Tiffany Q. Liu
April 18, 2011
CSC 270
Lab #11

Lab #11: Building a 1-Bit Input I/O Controller

Introduction

For this lab, we worked towards building a circuit with the 6811 kit that acts as a 1-bit input I/O controller. This lab assumes experience from a previous lab of an assembled, designed, and fully tested 1-bit output controller. It is important to note that steps in this lab are to be followed carefully for a design flaw could kill components in the kit.

Materials

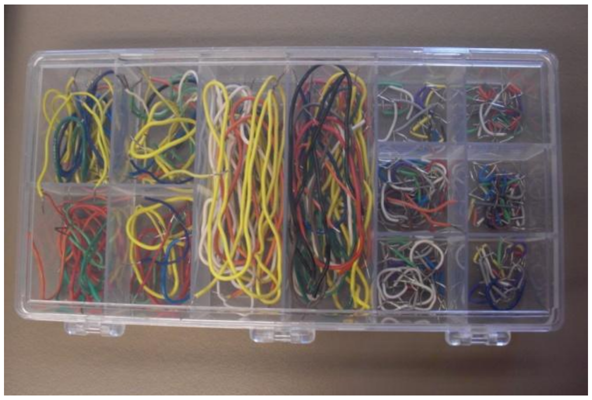


Figure 1. Wiring Kit.

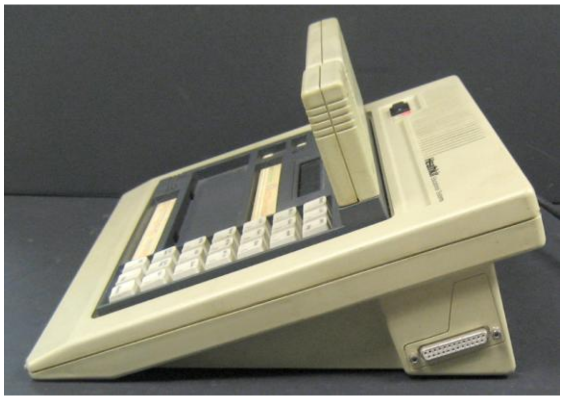


Figure 2. 6811 Microprocessor Kit (Taken from D.Thiebaut).

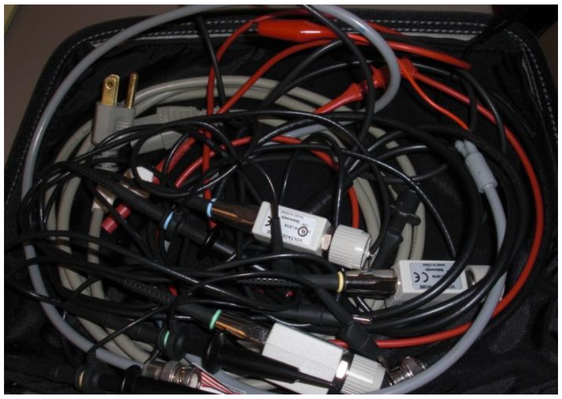


Figure 3. Oscilloscope Cables.

