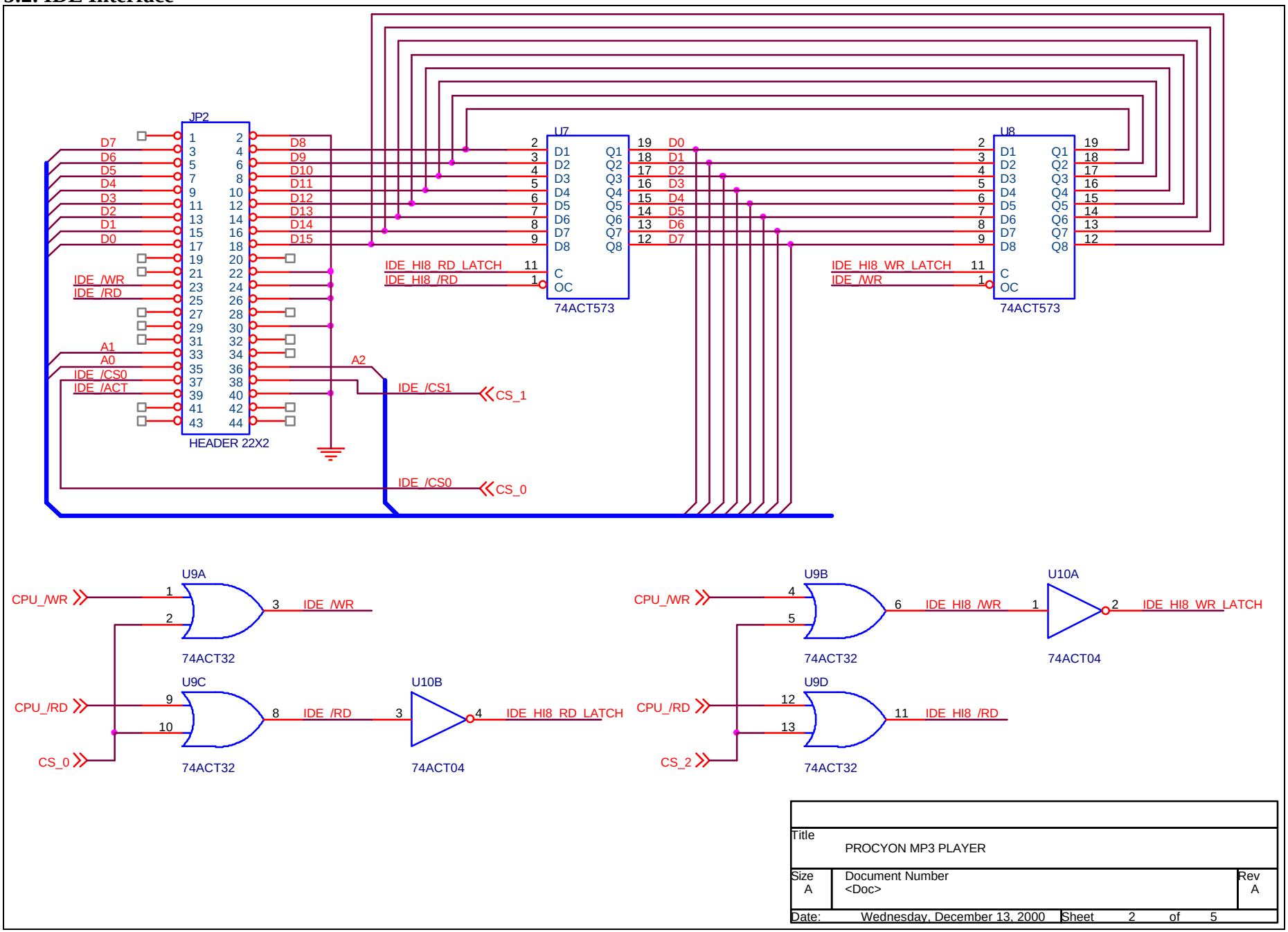
Please see accompanying code files for source listing

