Design Report

CS-400

Solid State MP3 Player

Submitted To	Prof. B. Barnekow
Submitted On	2/7/01
Submitted By	Jason Munzel
	Nathaniel Stoddard
	Jesse Gilles
	Ian Atkinson

Table of Contents

List of Tables

List of Figures

Project Overview

The goal of the project is to design and implement a portable solid-state MP3 player. The product will utilize removable storage in the form of a MultiMedia card for storing MP3 files. It will have an LCD to display information to the user and buttons to allow the user to change songs, start and stop playback, and change the volume. After developing a successful prototype, the product will be implemented on a printed circuit board and packaged by the sponsoring company, Progressive Engineering

Packaging Information

This project is part of a joint venture with Progressive Engineering. Progressive Engineering is an Engineering Services firm providing Implementation & Product Design and Development capabilities with Pro/ENGINEER solid modeling technology. Their mission statement is as follows:

> *"Progressive Engineering is a Product Development Solutions Provider, operating to achieve goals with our clients in a team-centric environment. We will enter into collaborative relationships in which our experience is shared with and transferred to the client. We will act responsibly to improve our clients condition through efficiency, hard work and experience."*

Progressive will be designing all of the packaging for the device. They are allocating highly skilled engineers to design and create the packaging in Pro/Engineer, and will use rapid prototyping techniques to generate actual samples of the finished product. They are providing these services at no charge in exchange for publicity, tax benefits, and experience in being involved with the entire development process for a consumer electronics device.

Major System Components

The seven major system components will be the MultiMedia card, two PIC processors, an MPEG decoder, a DAC, an LCD, and 7 buttons. The following table shows the actual components used for the major systems.

Table 1: System Components

System	Part
MCP	PIC16C67
DTP	PIC16C66
MPEG Decoder	STA013

DAC	CS4334
MMC	MultiMedia card
LCD	DMC16105

The master controller PIC (MCP) will interface with the buttons, using direct connections to pins, the LCD, using parallel communication, and the decoder, using I^2C communication. Button presses will be read by the MCP and translated into actions within the system. Changes to volume, bass and treble will be communicated to the decoder via the I^2C interface. Visual output to the user will be done using the LCD that the MCP interfaces to. Communication with the LCD will be done over an 8-bit parallel connection. The MCP will also communicate with the slave PIC (DTP), using an RS232 connection. All MMC access will be performed by the DTP so that data streaming can never be interrupted or slowed by user actions. The MCP will issue commands to the DTP to *tell* it to perform an action, such as start streaming a track. Normal operation will be for the DTP and MMC to stream data to the DSP. However, in order to display the current track information to the user on the LCD, the DTP will also be able to extract track information and return it to the MCP. The DAC will be directly connected to the decoder and its analog output to a headphone jack.

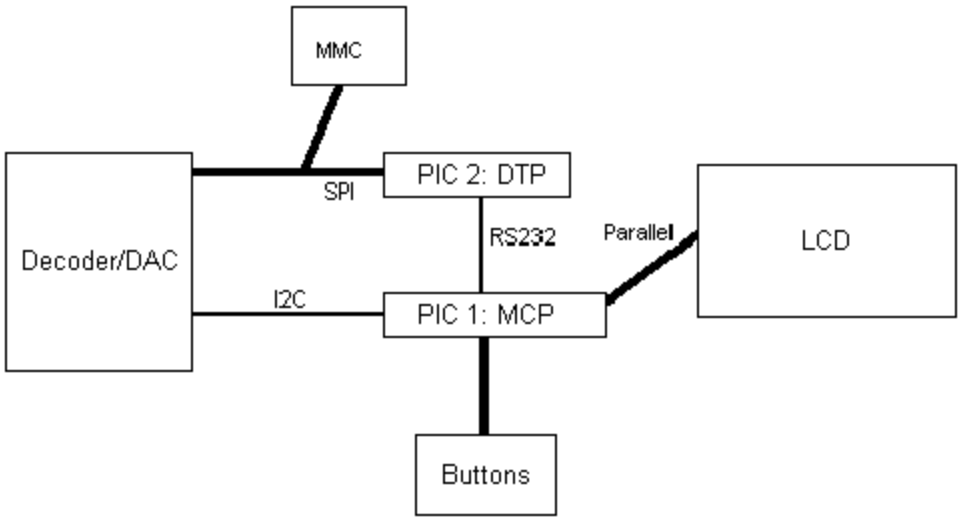


Figure 1: System Block Diagram

Hardware Design

System Communication

Several forms of communication will take place between the various subsystems. These communication protocols are outlined, with the associated subsystems, in the following table.

Table 2: Subsystem Communication

System 1	Communication	System 2
MCP	Parallel	LCD

MCD	Serial - RS232	DTP
DTP	Serial – SPI	MMC
MMC	Serial – SPI	MPEG Decoder
MCP	Serial – I^2C	MPEG Decoder

Connection Selections

Since both the MCP and the DTP have several I/O pins (33 and 22 respectively) there were a number of options when connecting the subsystems. Connections were selected such that they would provide as much help to the software as possible. For example, on the MCP, the processor has certain pins that can generate interrupts. These pins were used for connections that the software needs to know as soon as something happens, such as the MMC being inserted or removed. The following tables outlines the connections and the reasons for the pins used.

Table 3: MCP Connections

Connection	Pin(s)	Reason
LCD	Port D, Port E	PSP support, which helps with communication to LCD
DTP	Port C[6..7]	Default RS232 pins
Decoder	Port C[3..4]	I^2C interface pins
MMC Detection	Port B[7]	Can generate interrupts so card insertions/removals can be detected
Buttons	Port B[0..6]	Sequentially laid out, available

Table 4: DTP Connections

Connection	Pin(s)	Reason
MMC	Port C[3..5]	SPI interface pins
Decoder Data	Port C[3..4]	SPI interface pins (except data in)
MMC Detection	Port B[7]	Can generate interrupts so card insertions/removals can be detected
MCP	Port C[6..7]	Default RS232 pins

Hardware Maintenance

The nature of the final project is such that there is virtually no need for hardware maintenance. Swapping of memory cards will be supported so that users can have more than one card with audio tracks. However, the remaining components will all be soldered directly to the printed circuit board. Since most of the component interfaces are proprietary, or at least unique with respect to pin positions, a redesign would be required to replace any major component. Just as similar consumer small electronics, such as calculators and portable compact disc players, the final product will be sealed from the user and there will be no low-level hardware upgrades available.

