Implementing CPLD-Based Interfaces for Sensors and Actuators in a Mechatronics Design Course

Nicholas Krouglicof
Union College, Mechanical Engineering Department
E-Mail: krouglin@union.edu

Abstract
Complex **P**rogrammable **L**ogic **D**evices (*CPLDs)* are a class of programmable logic device that are commonly used to implement complex digital designs on a single integrated circuit. Current applications of CPLDs in the field of computer engineering include the implementation of bus controllers, address decoders, communication interfaces, etc. This paper outlines a novel application for CPLDs in the field of mechatronics. A low cost, microcontroller-based data acquisition system has been developed that incorporates both a user programmable microcontroller and a user reconfigurable CPLD. The CPLD basically provides reconfigurable digital I/O that permits the implementation of interfaces for smart sensors and actuators. Typical applications include quadrature decoder/counter interfaces for optical encoders, stepper motors controllers and **P**ulse-**W**idth **M**odulation (PWM) motor drives. By incorporating a CPLD that supports **I**n-**S**ystem **P**rogrammability (ISP) the target device can be reprogrammed by the user for a variety of applications without removing it from the host system.

Introduction

Mechatronics can be defined as a design philosophy which encourages engineers to integrate precision mechanical engineering, digital and analog electronics, control theory and computer engineering in the design of "intelligent" products, systems and processes rather than engineering each set of requirements separately. The advantages of the mechatronics approach to design are shorter design cycles, lower costs, and elegant solutions to design problems that can not easily be solved by staying within the bounds of the traditional engineering disciplines.

With an underlying focus on integration, the *Mechatronics Design* course (MER-180) at Union College emphasizes the fundamental technologies on which contemporary mechatronic designs are based: sensors and actuators, system dynamics and control, analog and digital electronics, microcontroller technology, interface electronics and real-time programming. The laboratory sessions focus on small, hands-on interdisciplinary design projects in which small teams of students configure, design, and implement a succession of mechatronic subsystems, leading to system integration in a final project.

For example, as an introduction to digital design, students apply the fundamental principals of combinatorial and sequential logic to the design of a quadrature decoder/counter circuit that is used to interface an incremental optical encoder to a microcontroller. The design is implemented using the appropriate software development tools and tested on a **C**omplex **P**rogrammable **L**ogic **D**evice (CPLD). **C**omplex

Programmable **L**ogic **D**evices (*CPLDs)* are commonly used to implement complex digital designs on a single integrated circuit. As part of the final design project, students integrate the interface circuit and optical encoder with a DC servomotor / lead-screw assembly to construct a servomechanism which controls one axis of a simple machine tool.

While there are numerous applications for CPLDs in the field of computer engineering [1] (e.g., bus controllers, address decoders, communication interfaces, simple microprocessors), this paper outlines several novel "mechatronic" applications for CPLDs that can be readily implemented in an undergraduate laboratory setting. Pedagogically these applications serve as both case studies for introducing various digital design methodologies as well as a "toolbox" of predefined modules that can be integrated into a final design project.

Hardware Platform

The primary hardware platform for the Mechatronics Design course (MER-180) at Union College is the UC^2 system (***U****nion* ***C****ollege* ***U****niversal* ***C****ontroller*); a system tailored to the needs of engineering students at Union College. This novel, low cost, microcontroller-based system enables students to interface a variety of sensors and actuators to their laptop computers in a laboratory or studio classroom environment. The system is unique in that it functions as a data acquisition system, stand-alone controller or data logger. As illustrated in Figure 2, the UC^2 system incorporates both a user programmable microcontroller and a user configurable Complex Programmable Logic Device (CPLD). For a more detailed description of the UC^2 system, refer to reference [2].

