Parallel-to-Serial and Serial-to-Parallel Converters

Max Rabiee, Ph.D., P.E. University of Cincinnati

Abstract:

Microprocessors (MPUs) on a computer motherboard communicate in a parallel format with the memory system and support chips. The memory system consists of Read Only Memory (ROM), and Random Access Memory (RAM). The 8255 Programmable Peripheral Interface (PPI) is an example of such a support chip [1]. Microprocessors (MPUs) communicate in a serial format with outside peripheral devices. Serial-to-parallel and parallel-to-serial conversion must be performed to establish this communication. A few examples of outside computer peripheral devices are the modem, the printer, and the CRT terminal. In parallel communication format, all data bits are transferred at the same time through a computer data bus, while in serial communication, bits are transferred through one data line in a pulse format. Parallel data must be converted to serial data form prior to transmission to the outside peripheral device(s). The serial data received by the computer from an outside peripheral device must be converted back to a parallel format and then placed on the computer data bus [2]. The parallel data on the computer data-bus is represented by either zero or five volts (0V or 5V). However, the serial data, which is transmitted outside the computer, must be set to an industry standard voltage/current level. These standards are designed to insure that the transmitted data is immune to outside electromechanical and electromagnetic noise interference.

In this paper, we will describe the terms used in serial communication systems. Then, we will describe a project in which a digital circuit is designed to convert the parallel data to serial format. Application specific chips are available to perform the task of parallel-to-serial and serial-to- parallel conversion. One example of such an application specific chip, is the INTEL 8251A [1], which is a Universal Synchronous/Asynchronous Receiver/Transmitter (USART) chip. This chip can be programmed to convert parallel data to serial and vice versa. It can also be programmed to either transmit or receive data in asynchronous or synchronous forms. If the 8251A is programmed for asynchronous transmission, its framed data words must have at least one start bit, or possibly a one and a half, or two stop bits. In addition, the framed data words may be set to include parity bits. The 8251A is also programmed to set the serial data transmission speed rate in bits per second.

Semiconductor chips similar to the INTEL 8251A are utilized by the computer motherboard to convert, receive, and transmit data for the microprocessor. We will also describe one of the popular serial communication standard that is utilized in industry. This serial communication protocol is used to protect the serial data from distortion due to outside electrical noise pollution.

Students in a microprocessor interfacing class will learn how to program the 8251A chip and similar types of chips [2]. However, projects that encourage the students to design and build a data transmission system will motivate them to enjoy programming the microprocessor and its peripheral chips. The project in this paper will familiarize students with the integral working structure of chips that are used for parallel-to-serial conversions.

Introduction:

Within a computer system, microprocessors communicate in a parallel format with their support chips (e.g., ROM, RAM, etc.). Outside devices, such as modems and printers are serial devices. Therefore, serial-to-parallel and parallel-to-serial conversion must be accomplished.

A typical 7-bit, ASCII (American Standard Code for Information Interchange) character, such as "E" could be transmitted in the asynchronous serial form displayed in **Figure 1**[2].

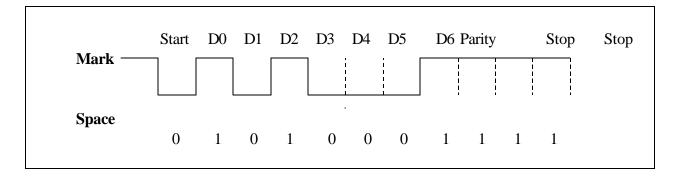


Figure 1- Seven-bit Framed ASCII Serial Word for the Letter "E"

In asynchronous communication there is no common clock between the transmitter and receiver. While for synchronous communication there must be a common clock connection between the transmitter and receiver. In synchronous transmission, after transmission of a few start bit signals, data is transferred continuously.

Asynchronous communication is used more often than synchronous communication. In asynchronous

communication, Start and Stop bits must be added to the ASCII (American Standard Code for Information Interchange) data prior to transmission. One (1), one and half (1.5), or two (2) high pulses indicate the termination of word transmission. Sometimes a parity bit is also added to the data for utilizing the parity check process. Adding start, stop, and parity bits to the ASCII data is called "framing" the data. The transmitter will frame the ASCII word prior to transmission. The receiver will strip the frame bits (i.e., start, stop and parity bits), and save the ASCII word.

