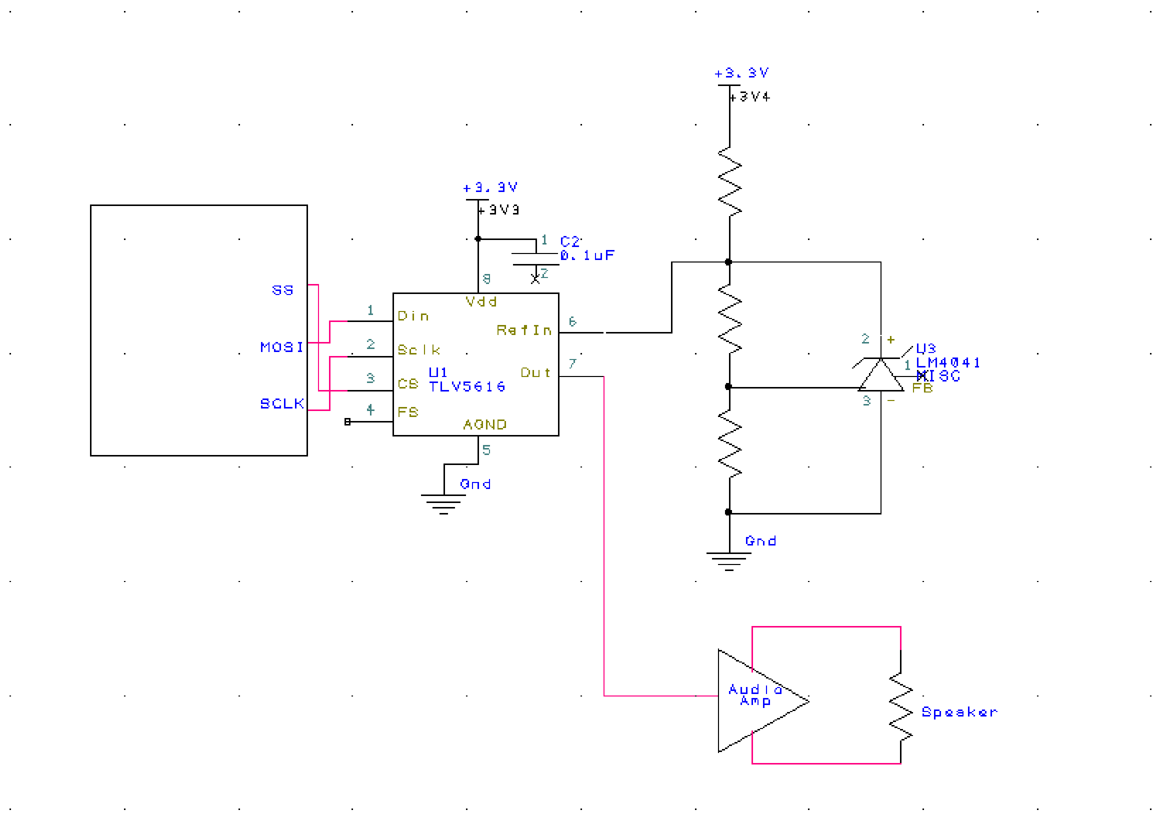
Lab 5 Prep

Cooper Carnahan and Faisal Mahmood



Requirements document

As always, feel free to adjust the syntax and format of your requirements document as you think appropriate. The goal of the document is to provide a clear an unambiguous description of what the project does.

1. Overview

1.1. Objectives: Why are we doing this project? What is the purpose?

The objectives of this project are to design, build and test a music player. Educationally, students are learning how to interface a DAC, how to design a speaker amplifier, how to store digital music in ROM, and how to perform DAC output in the background. Your goal is to play your favorite song.

1.2. Process: How will the project be developed?

The project will be developed using the TM4C123 board. There will be two or three switches that the operator will use to control the music player. The system will be built on a solderless breadboard and run on the usual USB power. The system may use the on board switches or off-board switches. A hardware/software interface will be designed that allows software to control the player. There will be at least three hardware/software modules: switch input, DAC output, and the music player. The process will be to design and test each module independently from the other modules. After each module is tested, the system will be built and tested.

1.3. Roles and Responsibilities: Who will do what? Who are the clients?

EE445L students are the engineers and the TA is the client. Students are expected to make minor modifications to this document in order to clarify exactly what they plan to build. Students are allowed to divide responsibilities of the project however they wish, but, at the time of demonstration, both students are expected to understand all aspects of the design.

1.4. Interactions with Existing Systems: How will it fit in?

The system will use the TM4C123 board, a solderless breadboard, and the speaker as shown in Figure 5.1. It will be powered using the USB cable. You may use a +5V power from the lab bench, but please do not power the TPA731 or the speaker with a voltage above +5V.

1.5. Terminology: Define terms used in the document.

Definitions for the terms SSI, linearity, frequency response, loudness, pitch, instrument, tempo, envelope, melody and harmony can be found in the textbook. ***(Note to students: add any addition terms you feel are needed)***

1.6. Security: How will intellectual property be managed?

The system may include software from StellarisWare and from the book. No software written for this project may be transmitted, viewed, or communicated with any other EE445L student past, present, or future (other than the lab partner of course). It is the responsibility of the team to keep its EE445L lab solutions secure.

