9 Peripherals

In this Chapter, the digital protoboard, Commodore mouse, hexadecimal keypad, and logic analyzers are described. The LEGO control board is described separately in the next Chapter.

9.1 Digital Protoboard

In the lab, each Ultragizmo board station is equipped with a digital protoboard (shown in Figure 51). The protoboard is a versatile tool providing simple user I/O to the Ultragizmo board. It is especially handy for debugging SFPGA hardware designs. Each protoboard has 8 switches (SW1-SW8), a push button (PULSE), 8 LEDs ($\overline{\text{LED1}}$-$\overline{\text{LED8}}$) and a 40-pin connector. The 40-pin connector is connected to the switches, buttons and LEDs. Table 29 shows the connection of the 40-pin connector. Pins 2, 4, 6, 8, 10, 12, 14, 16, and 18 are outputs from the 40-pin connector driven by the switches and the pulse button. Pins 22, 24, 26, 28, 30, 32, 34, and 36 are inputs to the 40-pin connector driving LEDs. The protoboard can be easily connected to the SFPGA by connecting the 40-pin connector of the protoboard with the pin-compatible **SFPGA_DIGITAL** port of the Ultragizmo board using a 40-pin ribbon cable.

Pin	Connection	Pin	Connection
1	GND	2	SW1
3	GND	4	SW2
5	GND	6	SW3
7	GND	8	SW4
9	GND	10	SW5
11	GND	12	SW6
13	GND	14	SW7
15	GND	16	SW8
17	GND	18	PULSE
19	GND	20	GND
21	GND	22	$\overline{\text{LED1}}$
23	GND	24	$\overline{\text{LED2}}$
25	GND	26	$\overline{\text{LED3}}$
27	GND	28	$\overline{\text{LED4}}$
29	GND	30	$\overline{\text{LED5}}$
31	GND	32	$\overline{\text{LED6}}$
33	GND	34	$\overline{\text{LED7}}$
35	GND	36	$\overline{\text{LED8}}$
37	GND	38	GND
39	GND	40	CLK