A parity check can be used to detect the loss of one bit during the transmission. If the number of high bits in the transmitted word is even, the word has EVEN parity. If the number of high bits in the transmitted word is odd, the word has ODD parity. Prior to transmission, the communication system is set for EVEN, ODD, or NO parity check. Parity bit is used to set the number of high bits to even or odd for the transmitted word. Then, the receiver will check to see if the parity setting is correct. Note that in serial communication, the term "mark" indicates logic one (1) and the term "space" indicates logic zero (0). A low bit indicates the start of data transmission. The transmitted word displayed in Figure 1 has a one start bit (one Mark), and two stop bits (two Space). This transmitted word, has been set for EVEN Parity.

Devices that are used to send or receive serial data (e.g., modem) are called Data Communication Equipment (DCE). The terminals or computers that are sending or receiving data are refereed to as Data Terminal Equipment (DTE). In a **Simplex** communication system, data is transmitted in only one direction. This means that data can either flow from the Data Communication Equipment (DCE) to Data Terminal Equipment (DTE) or vice versa. In a **Half Duplex** communication system data can be transmitted in both directions but not at the same time. This means that data can flow from the DTE to the DCE or vise versa but not simultaneously. In a **Full Duplex** communication system data can be transmitted in both directions simultaneously. This means that both the DTE and the DCE can transmit and receive at the same time.

Parallel-to-serial and serial-to-parallel conversions at the TTL (Transistor -Transistor Logic) level are carried out by support chips such as the 8251A. In the Transistor -Transistor Logic (TTL), the low level logic is between approximately 0 to 1.5V, while the high level logic ranges between 4.5V to 5.5V. There are three types of serial-to-parallel/parallel-to-serial converter chips; (1) Universal Synchronous Receiver/Transmitter (USRT), (2) Universal Asynchronous Receiver/Transmitter (UART), and (3) Universal Synchronous/Asynchronous Receiver/Transmitter (USART). The 8251A is a USART and is capable of Full Duplex communication.

Serial data in the TTL (Transistor / Transistor Logic) level cannot be transmitted due to the transmission line's capacitance. Transmission line capacitance will cause digital signals to be distorted. The Electronic Industrial Association (EIA) has set the standard for a RS232C (Recommended Standard 232C) plug compatibility between equipment made by different manufacturers. In the RS232 standard, logic zero (0) is represented by voltage between +3V and +25V, while logic one (1) can be anywhere

between -3V and -25V. Most RS232C microcomputer interfaces use +12V or +15V to represent a logic zero, and -12V or -15V to represent logic one level. Chips are available for conversion of data from the TTL level to the RS232 standard level and vice versa. For example, the MC1488 manufactured by the Motorola Company can be used to convert TTL to RS-232C. Also, a MC1489 is used to convert RS-232C to TTL [3].

In the following sections a system is described that uses a computer to convert parallel data to framed serial words. Then, the framed words are transmitted in an asynchronous communication protocol format. The system in this project is a Simplex Communication system. We will first describe the system hardware, and then we will describe the system software.

System Hardware:

Figure 2 displays the schematic diagram of a circuit for converting eight-bit parallel data to a framed serial word. The framed word is transmitted by the asynchronous serial communication system. A 74LS245 Octal Transceiver is connected to an eight-bit Dip Switch. The switch is used to set one byte of parallel data for the system. The 74LS245 Octal Transceiver chip is used to transfer the data into a IBM compatible personal computer (PC) via a PC Interface card. The Octal Transceiver will act as a buffer to hold the data for the computer's microprocessor unit (MPU). When the MPU is ready to read the data, it will read the output of the 74LS245 chip.