Logic Schematic

The following image shows the detailed connections required for the project logic.

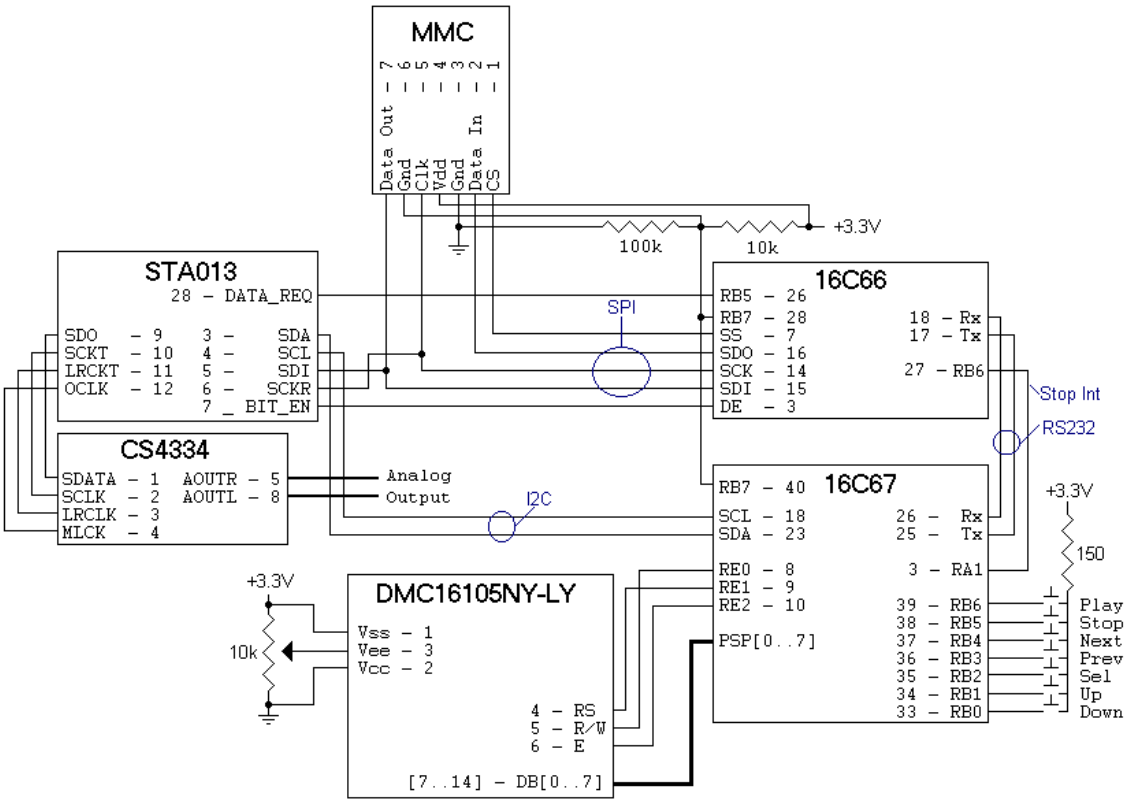


Figure 2: Schematic of Hardware Logic Connections

Software Design

Subsystem Libraries

Each subsystem within the project, that will be frequently interacted with, has its own software library. The following table lists the subsystem with its associated library. A complete listing of the library interfaces can be found in the appendixes.

Table 5: Subsystems and Associated Software Libraries

Subsystem	Software Library or Header	Purpose
MCP	MCP	Code for MCP
DTP	DTP	Code for DTP
MPEG Decoder	DecoderLib	Initialization and management of STA013 MPEG Decoder
MMC	MMCLib	Initialization and data access of MMC
LCD	LCDLib	Initialization and management of LCD