5. Schematics

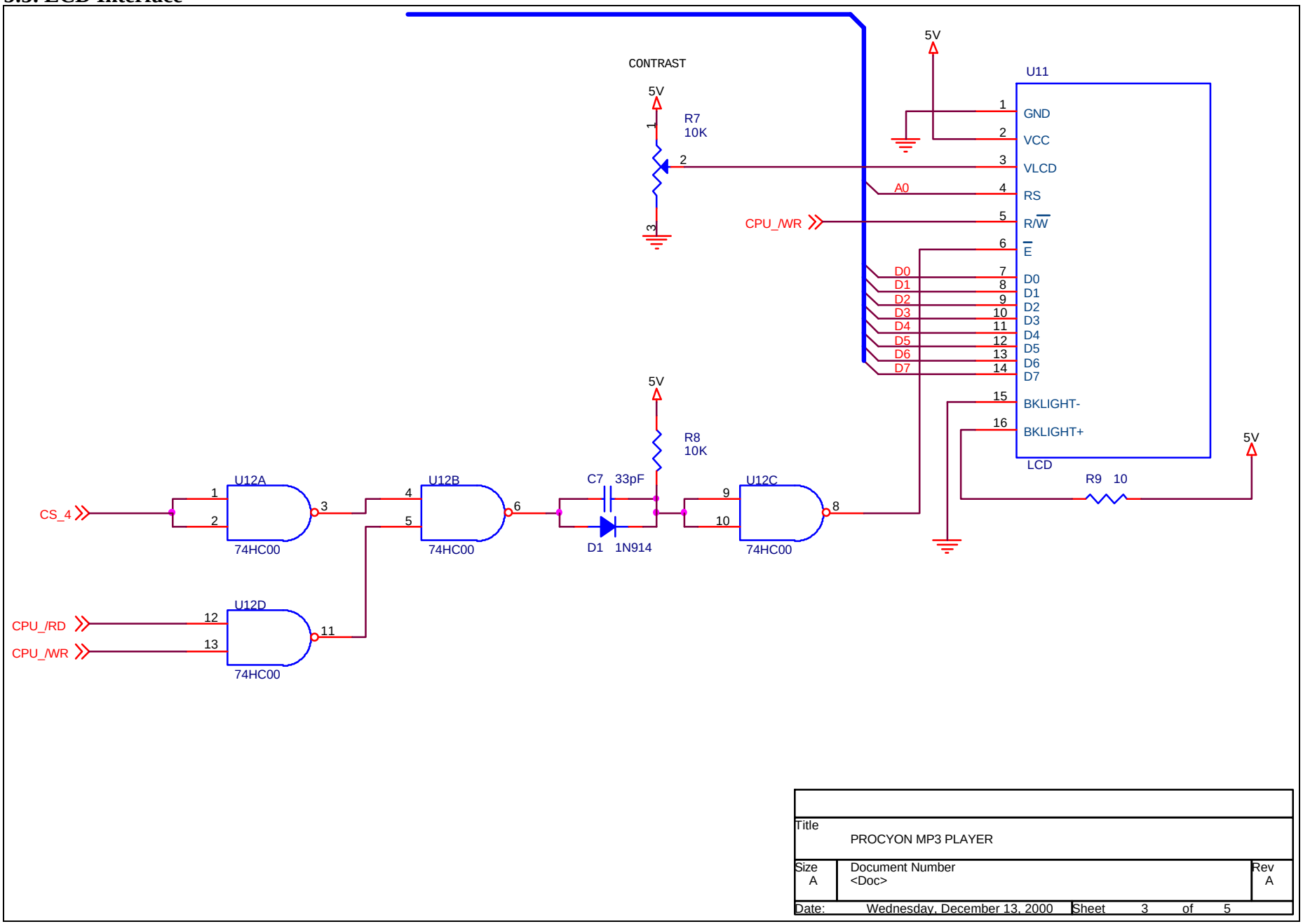
5.1. CPU System



5.2. IDE Interface

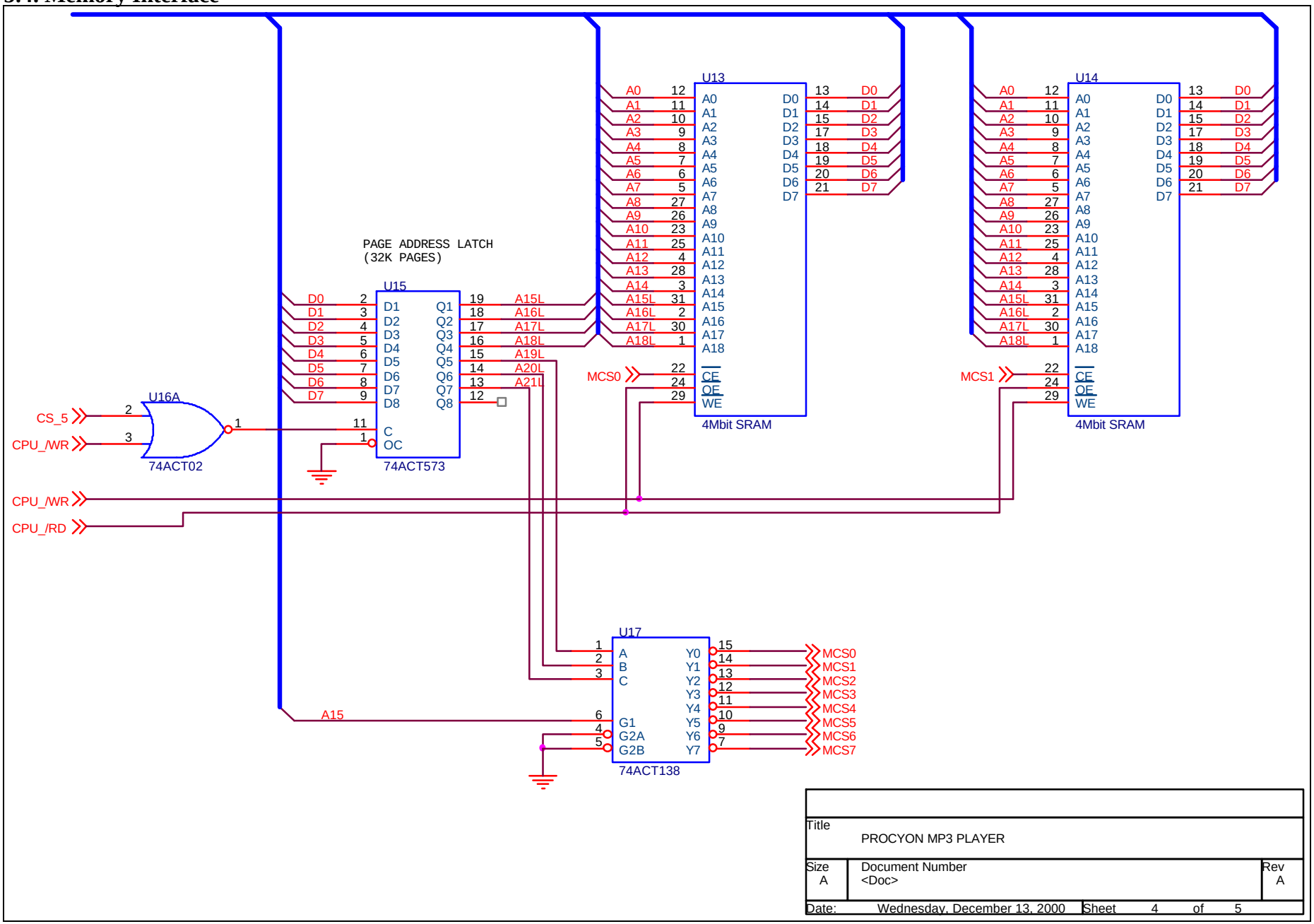


5.3. LCD Interface



Title: PROCYON MP3 PLAYER		
Size: A	Document Number: <Doc>	Rev: A
Date: Wednesday, December 13, 2000	Sheet 3 of 5	

5.4. Memory Interface



5.5. MP3 Decoder