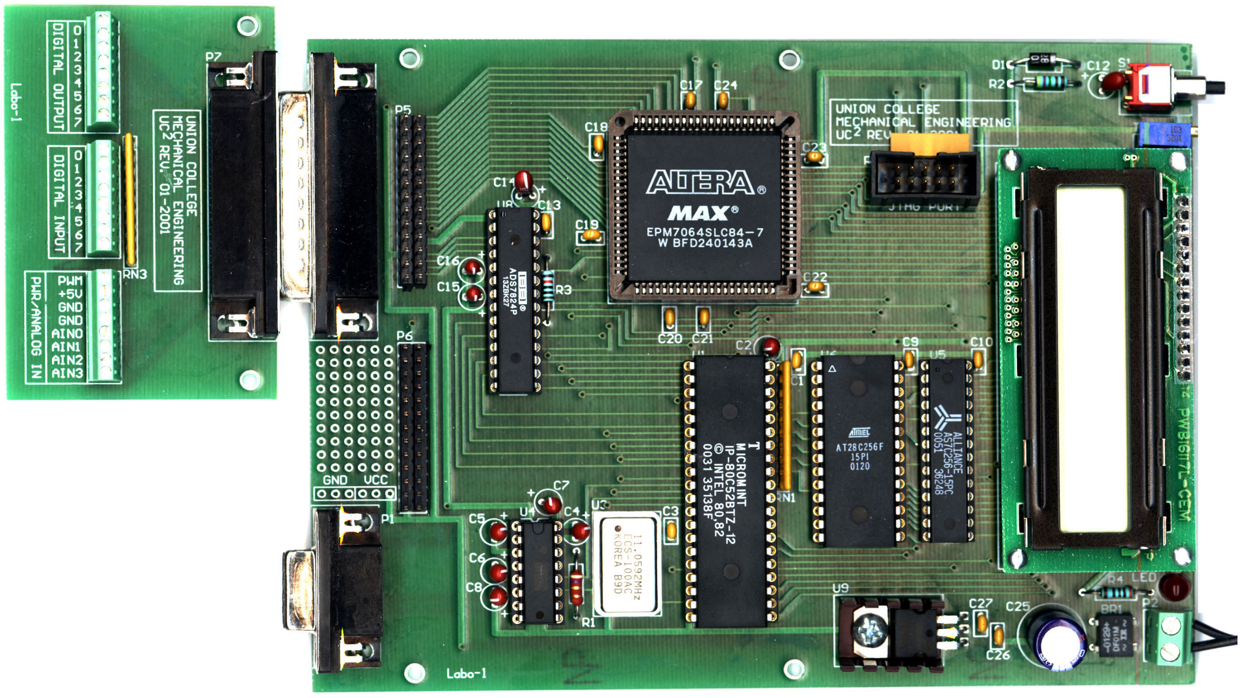


Figure 1: Image of populated printed circuit board for the UC^2 system.

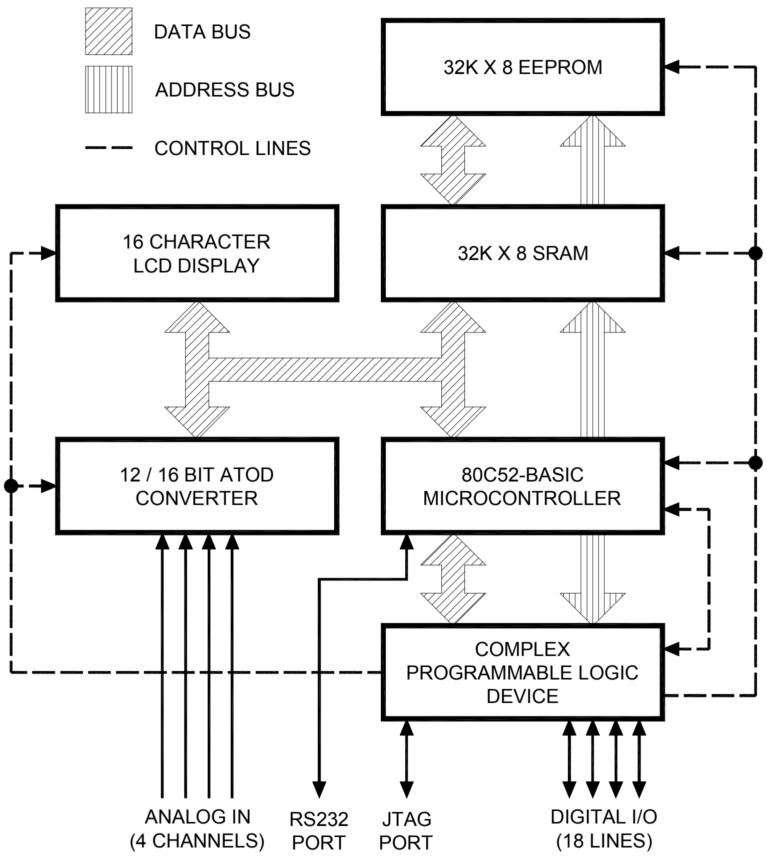


Figure 2: Block Diagram of System

Complex Programmable Logic Device

Perhaps the most innovative feature of the UC^2 system is the incorporation of a user configurable Complex Programmable Logic Device (CPLD). The CPLD handles all the routine tasks normally associated with embedded controller applications; data latching, address decoding, and memory management. This effectively eliminates the need for any discrete logic on the board. In addition, approximately 75% of the internal resources and 18 IO pins on the CPLD are made available to the user for custom applications. The CPLD supports in-system programmability (ISP) via the IEEE 1149.1 **J**oint **T**est **A**ction **G**roup (JTAG) test port. This permits the target device to be reprogrammed by the user without removing it from the host system. Code for the CPLD is developed in either in Verilog HDL (**H**ardware **D**escription **L**anguage), VHDL (**V**ery High Speed Integrated Circuit **H**ardware **D**escription **L**anguage) or AHDL (**A**ltera **H**ardware **D**escription **L**anguage) using the MAX+plus II 9.23 Baseline development environment from ALTERA. Educational institutions can obtain a MAX+plus II software license for the specific device used in the UC^2 system at no cost over the WEB.

