ECE 353 Lab 4

General MIDI Explorer

Professor Daniel Holcomb
Fall 2015
Where are we in Course

- Lab 0  Cache Simulator in C
  - C programming, data structures
  - Cache architecture and analysis

- Lab 1  Heat Flow Modeling
  - C programming, data structures

- Lab 2  Pipelined Machine
  - C programming, data structures
  - Pipeline architecture and analysis

- Lab 3  MIDI Receiver
  - Hardware design
  - Proper Verilog coding
  - Functional Simulation, GoLogic, MIDI-Ox

- Lab 4  General MIDI Explorer with Record/Playback
  - Microcontroller programming in C
  - Breadboarding and Debugging
Microprocessors, Microcontrollers, Programmable Logic

- Microprocessors are higher performance capable and larger memory
  - From 50MHz to GHz
- FPGA systems often contain CPUs in softcore (synthesized) or hardcore (part of die) format but can also contain logic blocks for other hardware, e.g., state machines, etc
  - 45-65 nm FPGAs (Virtex 5-7) can achieve high performance.
- Microcontrollers are more limited in functionality and often do not include support for virtual memory and caches
  - Up to 50MHz
Lab 4 Objectives

- Build a complete system
- Record MIDI packets from computer when prompted
- Store on breadboard
- (modify them)
- Playback to computer when prompted
- Hear music

Record → ATmega32 → Computer

Playback → MIDI → Computer
Lab 4 Objectives

- Exposure to microcontroller programming vs. design of hardware
  - Compiled Language (C)
  - Data Structure in Memory
  - Low level interaction with hardware components
- Continuation of MIDI theme
  - Serial communication
  - Notes and instruments
- Intended to be a FUN project
ATmega32 AVR

- 16MHz CMOS 8-bit microcontroller with AVR RISC instruction set
- 32 general purpose registers
- 32KB of Flash
- 2Kb internal SRAM
- 1KB E2PROM – 100k erase cycles
- 8 and 16 bit counters
- USART/SPI/I2C
- 8-channel 10-bit ADC
- JTAG
- 4 8-bit I/O ports
- mW power
  - Power-on Reset
    - Circuitry that detects when power-on and resets all registers
    - Different than Lab3
Flexible Tool. Many Configurations. Use Carefully

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(XCK/T0)</td>
<td>PB0</td>
<td>1</td>
<td></td>
<td>PA0</td>
</tr>
<tr>
<td>(T1)</td>
<td>PB1</td>
<td>2</td>
<td>40</td>
<td>PA1</td>
</tr>
<tr>
<td>(INT2/AIN0)</td>
<td>PB2</td>
<td>3</td>
<td>39</td>
<td>PA2</td>
</tr>
<tr>
<td>(OC0/AIN1)</td>
<td>PB3</td>
<td>4</td>
<td>38</td>
<td>PA3</td>
</tr>
<tr>
<td>(SS)</td>
<td>PB4</td>
<td>5</td>
<td>37</td>
<td>PA4</td>
</tr>
<tr>
<td>(MOSI)</td>
<td>PB5</td>
<td>6</td>
<td>36</td>
<td>PA5</td>
</tr>
<tr>
<td>(MISO)</td>
<td>PB6</td>
<td>7</td>
<td>35</td>
<td>PA6</td>
</tr>
<tr>
<td>(SCK)</td>
<td>PB7</td>
<td>8</td>
<td>34</td>
<td>PA7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>VCC</td>
<td>32</td>
<td>AREF</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GND</td>
<td>31</td>
<td>GND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>30</td>
<td>AVCC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>XTAL2</td>
<td>29</td>
<td>PC7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>28</td>
<td>PC6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XTAL1</td>
<td>27</td>
<td>PC5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td>26</td>
<td>PC4</td>
</tr>
<tr>
<td>(RXD)</td>
<td>PD0</td>
<td>14</td>
<td>25</td>
<td>PC3</td>
</tr>
<tr>
<td>(TXD)</td>
<td>PD1</td>
<td>15</td>
<td>24</td>
<td>PC2</td>
</tr>
<tr>
<td>(INT0)</td>
<td>PD2</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(INT1)</td>
<td>PD3</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(OC1B)</td>
<td>PD4</td>
<td>18</td>
<td>23</td>
<td>PC1</td>
</tr>
<tr>
<td>(OC1A)</td>
<td>PD5</td>
<td>19</td>
<td>22</td>
<td>PC0</td>
</tr>
<tr>
<td>(ICP1)</td>
<td>PD6</td>
<td>20</td>
<td>21</td>
<td>PD7</td>
</tr>
</tbody>
</table>

www.wengerna.com
Lab 4 Requirements

- Part 1:
  - Send & Receive MIDI Messages
  - Play and Recording Modes
  - Store and Replay MIDI Messages