Figure 4. Tektronix Oscilloscope

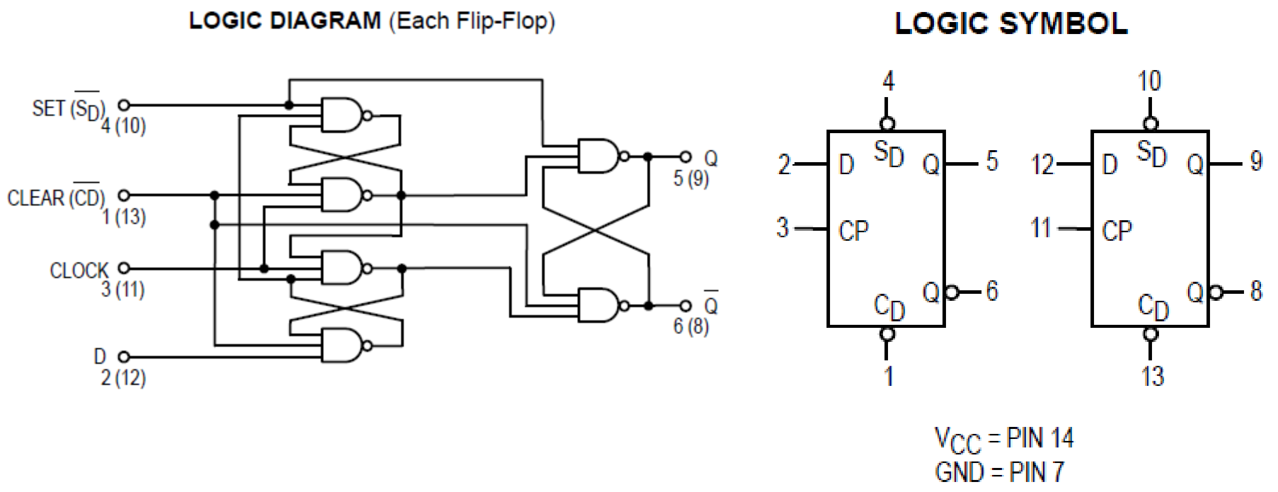

LOGIC DIAGRAM (Each Flip-Flop)
SET ($\overline{S_D}$) 4 (10)
CLEAR ($\overline{C_D}$) 1 (13)
CLOCK 3 (11)
D 2 (12)
Q 5 (9)
$\overline{Q}$ 6 (8)
LOGIC SYMBOL
V_{CC} = PIN 14
GND = PIN 7

Figure 5. Dual D-Type Positive Edge-Triggered Flip-Flop 74LS74.

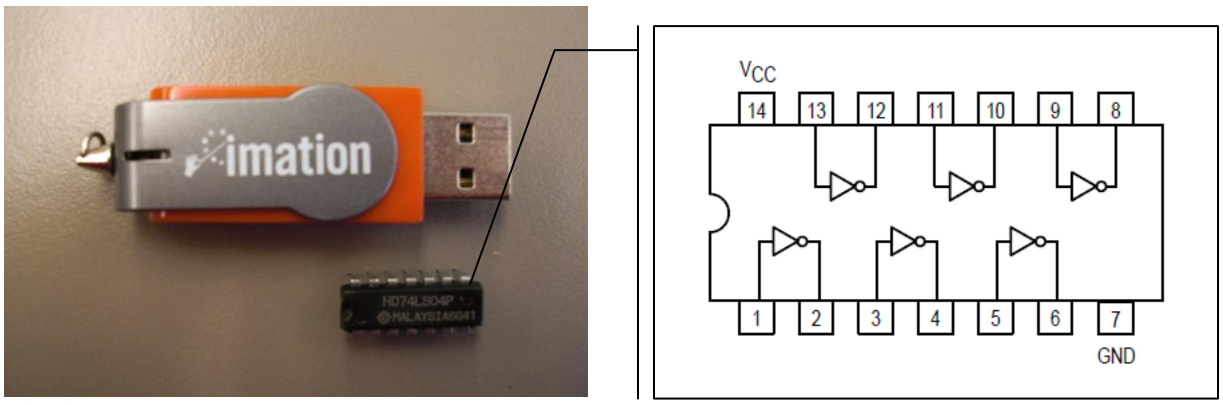


Figure 6. Hex Inverter 74LS04 Compared to a USB Flash Drive.

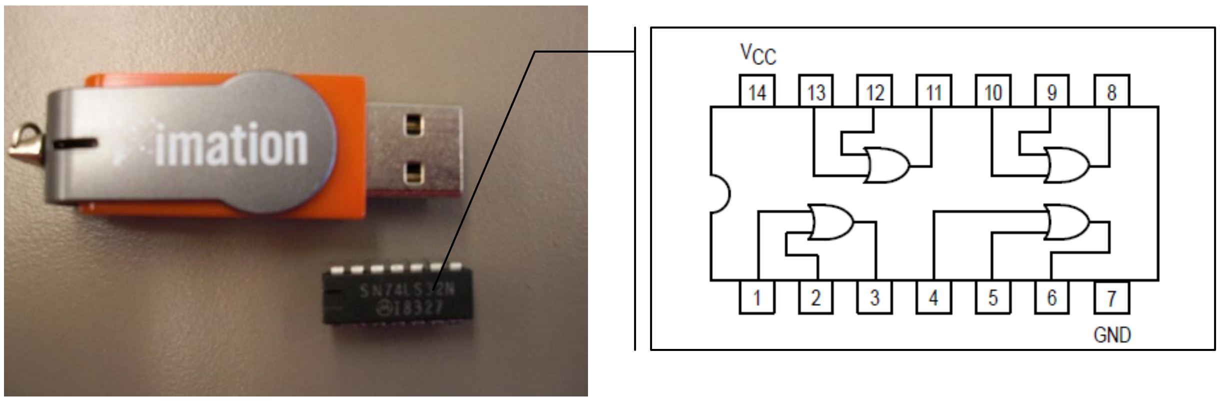


Figure 7. Quad 2-Input OR Gate 74LS32 Compared to a USB Flash Drive.