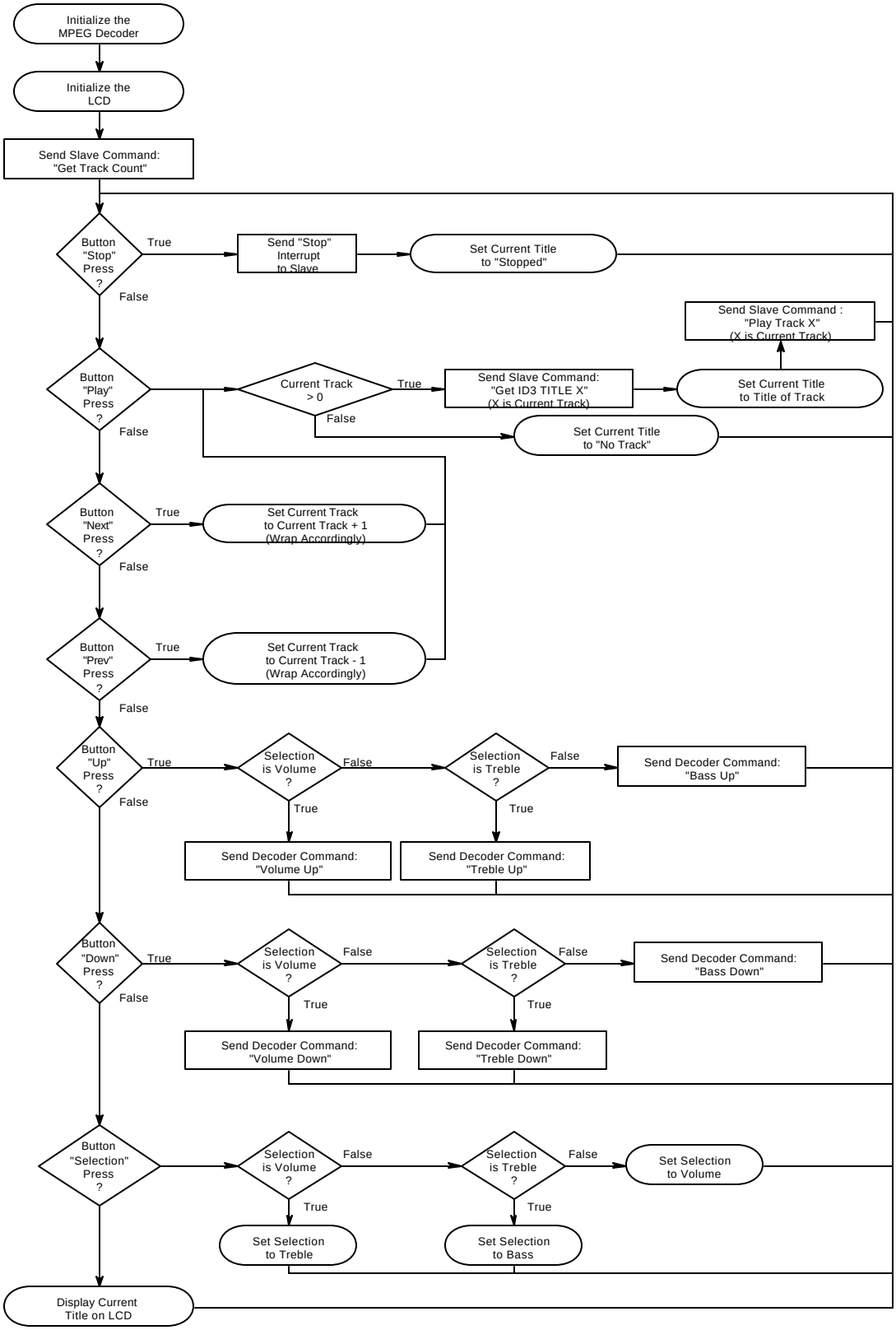
Software Maintenance

The software is designed as a number of libraries, one for each major subsystem. Each library is externally generic so that calls it can be made without detailed knowledge and understanding of the library. This provides a number of advantages. First, any problems associated with a specific subsystem can be immediately traced to the library. For example, if an error occurred while initializing the MPEG decoder, it would be known that the problem must be in the decoder library's *init* routine since it is externally generic. That is, regardless of how the *init* routine is invoked, it will either initialize properly or return an error. Second, only the author of the library needs to have intricate knowledge of the associated hardware. This the entire team to perform a task such as reading data from the MMC without knowing every minor nuance associated with the initialization and data transfer. Finally, it allows changes to be made to the underlying library without affecting any system using that library. As long as the interface to the routine is not changed, there is no need to update any code except in the library.

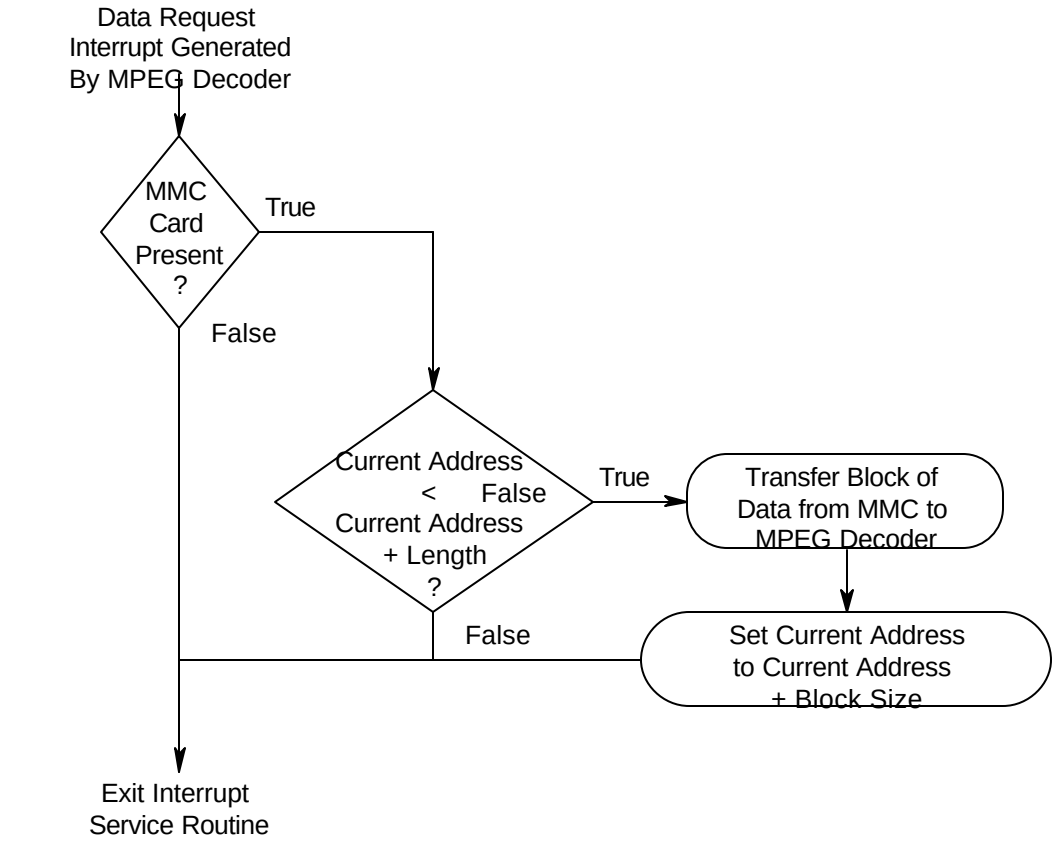
Software Flow

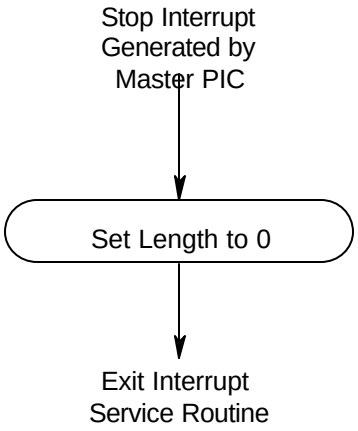
The following charts illustrate the software flow in both the MCP and DTP. Both processors will have *standard* code and interrupt code. These are broken apart for simplicity.

