Session 1420

An Embedded Systems Course Using the OOPic Microcontroller

Henry Chaya
Manhattan College
Riverdale, New York

Abstract

A new course titled "Introduction to Embedded Systems" based on the OOPic microcontroller from Savage Innovations has been offered at Manhattan College. The OOPic microcontroller was chosen for several reasons. It can be programmed in several high-level languages similar to Basic, Java or C; it has several predefined objects to facilitate interface programming with a variety of devices; in-circuit debugging is a simple procedure and the device is very affordable. The present course is for sophomore electrical engineering students who have had a course in Visual Basic. Two goals of the course are to teach elements of C++ and to provide interface-programming projects that are creative and interesting. Another goal is to address programming and performance issues relevant to embedded system programming such as timing and event-driven procedures. While the OOPic fulfilled the educational needs of this introductory course, its slow speed and limited memory preclude its use in more advanced courses.

Introduction

Recently the Electrical and Computer Engineering department at Manhattan College has been revising its curriculum for electrical engineering students. One of the goals is to give more attention to high-level programming languages. All engineering students take a required course in Visual Basic during the common freshman year. For electrical engineering students this was the only high level language covered in the curriculum. A course during the Spring of sophomore year covered assembly language programming for the 8051 microcontroller. Feedback from employers and alumni clearly indicated that we needed to improve the coverage of high-level languages.

A course in C or C++ was indicated. With changes in the curriculum, three credits became available during the Fall of the sophomore year. Requiring the introductory C++ course offered by the Computer Science department was an option. However, the department felt hands-on experiments with hardware would stimulate student interest in electrical engineering. So the author prepared a new course that would cover C++ and interfacing to sensors and actuators.

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

Since the students had no formal training in analog or digital electronics, the interfacing hardware had to be kept simple. I considered using the printer port of a PC as an interface, but the complexity of getting by the Windows operating system to control the printer port made this approach unappealing. Since much of our curriculum is based on the 8051, teaching students to program this device in C was an obvious consideration. However, I felt that writing effective C programs for the 8051 would require knowledge about its architecture, and I did not want to duplicate material that would be presented in a later course. I also considered using the Handy Board that is popular in many engineering programs, but this is geared toward the 68HC11 microcontroller and it involved some expense.

I chose to use the OOPic microcontroller from Savage Innovations for the interfacing projects in the course. There are several reasons for this, which are presented below. First, there are some unique features of the OOPic design which merit further explanation

What is an OOPic?

The OOPic from Savage Innovations is a programmable microcontroller that uses an object-oriented language. The compiler offers a choice of syntax based on Basic, C or Java. The microcontroller itself is a PIC 16C74 from Microchip. Its firmware contains a proprietary operating system with a library of predefined objects. Several of these objects support the microcontroller's internal interface hardware. These include 31 digital I/O lines, a four-channel A-to-D converter, two PWM outputs, a high-speed timer/counter, an I2C network and a serial port. Some objects support the interfacing of specific hardware such as a serial EEPROM, an RC servo, a keypad or an LCD display. These objects make it easy for a novice to develop programs to exploit these devices.

The OOPic development board is a compact size of 2" x 3.5". Besides the microcontroller, it has up to two non-volatile EEPROMs to store data and programs, a small prototype area, power regulator and a 40-pin header for interface connections. It also has three I2C serial port connectors. Using these ports, the board can communicate with a host PC though a special cable connected to the PC's printer port. They can also be used to communicate with other OOPics to form a simple network. Low power consumption permits a board to run off a 9V battery.

The OOPic web site at www.oopic.com offers a free integrated development environment, IDE, to support the OOPic. The IDE, which runs under Windows, can compile and download programs to the OOPic board though the I2C interface. It can also query the value of any variable or object property during program execution. This permits a useful form of in-system debugging. The documentation that comes with the IDE is well organized with tutorials, example projects and a complete reference manual for the OOPic's objects and language syntax.

OOPIC Software Organization and the Virtual Circuit

The OOPic language itself has some unique features of software organization. All variables must be declared as objects. The OOPic supports only integers of 1, 4, 8 or 16 bits. A buffer object can store up to 32 bytes that can be interpreted as a string.