- Part 2: (optional for extra credit)
  - Use hex switch and photo cells
  - Vary speed of messages or repetition
  - Change note or velocity

HW Components

- Data EEPROM (Storage)
- USART (Communication)
- Timer1 (Counter)

- More GPIO
- ADC (Analog to Digital Converter)
Record and playback switches not shown
Required Part of Lab:

- Develop, program AVR through JTAG, for:
  - Getting MIDI messages through MIDI OX from the PC serially
  - Similar to Lab 3 but using software
  - AVR Studio and AVR gcc compiler
  - Configure USART to read MIDI bytes

- Once messages received in AVR
  - Store in E2PROM if switch set to Record
  - Compress before storing, decompress after
  - Use any algorithm you find suitable
  - **Record timing** also so that you can replay exactly the same way

- Playback when prompted
Compression

- Why compress?
  - Store more data in same persistent memory
  - Persistent data limited in # of erase cycles
  - Save on bandwidth when communicating

- Lossless compression (LZ, Huffman, RLE):
  - Decompress and get the same data, bit - for - bit

- Lossy compression (JPEG,…):
  - Decompress and get something similar

- Tradeoff between quality and compression

- Domain specific data patterns can be exploited
Run Length Encoding

- Simple lossless encoding of repeating patterns
  - Example: Imagine a stream of bits with many 0s and 1s
  - Source: 000000000111111110001111111
  - RLE Version: 90813071

- Limit maximum sequence value to avoid too much overhead for very random sequences.
  - limit to maximum 16 repetition of 1s or 0s: represent with 4 bits
    - 1001(0)1000(1)0011(0)0111(1) for above sequence
  - Compression ratio is 27/20

- What is best possible compression ratio in this scheme? Provide sequence that realizes this ratio
- What is the worst possible compression ratio in this scheme? Provide a sequence that realizes this ratio
Pseudocode of a Possible Implementation

Configure timers/USART/interrupts/etc
While (1) {
    If (recording) {
        USART_Read()
        Compress
        EEPROM_Write()
    }
    If (playing) {
        EEPROM_Read()
        Decompress
        USART_Write()
    }
}
Figure Out How To…

- Initialize the USART
  - Set baud rate for MIDI (remember clock divider from lab 3?)
  - Enable the receiver and transmitter
  - Initialize to 8 data bits, 1 stop, and 1 stop bit

- Create functions for receive, transmit, and flush USART

- Create functions to read and write E2PROM

- Work with timers and interrupts
  - Use 16-bit counter and generate interrupts for overflows; need to setup ISR (interrupt service routine)

- Main function

- Can use LEDs to debug, try something simple first

- **ATmega datasheet is your friend in this lab**
  - All information needed is found there, including sample code
Using the USART

- UCSRA/UCSRB/UCSRC (USART Control Status Register A/B/C)
- UBRRH/UBRRL (USART Baud Rate Register High/Low)
- Lots of bit-set and bit-check operations

**USART**

The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a highly flexible serial communication device. The main features are:

- Full Duplex Operation (Independent Serial Receive and Transmit Registers)
- Asynchronous or Synchronous Operation
- Master or Slave Clocked Synchronous Operation
- High Resolution Baud Rate Generator
- Supports Serial Frames with 5, 6, 7, 8, or 9 Data Bits and 1 or 2 Stop Bits
- Odd or Even Parity Generation and Parity Check Supported by Hardware
- Data OverRun Detection
- Framing Error Detection
- Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter
- Three Separate Interrupts on TX Complete, TX Data Register Empty, and RX Complete
- Multi-processor Communication Mode
- Double Speed Asynchronous Communication Mode
Suggestions for building project incrementally