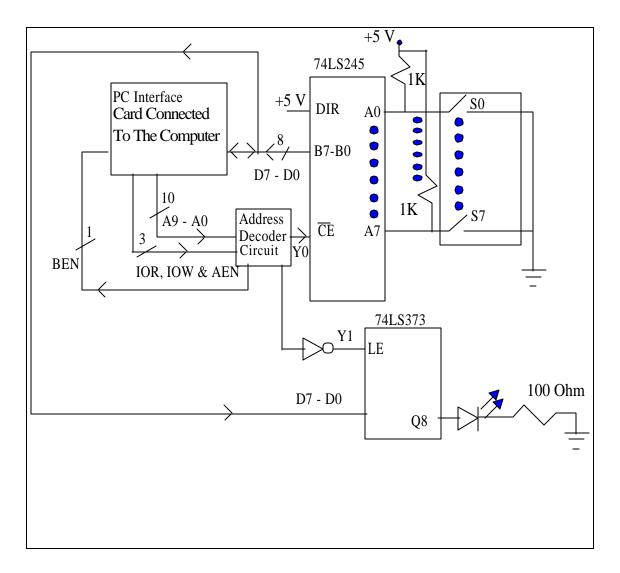


Figure 2- Schematic Block Diagram of the System

The software routine within the computer contains instructions that will frame the parallel data for asynchronous serial transmission. Then the framed serial word is transmitted through the most significant bit (MSB) of the 74LS373 Octal Latch. The 74LS373 latch will hold the transmitted pulse that is displayed on the light emitting diode (LED). A one hundred-ohm resistor is used as a current limiting device so that the output of the 74LS373 chip is not over- loaded. Note that the 74LS373 Octal Latch is also connected to the personal computer via a PC Interface card.

Figure 3 illustrates the pin-out diagram of a PC-Interface card. This card can be placed in a 32-pin Industry Standard Association (ISA) port of an IBM compatible personal computer. The PC-Interface card can be purchased through the Electronic Industries Association (EIA) [4].

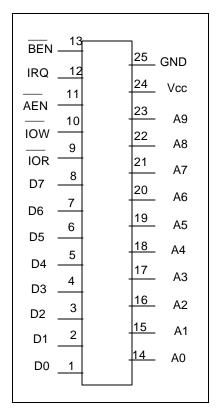


Figure 3 Pin-out Diagram of the PC Interface Card

The PC Interface card will enable us to access eight (8) computer data lines (D7-D0), ten (10) address lines (A9-A0), five control lines, and the five volt computer's power supply (+5 V) and ground (GND). The five-volt (+5V) supply can be used to power the external digital circuit that is connected to the computer via the PC interface card. The control lines consist of Input/Output Read and Write (IOR/IOW), valid Address Enable (AEN), valid data Bus Enable (BEN), and Interrupt Request (IRQ) lines. In the following section we will describe the System Hardware and its connections to the computer.

An address decoder circuit enables the computer to read from, and write to, various peripheral devices. The address decoder actually performs two functions; (1) to connect a decoder device to the data bus, and (2) to make sure only one device is enabled at a time. On most personal computers the address range 300 Hex-to-31F Hex is available for external use. The decoder circuit displayed in **Figure 4** was built using Transistor-Transistor Logic (TTL) gates.

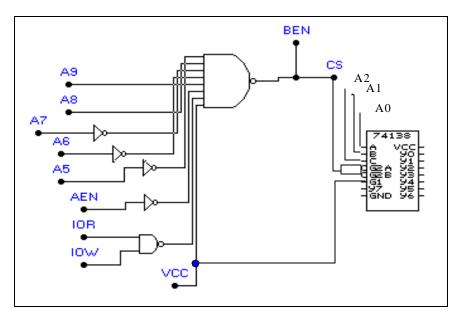


Figure 4 Address Decoder Circuit

The decoder circuit displayed in Figure 4 will generate chip-enable signals for the address range 300H through 307H. Decimal values for these decoded addresses are 768, 769, 770, 771, 772, 773, 774 and 775. Figure 2 displays a circuit used to decode the 74LS245 Octal Transceiver chip and the 74LS373 Octal Latch for addresses 768 (300 Hex) and 769 (301H), respectively.

Five (5) address lines (A5, A6, A7, A8 and A9), and three (3) control lines (IOR, IOW and AEN) are used in the address decoder circuit. The output of the decoder circuit is connected to the Bus Enable (BEN') pin of the PC interfacing card, and the Chip Enable (CE') pin on the 74LS245 Octal Buffer chip.

Table 1 shows the address map of the decoder circuit. If we utilize four (4) 74LS138 (three-to-eight) decoders, this address decoder map can have up to thirty-two (32) addresses, starting with decimal numbers 768 up to 799. This will allow the computer to decode thirty-two (32) different addresses.

																HEX Address Range
0	0	0	0	0	0	1	1	0	0	0	X	X	\$	\	\leftrightarrow	0300h TO 031Fh

Table 1. The Address Decoder Map

Appendix A contains the thirty-two (32) decoded blocks of addresses. However, in the circuit displayed in Figure 4, we have utilized only one (1) 74LS138 chip. Therefore, the computer can communicate with eight (8) different peripheral devices.

