* 12/28/20
* 12/29/20
* 12/30/20
  + I connected the first 5 bits of the 6502 data bus to the Arduino, all 8 data pins, the RW pin, and the clock pin. I added an external “wall wart” type 5V power supply for the 6502 breadboard, as the Arduino would shut down from the high power draw from its 5V pin. The LEDs were still connected to the address bus.
  + I wrote an Arduino sketch that controlled the clock, sent instructions over the data bus, and read the data and address busses. The final version of the sketch was called 6502\_driver\_4. The program waited for a reset sequence (by checking for the bytes 0xC and 0xD (the end of 0xFC and 0xFD reset vectors) on the data bus, and sending the address 0xF100 over the data bus). After the reset sequence was over, the program sent instructions (in an array in the sketch) over the data bus (there were debug printouts to see what). Whenever the 6502 wrote to memory (/RW pin is low, and then the Arduino sets the pinMode to input, the /RW pin is set when PHI2 is low, data is read/wrote when PHI2 is high), the sketch would write the 8-bit ASCII character on the data bus over the serial port.
  + I chose to use the Arduino as a simulated ROM instead of an actual EEPROM because I didn’t want to buy an EEPROM programmer and it would reduce the number of wires and components needed (cleaner design). Designs with just a microcontroller, MPU, and RAM are known as 3-chip designs, although at this point it was a 2-chip design as I had no RAM. This has been done with the Z80 as well as the 6502.
  + However, there was a problem. No matter how many STA instructions I tried, I could never get the 6502 to write anything to the data bus. I checked the connection of the /RW pin and it was fine. I checked the clock (PHI2) timing, and it was also fine.
  + I then decided to just write random data to the data bus and see if the 6502 would ever write to the data bus, and it did. I then checked the data bus lines and discovered 2 were flipped. I fixed the flipped wires, and then my program worked just fine! It simply wrote “Hi” to the serial port. I turned off the debug print statements, and I saw the 2 characters “H” and “i” written to the screen.
* 12/31/20
  + I ordered some 22AWG hookup wire, so I wouldn’t have the long and messy DuPont jumper wires everywhere. I’m going to use this hookup wire for more permanent connections.
* 1/18/21
  + I attached an SRAM chip to the board, and hooked up the power and data pins (no address lines yet).
* 1/20/21
  + I decided to start making my “loader program” from the 6502. The Arduino would no longer depend on the address bus from the 6502, but rather read and write from the data bus only. In essence the Arduino would send commands to the 6502 to upload a program to the SRAM. I made a sketch that sent each byte of the program to the 6502 to run. I made a simple ‘Hello World’ 6502 program, but it didn’t work at first. I had to add a NOP statement between each block of LDA and STA statements. This is because the Arduino didn’t know the address bus of the 6502, so it couldn’t dynamically read from memory locations. I gave a grace period of 2 seconds for the user to reset the MPU, before the Arduino started sending code to the 6502. The code usually worked fine, but sometimes there were errors. Sometimes it didn’t print ‘Hello World’, but that text with most of the letters changed or altered. This usually happened when I reset the MPU, but rarely when the MPU wasn’t reset. I checked and reseated the resistor on the 6502’s reset pin. It didn’t change much. I kept the Arduino serial monitor open, and kept pressing the reset button on the Arduino to check for errors, which were far more common after resets. I then decided to add a string of NOPs before we executed the actual code, which also didn’t help. I thought of the idea of adding some kind of checksum for the program to automatically retry sending the program to the 6502. I finally decided to add a string of NOPs on the end of the 6502 program, and increase the length of the program (increasing the program length variable in the sketch). This seemed to have solved the problem, as no more errors are occurring, even after reset. I may still implement the checksum idea. Actually, sometimes there are still errors after reset. Maybe this didn’t eliminate any errors, but it did seem to reduce them. I also tried pushing in the data lines more, but it wouldn’t make sense for the problem to be a data line connection considering the correlation with the reset sequence.
  + It also seems like if I hold the reset jumper on the negative rail for a solid second, instead of very quickly, errors seem to be reduced.
  + Some examples of errant strings: “@eddo Wo2($”, “Hehhj Ejbh`”, “dpWorld”.
  + Arduino has a built-in checksum function, CRC16. I will probably use that.
  + My next step is a byte loader program, so I don’t have to manually add the LDA and STA instructions, saving space on the Arduino sketch.
  + Another idea I have is to control the reset pin with the Arduino itself. This would probably fix the errors.
* 1/21/21
  + My final plan for this system is to have the 65c02 and the SRAM on a