

Digital Signal Input and Output

Learning Objectives:

After successfully completing this lab, students will be able to:

- Describe the digital input and output functions of the ATmega16 microcontroller
- Read logic level signals from an input port
- Send logic level signals to an output port
- Build a circuit with ICs and passive components from a schematic diagram

Components:

<u>Qty.</u>	<u>Item</u>
1	Atmel ATmega16 microcontroller with STK500 and serial port cable.
1	Solderless Breadboard
1	7-segment LED (common anode (CA))
2	470 Ω DIP resistor pack
2	7447 BCD to 7-segment LED decoder IC

Introduction:

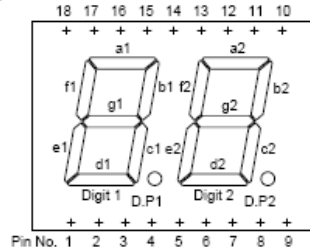
In this lab you will explore the input/output capability of the ATmega16 microcontroller. Microcontrollers are inherently digital devices, which means they operate with discrete values, usually the binary values 0 and 1 corresponding to the voltages 0 V and 5 V respectively. The discrete value of 0V is considered a “Low” or a logical 0 and that of 5V is considered a “High” or a logical 1. The ATmega16 microcontroller can service up to **32 digital digital inputs or outputs**. A digital output means that a program running on the ATmega16 can change the pin voltage to be either at common potential (0V) or at 5 V by writing a 0 or 1 to that pin. A digital input means that the world *outside* the microcontroller can change the voltage on the pin to either 0 V or 5 V, and the microcontroller can record the value as a 0 or 1 respectively.

7-segment LED display

You will use a 7-segment light emitting diode (LED) display as a digital output device and push-button switches as digital input devices. A 7-segment LED is nothing more than 7 LED’s arranged in a pattern that can form a character when the appropriate segments are lit. These displays come in two basic types: common anode (CA) and common cathode (CC). CA types have all of the anodes of the 7 LED’s connected together, and each of the 7 cathodes independent. Power is applied to the common anode, and a segment will be lit when its cathode is grounded. **(Don’t forget to use a current limiting resistor between the cathode and ground!)** The reverse is true for the CC types. Figure 1 below shows a schematic diagram of a CA 7-segment LED. The letters as shown denote the particular led segment.

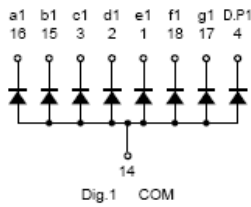
Two of the physical pins on the display are tied together (made ‘common’). For the CA-type of display, the two pins connect to the common anode. For the CC-type of display, the two pins connect to the common cathode.

●Pin assignments



Pin No.	Function	Pin No.	Function
1	Segment "e1"	10	Segment "b2"
2	Segment "d1"	11	Segment "a2"
3	Segment "c1"	12	Segment "f2"
4	D.P1	13	Digit 2 Common
5	Segment "e2"	14	Digit 1 Common
6	Segment "d2"	15	Segment "b1"
7	Segment "g2"	16	Segment "a1"
8	Segment "c2"	17	Segment "g1"
9	D.P2	18	Segment "f1"

●Equivalent circuit (anode common)



(cathode common)

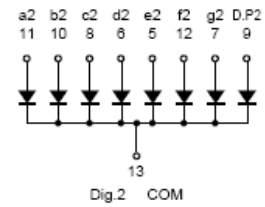
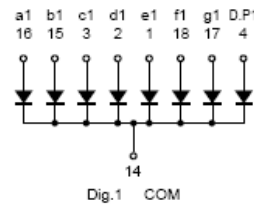
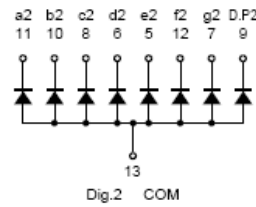


Figure 1. 7-segment LED Display (Common Anode (CA) type). Power is applied to both common anode pins (pin 13 and pin 14). A segment is lit when its cathode is then grounded through a current limiting resistor. Pin positions are numbered from 1 to 18.