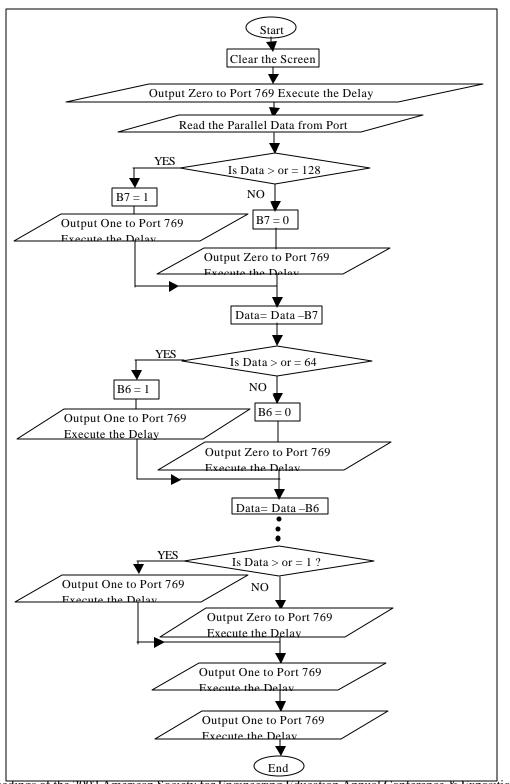
System Software:

An eight-bit Dip Switch is used to generate a random one-byte number. The computer thru the 74LS245 buffer reads this one-byte data that is in parallel data format. Then, the program in the computer is used to convert the data to serial form and frame it for asynchronous transmission. The program will also set the transmission Baud Rate. Baud Rate is the number of pulses traveling per second though the transmission line. The transmission line is connected to the Most Significant Bit (MSB) output pin (Q8) of the 74LS373 Latch. Once the parallel data is converted to serial form and framed for transmission, it will be transmitted at the set Baud Rate. The light emitting diode (LED) on the transmission line is utilized to show the transmitted serial bits.

The program for this project is a "software driver" that binds the computer and the external system hardware. **Figure 5** displays the schematic diagram of the system software. The program code in Quick Basic (Q-Basic) language is listed in Appendix A, and the program code in C language is listed in Appendix B. Therefore, this project can be assigned to students that have not taken high-level computer programming language classes as well as those upper class students who have taken high-level courses.

The computer screen is cleared and then a low bit is sent to the output port address 769 to signal the start of data transmission. Note that output port 769 is connected to the light emitting diode (LED). Next, the parallel data is read from the Dip-Switch setting from input port address 768. Then, compare functions are used to see which bits in the one-byte parallel data are high. Once the status of a bit is determined, the high or low pulse for each bit is sent to the output port address 769. These actions are followed by executions of software delay routines.

When all eight bits in the parallel data are converted to serial format and transmitted, two stop bits will be sent to output port 769. Since the stop bits are high pulses, the light emitting diode (LED) is normally ON. A low pulse will turn the LED off to signal the start of data transmission. This is followed by eight (8) high and/or low pulses that form a one-byte (8-bit) data. Finally, two high pulses will signal the end of the data transmission. The data in this example is framed to have one start bit, eight data bits, no parity bit, and two stop bits.



Proceedings of the 2002 American Society for Engineering Education Annual Conference & Exposition Copyright ©2002, American Society for Engineering Education

Figure 5- The Program Flow Chart

Conclusion:

This project allowed students to be involved in the design and development of digital circuits. They learned how to interface an outside digital circuit to a microprocessor in a personal computer. They also wrote a program that was a software driver for communication between a microprocessor and it's circuit.

Students involved in this project learned in detail the concept of parallel to serial conversion. A project that will read the serial data and convert it to parallel data can also be very beneficial in teaching students the concept of serial to parallel conversion. A similar project could be used to motivate students to explore in detail microprocessor hardware and software engineering. Note that the components utilized for this project are reasonable in cost and available in the commercial market. Therefore, the logistics of supervising the development of projects similar to the one presented in this paper are not costly. However, the level of student participation is high, and the project teamwork will teach students how to interact with hardware and software engineers.