Other objects support the on-chip interface hardware. For example, the oA2D object controls the on-chip A-to-D converter. Assigning the Value property of the oA2D object to a variable transfers the A-to-D output to memory. The data can then be manipulated using a conventional program with line-by-line execution. However, OOPic objects can also be "linked" to one another so that data is transferred directly from one object to another in what is termed a "virtual circuit". A virtual circuit is an example of the data-flow model of software organization used in graphical languages such as Labview. Some example programs will help to clarify these ideas.

The following program initializes an oA2D object to read a potentiometer and an oPWM object to control the brightness of an LED. It uses a perpetual while loop to transfer data from the A-to-D converter to the PWM generator.

```
oA2D POT = new oA2D;
oPWM LED = new oPwm;

sub void main(void)
{
   OOPic.Node = 1;
   POT.IOline = 1;
   POT.Operate = cvTrue;
   LED.IOline = 18;
   LED.operate = cvTrue;

   while(1)
     LED.value = Pot.value;
}
```

The same functionality is obtained by "linking" the two objects with an oBus object to form a virtual circuit. The OOPic operating system will continuously transfer data between the A-to-D and PWM even if a program is executing in the foreground.

```
oA2D POT = new oA2D;
oPWM LED = new oPwm;
oBus BUS = new oBUS;

sub void main(void)
{
   OOPic.Node = 1;
   POT.IOline = 1;
   POT.Operate = cvTrue;
   LED.IOline = 18;
   LED.operate = cvTrue;
   BUS.Input.Link(POT);
   BUS.Output.Link(LED);
   BUS.Operate = cvTrue;
}
```

Besides the oBus, there are a number of "processing" objects that provide various functions and data manipulation operations for virtual circuits. In the OOPic, virtual circuits run faster performing 100,000 instructions per second as compared to 2000 for a conventional program.

Event-Driven Programming

Of special interest is the oEvent object, which will execute a specific subroutine each time a certain logic value in the virtual circuit goes high. This can be used to demonstrate how event-driven routines such as interrupts are used in embedded software. The following program demonstrates how on event-driven counter can be used for frequency measurement. A second event-driven routine updates the frequency variable and resets the counter once a second.

```
// VeryLowFreq.osc

// This program uses an event-driven function to count
// pulses.  It can only count up to 80 counts per second.

oDIO1 Counter_input = new oDIO1;
oEvent Very_Slow_Counter = new oEvent;
oWord Count = new oWord;
oWord Frequency = New oWord;
oEvent Load_Freq = new oEvent;
oGate Wire_load = new oGate;
oGate Wire_count = new oGate;

Sub void main(void)
{
  OOpic.node = 1;

  Counter_input.IOline = 16;
  Counter_input.Direction = cvInput;

  Wire_count.input1.link(Counter_Input.Value);
  Wire_count.output.Link(Very_Slow_Counter.operate);
  Wire_count.Operate = cvTrue;

  Wire_load.input1.link(oopic.Hz1);
  Wire_load.output.Link(Load_Freq.operate);
  Wire_load.Operate = cvTrue;

}
sub void Very_Slow_Counter_Code(void)
{
    count++;
}

sub void Load_Freq_Code(void)
{
  Frequency = count;
  count = 0;
}
```

This program can only measure frequencies up to 80 Hz because the program execution rate is so slow. Using a virtual circuit to increment the count variable, frequencies up to 7 kHz can be

measured due to increased execution speed. It is also possible to measure frequencies up to 5 MHz using the OOPic's built-in hardware counter. This example clearly demonstrates the tradeoffs between using hardware and software to implement an interface.

Why Choose the OOPic for the Course?

One goal of the present course was to introduce software design for embedded systems. This included multi-tasking, interfacing sensors and actuators, timing requirements, event-driven programming, memory resources and inter-processor communication. The OOPic provided a platform where all these topics could be explored. The following features made it particularly attractive:

1. Use of a PIC microcontroller with built-in interfacing hardware
2. Ease of interfacing a variety of devices supported by the object library.
3. Programming in a high-level language with C-style syntax.
4. In-system debugging capability.
5. Good documentation and support.
6. Support for a variety of software organization paradigms.
7. Networking capability via the I2C interface
8. Portability. (Students could check out an OOPic "kit".)
9. Low-cost. (The IDE is free.)

There were also some drawbacks.

1. Slow execution speed (2000 instructions per second).
2. Limited data types, (only unsigned integers for OOPic I).
3. Limited memory for variables and objects (86 bytes)
4. No support for local variables.