Clearly reconfiguring the CPLD is beyond the capabilities of many students. In recognition of this, the 18 free IO pins on the device are predefined as 8 digital input lines and 8 digital output lines plus a free chip select (CS) and a PWM (**P**ulse-**W**idth-

Modulation) signal. These IO signals are all readily accessible by the user through the microcontroller via the MCS BASIC-52 programming language.

For the advanced user the CPLD provides reconfigurable digital I/O that facilitates the implementation of hardware interfaces for smart sensors and actuators. This paper details several typical "mechatronic" applications that have been successfully implemented in hardware (i.e., in the CPLD) on the UC^2 system. These applications include:

- A quadrature decoder/counter interface for an incremental optical encoder
- A unipolar, half-stepping, stepper motor controller
- A Pulse-Width-Modulation (PWM) driver for controlling DC motors.

Conventional tasks (e.g., address decoding) are not the primary reason for incorporating the CPLD, but can also be explored in a laboratory environment.

Quadrature Decoder/Counter Interface

In digital closed loop motion control systems, optical encoders are customarily used to translate the rotary motion of a shaft into digital form. Optical encoders typically employ a Light Emitting Diode (LED) as a light source (or emitter) and a photodiode as a detector. A codewheel (Figure 3) rotates between the emitter and detector, causing the light from the emitter to be interrupted by the radial slots in the codewheel. The angular position of the shaft is evaluated by counting the pulses generated by the detector. For bidirectional operation, a second emitter/detector pair is positioned on the circumference of the code wheel so that when the first detector (channel A) reads a slot, the second detector (channel B) reads a bar. The digital output of channel A is said to be in quadrature with that of channel B (i.e., 90 degrees out of phase). When the codewheel rotates in the counterclockwise direction, channel A will lead channel B and the system must count up. In the clockwise direction, channel B leads channel A and the system must count down (Refer to Figure 4).

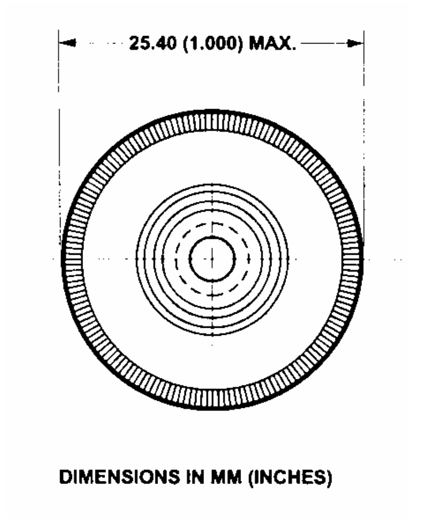


Figure 3: HEDS-5120 Codewheel from Agilent Technologies.

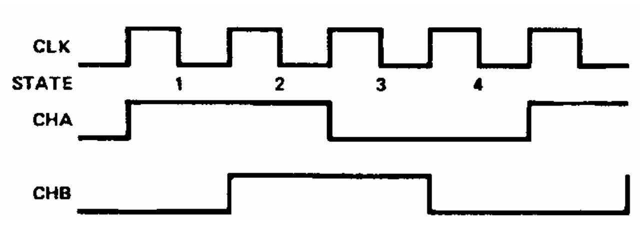


Figure 4: Quadrature decoding timing diagram illustrating the four possible states.

As illustrated in Figure 4, channels A and B can be in one of four possible states. Based on the past binary state of the two signals and the present state, the binary counter must be either incremented or decremented as illustrated in Figure 5. By counting high-to-low or low-to-high transitions on both channels, a resolution corresponding to four times the basic resolution of the codewheel can be achieved. Thus a typical 512 slot codewheel yields an effective resolution of 2048 counts per revolution.

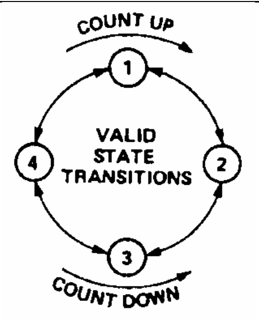
 COUNT UP 1 2 3 4 VALID STATE TRANSITIONS COUNT DOWN	Channel A	Channel B	State
	1	0	1
	1	1	2
	0	1	3
	0	0	4

Figure 5: Valid state transitions.

CPLD Implementation

The CPLD implementation of the quadrature decoder/counter interface consists of four modules; edge detectors for channels A and B, quadrature decoder logic, a 16-bit binary counter and bus interface circuitry.