7447 BCD-to-7-segment (CA) display driver

The most common way 7-segment displays are implemented is with a BCD-to-7-segment decoder/driver chip (BCD stands for Binary Coded Decimal). This chip takes a 4-bit binary number (like 0101, which corresponds to decimal 5) as an input, and when connected to a 7-segment display, it causes the proper LED segments to turn on and display the corresponding decimal number. The chip used with CA displays is the 7447. For CC displays it is the 7448. Figure 2 shows the pinout diagram for the 7447 and describes its operation. (The actual lettering on the chip may include other letters and numbers like, SN74LS47). By standard convention, pin 1 on any IC package is always the lower leftmost pin when the IC is oriented as shown with the U-shaped notch, or dot toward the left. Pin numbers proceed to the right and loop around the right end of the chip as shown. Some IC's will only have the notch or dot; some have both.

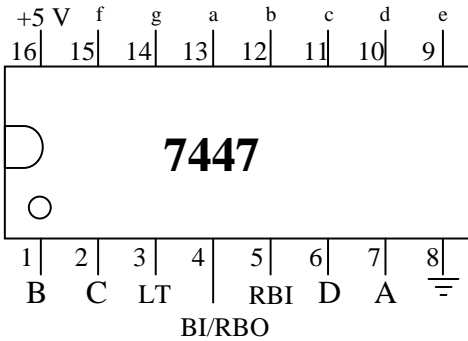


Figure 2. 7447 BCD-to-7-segment decoder driver chip. This chip takes a 4-bit binary number applied to DCBA, where A is the least significant bit (LSB), and grounds the appropriate pin 9-15, so that when these pins are connected to a 7-segment LED display through current limiting resistor, the corresponding decimal number will appear on the display. The 7447 is used to drive common anode (CA) displays.

Switch input

You will use two of eight switches mounted on the STK500 to provide digital inputs. A switch is either on or off, hence it makes for a very simple digital sensor. These switches are momentary, normally open (NO) single pole, single throw pushbutton switches called 'tact' switches. These switches are made to be soldered directly to a PCB. Two leads on the same side of the switch are tied together **internally**,

and, likewise, the two leads on the opposite side are tied together **internally**. When the button is pressed, an electrical connection is made between the two sides. One side of each switch is connected to ground, while the other side is pulled up to +5 V through a 10k Ω resistor and is accessible through the appropriately labeled SWITCHES header. Part **B** in the circuit shown in Figure 3 shows the two switches. Note: in order for the microcontroller to be able to determine whether the switch is open or closed, you *must* make an electrical connection (via a jumper wire) from the associated pin on switch header to a pin associated with the microcontroller port you will use in your program.

Hexadecimal (Hex) Numbers

Hexadecimal (also called base 16) is a numbering system in which there are 16 distinct symbols, 0-9 and A-F. While the digits 0-9 in a hex number represent the same decimal numerical values you are familiar with, the digit A in hex represents 10, B 11, C 12, D, 13, E 14 and F 15. Hexadecimal is often used in programming because a two-digit hex number can be used as shorthand to represent 8 binary values: the leftmost hex digit covers the leftmost four binary places and the rightmost hex digit covers the remaining 4 binary places. Just like with decimal numbers you are familiar with, binary numbers have place values. In a decimal number the rightmost place is the ones place (10^0), the next place to the left is the tens place (10^1), the next is the hundreds place (10^2), and so on. In a binary number the place values for the first four places go like this from right to left: ones (2^0), twos (2^1), fours (2^2), eights (2^3).

To convert four binary digits into a hexadecimal digit, just add up each binary digit multiplied by its place value. Thus, for the binary number, 1010, $1(8)+0(4)+1(2)+0(1) = 10$, which in hex is A.

The same logic applies going from hexadecimal values to binary. Choose the binary digits (0 or 1), so that when each is multiplied by its place value, it gives the hex value you are looking for. For example, hex F represents decimal 15, and $1(8)+1(4)+1(2)+1(1) = 15$, so the corresponding binary number is 1111.

For the purposes of our class we will be using two hexadecimal digits, because the ports and registers on the microcontroller have groupings of eight pins or eight locations. Table 1 lists binary, hex, and decimal numbers from zero through 15.

Table 1. Binary, Hexadecimal, and Decimal Values. HEX is a shorthand way to represent 4 binary digits.