Master Controller PIC Main Code

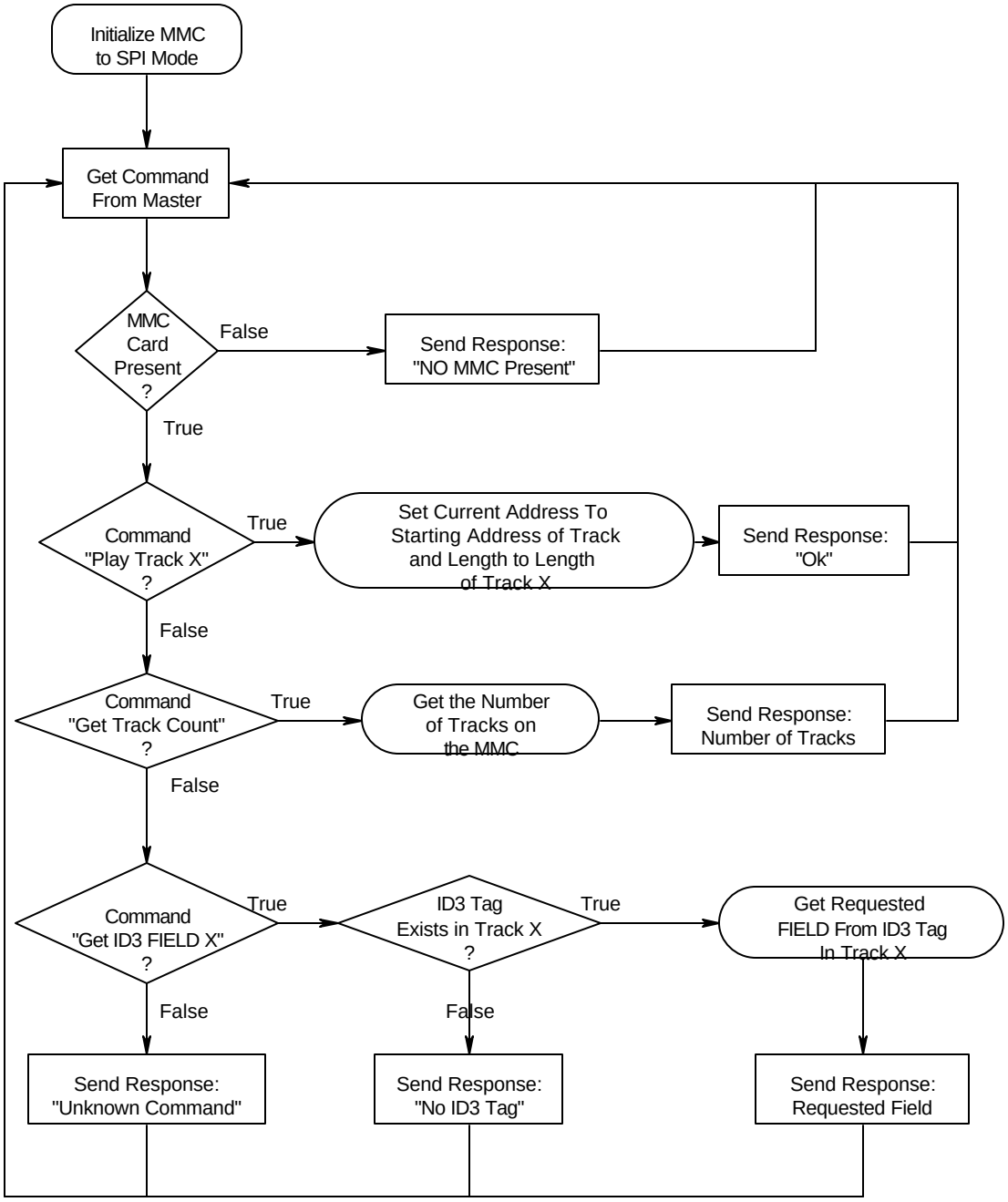


Master Controller PIC Interrupt Code

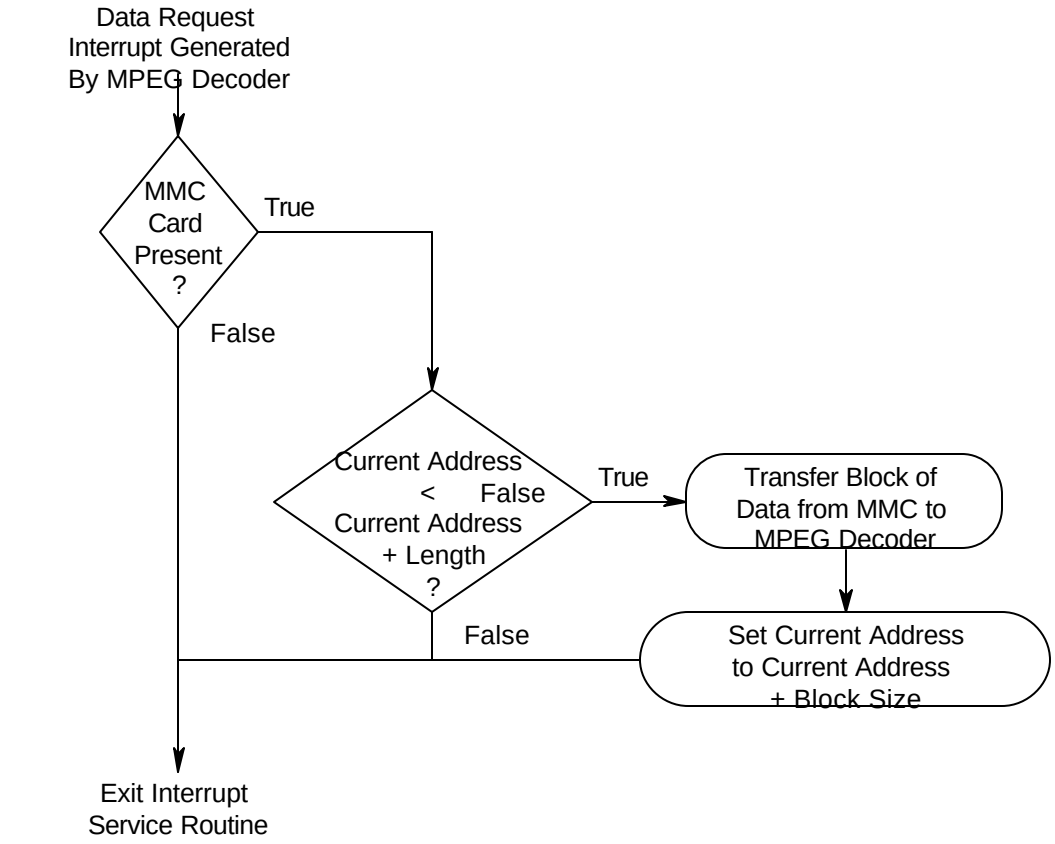


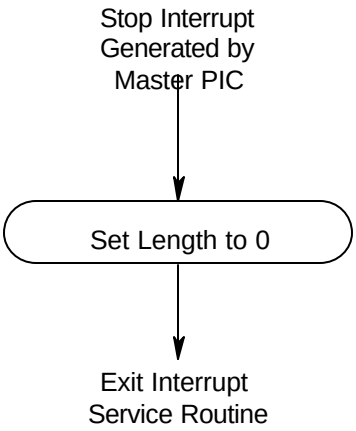


Data Transfer PIC Main Code



Data Transfer PIC Interrupt Code





Product Cost

The following table outlines the required components for the end product and their individual cost.

