16.070 Introduction to Computers and Programming

March 22 **Recitation 7** **Spring 2001**

Topics:

Input / Output

- Formatting Output with *printf*
- File Input / Output

Data Conversion

- Analog vs. Digital
- Analog → Digital
- Digital → Analog

Input / Output

Formatting Output with *printf*

There are several characters that can be added to the format specifiers, %d, %c, %f & %lf, %e, and %s, used in *printf* in order to achieve special formatting in the output. The table below contains the characters that can be added between the % symbol and the character in the format specifier and their resulting format.

Character	Example	Special Formatting
integer	%**10**d	prints the data in a field that is 10 spaces wide
-	%-10d	prints the data left justified
0	%**0**10d	prints the data with zeros filling the field to the left
+	%+10d	prints positive data with a plus sign
.decimal	%10.**3**f	prints the data with a field width of 10 with 3 decimal places

If your program uses many *printf* statements and you want all of your data formatted the same way you can make use of variables for determining field width or precision. For example if we define a variable *width* to represent the field width that we want and a variable *precision* to represent the number of decimal places that we want. It is possible to write our output statements as follows:

```
printf("%*.*f", width, precision, data);
```

Using this format allows you to change multiple output statements at once.

File Input / Output

In order to use file I/O you must declare every input and output file with the following syntax:

```
FILE  *file_name
```

This declares the internal *file_name* for the file you want to operate on, sometimes called a "file handle". Next you must point this internal file name to an external file using the *fopen* command. The format of the *fopen* command is as follows:

```
file_name = fopen("external_file_name", Openmode);
```

Where *Openmode* is one of the following:

Open Mode	Resulting Permissions
"r"	read access

“w”	write access
“a”	append data to the end of an existing file
“r+”	opens a file for input & output, if it exists already, the contents are not destroyed
“w+”	destroys the file if it exists and creates a new one for update
“a+”	opens a file for update so that writing is done at the end of the file

The “w” and “w+” modes are destructive! The “a” and “a+” modes are not destructive. They append data to already existing files. The difference is that you can read from a file that was opened with “a+” but not one that was opened with “a”.

If the *fopen* command fails for some reason the function will return a NULL pointer value. This can be useful when determining whether a file has opened successfully or not. After opening and operating on a file it must be closed. This is done with the *fclose* command:

```
flcose(file_name);
```

Another very useful function is the *feof* function. It returns a true (1) if the most recent input operation on the file returned an end of file character, and returns false (0) otherwise. There is also a constant EOF that is defined to represent the end of file character. This can also be used in various ways to test for the end of a file.

Writing Data to Files
The *fprintf* function is used to write data to a file. It is very similar to the *printf* function except it requires a file name to write to.

```
fprintf(file_name, format_specifiers, variables);
```

There are two other functions that can be used to write to files. The *fputc* function writes a character to a file and the *fputs* function writes a string to a file.

```
fputc(CharacterExpression, file_name);       fputs(StringExpression, file_name);
```

Reading Data from Files
The functions used to read data from files are similar to those seen earlier. The three functions are *fscanf*, *fgetc*, and *fgets*.

```
fscanf(file_name, format_specifiers, variables);

fgetc(file_name);             fgets(StringExpression, Size, file_name);
```

Since *fgetc* has no variable specification it is usually used in an expression in the form of

```
CharacterVariable = fgetc(file_name);
```

Input from Preformatted Data Files
The data can be contained in fixed width fields, it is possible to read it by simply using the proper width declaration in the format specifiers of the *fscanf* function. The data can also be separated by a specific character, such as a comma in a comma separated value (CSV) file. It is possible to store the separation character in a dummy variable or include the character between the specifiers in the function call.