Binary				HEX	Decimal
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	2	2
0	0	1	1	3	3
0	1	0	0	4	4
0	1	0	1	5	5
0	1	1	0	6	6
0	1	1	1	7	7
1	0	0	0	8	8
1	0	0	1	9	9
1	0	1	0	A	10
1	0	1	1	B	11
1	1	0	0	C	12
1	1	0	1	D	13
1	1	1	0	E	14
1	1	1	1	F	15

STK 500 Port Header Pins

As you know from the Intro to the ATmega16 lab, the STK500 has different port header pins, ranging from PORT A, PORT B, PORT C, PORT D and PORT E, as shown in Figure 3.

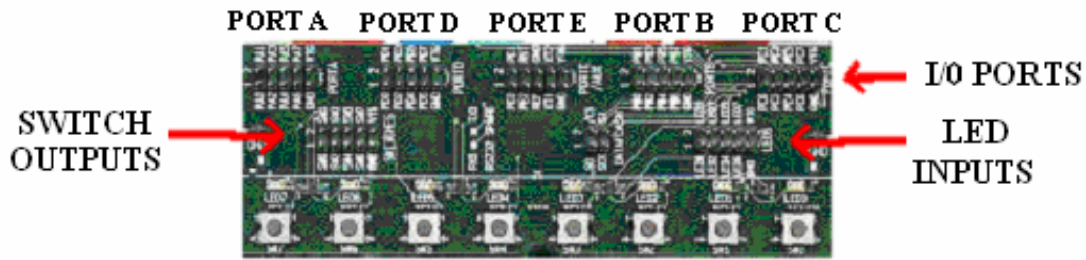


Figure 3. Output headers that include PORT A, PORT B, PORT C, PORT D and PORT E pins. Input headers include the SWITCH port and the LED port. Note, in order to use the SWITCH and LED ports a 10 pin ribbon cable needs to be jumpered between the input and I/O ports so they will be connected to the ATmega16.

Each header has 10 pins, one for power (VTG), one for ground (GND), and 8 pins that correlate with the 8 binary placeholders from above. Under program control, we can cause a pin to go to either 0 V or 5 V, which we can represent by the corresponding binary values of zero (0) or 1 (1). The 8 pins that correlate with the 8 binary placeholders start their count from 0 and go through 7 (where Pin 0 represents the ones place, and Pin 7, the eights place). We refer to the headers in code by “PORTx”. If we wanted to interface to Port A we would type PORTA.

If you followed everything above, you now have the capability to use hex numbers and output either 0V or 5V on different pins for the ATmega16.

Note: At this point in the ME 106 lecture you should understand how to use hex values to cause port pin values to take on voltage values that you wish to set. On the ME 106 website, under Lecture Notes, there is a program called ‘BlinkLED3.c’ and a program called ‘Simple-SW_LED3.c’ that you should review at this time. Understanding these programs will help you with this lab.

Interfacing to Port Pins

When we are programming the microcontroller, it is a good idea to assign the pins you want to be inputs or outputs and set their voltage levels before you get into the main part of your code. This is usually done in an initialization function, such as `init()` in this lab. There are two-steps involved: setting the pin’s direction (making it an input or an output), and if it is to be an output, setting its voltage (either high (5V) or low (0 V)). Setting bits in the Data Direction Register for a port will determine whether the corresponding pin will be an input or output. For example, if we wanted to make all the pins associated with Port A be inputs we would type: `DDRA = 0x00`; If we wanted all the pins of Port A be outputs we would write: `DDRA = 0xFF`;

(NOTE: the notation ‘0x__’ denotes that what follows the ‘0x’ is a hexadecimal number. If you wanted to instead work with a binary number, (say 1101_b), you would write, ‘0b1101’. Internally, the value is ultimately converted to a binary number, so writing, `DDRA=0x1A`; is equivalent to `DDRA=0b11010`; which is equivalent to, `DDRA=26`;))

If a pin is specified as an *output*, our program can specify its voltage level. This is especially important during initialization, because if one of the microcontroller pins was, say connected to a power interface, which in turn operated a motor, we might want to make sure that the pin voltage was such that the motor did not turn on until we wanted it to. We can change the voltage of a pin that is set to be an output by writing a binary digit to the corresponding PORT variable. For example, if all the pins of PORTA were designated as outputs, and we wanted them all to be at 0 V, we would write, `PORTA = 0x00`; If instead you wanted only Pin 7 and Pin 1 to be at 5 V, and the rest at 0 V, you would write, `PORTA = 0x82`;