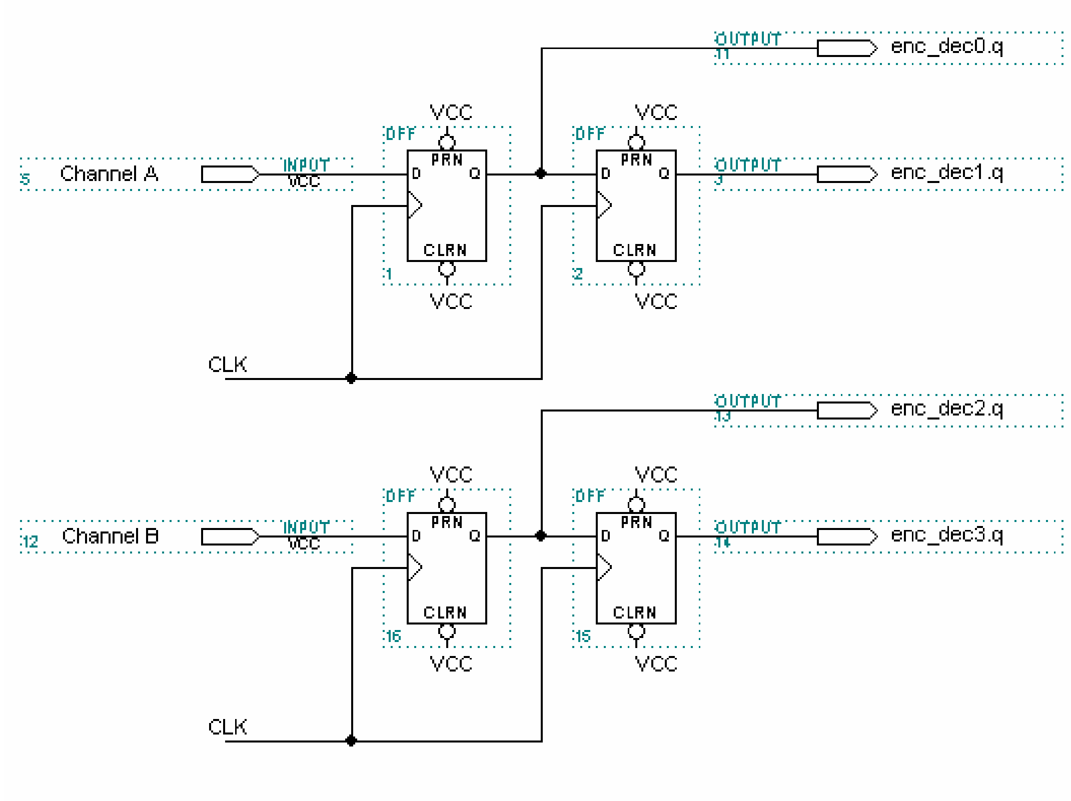


Figure 6: Edge-detection circuitry.

The edge detectors operate by sampling the incoming channels at a frequency significantly higher than the fundamental encoder frequency. In the case of the UC^2 system, the global clock is set at 11.0592 MHz which results in a maximum operational speed of over 300,000 rpm for a typical 512 slot codewheel. As illustrated in Figure 6, a two-bit shift register employing D-type flipflops is used to detect transitions on channels A and B.

By way of example, a rising edge on channel A is detected when *enc_dec0.q* and *enc_dec1.q* are "1" and "0" respectively. The transitions between the various states illustrated in Figure 5 can be decoded by analyzing the outputs of the four D-type flipflops as shown in Table 1 below.

State Transition			**Encoder Signals**		**Flipflop outputs: *enc_decX.q***			
Past State	**Present State**	**Count**	**Chan A**	**Chan B**	**0**	**1**	**2**	**3**
1	2	UP	1	↑	1	1	1	0
2	3	UP	↓	1	0	1	1	1
3	4	UP	0	↓	0	0	0	1
4	1	UP	↑	0	1	0	0	0
1	4	DOWN	↓	0	0	1	0	0
4	3	DOWN	0	↑	0	0	1	0
3	2	DOWN	↑	1	1	0	1	1
2	1	DOWN	1	↓	1	1	0	1

Note: ↑ and ↓ indicate a rising and falling edge respectively.

Table 1: Summary of quadrature decoding logic.

Although there are several methods of implementing state machines in VHDL or AHDL, this particular example was implemented by means of a truth table. In an AHDL truth table, each entry in the table contains a combination of input values that will produce specified output values.

```
TABLE
        enc_dec[3..0].q       =>        dwn_cnt,        up_cnt;

        B"0000"                     =>     0,              0;
        B"0001"                     =>     0,              1;
        B"0010"                     =>     1,              0;
        B"0011"                     =>     0,              0;
        B"0100"                     =>     1,              0;
        B"0101"                     =>     0,              0;
        B"0110"                     =>     0,              0;
        B"0111"                     =>     0,              1;
        B"1000"                     =>     0,              1;
        B"1001"                     =>     0,              0;
        B"1010"                     =>     0,              0;
        B"1011"                     =>     1,              0;
        B"1100"                     =>     0,              0;
        B"1101"                     =>     1,              0;
        B"1110"                     =>     0,              1;
        B"1111"                     =>     0,              0;

END TABLE;
```

Table 2: Truth table implementation of quadrature decoder.