(Example of file I/O)

```
/* TJ, March 2001 */
/* Recitation 7   */
/* File I/O       */

/* This program writes integers to a file and reads the same
   integers from it again */

#include <stdio.h>
```

```
int main(void)
{
	int a,b,c;   /* declare 3 integer variables to write to file */

	/* declare file handle/internal name */
	FILE *file_name;

	/* open file */
	file_name=fopen("test_file.txt","w+");

	/* print to file */
	fprintf(file_name,"%d ",1);
	fprintf(file_name,"%d ",2);
	fprintf(file_name,"%d ",3);

	/* close file */
	fclose(file_name);

	/* re-open file */
	file_name=fopen("test_file.txt","r");

	/* read from file */
	fscanf(file_name,"%d",&a);
	fscanf(file_name,"%d",&b);
	fscanf(file_name,"%d",&c);

	/* close file */
	fclose(file_name);

	/* print contents to screen */
	printf("\n File contains: %d %d %d\n\n",a,b,c);

	return 0;

}	/* end main */
```

Data Conversion

Reference: "Data and Computer Communications" by William Stallings, 4th Ed., MacMillan Publisher
This section deals mostly with modulation techniques involved when actually transferring (communicating) analog information via a digital format, or when transferring (communicating) digital information via an analog format, i.e. so-called modulation techniques. This is different from the actual *digital to analog* and *analog to digital* **conversion** processes, which entail converting single digital numbers to analog outputs and converting an analog input to single sampled digital values respectively.

An example of simple analog to digital conversion:

> A sensor outputs a voltage in the range 0 to 10V, signifying the rotational angle of a solar array. If your computer needs to know the solar array angle, it reads the sensor voltage at a specific time and converts the voltage between 0 and 10 to a digital value between, say, 0 and 255, if the value is represented by a single byte.

An example of simple digital to analog conversion:

> A DC motor's angular velocity needs to be controlled with a voltage input between 0 and 12V. Your computer computes the angular velocity required of the motor. The motor can be controlled by a circuit converting a digital number, say 0 to 255 (8-bits again), to an analog voltage scaled between 0 and 12V.

In general, the difference between modulation/de-modulation techniques and actual direct analog to digital (A/D) and digital to analog (D/A) conversion, is that modulation techniques are concerned with conveying information by employing specific signal patterns (as you will see) while direct D/A and A/D conversion is concerned with converting explicit voltage levels to explicit digital values and explicit digital values to explicit voltage levels.

Analog vs. Digital

Characteristics of each type of signal:

Analog	Digital
easy for humans to process	easy for computers to process
natural	artificial
qualitative	quantitative
infinite detail	finite detail

Examples of each type of signal:

Analog	Digital
copper-wire telephone	ISDN telephone
photograph	digital image
cassette tape	compact disc
human brain	computer processor

Benefits of each type of signal:

Analog	Digital
easily processed by humans	easily processed by computers
high detail	data can be transmitted with zero loss
exact	data can be reproduced