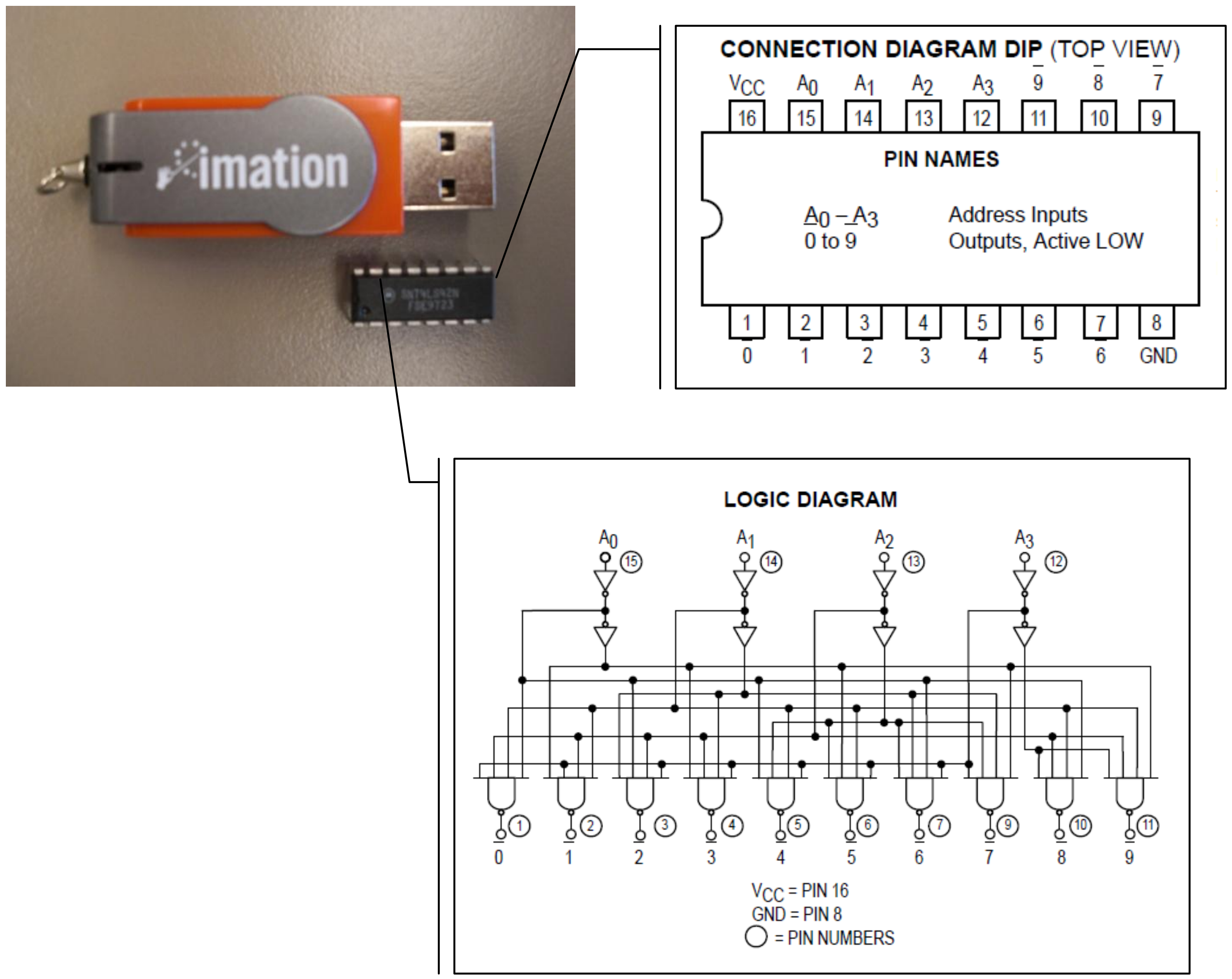

imation
CONNECTION DIAGRAM DIP (TOP VIEW)
VCC 16, A0 15, A1 14, A2 13, A3 12, 9 11, 8 10, 7 9
PIN NAMES
A0 – A3 Address Inputs
0 to 9 Outputs, Active LOW
1 0, 2 1, 3 2, 4 3, 5 4, 6 5, 7 6, 8 GND
LOGIC DIAGRAM
A0 15, A1 14, A2 13, A3 12
0 1, 1 2, 2 3, 3 4, 4 5, 5 6, 6 7, 7 9, 8 10, 9 11
VCC = PIN 16
GND = PIN 8
◯ = PIN NUMBERS

Figure 8. One-of-Ten Decoder 74LS42 Compared to a USB Flash Drive.