The two nodes *up_cnt* and *dwn_cnt* are subsequently used to increment or decrement a 16-bit binary counter which holds the encoder pulse count. Note that for the majority of states, nodes *up_cnt* and *dwn_cnt* are both “false” and the current count is simply maintained. In the AHDL implementation of the counter module shown below, the variable *encoder* is a 16-bit array of D-type flipflops which allows for 32 complete shaft revolutions for a 2048 (effective) pulse per revolution encoder.

```
IF up_cnt THEN
                encoder[].d = encoder[].q + 1;
        ELSE
                IF dwn_cnt THEN
                        encoder[].d = encoder[].q - 1;
                ELSE
                        encoder[].d = encoder[].q;
                END IF;
        END IF;
```

The interface circuitry allows the microcontroller to both read and reset the binary counter. In many respects this is the most complex part of the design since it involves implementing a tri-state, bidirectional bus. In addition, since the microcontroller data bus is 8 bits wide, the **M**ost **S**ignificant **B**yte (MSB) of the encoder count must be latched into a data buffer when the **L**east **S**ignificant **B**yte (LSB) is read to prevent a false count due to delays between subsequent microcontroller reads. The interface circuitry is beyond the scope of this paper but the basic implementation is summarized below.

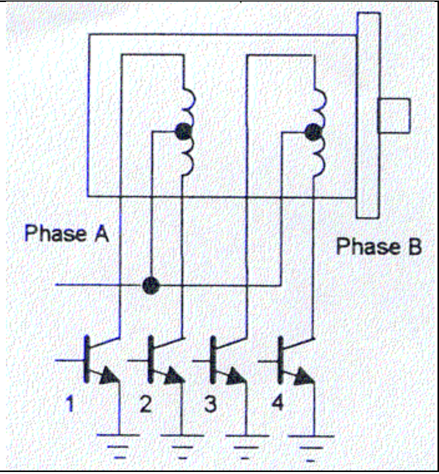
```
        tri_node[7..0]          = tri_enc_low[7..0].out;
        tri_enc_low[7..0].in    = encoder[7..0].q;
        tri_enc_low[7..0].oe    = enc_low & !read/;

% MSB of encoder count is latched when LSB is read to prevent a false
count due to delay between reads %

        enc_latch[].clk         = clk;
        enc_latch[].clrn        = VCC;
        enc_latch[].prn         = VCC;
        enc_latch[].ena         = enc_low & !read/;
        enc_latch[].d           = encoder[15..8];

        tri_node[7..0]          = tri_enc_high[7..0].out;
        tri_enc_high[7..0].in   = enc_latch[7..0].q;
        tri_enc_high[7..0].oe   = enc_high & !read/;
```

The LSB and MSB of the binary counter are mapped to memory locations *enc_low* and *enc_high* respectively. The bidirectional data bus is accessed via a tri-state node (*tri-node[7..0]*) which is connected to two tri-state primitives corresponding to the LSB and MSB of the binary counter.

Unipolar Half-Stepping, Stepper Motor Controller

Stepper motors are characterized as *bipolar* or *unipolar*. Bipolar stepper motors have four lead wires and require a total of eight drive transistors (i.e., two full H-bridges). Unipolar have an additional center-tap on each phase for a total of six lead wires. With the center-taps connected to a common voltage source, unipolar stepper motors can be controlled with four identical NPN or N-channel drive transistors (Figure 7). In conventional full-stepping mode, one motor phase is energized at a time resulting in minimum power consumption and high positional accuracy regardless of winding imbalance. Half-stepping control alternates between energizing a single phase and two phases simultaneously resulting in an eight-step sequence which provides higher resolution, lower noise levels and less susceptibility to motor resonance.

The desired drive waveforms are illustrated in Figure 7. The eight step drive sequence shown (steps 1 through 8) advances the stepper motor four full steps or eight half steps. Reversing the drive sequence (i.e., from step 8 towards 1) reverses the direction of rotation.



STEP	TRANSISTORS			
	1	2	3	4
1	ON	OFF	ON	OFF
2	ON	OFF	OFF	OFF
3	ON	OFF	OFF	ON
4	OFF	OFF	OFF	ON
5	OFF	ON	OFF	ON
6	OFF	ON	OFF	OFF
7	OFF	ON	ON	OFF
8	OFF	OFF	ON	OFF

Figure 7: Half-step switching sequence for unipolar stepper motor.

CPLD Implementation

As with the decoder/counter interface, there are several possible methodologies for implementing a stepper motor controller on a CPLD [3] [4], however synthesizing the desired waveforms by applying the basic principles of synchronous sequential circuit design has significant pedagogical value.

The basic design is that of a conventional synchronous counter employing JK-type flipflops running off a common clock. In this particular case, four flipflops are required to generate the required waveforms for the four drive transistors. In order to achieve speed control, the common clock is generated by the microprocessor programmable timer. As shown in Tables 3 and 4, the two inputs to the flipflops, J and K, are synthesized from the current output state of the flipflops using conventional combinatorial logic. The appropriate Boolean logic functions are determined by applying Karnaugh maps of five variables. Four of the variables are the current outputs of the flipflops (Q0 to Q3). The fifth variable, Qt, is a direction bit. If Qt equals zero the sequence advances from step 1 towards 8, whereas if Qt equals 1, the direction is reversed. The complete schematic for the unipolar, half-stepping, stepper motor controller is shown in Figure 8 while the AHDL implantation is shown in Table 5.