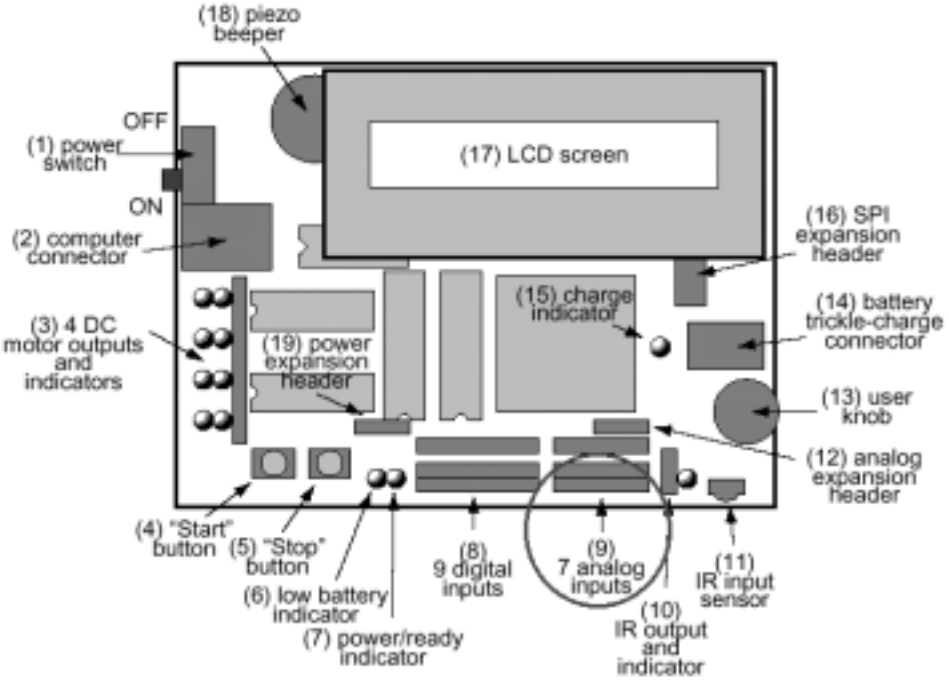


Figure 1 Your Handy Board can convert 7 analog inputs between 0 and 5V directly to digital values between 0 and 255.

Pitfalls of direct A/D and D/A conversion

1. If you sample an analog signal digitally at a constant sampling rate F, then you need to remove (filter) all frequencies higher than 0.5*F from the analog signal before any conversion to a digital value is attempted. This pitfall is called aliasing and occurs when some signals of different frequencies convert to exactly the same digital values, because of the sampling rate.
2. Digital discretization noise results when an analog signal is sampled digitally, since the analog value is by definition of a higher numerical precision (infinite) than the digital value. Also, digital values converted to analog voltages will always be "step-wise", with certain voltages being simply to precise to represent with a given number of bits and the available D/A converters voltage output range.
3. A/D and D/A voltage ranges are usually fixed for a specific converter. It is important not to overload the voltage input to A/D converters or to saturate the voltage output of D/A converters, while using as much of the voltage range as possible, in order to decrease discretization noise.