Procedure:

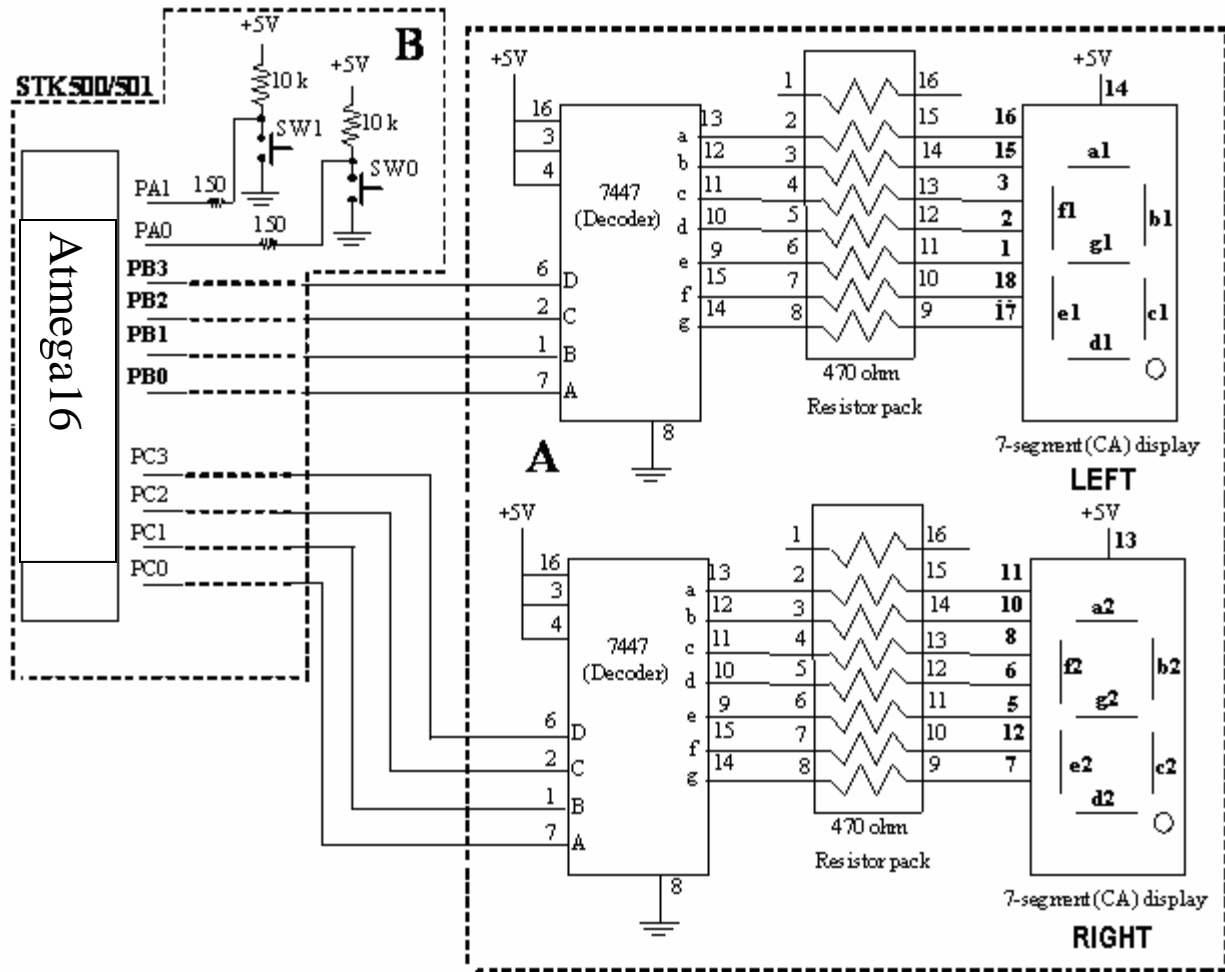


Figure 4. Switch-controlled display circuit. Two tact switches (SW1 and SW2) mounted on the STK500 provide digital inputs to the ATmega16, which in turn drives (by four digital outputs) a 7447 seven-segment decoder IC. Note the 10 kΩ resistors connected to the switches. These are called ‘pull up’ resistors, because they pull the voltage of pins PA0 and PA1 up to 5 volts when the switch is not pressed. They also limit the current to ground when the switches are pressed.