Although this particular design may appear somewhat complex, it was successfully completed by senior undergraduate students who genuinely appreciated seeing the fruits of their labors implemented in hardware and tested on a real stepper motor.

Half-Stepping Forward and Backward

Present Q3Q2Q1Q0	Input Qt	Next Q3Q2Q1Q0	J3	K3	J2	K2	J1	K1	J0	K0
1010	0	1000	X	0	0	X	X	1	0	X
1010	1	0010	X	1	0	X	X	0	0	X
1000	0	1001	X	0	0	X	0	X	1	X
1000	1	1010	X	0	0	X	1	X	0	X
1001	0	0001	X	1	0	X	0	X	X	0
1001	1	1000	X	0	0	X	0	X	X	1
0001	0	0101	0	X	1	X	0	X	X	0
0001	1	1001	1	X	0	X	0	X	X	0
0101	0	0100	0	X	X	0	0	X	X	1
0101	1	0001	0	X	X	1	0	X	X	0
0100	0	0110	0	X	X	0	1	X	0	X
0100	1	0101	0	X	X	0	0	X	1	X
0110	0	0010	0	X	X	1	X	0	0	X
0110	1	0100	0	X	X	0	X	1	0	X
0010	0	1010	1	X	0	X	X	0	0	X
0010	1	0110	0	X	1	X	X	0	0	X

Table 3: Truth Table for half-step switching sequence.

	000	001	011	010	110	111	101	100
00	X	1	0	0	X	X	X	X
01	X	0	0	0	X	X	X	X
11	1	X	X	0	X	X	X	X
10	0	X	X	0	X	X	X	X

J3 = Q2' *Q0' *Qt' + Q2' *Q1' *Qt

	000	001	011	010	110	111	101	100
00	X	X	X	X	X	X	0	0
01	X	X	X	X	X	X	1	0
11	X	X	X	X	X	X	X	0
10	X	X	X	X	X	X	X	1

K3 = Q3*Q1*Qt + Q0*Qt

	000	001	011	010	110	111	101	100
00	X	0	X	X	X	X	0	0
01	X	1	X	X	X	X	0	0
11	0	X	X	X	X	X	X	0
10	1	X	X	X	X	X	X	0

J2 = Q3' *Q1*Qt + Q3' *Q0*Qt'

	000	001	011	010	110	111	101	100
00	X	X	1	0	X	X	X	X
01	X	X	0	0	X	X	X	X
11	X	X	X	1	X	X	X	X
10	X	X	X	0	X	X	X	X

K2 = Q3' *Q1*Qt' + Q0*Qt

	000	001	011	010	110	111	101	100
00	X	X	X	1	X	X	X	0
01	X	X	X	0	X	X	X	1
11	0	X	X	0	X	X	X	0
10	0	X	X	0	X	X	X	0

J1 = Q3' *Q0' *Qt' + Q3*Q0' *Qt

	000	001	011	010	110	111	101	100
00	X	0	0	X	X	X	1	X
01	X	0	1	X	X	X	0	X
11	X	X	X	X	X	X	X	X
10	X	X	X	X	X	X	X	X

K1 = Q2*Qt + Q3*Q0' *Qt'

	000	001	011	010	110	111	101	100
00	X	0	0	0	X	X	0	1
01	X	0	0	1	X	X	0	0
11	X	X	X	X	X	X	X	X
10	X	X	X	X	X	X	X	X

J0 = Q2*Q1' * Qt + Q2' *Q1' *Qt'

	000	001	011	010	110	111	101	100
00	X	X	X	X	X	X	X	X
01	X	X	X	X	X	X	X	X
11	0	X	X	0	X	X	X	1
10	0	X	X	1	X	X	X	0

K0 = Q3*Qt + Q2*Qt'

Table 4: Karnaugh maps for stepper motor half-stepping synchronous counter design.

