Reimagining Orchestral Chimes Electronically

Project Presentation by Johnny Console

Overview

- Project Overview
- Project Motivation
- Materials Used
- How it Works
- Future Work
- Issues Faced
- Project Demonstration

Project Overview

- This project aims to reproduce orchestral chimes using a microcontroller chip
- Using the microcontroller's internal program memory, the sound data is stored on the chip
  - This can be improved in multiple ways, which will be discussed later
- The code for this project was written in the MPASM assembler for Microchip 8-bit Microcontroller Unit (MCU) chips

Project Motivation

- Orchestra quality chimes are very expensive (~$4,000 to $45,000 for a set)
- The parts for this project come out to about $200 for one chime.
- Comparison of these costs shows that this could be a cheaper alternative

Materials Used

- The materials used in this project include:
  - Two 40-hole breadboards (for prototyping),
  - One PIC18F47Q10 MCU (Microchip),
  - Two AT24CM02 flash memory chips (Microchip),
  - One DAC8571 digital to analog converter (DAC, Texas Instruments),
  - One 64-megahertz crystal oscillator,
  - Five ten-kilohm resistors
  - Two one-kilohm resistors
  - Eight LED, for debugging purposes
  - Eight 220-Ohm resistors, for debugging purposes
  - Jumper Wires

How it Works – An Overview

- The product will ship with a loaded sample
- The user can connect the product to a computer with Java installed and use the included uploading tool to serially change the sample
- Before uploading a new sample, the user must press the yellow download button to enable receiving of the sample data
- The sample is played by pressing the green play button
- The sample can be stopped, or dampened, by using the red stop button

How it Works – In Detail

- The sample is stored in the MCU program memory at address 0x600 until address 0xFA0
- When the user presses the download button, the first two 256-byte pages are loaded from the program table
- The MCU sends the page bytes, two at a time, to the DAC chip which converts their digital bit pattern to an analog data stream
- When a page has been sent to the DAC, the next page is loaded in parallel.

How it Works - In Detail

- When the user presses the dampening button, the sample stops immediately, and the device is prepared to begin playing.
- In the event of a timeout, the user can reset the MCU by pressing the white reset button. This sends a pulse to the MCU’s master clear pin.

How it Works – Serial Protocol

- The protocol begins when the user presses the upload button.
- The sample upload utility sends the first 256-byte block of data to the MCU.
- The MCU calculates the checksum of the data and requests it from the host by sending a “C”.
- The host sends the calculated 8-bit checksum and the MCU compares it to the one it calculated.
- If they are equal, the MCU places the block in the memory page and requests the next block by sending an “N”
- If they are not equal, the MCU discards the block and requests the host to resend it by sending an “R”
- Steps 2 to 5/6 repeat until all blocks are successfully received. At this point, the MCU sends an “X” to end the protocol.

How it Works – I2C Protocol Overview

- An I2C protocol transmission consists of a start condition, a device address byte, data bytes, acknowledgements/non-acknowledgements and a stop condition.
- The start condition is resembled by a transition from logic high to logic low while the clock line is active.
- Similarly, the stop condition is resembled by a transition from logic low to logic high.
- The device address byte format, as well as the number of data bytes is dependent on the device being used and the operation being performed.
- The device sending commands is known as the master, and the device sending data is known as the slave.

How it Works – I2C Protocol Overview

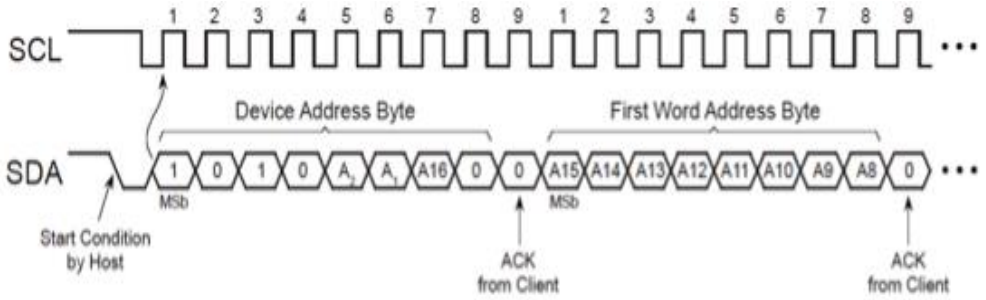
- When sending to the DAC, the MCU acts as a master – sending commands and data to the DAC, and the DAC acts as a slave – performing operations on the commands and data sent by the MCU.
- With the flash memory, the master-slave assignment changes depending on what operation is being performed.
- For a write operation, the MCU is the master, and the flash memory is the slave
- For a read operation, the MCU becomes the slave, and the flash memory becomes the master.