1. In this lab we are going to build a display circuit in which the onboard switches of the STK500 will act as our input based upon their hexadecimal value.
2. Build the section of the circuit shown in the dashed rectangle A in Figure 4 on the breadboard. You should think about where you are going to place the components so that you do not connect two wires to the same place in the breadboard. **Note the numbers next to different connections represent the pin numbers; you cannot just lay the chips next to each other and start connecting across.** Do not connect the 7447 to the ATmega16 yet! (As we have emphasized in previous labs, you will save yourself lots of time, effort, and frustration by building and testing **pieces** of a complicated circuit rather than trying to wire up everything in one shot.) Use the pinout diagrams shown in Appendix A for the 7447 and the LED display to see where the pins are actually located.

Test the circuit in A by grounding the inputs DCBA. This can be done by utilizing a “VTG” pin which will give 5V and grounding out from one of the “GND” pins. The decimal digit ‘0’ should appear on the 7-segment display.

After you have proven that the both 7447's and 7-segment display's have been wired correctly, connect power and the serial cable to the STK500 board (connect the serial cable to the RS232 CTRL connector). Don't connect anything else to the ATmega16 yet! Note that when you power up the ATmega16 it will run the program you downloaded last, so it is a good idea to hook up the ATmega16 after you know it has been reprogrammed successfully.

3. Create a new project and enter and run the **mainDIO1.c** code from for the course website. **What does the double 7-segment display show? What is the voltage at pins 9, 10, 11, 12, 13, and 15 of the 7447 chip? What is the voltage at pin 14 of the 7447 chip?**
4. Modify the program to write a '01' to Display instead of '55'. **Which pins of the 7447 do you expect to be at 5 V, and which are low?** Experiment by writing numbers between 0 and 9 until you are satisfied with your understanding of what is happening with the ATmega16 pins and your circuit.
5. Modify the program so that the display will count from 0 to 99 continually with a delay of 100 milliseconds second in between numbers. This is how you make digital clocks or timers. **Include this code in your lab report. Hint:** Very easy to use a 'for' loop. If you don't know the syntax of a 'for' loop, Google it. As an engineer, you will encounter many problems that will require you to investigate outside sources. You should find significant hints by reading through the provided code. So far we've been dealing with the digital output capabilities of the ATmega16. Now let's bring in the digital input capabilities.
6. Let's make the display circuit output "00" if neither button is pressed, "11" if SW0 is pressed, or "22" if SW1 is pressed. Complete the circuit in Figure 3 by wiring the two switches to the appropriate PORTA pins. Create a new project and enter and run the **mainDIO2.c** code from the course website. **Explain how the switch statement reads in the inputs from the switches connected with PORTA.**
7. To test your understanding of digital input and output write a program that will cause the display to count up (continuously, with 1 second pauses between numbers) when SW0 is pressed and held down, and count down when SW1 is pressed and held down. **Enter this code in your lab report for counting up with a 1 second pause in between numbers. (Once more, significant hints on how to do this are available in the code, READ THROUGH IT!)**

Challenge (6pts Extra Credit!): How would you interface a dc motor and switch to the ATmega16, so that you could turn the motor on when the switch is pressed? Draw a schematic for your interface and write a short program C to implement your idea.