- Write simple sketch of code
  - Verify that playback/record switches lead to the expected points in code
- Verify that USART receives MIDI correctly
  - Use watch window or light up LEDs (just like lab 3)
- Verify that USART correctly sends MIDI to PC
  - Start by sending hard-coded constants
- Verify functions for reading and writing EEPROM
  - Simple test routine applying a few values
- Then worry about compression
# Example Code

```c
// blinky.c
#include <avr/io.h>    // Standard AVR header
#include <avr/delay.h> // Delay loop functions

int main(void)
{
    DDRA = 0xFF;    // PORTA is output
    while (1) {
        for (int i=1; i<=128; i*=2) {
            PORTA = i;
            _delay_loop_2(30000);
        }
        for (int i=128; i>1; i/=2) {
            PORTA = i;
            _delay_loop_2(30000);
        }
    }  // end while
}
```
Example Code

#include <avr/io.h>

int main(void) {
    int data;
    DDRA = 0xFF;         // (PORTA output)
    DDRB = 0x00;         // (PORTB input)
    PORTA = 0x55;        // (Output 0x55 or 0b01010101)
    data = PINB;         // (Store input values from port B)
}

Port A Data Direction Register – DDRA

<table>
<thead>
<tr>
<th>Bit</th>
<th>DDA7</th>
<th>DDA6</th>
<th>DDA5</th>
<th>DDA4</th>
<th>DDA3</th>
<th>DDA2</th>
<th>DDA1</th>
<th>DDA0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
<tr>
<td>Initial Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Port A Input Pins Address – PINA

<table>
<thead>
<tr>
<th>Bit</th>
<th>PINA7</th>
<th>PINA6</th>
<th>PINA5</th>
<th>PINA4</th>
<th>PINA3</th>
<th>PINA2</th>
<th>PINA1</th>
<th>PINA0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read/Write</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>Initial Value</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Optional Part: Integrate Additional Inputs

- Use three optical sensors connected to ADC inputs on AVR
  - Move hand over sensor to change characteristics of playback
- Use 4-bit digital input from rotary hex switch
- What to do with these new inputs?
  - Change note and velocity MIDI bytes
  - Speed up or slow down playback
  - Repeat notes during playback
  - Or anything else you can think of...
- Make sure to leave room on board if planning to do optional part of lab
General Advice

- Consult the Atmega32 datasheet
  - Pin layouts
  - Code examples
  - Everything you need to manipulate the microcontroller for this project is here!
- Use the AVR studio debugger
  - Check the value of pins and registers
  - Single step through code to find source of problems
- Debug with logic analyzer and oscilloscope
  - Check frequency of USART outputs on scope
- Use internet resources
  - avrfreaks.net is a good resource for examples
- Start early and get help early if stuck
Build Process

- Board Assembly
- C Programming
  - WinAVR
  - AVR Studio (IDE)
  - JTAG Programmer
- Testing
  - In-Circuit Emulator
  - MIDI-OX
  - GoLogic Logic Analyzer
  - Oscilloscope
WinAVR

- WinAVR is a set of development tools for Atmel AVR RISC microprocessors
- Programs written in C, compiled with GCC and avr-libc
- Open source, can be obtained at: winavr.sourceforge.net
AVR Studio Demo: Getting Started

- Open AVR Studio
- Select AVR GCC for the project type
- Type in a project title and choose project directory if necessary
- Click next and select the debugging platform (JTAG ICE) used to program the MCU as well as the MCU that you are using (ATmega32)
- Finish
AVR Studio Demo: Configure Project

- go to Project --> Configuration Options
- Choose the MCU clock frequency
- Under custom options, should see WinAVR for avr-gcc and make
- May need to add include directories and libraries if not seeing any external dependency files listed in project
AVR Studio Demo: Program Device

- Start Coding, then…
- 1. build your project, and
- 2. connect to the MCU

- Once you connect to your MCU, erase the device and then load to flash memory the .hex file created in your project directory
- Use external clock (our 4MHz oscillator)

- Now ready to step through code!
In Circuit Emulator

- Debug code while it runs on ATmega32
- Step through code one instruction at a time
- Breakpoints
- Can view current state of all registers using watch windows