Table 6: Product Cost Breakdown

Item Description	Part Number	Supplier	Quantity	Cost Per	Total Cost
STA013 MPEG Decoder Chip	STA013_SOP32	PJRC.com	1	$20.00	$20.00
Crystal Stereo DAC Chip	CS4334_SOIC	PJRC.com	1	$5.00	$5.00
MicroChip PIC16C66 MCU	PIC16C66/JW	DigiKey	1	$13.80	$13.80
MicroChip PIC16C67 MCU	PIC16C67/JW	DigiKey	1	$13.89	$13.89
Maxim 1709 3.3V - 5V Step-Up DC-DC Converter	MAX1709ESE	Arrow	1	$6.08	$6.08
Maxim 1760 3.3V Step-Up DC-DC Converter	MAX1760EUB	Arrow	1	$4.35	$4.35
Right Angle Tactile Switch	SW407-ND	DigiKey	7	$0.30	$2.10
Common Parts (resistors, capacitors crystals, etc.)	-	DigiKey	-	$10.00	$10.00
Printed Circuit Board	MiniBoard Service	ExpressPCB	1	$20.00	$20.00
32 MB MultiMedia Card	32 MB MMC Card	sandiskstore.com	1	$74.99	$74.99
Optrex 16x1 Backlit LCD	DMC-16105NY-LY	DigiKey	1	$26.94	$26.94
			Total Cost:		$197.15

Implementation Timeline

Table 7: Implementation Timeline

Week	Milestones
Week 1	?? Prototype interface MMC, DTP
Week 2	?? Prototype interface Decoder/DAC, MCP
Week 3	?? Prototype interface MCP, DTP and buttons
Week 4	?? Prototype interface LCD, MCP
Week 5	?? Prototype complete system
Week 6	?? Create PCB
Week 7	?? Connect and start testing PCB (must submit packaging data)
Week 8	
Week 9	?? Complete testing PCB
Week 10	?? Finish PCB and package
Week 11	?? Prepare for show

Project Management

Table 8: Previous Time Log

Activity	Atkinson	Gilles	Munzel	Stoddard
Hardware Design	8			1
Install software (PIC)	1	4	1	2
Researched Bluetooth			2	40
Researched Connectors/Adapters	2	2	2	2
Researched Decoder/DAC	25			
Researched LCD			5	
Researched Memory	1	8	1	
Researched PCB			7	3
Researched PIC	2	6		
Researched Power				10
Resource Acquisition			15	
Software Design	10	1		
Tested basic PIC features		3	1	3
Tested LCD Interface	1		1	11
Tested RS232 Interface	1			1
Totals	51	24	35	73

Assuming a rate of $30.00 per hour, the total development cost thus far $5490.00.

Table 9: Future Time Log

Activity	Estimated Hours	Actual Hours
Interface MMC to DTP	30	
Interface MCP to DCP	7	
Interface Decoder/DAC to MCP	25	
Interface Decoder/DAC to DTP	5	
Interface LCD to MCP	5	
Create PCB	15	
Solder PCB	5	
Test PCB	10	
Packaging (Progressive)	3 weeks	
Interface MCP to Buttons	5	

Glossary

MCP – Master Controller PIC: The PIC processor that handles user I/O (LCD, buttons) and manages the decoder.

DTP – Data Transfer PIC: The PIC processor that is responsible for accessing the MMC and streaming data to the decoder.

MMC – MultiMedia Card: Solid-state removable memory used for storage of audio tracks.

Appendix A – Headers and Software Library Interfaces

Globals

```
// Global.h
//
// Author: Ian Atkinson
// Date  : 2/4/01
// Declares constants and macros used globally by all
// modules this file should be included by every module
// to garuntee consistency
//

#ifndef _GLOBAL_H
#define _GLOBAL_H

// define a datatype of byte
// since a byte is 8-bits it is really just a character
#define byte char

// there is no "bool" type in c
// it can be faked using a byte however
#define bool byte
#define true 1
#define false 0

// macro defines
#define MAX(a,b) (a>b?a:b)
#define MIN(a,b) (a<b?a:b)

#endif
```