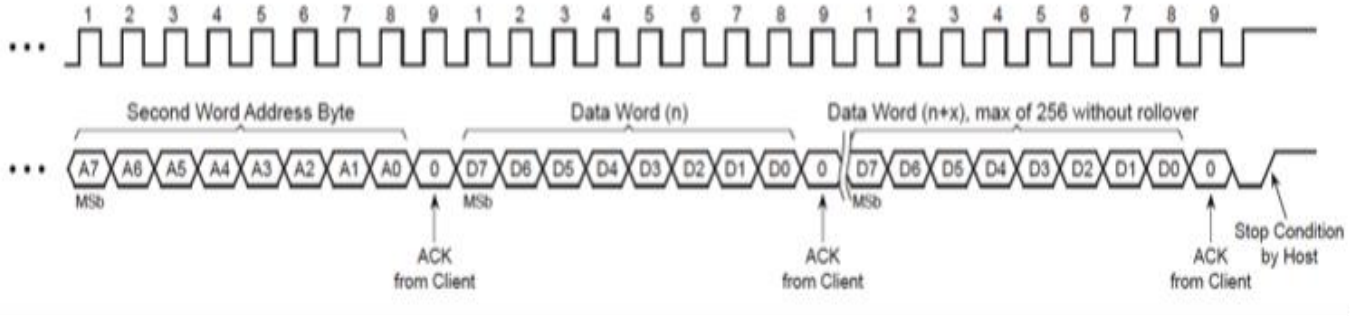
How it Works – I2C Protocol In Detail

Transmitter	Data								Comment
Master	Start Condition								Begin Sequence
Master	1	0	0	1	1	A0	0	RW	Address
Slave	Acknowledge								
Master	0	0	0	0	0	0	0	0	Control
Slave	Acknowledge								
Master	15	14	13	12	11	10	9	8	Data MSB
Slave	Acknowledge								
Master	7	6	5	4	3	2	1	0	Data LSB
Slave	Acknowledge								
Master	Stop Condition, or Data MSB and LSB								End Sequence

- The master creates a start condition
- The master sends the device address byte: 1 0 0 1 1 A 0 RW
- Slave Acknowledges
- Master sends the control byte: 0 0 0 0 0 0 0 0
- Slave Acknowledges
- Master sends MSB to be converted
- Slave Acknowledges
- Master sends LSB to be converted
- Slave Acknowledges
- MSB/ACK/LSB/ACK can repeat until done
- Master creates a stop condition to end the sequence

