University of Massachusetts Amherst
Department of Electrical and Computer Engineering
ECE 353, Computer Systems Lab 1
Fall 2017

Lab 4: MIDI Receiver in Verilog

Associated lectures:
Lab assignment and demonstration, Verilog HDL

Lab objectives:
Gain experience designing simple sequential and combinational logic in a hardware description language
Gain experience working with simple Verilog modules, and a basic hardware design flow
Gain experience simulating and debugging hardware designs

1. Introduction

In this lab you will design and implement with programmable logic a serial input port for MIDI (Musical Instrument Digital Interface) data. MIDI messages are described in the Lab 2 assignment. As we know, MIDI messages are transmitted *asynchronously,* as groups of bytes. Each byte is preceded by one START bit and followed by one STOP bit in order to synchronize reception of the data, as shown in Figure 1. A typical MIDI message is composed of three bytes. The second byte contains the *note number*; it can specify 128 different musical note numbers, spanning about 10 octaves. In this assignment, you are going to store and display only the 7 least significant bits of the second byte, which is the note number, in binary.

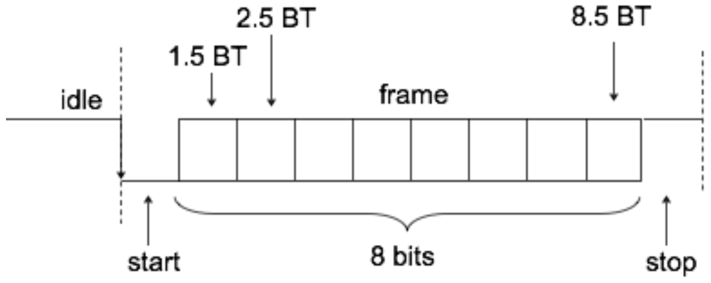

Figure 1. Timing of a MIDI byte

2. Project Description

You are to design a serial port for the MIDI device that will read a MIDI signal, interpret its content and display the note number (from the second byte) in binary on seven LEDs. The note number will remain on the LEDs only as long as the note remains on. The MIDI signal will come from the PC via MIDI OX as in Lab 2. The notes played on the computer's keyboard will cause MIDI data to be sent serially out the MIDI OUT connector. This signal will be connected to a MIDI IN connector on your breadboard. The input stage of your circuit is the same as in lab 2, and uses an optical isolator device to ensure that there is no direct electrical link between the devices. The output from the isolator, which was received on the RX pin of the microcontroller in lab 2, will in this lab be received on a pin of the programmable logic device.

3. Design Specification

The MIDI standard specifies a uni-directional serial interface running at 31,250 bits/s ±1%, a convenient division of the 4MHz clock you will use in this lab. In lab 2 the serial communication was handled by the USART module on the microcontroller, and your program interacted with this module using software. In lab 4, you are designing the hardware for a receiver module. Your module is customized for receiving MIDI, and is therefore simpler and less general than the microcontroller's USART. Specifically, the hardware you design will only operate in receive mode, and only receives bytes that match the MIDI format -- baud rate of 31,250 with a single start bit and stop bit framing each byte.

For transmission at 31,250 bits/s, each bit persists on the line for 32 µs, called the bit time (BT). The addition of the start bit and stop bits means that each MIDI byte takes 10 bit periods to transmit. However, the consecutive 10-bit frames may be separated by no time at all or an idle time of an undefined duration. Your design must not make the assumption that a stop bit is followed immediately by a start bit.

Before each arriving MIDI byte, the signal line idles high (1). The receiver monitors the line, waiting for the signal to drop to 0. Once the negative-going transition is detected, the receiver synchronizes on this transition and starts sampling the incoming serial stream. Conceptually, the receiver reads the 8 bits of the serial data by sampling the input "in the middle" of each bit, i.e., at 1.5 BT (bit0), 2.5 BT (bit1), …, 8.5 BT (bit7), as shown in Figure 1. The STOP bit could be sampled at 9.5 BT but in this lab you need not sample the stop bit.

The simple receiver that you design will lack many of the reliability checks that exist in a real USART. Typically microcontroller USART modules sample the data at a rate 16x or 8x of the transmitted message. Several samples would be made for each bit and "voting" would be performed to minimize the chance of a sampling error. Furthermore, if the sampling at 9.5 BT (the STOP bit) does not produce the expected high value, a real USART receiver would set a flag to indicate a *framing error*. In this assignment you are not required to implement voting or framing error detection, but without these safety features in place you must be careful to sample the bits correctly.

4. **Implementation**

In this lab you will design a hardware version of a serial MIDI receiver. You'll find the schematic in the slides on the course site. The design is to be implemented in Altera's Complex Programmable Logic Device (CPLD) MAX 7000S. Specifically, you will use a device from the MAX 7000S family with part number EPM7064SLC44-10. Input to the Altera device is the single-bit input line from the MIDI interface cable through the opto-isolator circuit. The Altera chip will be clocked by a 4MHz crystal oscillator, from which you may want to derive local clock for sampling (or, alternatively, count the faster clock to make sure you sample in the middle of each bit). Output pins of the device will drive seven LEDs to display the note number of the note played on computer keyboard in binary.