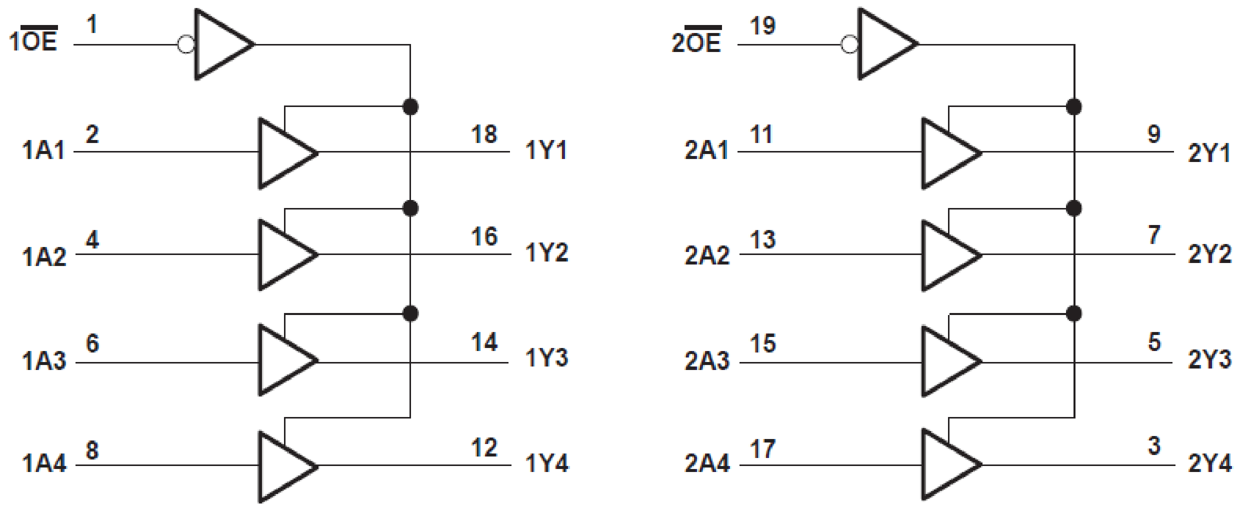


Figure 9. Logic Diagram of the 74HCT244 Driver.

Step 1: Basic Design

To start, the E signal and address bits were connected to the decoder the same exact way as they were connected in the previous lab when we were creating a 1-bit output controller. In the previous lab, we assigned our flip-flop to hexadecimal address 8000H, which was activated by decoder output y4. This is the same address we will use for our 1-bit input port. Since the output port is only energized when the R/W’ signal is low and the input port is only energized when the R/W’ signal is high, we can use the same address for our flip-flop for both input and output. Knowing this we came up with the following circuit and timing diagram for an input/output port:

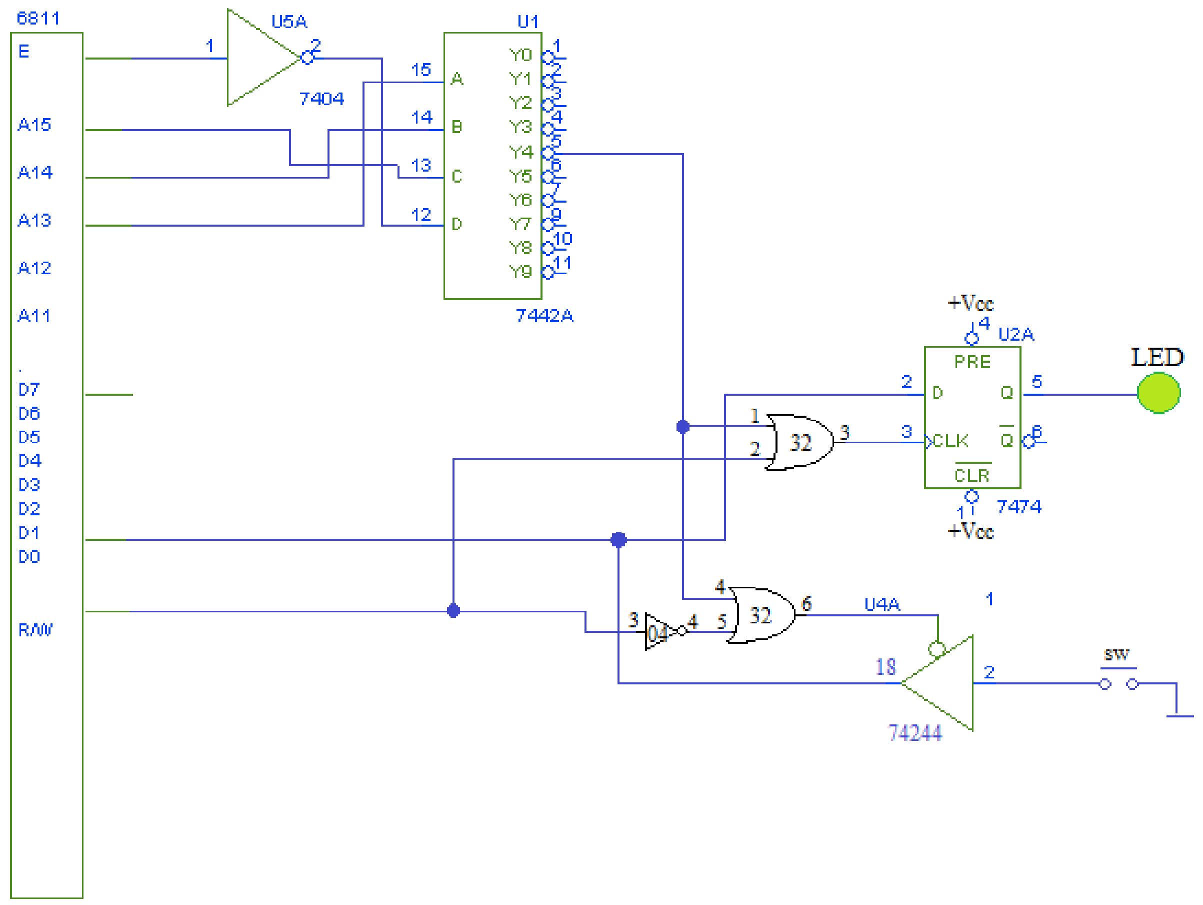


Figure 10. Circuit Diagram for a 1-Bit Input Controller.

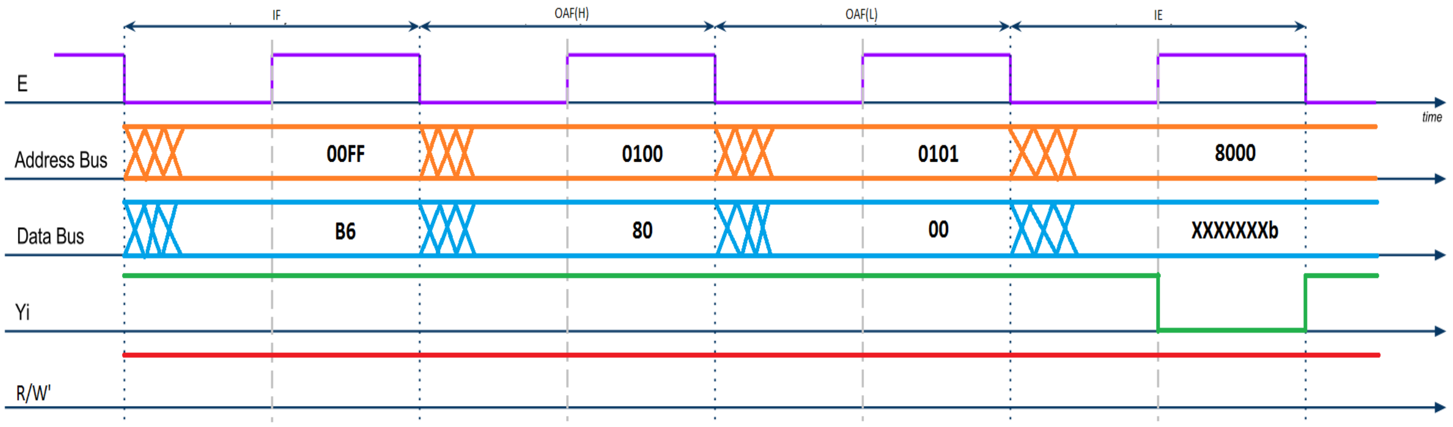


Figure 11. Timing Diagram for LDAA Instruction.