Appendix A:

Decoded Block Addresses;

AE	N IOI	R IO	W A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	ADDRESSES
0	0	0	1	1	0	0	0	0	0	0	0	0	300H = 768
0	0	0	1	1	0	0	0	0	0	0	0	1	301H = 769
0	0	0	1	1	0	0	0	0	0	0	1	0	302H = 770
0	0	0	1	1	0	0	0	0	0	0	1	1	303H = 771
0	0	0	1	1	0	0	0	0	0	1	0	0	304H = 772
0	0	0	1	1	0	0	0	0	0	1	0	1	305H = 773
0	0	0	1	1	0	0	0	0	0	1	1	0	306H = 774
0	0	0	1	1	0	0	0	0	0	1	1	1	307H = 775
0	0	0	1	1	0	0	0	0	1	0	0	0	308H = 776
0	0	0	1	1	0	0	0	0	1	0	0	1	309H = 777
0	0	0	1	1	0	0	0	0	1	0	1	0	30AH = 778
0	0	0	1	1	0	0	0	0	1	0	1	1	30BH = 779
0	0	0	1	1	0	0	0	0	1	1	0	0	30CH = 780
0	0	0	1	1	0	0	0	0	1	1	0	1	30DH = 781
0	0	0	1	1	0	0	0	0	1	1	1	0	30EH = 782
0	0	0	1	1	0	0	0	0	1	1	1	1	30FH = 783
0	0	0	1	1	0	0	0	1	0	0	0	0	310H = 784
0	0	0	1	1	0	0	0	1	0	0	0	1	311H = 785
0	0	0	1	1	0	0	0	1	0	0	1	0	312H = 786
0	0	0	1	1	0	0	0	1	0	0	1	1	313H = 787
0	0	0	1	1	0	0	0	1	0	1	0	0	314H = 788
0	0	0	1	1	0	0	0	1	0	1	0	1	315H = 789
0	0	0	1	1	0	0	0	1	0	1	1	0	316H = 790
0	0	0	1	1	0	0	0	1	0	1	1	1	317H = 791
0	0	0	1	1	0	0	0	1	1	0	0	0	318H = 792
0	0	0	1	1	0	0	0	1	1	0	0	1	319H = 793
0	0	0	1	1	0	0	0	1	1	0	1	0	31AH = 794
0	0	0	1	1	0	0	0	1	1	0	1	1	31BH = 795
0	0	0	1	1	0	0	0	1	1	1	0	0	31CH = 796
0	0	0	1	1	0	0	0	1	1	1	0	1	31DH = 797
0	0	0	1	1	0	0	0	1	1	1	1	0	31EH = 798
0	0	0	1	1	0	0	0	1	1	1	1	1	31FH = 799

Appendix B:

The Program Code in Q-Basic Language;