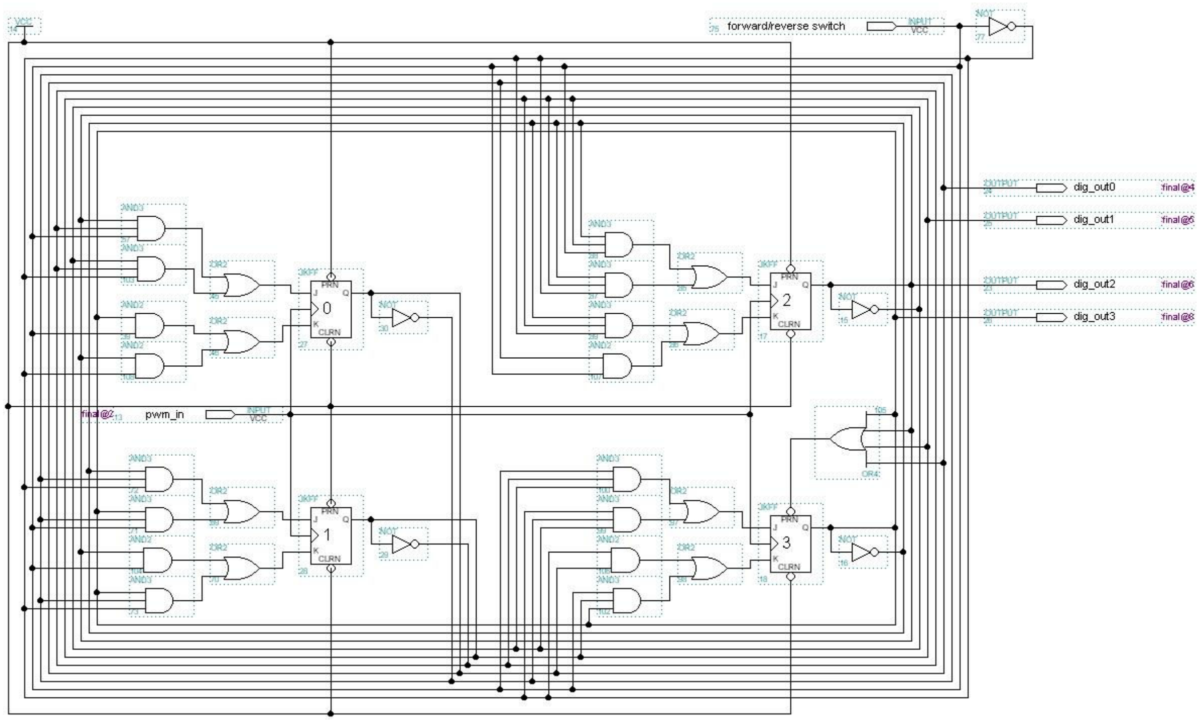


Figure 8: Schematic of stepper motor controller.

```
stepmot3.CLK      =pwm_in;
stepmot3.K =stepmot3.q & stepmot1.q & data0 # stepmot0.q & !data0;
stepmot3.J =!stepmot2.q & !stepmot0.q & !data0 # !stepmot2.q & !stepmot1.q & data0;
stepmot3.clrn =VCC;
stepmot3.prn =stepmot3.q # stepmot2.q # stepmot1.q # stepmot0.q;

stepmot2.CLK      =pwm_in;
stepmot2.K =!stepmot3.q & stepmot1.q & !data0 # stepmot0.q & data0;
stepmot2.J =!stepmot3.q & stepmot1.q & data0 # !stepmot3.q & stepmot0.q & !data0;
stepmot2.clrn =VCC;
stepmot2.prn  =VCC;

stepmot1.CLK      =pwm_in;
stepmot1.K =stepmot2.q & data0 # stepmot3.q & !stepmot0.q & !data0;
stepmot1.J =!stepmot3.q & !stepmot0.q & !data0 # stepmot3.q & !stepmot0.q & data0;
stepmot1.clrn =VCC;
stepmot1.prn =VCC;

stepmot0.CLK      =pwm_in;
stepmot0.K =stepmot3.q & data0 # stepmot2.q & !data0;
stepmot0.J =!stepmot1.q & stepmot2.q & data0 # !stepmot2.q & !stepmot1.q & !data0;
stepmot0.clrn =VCC;
stepmot0.prn  =VCC;

dig_out0 =stepmot3.q;
dig_out1 =stepmot2.q;
dig_out2 =stepmot1.q;
dig_out3 =stepmot0.q;
```

Table 5: AHDL implementation of stepper motor controller.

Pulse Width Modulation Controller
Pulse Width Modulation (PWM) amplifiers are commonly used to control the speed or torque or DC servomotors. They operate by switching the DC supply voltage to the motor "on" and "off" at a fixed frequency (typically 10 kHz). The average current through the motor is controlled by varying the duty cycle, i.e., the ratio of the "on" time to the period of the PWM waveform. This effectively changes the speed and torque at the output of the motor.

CPLD Implementation
The CPLD implementation of the PWM controller is perhaps the simplest mechatronic example presented in this paper. The microprocessor basically sets the period of the PWM waveform (*totaltime*) and the "off" time (*lowtime*) in clock cycles (Figure 9). These 8 bit values are latched by the CPLD into two 8 bit registers consisting of D-type flipflops. These values are compared to a free running counter (*cntr*). An output pin is set when *cntr* exceeds *lowtime* and *cntr* is reset when it reaches *totaltime*. The complete AHDL implementation of the PWM is given in Table 6.

This particular implementation allows both the duty cycle and base frequency to be set by the microcontroller. If the base frequency is fixed at say 256 clock cycles then an even simpler implementation is possible.