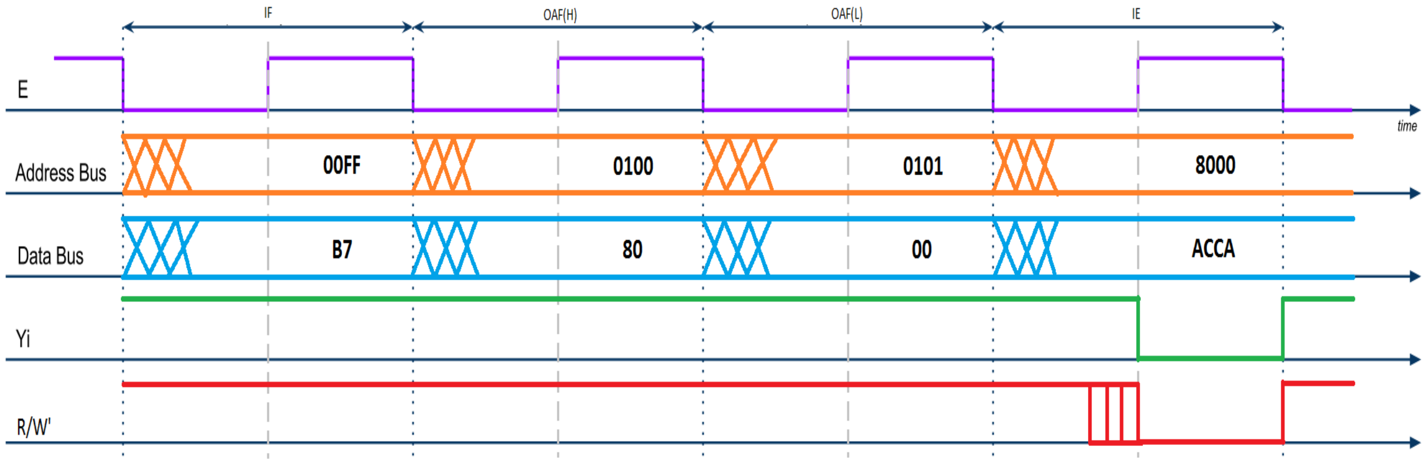


Figure 12. Timing Diagram for STAA Instruction.

Step 2: Wiring

Next, we wired our kit according to the circuit diagram in Figure , connecting everything except the output of the tri-state driver to D0 (this will be done only when we are sure that the rest of our circuit is wired correctly).