2. Function Description

2.1. Functionality: What will the system do precisely?

If the operator presses the play/pause button the music will play or pause. If the operator presses the play/pause button once the music should pause. Hitting the play/pause again causes music to continue. The play/pause button does not restart from the beginning, rather it continues from the position it was paused. If the rewind button is pressed, the music stops and the next play operation will start from the beginning. There is a mode switch that allows the operator to control some aspect of the player. Possibilities include instrument, envelope or tempo. *(Note to students: if you use the internal switches you could rename the switches SW1 and SW2 to match the switches you use) (Note to students: specify exactly what your mode button does.)*

There must be a C data structure to hold the music. There must be a music driver that plays songs. The length of the song should be at least 30 seconds and comprise of at least 8 different frequencies. Although you will be playing only one song, the song data itself will be stored in a separate place and be easy to change. The player runs in the background using interrupts. The foreground (main) initializes the player, then executes **`for(;;){}`** do nothing loop. If you wish to include LCD output, this output should occur in the foreground. The maximum time to execute one instance of the ISR is xxxx *(note to students: replace the xxxx with performance measure of your solution)*. You will need public functions **`Rewind`**, **`Play`** and **`Stop`**, which perform operations like a cassette tape player. The **`Play`** function has an input parameter that defines the song to play. A background thread implemented with output compare will fetch data out of your music structure and send them to the DAC.

There must be a C data structure to store the sound waveform or instrument. You are free to design your own format, as long as it uses a formal data structure (i.e., **`struct`**). The generated music must sound beautiful utilizing the SNR of the DAC. Although you only have to implement one instrument, it should be easy to change instruments.

2.2. Scope: List the phases and what will be delivered in each phase.

Phase 1 is the preparation; phase 2 is the demonstration; and phase 3 is the lab report. Details can be found in the lab manual.

2.3. Prototypes: How will intermediate progress be demonstrated?

A prototype system running on the TM4C123 board and solderless breadboard will be demonstrated. Progress will be judged by the preparation, demonstration and lab report.

2.4. Performance: Define the measures and describe how they will be determined.

The system will be judged by three qualitative measures. First, the software modules must be easy to understand and well-organized. Second, the system must employ an abstract data structures to hold the sound and the music. There should be a clear and obvious translation from sheet music to the data structure. Backward jumps in the ISR are not allowed. Waiting for SSI output to complete is an acceptable backwards jump. Third, all software will be judged according to style guidelines. Software must follow the style described in Section 3.3 of the book *(note to students: you may edit this sentence to define a different style format)*. There are three quantitative measures. First, the SNR of the DAC output of a sine wave should be measured. Second, the maximum time to run one instance of the ISR will be recorded. Third, you will measure power supply current to run the system. There is no particular need to optimize any of these quantitative measures in this system.

2.5. Usability: Describe the interfaces. Be quantitative if possible.

There will be three switch inputs. The DAC will be interfaced to a 32-ohm speaker. *(note to students: you could use 8 ohm speaker)*

2.6. Safety: Explain any safety requirements and how they will be measured.

If you are using headphones, please verify the sound it not too loud before placing the phones next to your ears.

3. Deliverables

3.1. Reports: How will the system be described?

A lab report described below is due by the due date listed in the syllabus. This report includes the final requirements document.

3.2. Audits: How will the clients evaluate progress?

The preparation is due at the beginning of the lab period on the date listed in the syllabus.

3.3. Outcomes: What are the deliverables? How do we know when it is done?

There are three deliverables: preparation, demonstration, and report. *(Note to students: you should remove all notes to students in your final requirements document)*.

// Switch.h

// Initializes and handles switch input.

// Cooper Carnahan and Faisal Mahmood

// 10/04/17

// Lab 5

// Saadallah

// 10/04/17

/***

Switch_Init()

Initializes switch input to the TM4C123.

inputs:

none

outputs:

none

***/

void Switch_Init(int num, int interrupts);

/***

Switch_Handler()

Called when a switch is pressed, handles user input. Will either

call function Play(), Pause(), Rewind(), or Change_Mode()

inputs:

switchID - Identifier of which switch was pressed

outputs:

none

```
***************************************************************/
void Switch_Handler(int switchID);

/**************************************************************
Play()
	Called when switch 2 is pressed, handles user input. Will
	cause the music to begin playing

inputs:
	none

outputs:
	none
***************************************************************/
void Play(void);

/**************************************************************
	Rewind()
	Called when switch 1 is held, handles user input. Will
	cause the music to go back to the beginning and wait for
	user to begin playing again

	inputs:
	none

	outputs:
	none
***************************************************************/
void Rewind(void);
```

```
/************************************************************
Pause()
	Called when switch 1 is pressed, handles user input. Will
	cause the music to go back to the beginning and wait for
  user to begin playing again

inputs:
	none

outputs:
	none
************************************************************/
void Pause(void);
```

```
// MAX5353.h
// Cooper Carnahan and Faisal Mahmood
// 10/04/17
// Lab 5
// Saadallah
// 10/04/17
// Runs on LM4F120/TM4C123
// Use SSI0 to send a 16-bit code to the MAX5353.

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to Arm Cortex M Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2015

 Copyright 2015 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

// SSI0Clk (SCLK, pin 4) connected to PA2
// SSI0Fss (!CS, pin 2) connected to PA3
// SSI0Tx (DIN, pin 3) connected to PA5
// see Figure 7.19 for complete schematic
```

```
//********DAC_Init*****************
// Initialize Max5353 12-bit DAC
// inputs:  initial voltage output (0 to 4095)
// outputs: none
// assumes: system clock rate less than 20 MHz
void DAC_Init(uint16_t data);

//********DAC_Out*****************
// Send data to Max5353 12-bit DAC
// inputs:  voltage output (0 to 4095)
// outputs: none
void DAC_Out(uint16_t code);

//********DAC_Out2*****************
// Send data to Max5353 12-bit DAC
// inputs:  voltage output (0 to 4095)
// outputs: reply is returned
// send the 16-bit code to the SSI, return a reply
uint16_t DAC_Out2(uint16_t code);

//********DAC_InitRaw*****************
// Initialize Max5353 12-bit DAC with PA2,PA5
// with alternate functionality while PA3 is
// configured with regular functionality for
// FSS usage
// inputs:  initial voltage output (0 to 4095)
// outputs: none
// assumes: system clock rate less than 20 MHz
void DAC_InitRaw(void);
```

```
//********DAC_Out*****************
// Send data to Max5353 12-bit DAC
// inputs:  voltage output (0 to 4095)
// outputs: none
// send the 16-bit code to the SSI,
void DAC_OutRaw(uint16_t code);
```

```
// MAX5353TestMain.c
// Runs on LM4F120/TM4C123
// Test the functions provided in MAX5353.c by outputting
// a sine wave at a particular frequency.
// Daniel Valvano
// September 11, 2013

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to Arm Cortex M Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2014

 Copyright 2014 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

// SSI0Clk (SCLK, pin 4) connected to PA2
// SSI0Fss (!CS, pin 2) connected to PA3
// SSI0Tx (DIN, pin 3) connected to PA5
// see Figure 7.19 for complete schematic

#include <stdint.h>
#include "MAX5353.h"
```

```
#include "SysTick.h"

#include "Timer0A.h"

#include "Switch.h"

typedef struct noteStruct {
  int notePitch;                                  //What period to interrupt at
  int noteLength;                                 //Translates to eighth, quarter, half, full

           //(likely a 0-3 identifier)
} note;

typedef struct songStruct {
  note notes[50];                                 //Song max length of 50 notes
} song;

// 12-bit 32-element sine wave
// multiply each value by 2 to shift into bits 12:1 of SSI packet
// three control bits in 15:13 are all zero for immediate DAC update
// book figure shows MAX5353 in unipolar rail-to-rail configuration
// that means when wave[n] = 0x0000 (LSB = 0), output = 0
//              wave[n] = 0x1000 (LSB = 0), output = Vref
//              wave[n] = 0x1FFE (LSB = 0), output = 2*Vref
const uint16_t wave[32] = {
  2048*2,2448*2,2832*2,3186*2,3496*2,3751*2,3940*2,4057*2,4095*2,4057*2,3940*2,
  3751*2,3496*2,3186*2,2832*2,2448*2,2048*2,1648*2,1264*2,910*2,600*2,345*2,
  156*2,39*2,0*2,39*2,156*2,345*2,600*2,910*2,1264*2,1648*2};

int main(void){
  uint32_t i=0;
  DAC_Init(0x1000);               // initialize with command: Vout = Vref
```

```
  SysTick_Init();
  while(1){
    DAC_Out(wave[i&0x1F]);
    i = i + 1;
    // calculated frequencies are not exact, due to the impreciseness of delay loops
    // assumes using 16 MHz PIOSC (default setting for clock source)
    // maximum frequency with 16 MHz PIOSC: (8,000,000 bits/1 sec)*(1 sample/16 bits)*(1 wave/32
sample) = 15,625 Hz
    // maximum frequency with 20 MHz PLL: (10,000,000 bits/1 sec)*(1 sample/16 bits)*(1 wave/32
sample) = 19,531.25 Hz
//    SysTick_Wait(0);          // ?? kHz sine wave (actually 12,000 Hz)
//    SysTick_Wait(9);          // 55.6 kHz sine wave (actually 10,000 Hz)
//    SysTick_Wait(15);         // 33.3 kHz sine wave (actually 8,500 Hz)
//    SysTick_Wait(19);         // 26.3 kHz sine wave (actually 8,500 Hz)
//    SysTick_Wait(64);         // 7.81 kHz sine wave (actually 4,800 Hz)
//    SysTick_Wait(99);         // 5.05 kHz sine wave (actually 3,500 Hz)
    SysTick_Wait(1136);         // 440 Hz sine wave (actually 420 Hz)
//    SysTick_Wait(50000);      // 10 Hz sine wave (actually 9.9 Hz)
  }
}
```