DecoderLib

```
// DecoderLib.h
//
// Author: Ian Atkinson
// Date  : 2/4/01
// Declares the constants and prototypes to interface to the
// STA013 MPEG decoder
//

#ifndef _DECODER_LIB_H
#define _DECODER_LIB_H

#include <global.h>

// constant values for volume range
const byte DECODER_VOLUME_MAX = 0x00;
const byte DECODER_VOLUME_MIN = 0xFF;

// constance values for bass and treble range
const byte DECODER_ENHANCE_MAX = 0x0C;
const byte DECODER_ENHANCE_MIN = 0xF4;

//-------------------------------------------------
// DecoderInit
//-------------------------------------------------
// Author: Ian Atkinson
// Date  : 2/4/01
//    Initializes the MPEG decoder for use
//-------------------------------------------------
// Arugments
//-------------------------------------------------
// Return value: void
//-------------------------------------------------
// Revisions
// Date     Author     Change
//    --/--/-- ---------- -------------------------
//-------------------------------------------------
void DecoderInit();

//-------------------------------------------------
// DecoderPlay
//-------------------------------------------------
// Author: Ian Atkinson
// Date  : 2/4/01
//    Starts the decoder decoding
//-------------------------------------------------
// Arugments
//-------------------------------------------------
// Return value: true if decoder was successfully
//                           started playing
//                      false otherwise
//-------------------------------------------------
// Revisions
// Date     Author     Change
//    --/--/-- ---------- -------------------------
```

```
//-------------------------------------------------
bool DecoderPlay();

//-------------------------------------------------
// DecoderStop
//-------------------------------------------------
// Author: Ian Atkinson
// Date  : 2/4/01
// Stops the decoder from decoding
//-------------------------------------------------
// Arugments
//-------------------------------------------------
// Return value: true if decoder was successfully
//                            stopped playing
//                       false otherwise
//-------------------------------------------------
// Revisions
// Date     Author     Change
// --/--/-- ---------- --------------------------
//-------------------------------------------------
bool DecoderStop();

//-------------------------------------------------
// DecoderGetVolume
//-------------------------------------------------
// Author: Ian Atkinson
// Date  : 2/4/01
//    Gets the digital volume setting in the decoder
//-------------------------------------------------
// Arugments
//-------------------------------------------------
// Return value: the digital volume setting
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- --------------------------
//-------------------------------------------------
// Notes
//    The volume is actually stored as amount
//    attenuation.  This means that a larger value
//    is really a lower volume.  A volume of zero
//    would be the max volume
//-------------------------------------------------
byte DecoderGetVolume();

//-------------------------------------------------
// DecoderSetVolume
//-------------------------------------------------
// Author: Ian Atkinson
// Date  : 2/4/01
//    Sets the digital volume setting in the decoder
//-------------------------------------------------
// Arugments
//    value - byte : the new volume setting
//-------------------------------------------------
// Return value: true if value was applied
//                       false otherwise
```

```
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- -------------------------
//-------------------------------------------------
// Notes
//    The volume is actually stored as amount
//    attenuation.  This means that a larger value
//    is really a lower volume.  A volume of zero
//    would be the max volume
//-------------------------------------------------
bool DecoderSetVolume(const byte& value);

//-------------------------------------------------
// DecoderGetTreble
//-------------------------------------------------
// Author: Ian Atkinson
// Date  : 2/4/01
//    Gets the digital treble setting in the decoder
//-------------------------------------------------
// Arugments
//-------------------------------------------------
// Return value: the digital treble setting
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- -------------------------
//-------------------------------------------------
// Notes
//    See STA013 documentation for interpretations
//    of values
//-------------------------------------------------
byte DecoderGetTreble();

//-------------------------------------------------
// DecoderSetTreble
//-------------------------------------------------
// Author: Ian Atkinson
// Date  : 2/4/01
//    Sets the digital treble setting in the decoder
//-------------------------------------------------
// Arugments
//    value - byte : the new treble setting
//-------------------------------------------------
// Return value: true if value was applied
//                       false otherwise
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- -------------------------
//-------------------------------------------------
// Notes
//    See STA013 documentation for interpretations
//    of values
//-------------------------------------------------
bool DecoderSetTreble(const byte& value);
```

```
//-------------------------------------------------
// DecoderGetBass
//-------------------------------------------------
// Author: Ian Atkinson
// Date  : 2/4/01
//    Gets the digital bass setting in the decoder
//-------------------------------------------------
// Arugments
//-------------------------------------------------
// Return value: the digital bass setting
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- --------------------------
//-------------------------------------------------
// Notes
//    See STA013 documentation for interpretations
//    of values
//-------------------------------------------------
byte DecoderGetBass();

//-------------------------------------------------
// DecoderSetBass
//-------------------------------------------------
// Author: Ian Atkinson
// Date  : 2/4/01
//    Sets the digital bass setting in the decoder
//-------------------------------------------------
// Arugments
//    value - byte : the new bass setting
//-------------------------------------------------
// Return value: true if value was applied
//                       false otherwise
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- --------------------------
//-------------------------------------------------
// Notes
//    See STA013 documentation for interpretations
//    of values
//-------------------------------------------------
bool DecoderSetBass(const byte& value);

#endif
```

MMCLib

```
// MMCLib.h
//
// Author: Ian Atkinson
// Date  : 2/5/01
// Declares the constants and prototypes to interface to the
// MultiMedia Card
//

#ifndef _MMC_LIB_H
#define _MMC_LIB_H

#include <global.h>

/-----------------------------------------------
// MMCInit
//-----------------------------------------------
// Author: Ian Atkinson
// Date  : 2/5/01
//    Initializes the MMC into SPI mode
//-----------------------------------------------
// Arugments
//-----------------------------------------------
// Return value: void
//-----------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- -------------------------
//-----------------------------------------------
void MMCInit();

/-----------------------------------------------
// MMCReadByte
//-----------------------------------------------
// Author: Ian Atkinson
// Date  : 2/5/01
//    Reads a single byte from the MMC
//-----------------------------------------------
// Arugments
//    address - long : address to read from
//    value - byte : data read
//-----------------------------------------------
// Return value: true if data was successfully
//                            read
//                       false otherwise
//-----------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- -------------------------
//-----------------------------------------------
bool MMCReadByte(const long &address, byte &value);

/-----------------------------------------------
// MMCReadByte
//-----------------------------------------------
```

```
// Author: Ian Atkinson
// Date  : 2/5/01
//    Reads a single byte from the MMC
//-------------------------------------------------
// Arugments
//    address - long : address to start reading from
//    length - long : number of bytes to read
//    value - byte : allocated buffer to write to
//-------------------------------------------------
// Return value: true if data was successfully
//                            read
//                       false otherwise
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- -------------------------
//-------------------------------------------------
bool MMCReadBlock(const long &address, const long &length, byte*
value);

#endif
```

Commands

```
// Commands.h
//
// Author: Ian Atkinson
// Date  : 2/4/01
// Declares the constants and data structures used for RS232
// communication between the master controller PIC and the
// data transfer PIC
//
#ifndef _COMMANDS_H
#define _COMMANDS_H

#include <global.h>

// define ACK and NACK as printable characters
// to make debugging easier
const byte ACK    = "*";
const byte NACK   = "#";

// MCP/DTP command constants
const byte COMMAND_GET_TRACK_COUNT  = 'C';
const byte COMMAND_PLAY_TRACK       = 'P';
const byte COMMAND_GET_ID3_FIELD    = 'I';

// ID3 field constants
const byte ID3_FIELD_TITLE               = 'T';
const byte ID3_FIELD_ARTIST              = 'A';
const byte ID3_FIELD_ALBUM               = 'M';
const byte ID3_FIELD_YEAR                = 'Y';
const byte ID3_FIELD_GENRE               = 'G';

// command structure
struct command
{
      byte CommandCode; // the code for the command to be executed
      byte Param1;            // the first parameter for the command
(may be NULL)
      byte Param2;            // the second parameter for the command
(may be NULL)
};

#endif
```