Most of the advantages listed above were discussed earlier, but items 8 and 9 deserve additional comment. To support the course I purchased a variety of hardware and assembled it as a kit, which each team of two students could sign out. The kit included an LED array, a potentiometer, a sound transducer, a Sharp GP2D12 infrared distance sensor, a 4x4 keypad, a16x2 character LCD display, a dual seven segment display, a stepper motor, a protoboard and the necessary cables and interface ICs. A kit cost about $100 including the OOPic and fit into a box the size of a thin textbook.

The major disadvantages of the OOPic are its slow speed and limited memory. The execution speed of 2000 instructions per second was adequate for the projects used in the course. However, small memory size is a critical limitation. There are only 86 bytes available in the OOPic's RAM for variables and objects. This precludes the use of sophisticated data structures or even local variables. While the memory size could accommodate simple interfacing programs, students who attempted elaborate projects found it a limitation.

Another goal of the course was to introduce students to advanced programming concepts encountered in C programming such as call-by-value, pointers and linked lists. The OOPic could support none of these. To fulfill this goal, there were several programming assignments using Borland's C++.

Student Assignments and Projects

The students completed a number of interfacing experiments and a final project with the OOPic. A review of these will illustrate how the OOPic was used during the course.

The initial experiments dealt with digital IO, analog-to-digital conversion, and timers. The students interfaced an LED array, a potentiometer, a distance sensor and a dual seven-segment display. One assignment was to get the LEDs to flash in a particular pattern at a rate controlled by the potentiometer. Another assignment was to calibrate the distance sensor and display the distance on the seven-segment display.

The intermediate assignments dealt with using pulse-width modulation to generate sound, interfacing a keypad, a stepper motor and an LCD display. Students were asked to develop a library of subroutines to support the stepper motor and use them to execute a complex motion sequence using an event-driven program. Most students found this a significant challenge. Another demanding assignment was to convert an arbitrary integer into a string of ASCII digits to display on the LCD.

Two advanced applications were done with the OOPic as demonstrations. One was to control a DC servomotor with an H-bridge and a quadrature encoder. Another was to play the school Alma Mater with the tempo controlled by moving a hand near a distance sensor.

Each team of two students had to complete a project during the last three weeks of the course. One group chose to build an XY plotter run by stepper motors. The most sophisticated part of this project was the use of the Bresenham line-drawing algorithm. Another group tried to convert a remote-control vehicle built for a freshman design project into an autonomous robot. It was an ambitious effort that met with limited success due to time constraints. Another project was a security lock using the keypad and the LCD display. The fourth project was a tone generator for musical training.

Student Feedback on the OOPic

In the course assessment survey, students were asked to comment about the advantages and disadvantages of the OOPic. The response was overwhelmingly positive. Students appreciated the ease of programming, the portability and the cost effectiveness. They were appreciative of the opportunity to apply their programming skills with hands-on interfacing projects.

Only a few disadvantages were noted. Some students found that the small memory size was a limitation during their projects. A couple of students observed that the OOPic language was not ANSII C and did not reinforce the concepts learned in class such as local variables.

Conclusion

The OOPic offers some significant advantages, and it helped the course to meet its goals. Its biggest advantages are low cost and an easy learning curve. I highly recommend it for a high school or an introductory college level course on embedded systems. It can be very effective for stimulating student design ideas.

The OOPic is clearly not the platform for advanced programming concepts or advanced embedded system design. It was effective in the present course and as a supplement to traditional C++ programming topics. Clearly, however, the OOPic will not supplant the 8051 microprocessor in our curriculum. My recommendation to the curriculum committee is to place the present course after the introductory microprocessor course. Then the course could effectively cover C programming for the 8051. This would better integrate the course into the curriculum, and give students a marketable skill. However, if the course must remain where it is, then I would continue to use the OOPic.

Further Information About the OOPic

The web site www.oopic.com offers complete documentation on the OOPic including tutorials and sample projects.

Henry Chaya
Henry Chaya is an Associate Professor of Electrical and Computer Engineering at Manhattan College in Riverdale New York. His professional interests include robotics and automation, artificial intelligence, embedded control and digital systems design. He has contributed extensively to the development of laboratory courses including design of custom hardware and software. Dr. Chaya has taught introductory programming courses for many years. Recently, He completed a term as chairman of the department. He received a BS degree from Manhattan College in 1973 and a Ph.D. from Princeton University in 1981. Dr. Chaya is a member of the Brothers of the Christian Schools.