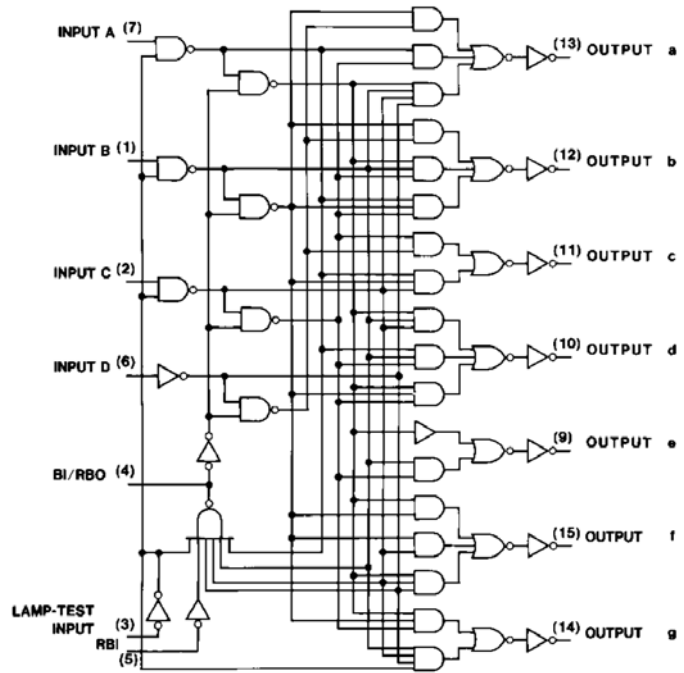
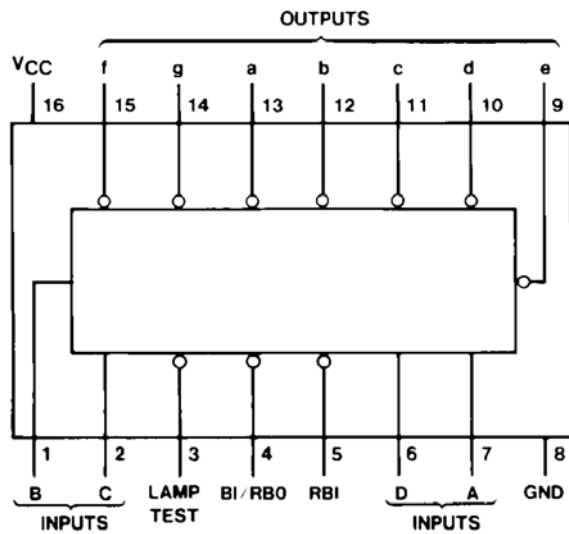
Note: The ATmega16 can only output 5V and 40 mA to a pin, which is likely not enough to power a motor or solenoid. You will most often need a power interface using a transistor or a MOSFET (shown in the next experiment) to power a motor or solenoid. (NOTE: 40 mA is an absolute *maximum*, which you should strive to be *well* below, like by a factor of 10, to avoid damaging the chip!)

Appendix A

DM7446A, DM7447A

BCD to 7-Segment Decoders/Drivers

Fairchild Semiconductor, 2001



Decimal or Function	Inputs						BI/RBO (Note 1)	Outputs							Note
	LT	RBI	D	C	B	A		a	b	c	d	e	f	g	
0	H	H	L	L	L	L	H	L	L	L	L	L	L	H	(Note 2)
1	H	X	L	L	L	H	H	H	L	L	H	H	H	H	
2	H	X	L	L	H	L	H	L	L	H	L	L	H	L	
3	H	X	L	L	H	H	H	L	L	L	L	H	H	L	
4	H	X	L	H	L	L	H	H	L	L	H	H	L	L	
5	H	X	L	H	L	H	H	L	H	L	L	H	L	L	
6	H	X	L	H	H	L	H	H	H	L	L	L	L	L	
7	H	X	L	H	H	H	H	L	L	L	H	H	H	H	
8	H	X	H	L	L	L	H	L	L	L	L	L	L	L	
9	H	X	H	L	L	H	H	L	L	L	H	H	L	L	
10	H	X	H	L	H	L	H	H	H	H	L	L	H	L	
11	H	X	H	L	H	H	H	H	H	H	L	L	H	L	
12	H	X	H	H	L	L	H	H	L	H	H	H	L	L	
13	H	X	H	H	L	H	H	L	H	H	L	H	L	L	
14	H	X	H	H	H	L	H	H	H	H	L	L	L	L	
15	H	X	H	H	H	H	H	H	H	H	H	H	H	H	
BI	X	X	X	X	X	X	L	H	H	H	H	H	H	H	(Note 3)
RBI	H	L	L	L	L	L	L	H	H	H	H	H	H	H	(Note 4)
LT	L	X	X	X	X	X	H	L	L	L	L	L	L	L	(Note 5)

H = HIGH level, L = LOW level, X = Don't Care