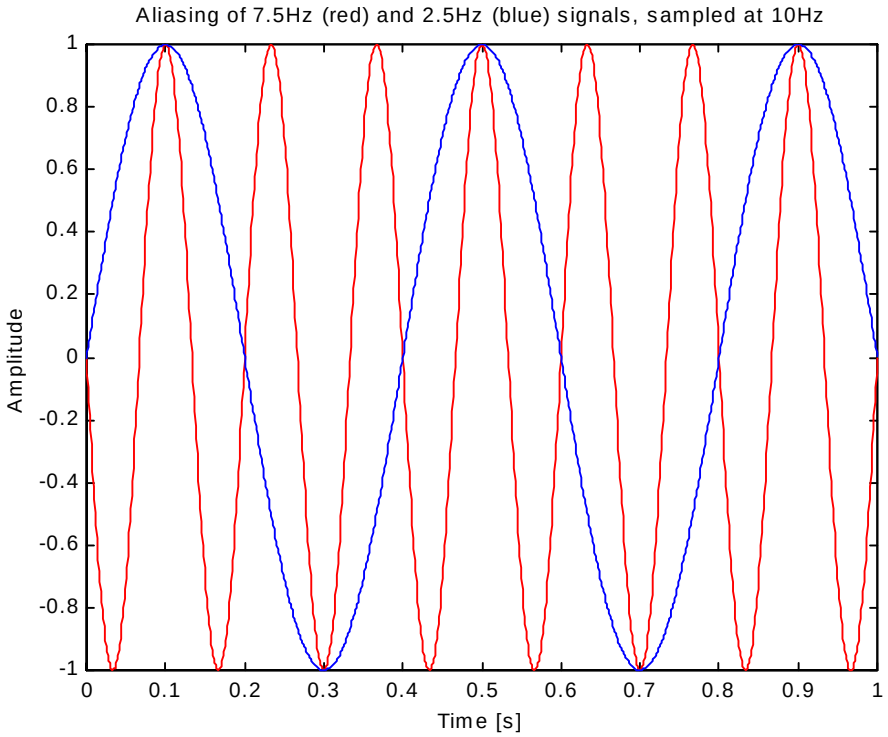
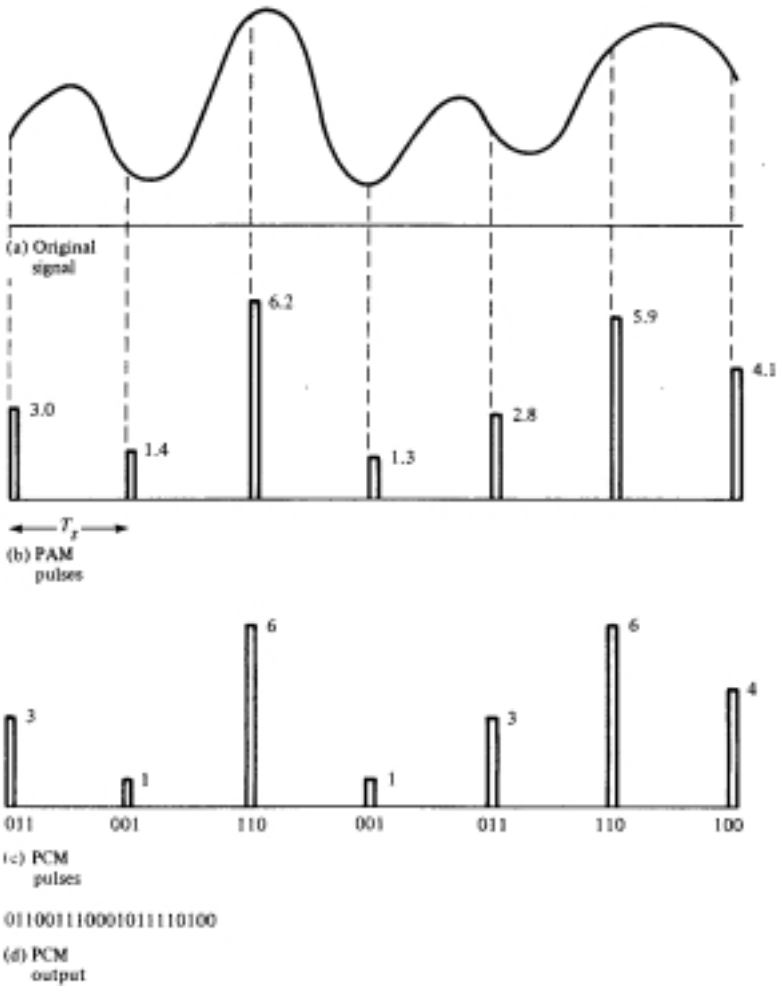


Figure 2 Sampling 7.5Hz and 2.5Hz signals at 10Hz result in exactly the same digital signal. The phenomenon is called "aliasing" and is avoided by filtering out all signal frequencies above 0.5*Sample Rate before sampling.

Transmitting analog data through a digital channel (Modulation)

As will be discussed in lecture, it is often necessary to take an analog signal from a sensor and convert it into digital data that the computer can manipulate. The device used for converting analog data into digital data, and recovering the analog signal on the other side is known as a *codec* (encoder-decoder). The two principle techniques used in encoding are *pulse code modulation* and *delta modulation*. We will be discussing pulse code modulation.


(a) Original signal
3.0
1.4
6.2
1.3
2.8
5.9
4.1
T_s
(b) PAM pulses
3
1
6
1
3
6
4
011
001
110
001
011
110
100
(c) PCM pulses
0110011100010111110100
(d) PCM output

PCM is based on the following sampling theorem,

> "If a signal *f(t)* is sampled at regular intervals of time and at a rate higher than twice the highest significant signal frequency, then the samples contain all the information of the original signal. The function *f(t)* may be reconstructed from these samples by the use of a low-pass filter."