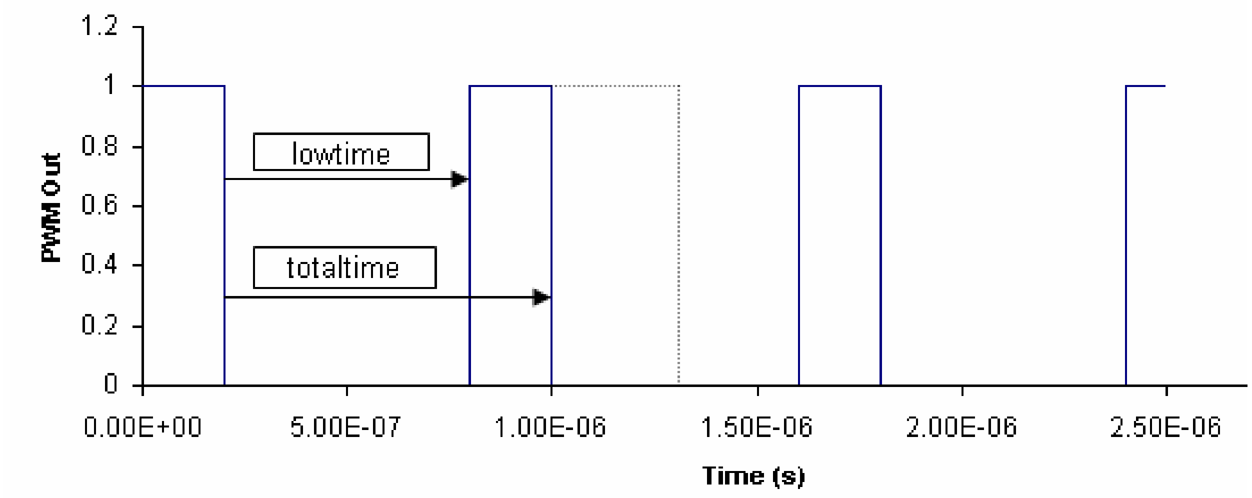


Figure 9: Drive waveform for PWM controller.

Conclusions
Undergraduate students are increasingly involved in mechatronic design projects that call for small, microcontroller-based, stand-alone controllers. For example, student participating in the SAE Walking Machine Challenge must design an intelligent, autonomous system that must perform a variety of simple tasks without human intervention. Both the UC^2 system and the CPLD-based interfaces described in this paper have been used extensively in such projects. Students have also adapted the three basic interfaces for specific sensors and actuators. For example, a custom PWM controller has been developed for driving **R**adio **C**ontrol (RC) servos directly from the UC^2 system.

In the *Mechatronics Design* course (MER-180) at Union College students apply the fundamental principals of combinatorial and sequential logic to the design of the various mechatronic interfaces described in this paper. The designs are implemented using the appropriate software development tools and tested on the UC^2 system with the appropriate sensors and actuators (e.g., encoder, stepper motor). As part of the final design project, students integrate the various modules developed into servomechanisms, robots or autonomous vehicles. As a general observation, students greatly appreciate the hands on experience and the ability to see their designs in operation (as opposed to on paper or in computer simulations).

```
totaltime[].clk  = clk;
totaltime[].clrn = VCC;
totaltime[].prn  = VCC;
totaltime[].ena  = io_space_2 & (add_latch[3..0].q == B"0110") & !write/;
%7ff6h%
totaltime[].d    = add_data[];

lowtime[].clk  = clk;
lowtime[].clrn = VCC;
lowtime[].prn  = VCC;
lowtime[].ena  = io_space_2 & (add_latch[3..0].q == B"0111") & !write/;
%7ff7h%
lowtime[].d     = add_data[];

cntr[].clk  = clk;
cntr[].clrn = !(cntr[].q == totaltime[].q);
cntr[].prn  = VCC;
cntr[].ena  = VCC;
cntr[].d = cntr[].q + 1;

switch.clk  = clk;
switch.clrn = VCC;
switch.prn  = VCC;
switch.ena  = VCC;
pwm_out = switch.q;

switch.d = (cntr[].q < lowtime[].q);
```

Table 6: AHDL implementation of PWM controller.

Bibliographical Information

1. M.E. Parten, “Teaching Digital Design with HDL,” *Proceedings of the 1997 American Society for Engineering Education Annual Conference and Exposition*, Milwaukee, WI, June 15-18, 1997.

2. N. Krouglicof, “Development of a Universal Controller for Pedagogical Applications Involving Data Acquisition, Data Logging and Control,” *Computers in Education Journal*, vol. XIII, No. 2, pp. 2-10, 2003.

3. D.J. Ahlgren and J.E. Mendelssohn, “The Trinity College Fire-Fighting Home Robot Contest: A Medium for Interdisciplinary Engineering Design,” *Proceedings of the 1998 American Society for Engineering Education Annual Conference and Exposition*, Seattle, WA, June 21-24, 1998.

4. O. Fucik, B. Wilamowski, and M. McKenna, “Laboratory for the Introductory Digital Course,” *Proceedings of the 2000 American Society for Engineering Education Annual Conference and Exposition*, St. Louis, MO, June 18-21, 2000.

Biographical Information

NICHOLAS KROUGLICOF joined the Mechanical Engineering Department at Union College in September 2000. Previously, he was a faculty member at the École de technologie supérieure in Montreal. He has taught and developed laboratories for a number of undergraduate courses relating to system dynamics and control, mechatronics, automation, and CAD. His research interests are in the areas of machine vision, intelligent sensors, and mechatronics.