Since two consecutive 10-bit frames may be separated by an unknown amount of idle time, you must devise a synchronization scheme whereby a START bit is detected for each byte of the three bytes messages separately. You cannot rely on synchronization of the START bit of the first byte only. Furthermore, since the MIDI signal is transmitted asynchronously, sampling it by the receiving synchronous device may violate the setup or hold time of the input flip-flop, which may result in metastability. A simple approach to avoid metastability in ICs is to add an additional flip-flop in the design to synchronize the incoming asynchronous signal with the new clock domain, as shown in lecture slides.

You will use Altera's Quartus II 9.1 software to design, simulate, synthesize and program your design on the MAX 7000S programmable logic device. The student version of the software is provided on the computers in Duda hall; it is recommended that you install the software on your own laptop. You will use Verilog as the hardware description language for your design, and will simulate it using Quartus II waveform simulator. A short tutorial on how to use Quartus is available in the lecture slides and video on the class website; a complete tutorial is available in the Help menu of the Quartus II software. We will give a lecture on Verilog, and there are numerous resources and examples of Verilog code available online.

To receive full credit in this lab your design must use multiple modules, so think about how to decompose your design. You should plan out the modules of your design and their connections before you start writing any Verilog code. Be sure to include only non-blocking assignments (and not combinational logic) in your sequential blocks, following the style indicated in the Verilog lecture slides and in the example Verilog code provided on the course webpage.

It is important that you simulate your design thoroughly before programming the chip and wiring up the board. After simulating the design, you will program it using Altera's USB-based JTAG programmer provided in your lab kit. You will also use a logic analyzer to analyze the design when running on the breadboard.

5. Demonstration and Report

Each group will demonstrate their design in the lab, and submit online the following three files by the due date: (1) project report, (2) self-completed assessment form, (3) Verilog code. The report should describe the contributions of each group member, and answer the questions in parts 1,2, and 3 below.

Apply the following settings in Quartus before completing parts 1 and 2. These settings will hide details of your combinational logic, and allow you to more easily distinguish combinational and sequential parts of the design.

[select **Tools**]
  ->[select **options**]
    ->[select **netlist viewers**]
      ->[select **RTL viewer**]
        ->[select **group combinational logic into logic cloud**]

[select **Assignments**]
  ->[select **settings**]
    ->[select **analysis & synthesis settings**]
      ->[click on **more settings**]
        ->[set **Extract Verilog State Machines** to **OFF**]

Part 1: Introduction: Synthesize the finite state machine (fsm.v) that is given on the course website.

- Within RTL Viewer ([Tools]->[Netlist Viewers]->[RTL Viewer]), find the fsm module in the hierarchy and expand its primitives to see how many registers and/or latches are in the synthesized design (if there are no latches, you won't see latches listed under primitives.)
- Modify the FSM design by removing the default case from the case statement. Now there is no value being assigned to state_nxt when state has a value of 2'b11.
  - o Report the number of registers and latches before and after the modification.
  - o Write a few sentences to explain why removing the default case leads to a different result. You may want to reference the Verilog lecture slides and the synthesis warnings of the modified design for hints.

Part 2: Design of MIDI Receiver:

- Include a screenshot of simulation results from Quartus showing a MIDI signal on the input pin and the LEDs lighting up when the second byte arrives. Annotate your design, and explain how you know that it works correctly or does not work correctly based on what the simulation shows.
- Include a screenshot of logic analyzer results. The logic analyzer printouts should include text annotation to indicate what value is being displayed on the LEDs, and should show the LEDs turning on and subsequently turning off.
- Describe the hierarchy of instantiated modules in your design. What is the top level module? What modules are instantiated within it? what modules are instantiated within those (if any)?
- Include a screenshot of the schematic diagrams shown by RTL Viewer for your design. This should use the same settings given at top of page. Show one schematic for the top level design, and show an additional schematic for each additional module instantiated in your design.
- Include a table that lists all registers used in the design. Indicate what module instance contains each register, the register name, the number of bits, and a short description of its purpose. The example table below is the table for the fsm module from part 1. State the total number of registers used in the entire design. Following good Verilog practices (separating combinational and sequential parts as shown in fsm.v) should make it easier to understand your registers.
  - * *Bonus points*: It is often desirable to minimize the number of registers in a design. The groups that use the minimum number of registers will receive bonus points. To be eligible for the bonus, the design must obey good practices by including two registers for preventing metastability on the MIDI line (that will be counted toward total), and must not include any latches. Groups achieving the minimum number of registers will get 5 points. If two or fewer groups achieve the minimum, then the bonus is increased to 10 points. If a group already has full credit on lab 2 and lab 4, they will not receive bonus points beyond full credit.

Instance	Register Name	Number of bits	Purpose
fsm (top)	State	2	Current state of FSM

Part 3: Robustness to clock imprecision: In lab, your design uses a 4MHz clock. Imagine what would happen if your 4MHz clock runs slightly faster than 4MHz. What is the highest clock frequency for which your design would still be able to always read the MIDI signal correctly? Show your calculation, and explain what would happen when that frequency is exceeded. You may want to validate your calculation with simulations using different clock frequencies, but simulation results are not required in the report.