- 10 CLS
- 15 OUT 769, 0
- 16 FOR T = 1 TO 1000: NEXT T
- 20 D = INP(768)
- 30 IF D > 128 OR D = 128 THEN B7 = 128 ELSE B7 = 0
- 40 IF B7 > 0 THEN OUT 769, 128 ELSE OUT 769, 0
- 50 FOR T = 1 TO 1000: NEXT T
- 60 D = D B7
- 70 IF D > 64 OR D = 64 THEN B6 = 64 ELSE B6 = 0
- 80 IF B6 > 0 THEN OUT 769, 128 ELSE OUT 769, 0
- 90 FOR T = 1 TO 1000: NEXT T
- 100 D = D B6
- 110 IF D > 32 OR D = 32 THEN B5 = 32 ELSE B5 = 0
- 120 IF B5 > 0 THEN OUT 769, 128 ELSE OUT 769, 0
- 125 FOR T = 1 TO 1000: NEXT T
- 130 D = D B5
- 140 IF D > 16 OR D = 16 THEN B4 = 16 ELSE B4 = 0
- 150 IF B4 > 0 THEN OUT 769, 128 ELSE OUT 769, 0
- 160 FOR T = 1 TO 1000: NEXT T
- 170 D = D B4
- 180 IF D > 8 OR D = 8 THEN B3 = 8 ELSE B3 = 0
- 190 IF B3 > 0 THEN OUT 769, 128 ELSE OUT 769, 0
- 200 FOR T = 1 TO 1000: NEXT T
- 210 D = D B3
- 220 IF D > 4 OR D = 4 THEN B2 = 4 ELSE B2 = 0
- 230 IF B2 > 0 THEN OUT 769, 128 ELSE OUT 769, 0
- 240 FOR T = 1 TO 1000: NEXT T
- 250 D = D B2
- 260 IF D2 OR D = 2 THEN B1 = 2 ELSE B1 = 0
- 270 IF B1 > 0 THEN OUT 769, 128 ELSE OUT 769, 0
- 280 FOR T = 1 TO 1000: NEXT T
- 290 D = D B1
- 300 IF D = 1 THEN B0 = 1 ELSE B0 = 0
- 310 IF B0 > 0 THEN OUT 769, 128 ELSE OUT 769, 0
- 320 FOR T = 1 TO 1000: NEXT T

```
330 OUT 769, 1
340 FOR T = 1 TO 100: NEXT T
350 OUT 769, 1
360 \text{ FOR T} = 1 \text{ TO } 1000 : \text{NEXT T}
370 PRINT B7, B6, B5, B4, B3, B2, B1, B0
390 END
Appendix C:
The Program Code in C Language;
/* This program will convert the Parallel data to a Serial format then transmit it. */
#include <stdio.h>
int t=500;
int d;
int set[8];
int on = 128;
int off = 0;
int main()
{
_{\text{outb}}(0,769);
for (t; t>0; t--);
d = inb(768);
if (d>=128)
set[1]=128;
}
else
 set[1]=0;
if (set[1]>0)
_outb(on,769);
else
_outb(off,769);
for (t; t>0; t--);
d=d - set[1];
```

```
if (d>=64)
{
 set[2]=64;
}
else
{
 set[2]=0;
if (set[2]>0)
  _outb(on,769);
else
  _outb(off,769);
for (t; t>0; t--);
d=d - set[2];
if (d>=32)
 set[3]=32;
}
else
{
 set[3]=0;
if (set[3]>0)
 _outb(on,769);
else
{
 _outb(off,769);
for (t; t>0; t--);
d=d - set[3];
if (d>=16)
{
 set[4]=16;
```

```
}
else
{
 set[4]=0;
if (set[4]>0)
 _outb(on,769);
else
{
_outb(off,769);
for (t; t>0; t--);
d=d - set[4];
if (d>=8)
{
 set[5]=8;
}
else
 set[5]=0;
if (set[5]>0)
 _outb(on,769);
else
{
 _outb(off,769);
for (t; t>0; t--);
d=d - set[5];
if (d>=4)
 set[6]=4;
}
else
{
```

```
set[6]=0;
}
if (set[6]>0)
  _outb(on,769);
}
else
  _outb(off,769);
for (t; t>0; t--);
d=d - set[6];
if (d>=2)
{
 set[7]=2;
else
 set[7]=0;
if (set[7]>0)
  _outb(on,769);
else
  _outb(off,769);
for (t; t>0; t--);
d=d - set[7];
if (d>=1)
{
 set[8]=1;
}
else
{
 set[8]=0;
if (set[8]>0)
```

```
{
    _outb(on,769);
}
else
{
    _outb(off,769);
}
for (t; t>0; t--);

_outb(on,769);
for (t; t>0; t--);
_outb(on,769);
for (t; t>0; t--);

_outb(on,769);
for (t; t>0; t--);

printf("%d %d %d %d %d %d %d %d %d",set[1],set[2],set[3],set[4],set[5],set[6],set[7],set[8]);
}
```

References:

- 1. Intel Peripheral Components, Intel Literature Sales, Mt. Prospect, Illinois
- 2. M. Mazidi and J. Mazidi, 80X86 IBM PC and Compatible Computers: Design and Interfacing of IBM PC, PS and Compatible Computers, Volume II, Third Edition, Prentice-Hall, 2000
- 3. Motorola Literature Distribution; P.O. Box 20291; Phoenix, Arizona 85036
- 4. Electronic Industries Association (EIA), Consumer Electronic Group, Washington, D.C.

Max Rabiee earned his Ph.D. in Electrical Engineering from the University of Kentucky in 1987. He is an Associate Professor of Electrical and Computer Engineering Technology at the University of Cincinnati. Dr. Rabiee is a registered professional engineer, and a member of the American Society of Engineering Education (ASEE), the Institute of Electrical and Electronic Engineering (IEEE), the National Association of Industrial Technology (NAIT), the Eta Kappa Nu Electrical Engineering Honor Society and the Tau Beta Pi Engineering Honor Society.