DTP

```
// DTP.h
//
// Author: Ian Atkinson
// Date  : 2/4/01
// Declares the prototypes for the routines supported on the
// data transfer PIC
//
#ifndef _DTP_H
#define _DTP_H

#include <global.h>
#include "commands.h"

// constant defines
#define MAX_TRACKS 20

// struct to hold information about a single track
struct DTPTrack
{
      long StartAddress;                  // starting address of track
      long Length;                        // length of track (in bytes)
      bool ID3;                           // ID3 tag present
};

// data members associated with the DTP
byte _DTPTrackCount = 0;
struct DTPTrack _DTPTracks[MAX_TRACKS];

//-------------------------------------------------
// DTPInit
//-------------------------------------------------
// Author: Ian Atkinson
// Date  : 2/4/01
//    Initializes all subsystems and settings for
//    the data transfer PIC
//-------------------------------------------------
// Arugments
//-------------------------------------------------
// Return value: void
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- -------------------------
//-------------------------------------------------
void DTPInit();

//-------------------------------------------------
// DTPGetCommand
//-------------------------------------------------
// Author: Ian Atkinson
// Date  : 2/4/01
//    Receives a single command from the MCP. Blocks
//    until a command is received
//-------------------------------------------------
```

```
// Arugments
//-------------------------------------------------
// Return value: command structure received
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- -------------------------
//-------------------------------------------------
const &command DTPGetCommand();

//-------------------------------------------------
// DTPSendResponse
//-------------------------------------------------
// Author: Ian Atkinson
// Date  : 2/4/01
//    Sends a command response to the MCP
//-------------------------------------------------
// Arugments
//    value - byte : response value
//-------------------------------------------------
// Return value: true if response was successfully
//                            sent
//                       false otherwise
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- -------------------------
//-------------------------------------------------
bool DTPSendResponse(const byte& value);

//-------------------------------------------------
// DTPSendResponse
//-------------------------------------------------
// Author: Ian Atkinson
// Date  : 2/4/01
//    Sends a command response to the MCP
//-------------------------------------------------
// Arugments
//    str - char* : response string/message
//-------------------------------------------------
// Return value: true if response was successfully
//                            sent
//                       false otherwise
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- -------------------------
//-------------------------------------------------
bool DTPSendResponse(const char* str);

//-------------------------------------------------
// DTPLoadCardTracks
//-------------------------------------------------
// Author: Ian Atkinson
// Date  : 2/4/01
//    Reads the track information off the MMC
//-------------------------------------------------
```

```
// Arugments
//-------------------------------------------------
// Return value: true if MMC was successfully
//                            read
//                       false otherwise
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- -------------------------
//-------------------------------------------------
bool DTPLoadCardTracks();

//-------------------------------------------------
// DTPCommandPlay
//-------------------------------------------------
// Author: Ian Atkinson
// Date  : 2/4/01
//    Plays a track of the MMC
//-------------------------------------------------
// Arugments
//    track - byte : track number to play
//-------------------------------------------------
// Return value: true if the track was
//                            successfully played (or
//                            stopped by an interrupt)
//                       false otherwise
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- -------------------------
//-------------------------------------------------
bool DTPCommandPlay(const byte& track);

//-------------------------------------------------
// DTPCommandStop
//-------------------------------------------------
// Author: Ian Atkinson
// Date  : 2/4/01
//    Stops any currently playing track
//-------------------------------------------------
// Arugments
//-------------------------------------------------
// Return value: true if the track was
//                            successfully stopped
//                       false otherwise
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- -------------------------
//-------------------------------------------------
void DTPCommandStop();

//-------------------------------------------------
// DTPCommandID3Field
//-------------------------------------------------
// Author: Ian Atkinson
// Date  : 2/4/01
```

```
//    Retrieves a field from the ID3 tag of a track
//    and sends it to the MCP
//-------------------------------------------------
// Arugments
//    track - byte : track number to read field from
//    field - byte : field code for field to read
//-------------------------------------------------
// Return value: true if the field was
//                            successfully read from track
//                       false otherwise
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- -------------------------
//-------------------------------------------------
void DTPCommandID3Field(const byte& track, const byte& field);

//-------------------------------------------------
// DTPISRStop
//-------------------------------------------------
// Author: Ian Atkinson
// Date  : 2/4/01
//    Handles the Stop Interrupt generated by
//    the MCP
//-------------------------------------------------
// Arugments
//-------------------------------------------------
// Return value: void
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- -------------------------
//-------------------------------------------------
void DTPISRStop();

//-------------------------------------------------
// DTPISRCardChange
//-------------------------------------------------
// Author: Ian Atkinson
// Date  : 2/4/01
//    Handles the Card Interrupt generated by
//    a MMC being inserted or removed
//-------------------------------------------------
// Arugments
//-------------------------------------------------
// Return value: void
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- -------------------------
//-------------------------------------------------
void DTPISRCardChange();

#endif
```