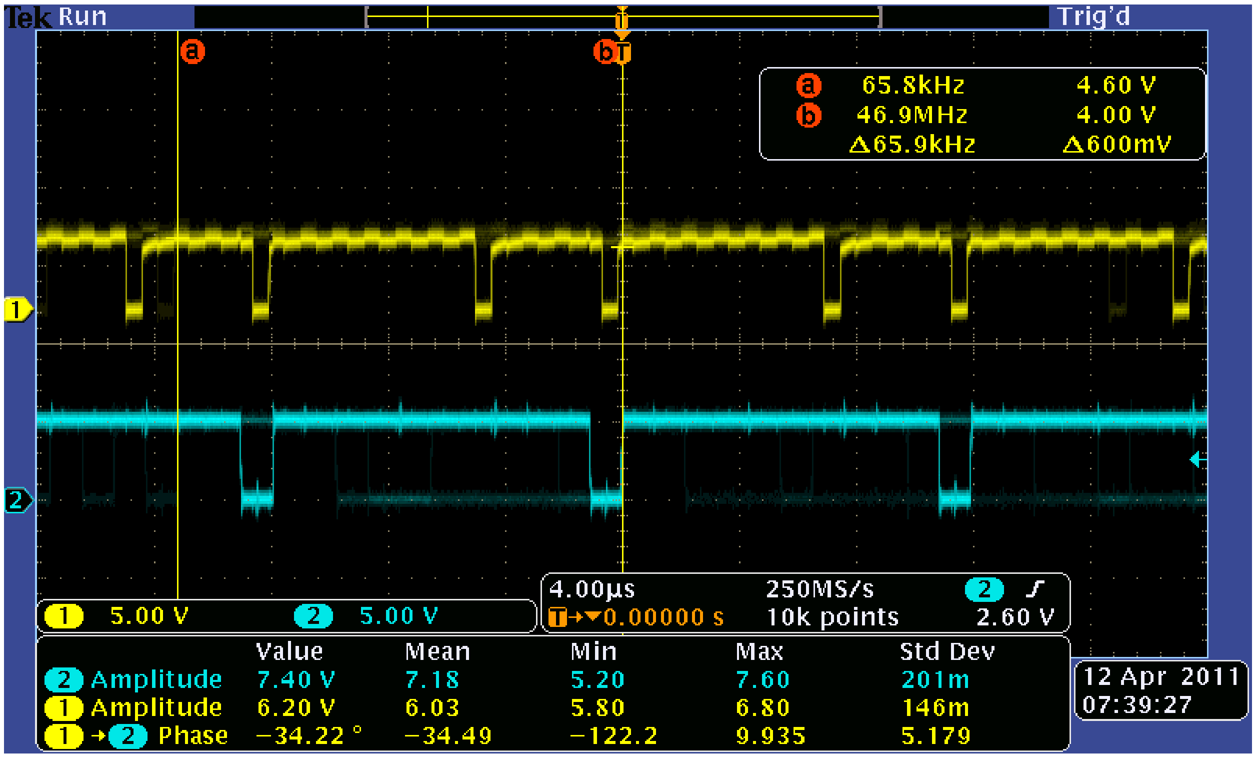
Step 3: Programming

We then wrote an endless loop that will constantly read a byte from the address corresponding to the 1-bit input port and write it back at the same address (storing it in the 1-bit flip-flop):

```
                      ;--- data section ---

                      ;--- code section ---
                              ORG    0000
                      LOOP:
0000 B6 80 00                 LDAA   8000        ; ACCA <-XXXXXXXb
0003 B7 80 00                 STAA   8000        ; flip-flop <- b
0006 7E 00 00                 JMP    LOOP        ; loop infinitely
```

After entering the assembled program into the kit and running it, we attached probes to the y4 output of the decoder and the R/W' to monitor the two signals generated by the program. We obtained the following screen capture:



```
a   65.8kHz    4.60 V
b   46.9MHz    4.00 V
    Δ65.9kHz   Δ600mV

1  5.00 V      2  5.00 V

4.00µs         250MS/s       2  ∫
T→▼0.00000 s   10k points    2.60 V

12 Apr 2011
07:39:27
```

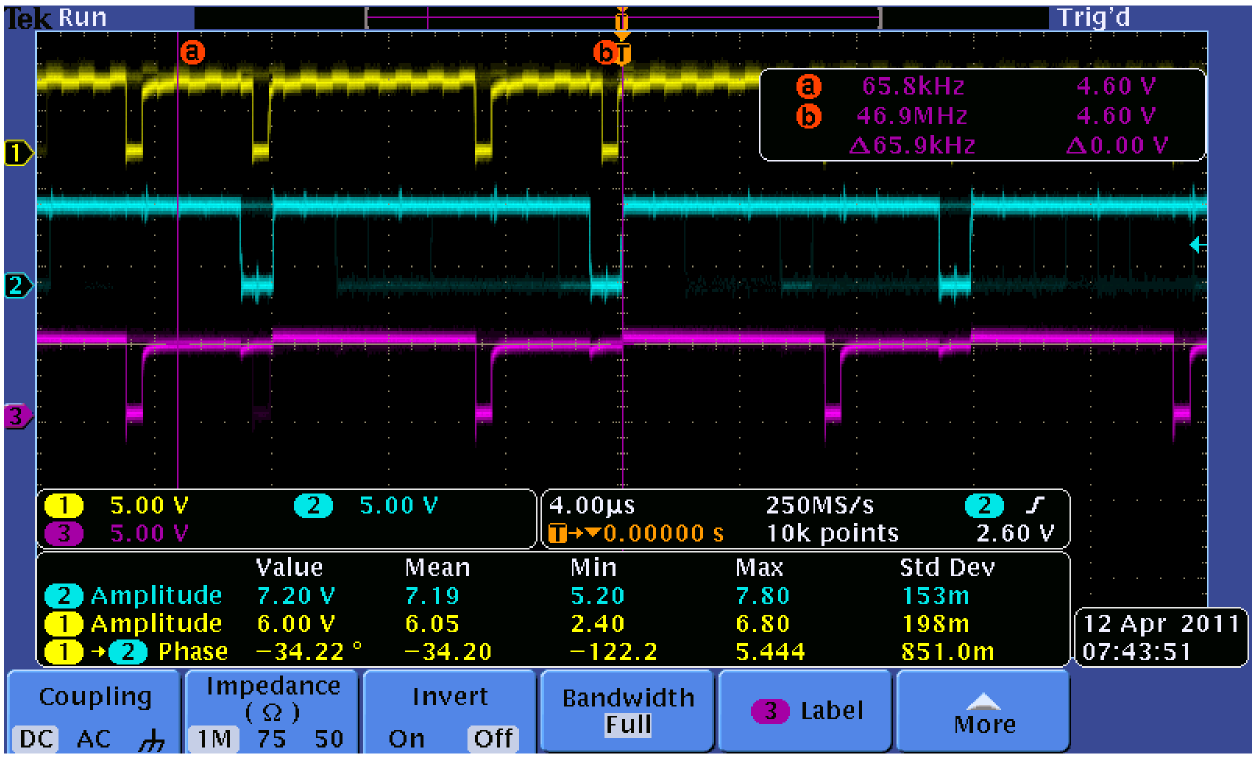
	Value	Mean	Min	Max	Std Dev
2 Amplitude	7.40 V	7.18	5.20	7.60	201m
1 Amplitude	6.20 V	6.03	5.80	6.80	146m
1 → 2 Phase	−34.22 °	−34.49	−122.2	9.935	5.179

Figure 13. Screen Capture of y4 Output Signal (Yellow) and R/W' Signal (Blue) from 1-Bit Input Controller.