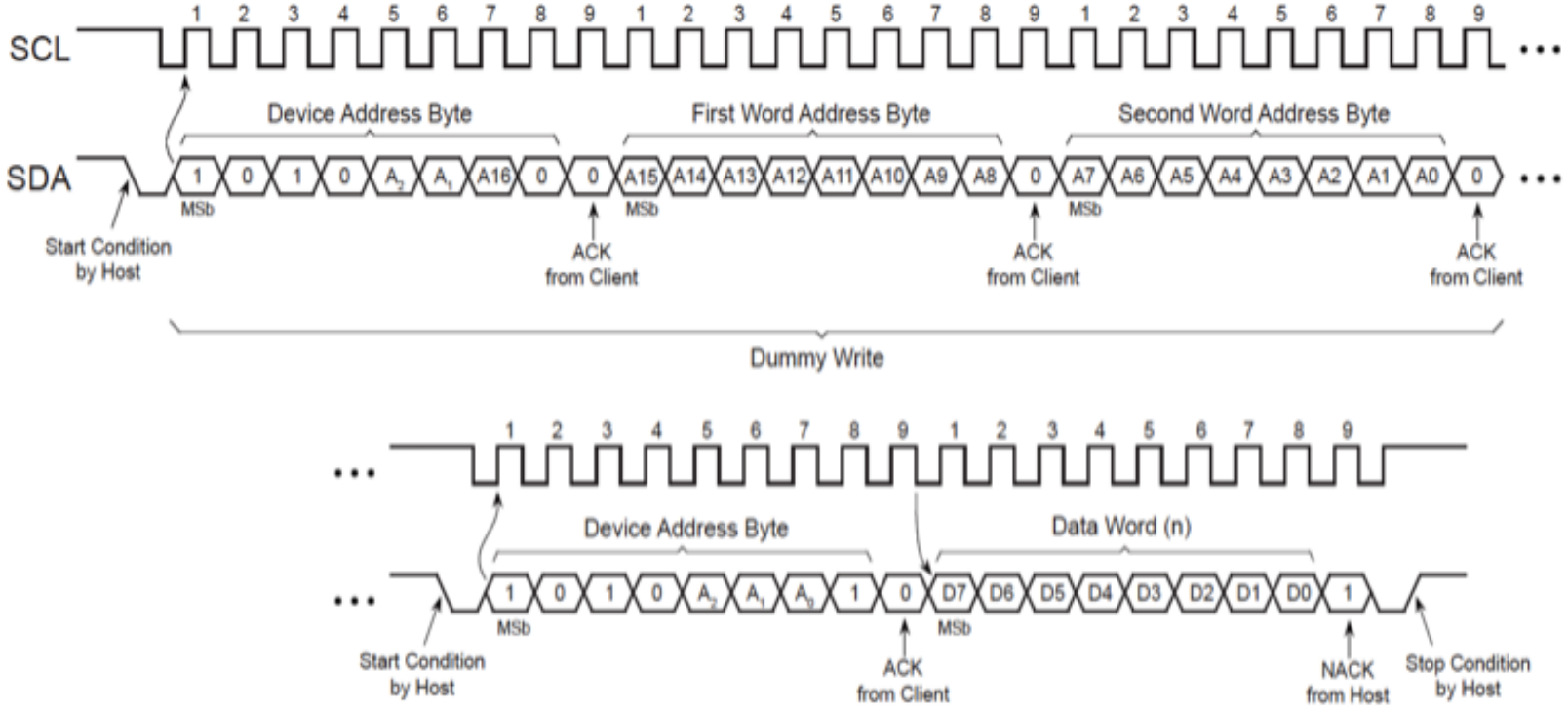
How it Works – I2C Protocol In Detail





- This is a timing sequence diagram
- Note the clock line and data line
- Writing to the flash ram

How it Works – I2C Protocol In Detail



- Random read from the flash ram

Future Work

- The device as presented is great, but it would be nice to have multiple samples on each MCU to reduce costs.
- This can be done by adding the flash memory chip, or multiple flash memory chips to the setup and running them on a separate I2C port to the DAC.
- However, this adds complexity to get the sample data to the flash memory from the serial port, and from the flash memory to the MCU memory.
- This was the original approach, but due to time constraints was unable to be implemented correctly.

Future Work

- It is also planned to replace the play button with a length of PVC pipe with a pressure switch that can be struck with a small mallet to trigger playing the sample.
- This will make the device appear closer to an orchestra chime setup.
- With this, the dampen button would be replaced with some sort of foot switch, such as those used by guitar players to add sound effects.

Issues Faced

- Learning a new Assembler
  - The MPASM Assembler is much different than MASM in x86 systems
  - Before working on the project, I did some simple IO manipulation to get used to the assembler
  - Reading the instruction set summary in the MPU datasheet also helped with instruction formats
- Serial Port Programming
  - Setting up the serial port was difficult
  - Example code was researched to overcome this

Issues Faced

- Interrupt-Driven Programming
  - Once the serial port code was finished and working, it was made interrupt-driven.
  - This was difficult, but with example code from the MPU datasheet, this was also overcome.
- I2C Programming
  - Programming one port was difficult, but example code was used to help with this
  - Programming both ports was extremely difficult
  - At times, the entire program would stop working
  - This was solved by reseating the pull up resistors on the clock and data lines of both ports

Issues Faced

- Flash RAM issues
  - When planning the project, the idea was to use flash ram chips to store the sample
  - This was difficult to read from because on occasion the flash ram would send the incorrect block
  - This is the reason the project was adapted to use the MCU's program memory table functions to read/store the sample
- Issues with writing to the Microcontroller's program flash memory
  - The process of writing to the program memory is complex: a sector must be read, erased and modified, in that order, with an unlock sequence performed before both steps.

Project Demonstration