MCP

```
// MCP.h
//
// Author: Jesse Gilles
// Date      : 2/6/01
// Declares the prototypes for the routines supported
//    on the master controller PIC

#ifndef _MCP_H
#define _MCP_H

#include <global.h>
#include <commands.h>


//-------------------------------------------------
// MCPInit
//-------------------------------------------------
// Author: Jesse Gilles
// Date  : 2/6/01
//    Initializes all subsystems and settings for
//    the master controller PIC
//-------------------------------------------------
// Arugments
//-------------------------------------------------
// Return value: void
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- -------------------------
//-------------------------------------------------
void MCPInit();

//-------------------------------------------------
// MCPGetResponse
//-------------------------------------------------
// Author: Jesse Gilles
// Date  : 2/6/01
//    Receives a response from the DTP. Blocks
//    until a command is received
//-------------------------------------------------
// Arugments
//-------------------------------------------------
// Return value: command structure received
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- -------------------------
//-------------------------------------------------
const &command MCPGetRespnse();

//-------------------------------------------------
// MCPSendCommand
//-------------------------------------------------
// Author: Jesse Gilles
```

```
// Date  : 2/6/01
//    Sends a command to the DTP
//-------------------------------------------------
// Arugments
//    value - byte : response value
//-------------------------------------------------
// Return value: true if response was successfully
//                            sent
//                       false otherwise
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- --------------------------
//-------------------------------------------------
bool MCPSendCommand(const byte& value);

//-------------------------------------------------
// MCPISRPlayButton
//-------------------------------------------------
// Author: Jesse Gilles
// Date  : 2/6/01
//    Interrupt service routine caused by the play
//    button being pressed
//-------------------------------------------------
// Arugments
//    none
//-------------------------------------------------
// Return value: nothing
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- --------------------------
//-------------------------------------------------
void MCPISRPlayButton();

//-------------------------------------------------
// MCPISRStopButton
//-------------------------------------------------
// Author: Jesse Gilles
// Date  : 2/6/01
//    Interrupt service routine caused by the stop
//    button being pressed
//-------------------------------------------------
// Arugments
//    none
//-------------------------------------------------
// Return value: nothing
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- --------------------------
//-------------------------------------------------
void MCPISRStopButton();

//-------------------------------------------------
// MCPISRPrevTrackButton
//-------------------------------------------------
```

```
// Author: Jesse Gilles
// Date  : 2/6/01
//    Interrupt service routine caused by the prev
//    track button being pressed
//-------------------------------------------------
// Arugments
//    none
//-------------------------------------------------
// Return value: nothing
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- --------------------------
//-------------------------------------------------
void MCPISRPrevTrackButton();

//-------------------------------------------------
// MCPISRNextTrackButton
//-------------------------------------------------
// Author: Jesse Gilles
// Date  : 2/6/01
//    Interrupt service routine caused by the next
//    track button being pressed
//-------------------------------------------------
// Arugments
//    none
//-------------------------------------------------
// Return value: nothing
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- --------------------------
//-------------------------------------------------
void MCPISRNextTrackButton();

//-------------------------------------------------
// MCPISRSelectButton
//-------------------------------------------------
// Author: Jesse Gilles
// Date  : 2/6/01
//    Interrupt service routine caused by the select
//    button being pressed
//-------------------------------------------------
// Arugments
//    none
//-------------------------------------------------
// Return value: nothing
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- --------------------------
//-------------------------------------------------
void MCPISRSelectButton();

//-------------------------------------------------
// MCPISRUpButton
//-------------------------------------------------
```

```
// Author: Jesse Gilles
// Date  : 2/6/01
//    Interrupt service routine caused by the up
//    button being pressed
//-------------------------------------------------
// Arugments
//    none
//-------------------------------------------------
// Return value: nothing
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- --------------------------
//-------------------------------------------------
void MCPISRUpButton();

//-------------------------------------------------
// MCPISRDownButton
//-------------------------------------------------
// Author: Jesse Gilles
// Date  : 2/6/01
//    Interrupt service routine caused by the down
//    button being pressed
//-------------------------------------------------
// Arugments
//    none
//-------------------------------------------------
// Return value: nothing
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- --------------------------
//-------------------------------------------------
void MCPISRDownButton();

//-------------------------------------------------
// MCPGenerateStopInterrupt
//-------------------------------------------------
// Author: Jesse Gilles
// Date  : 2/6/01
//    Generate an interrupt to tell the DTP to stop
//    streaming song data to the decoder
//-------------------------------------------------
// Arugments
//    none
//-------------------------------------------------
// Return value: nothing
//-------------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- --------------------------
//-------------------------------------------------
void MCPGenerateStopInterrupt();

//-------------------------------------------------
// MCPISRCardChange
//-------------------------------------------------
```

```
// Author: Jesse Gilles
// Date  : 2/6/01
//    Handles the Card Interrupt generated by
//    a MMC being inserted or removed
//-----------------------------------------------
// Arugments
//-----------------------------------------------
// Return value: void
//-----------------------------------------------
// Revisions
//    Date     Author     Change
//    --/--/-- ---------- --------------------------
//-----------------------------------------------
void MCPISRCardChange();

#endif
```

LCDLib

```
// LCDlib.h
//
// Author:        Nate Stoddard
// Date:          2-3-01
//
// This is the main library for the LCD.

#ifndef _LCD_LIB_H
#define _LCD_LIB_H

// Defines:
#define SENDINIT 0           // RS=0, R/W = 0 (Instruction Write)
#define SENDDATA 2           // RS=1, R/W = 0 (Data Write)
#define READBUSY 1           // RS=0, R/W = 1 (Read busy status)



//---------------------------------------
// Send2LCD (...)
//---------------------------------------
// Author:  Nate Stoddard
// Date:    2/2/01
//---------------------------------------
// Outputs a int (or char) to LCD
//---------------------------------------
// Arguments:
// Data = (int) byte to transfer
// Sel = (defined) what is being writting
//          SENDINIT = 00, sending instructions
//      SENDDATA = 10, sending data to display
//---------------------------------------
// Return value:
//
//---------------------------------------
// Revisions:
//
//---------------------------------------
// REQUIRED: PORTE 0,1 = OUTPUT!
//
void Send2LCD (int data, int Sel);



//---------------------------------------
//    Wait4LCD ()
//---------------------------------------
// Author:  Nate Stoddard
// Date:    2/2/01
//---------------------------------------
// Waits for LCD to NOT be busy
//---------------------------------------
// Arguments:
```

```
//
//---------------------------------------
// Return value:
//
//---------------------------------------
// REQUIRED: PORTE 0,1 = OUTPUT!
//
void Wait4LCD();



//---------------------------------------
// InitLCD ()
//---------------------------------------
// Author:  Nate Stoddard
// Date:    2-2-01
//---------------------------------------
// Initializes the LCD
// - 2 lines (setup for 1/16 duty) 5x7 font
// - cursor, blink, display = on
// - clear display
//---------------------------------------
// Arguments:
//---------------------------------------
// Return value:
//---------------------------------------
//
void InitLCD();


//---------------------------------------
// OutputMessage (...)
//---------------------------------------
// Author:  Nate Stoddard
// Date:    2-2-01
//---------------------------------------
// Outputs a message to LCD
//---------------------------------------
// Arguments:
// message =      (char*) of message to display
//                        ending in NULL
//---------------------------------------
// Return Value:
//---------------------------------------
//
void OutputMessage (char *message);



//---------------------------------------
// ShiftWait ()
//---------------------------------------
// Author:  Nate Stoddard
// Date;    2-3-01
//---------------------------------------
// This function shifts the cursor as the user
```

```
//  selects switches.
//----------------------------------------
// Arguments:
//----------------------------------------
// Return value:
//----------------------------------------
// NOTE:  For testing ONLY!
//
void ShiftWait ();
```

#endif