This capture makes sense in regards to the program we were running and the circuit we were using. In one iteration of the loop, we can see that the y4 output (Yellow) gets activated twice, while the R/W' (Blue) gets activated only once. Since we load to the flip-flop first, the y4 output gets activated since that's the output that corresponds to 8000H, but the R/W' signal at that time remains high since the processor is reading when executing a load instruction. The second time the y4 output gets activated, the R/W' signal goes low. This corresponds to the instruction that stores to the flip-flop since the y4 output gets activated when 8000H gets put on the address bus and the processor is writing when executing a store instruction.

Step 4: First-Phase Testing

At this point we are still not ready to connect our tri-state driver output to D0 yet. We connected a probe to the enable input of the driver to monitor the driver enable's signal on the oscilloscope with the y4 and R/W' signals. We obtained the following screen capture:



a	65.8kHz	4.60 V
b	46.9MHz	4.60 V
	Δ65.9kHz	Δ0.00 V

1 5.00 V 2 5.00 V
3 5.00 V
4.00µs 250MS/s 2 ⨍
T→▼0.00000 s 10k points 2.60 V

	Value	Mean	Min	Max	Std Dev
2 Amplitude	7.20 V	7.19	5.20	7.80	153m
1 Amplitude	6.00 V	6.05	2.40	6.80	198m
1 → 2 Phase	−34.22 °	−34.20	−122.2	5.444	851.0m

12 Apr 2011
07:43:51

Figure 14. Screen Capture of y4 Output Signal (Yellow), R/W' Signal (Blue), and Driver Enable Signal (Purple) of 1-Bit Input Controller.

From this screen capture we verified that the driver was activated (Purple) only when the R/W' signal (Blue) was high and the y4 signal (Yellow) was low.

Step 5: Connection to Data-Bus

Once we observed and verified the correct timing of the signals, we connected the output of the driver to D0 and the input of the driver to one of the switches on the kit. Using a small screwdriver, we verified that the LED connected to the output of the flip-flop turned on with the switch was on and the LED turned off when the switch was off.

Step 6: Understanding Address Decoding

After verifying that our circuit works, we modified our program such that the address of the I/O port was at something other than 8000H:

```
                  ;--- data section ---

                  ;--- code section ---
                         ORG    0000
                  LOOP:
0000 B6 80 00            LDAA   9000        ; ACCA <-XXXXXXXb
0003 B7 80 00            STAA   9000        ; flip-flop <- b
0006 7E 00 00            JMP    LOOP        ; loop infinitely
```

When we entered the assembled program into our kit, we were still able to turn the LED on and off by turning the switch on and off respectively. This was because we were using address 9000H, which fell in the range that can be controlled by the y4 output (8000H to 9FFFH) and also fell in the range that we are able to work with in terms of programmable memory space (7000H to AFFFH). In total there are 1FFFH + 1 addresses that can be assigned to the flip-flop, which is equal to 8192 addresses.

Step 7: More Programming

Finally, we wrote a program that makes the LED blink when the switch is on and then keeps the LED at the state it was previously in when the switch is off:

```
                   ;--- data section ---

                   ;--- code section ---
                          ORG   0000
0000 86 00                LDAA  #00        ; ACCA <- 0
                   LOOP:
0002 F6 80 00             LDAB  8000       ; ACCB <-XXXXXXXb
0005 C4 01                ANDB  #01        ; ACCB <- b
0007 1B                   ABA              ; ACCA <- ACCA + b
0008 B7 80 00             STAA  8000       ; flip-flop <- ACCA
000B 36                   PSHA             ; save ACCA
000C 37                   PSHB             ; save ACCB
000D BD C0 27             JSR   pause      ; wait for 1 second
0010 33                   PULB             ; restore ACCB
0011 32                   PULA             ; restore ACCA
0012 7E 00 02             JMP   LOOP       ; loop infinitely
```

When we entered the program into the kit and ran it, we saw that the LED light continued to blink when the switch was activated, and when we deactivated the switch the LED light stayed in the state that it was in immediately prior to the deactivation of the switch (ie. LED light stayed off if the light was off before the switch was deactivated and LED light stayed on if the light was on before the switch was deactivated).