This means that the encoder must measure the amplitude of the signal *f(t)* at a frequency that is at least twice that of the highest component frequency in *f(t)*. The samples are represented as narrow pulses whose amplitude is proportional to the value of the original signal. This is known as *pulse amplitude modulation* (PAM), and is the first step in PCM.

The amplitude of each PAM sample is then approximated by an *n*-bit integer. The value of *n* is determined by the number of different amplitude values that are needed. If $n = 3$ for instance that would mean that there are 8 (2^3) levels available to approximate the amplitude of each sample. It should be noted that it is necessary to know the encoding parameters used in PCM in order for the decoder to extract the analog signal from the digital data.

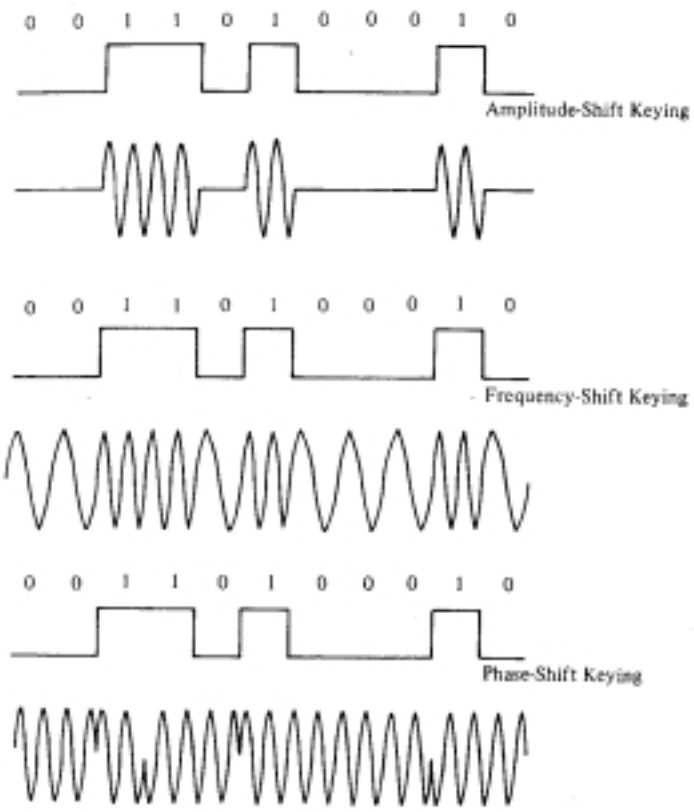
One drawback of PCM is that the size of the encoded signal increases with the number of amplitudes that must be approximated. Techniques such as *nolinear-encoding* or *companding* (compressing-expanding) the input signal are used to improve the performance of PCM.

Transmitting digital data through an analog channel (Modulation)

Converting digital data into an analog signal is necessary when you are trying to control motors, actuators, or other electro-mechanical devices with a computer (such as the Handy Board). The process is most commonly used to transmit digital data through the public telephone network (AOL, CompuServe, etc…). The network was designed to transmit analog signals in the voice-frequency range. This means that digital devices must be connected to the network via *modems* (modulator-demodulator), that convert digital data into analog signals and then back.

The basis for analog signaling is a continuous constant frequency signal known as the carrier signal. This signal is chosen to be compatible with the transmission medium being used. Data is transmitted by *modulating* the carrier signal. Modulation is done by manipulating one of the three fundamental frequency-domain parameters:

- Amplitude-Shift Keying (ASK)
- Frequency-Shift Keying (FSK)
- Phase-Shift Keying (PSK)



ASK uses two different amplitudes to represent the two binary values. Typically the binary 0 is represented by the absence of the carrier signal and the binary 1 is represented by the presence of the carrier signal.

FSK, uses two different frequencies near the carrier frequency to represent the two binary values. Typically f_1 and f_2 are offset from f_c by equal but opposite amounts.

PSK, uses phase shift of the carrier signal to represent binary data. In a two phase system a binary 0 is represented by sending a signal burst of the same phase as the previous signal burst. A binary 1 is represented by sending a signal burst of opposite phase as the previous signal.