Table 29 - *Digital Protoboard 40-Pin Connector Pin Assignment*

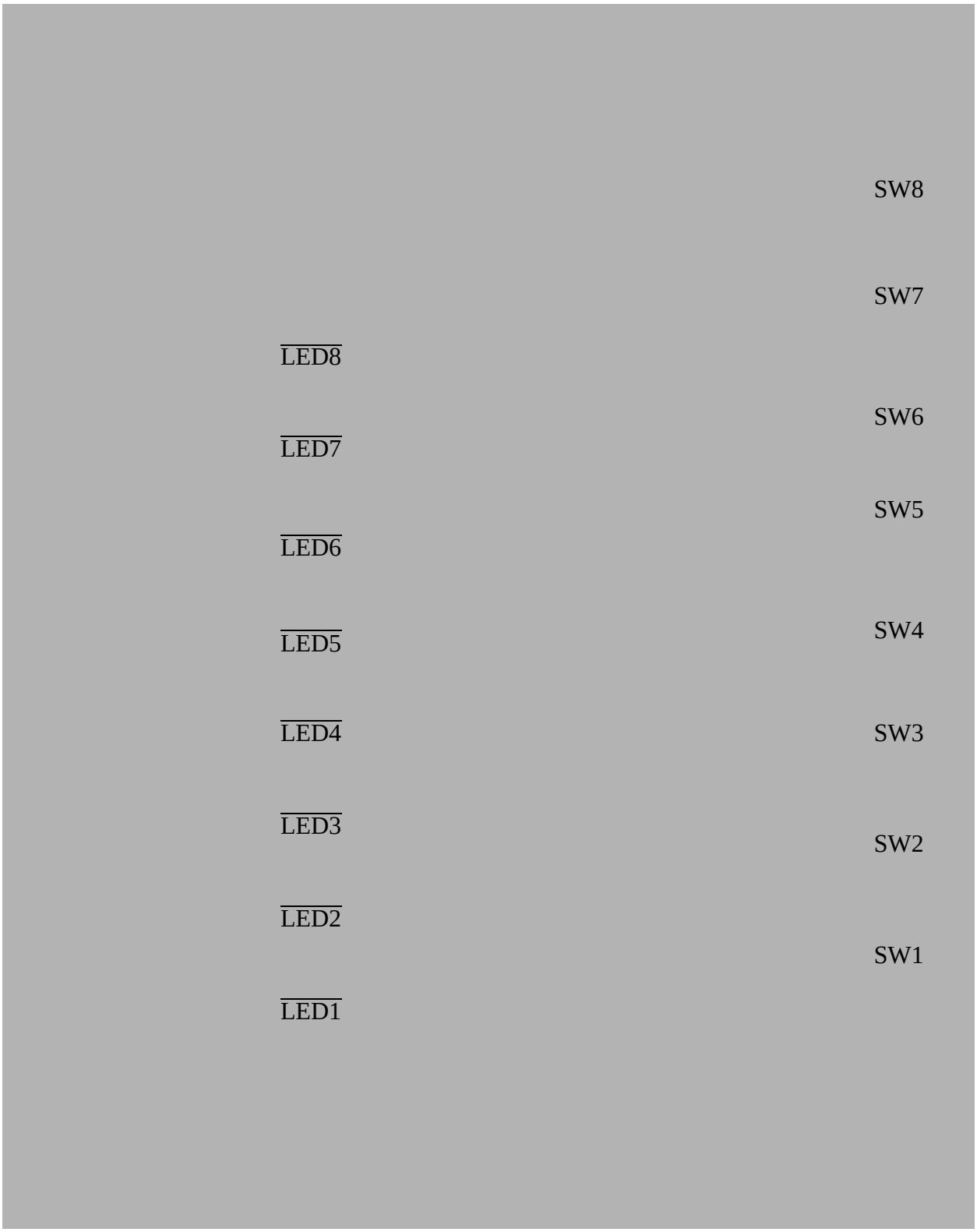

Figure 51 - *Digital Protoboard*

9.2 The Commodore 1352 Mouse

The Commodore 1352 mouse can be interfaced to the Ultragizmo board using the PIT. The mouse has 6 status lines that encode information about the motion of the mouse in both the x and y-directions, and whether or not each of the two buttons is pressed. These 6 status lines are connected to the PIT using the mouse cable adaptor. Connect one end of a 40-pin ribbon cable to the mouse cable adaptor and the other end to the PIT connector on the Ultragizmo board. When you plug the mouse into the mouse cable adaptor, the connections shown in Table 30 are made.

Mouse Pin (all outputs of mouse)	PIT Pin
Y0	PA0
Y1	PA1
X0	PA2
X1	PA3
Left Button	PA4
Right Button	PA5

Table 30 - *Mouse Status Line Connections*

Internally, the mouse has two rollers, one in the x-direction and one in the y-direction. Sensors determine the orientation of these rollers, and encode the position of each into a two-bit pattern. X1 and X0 encode the position of the x-roller, and Y1 and Y0 encode the position of the y-roller. When the mouse is not moving, the values on the X and Y lines remain constant. When the mouse moves, however, the rollers spin and the values on the X and Y lines (depending on the direction of the movement) change in a cyclical pattern. Figure 52 shows the state diagram for both the X and Y status lines. A transition in either X or Y indicates that the mouse has moved by one unit.

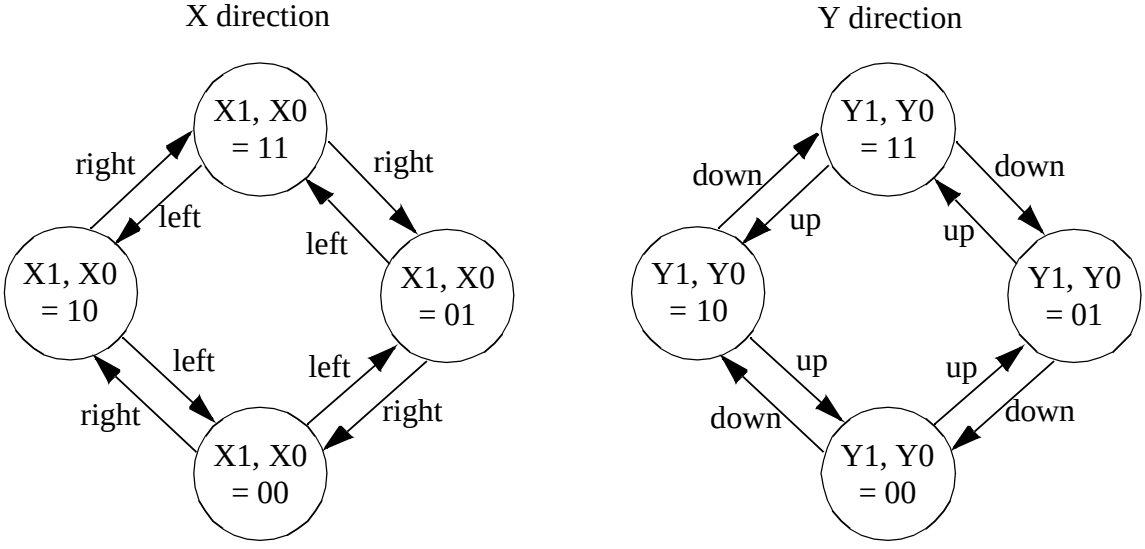

Figure 52 - *Mouse Status Line Encoding*

The state of the signals on X1, X0 (or Y1, Y0) can only change to an adjacent state, with the direction of mouse motion necessary for that transition indicated on the diagram. For example, if the X1 and X0 go went through the sequence 11-01-00-10-11-01-00, we can tell that the mouse has moved 6 units to the right.

When a button is pressed the corresponding status line is grounded; when not pressed the status line is set to logic 1 by a pullup resistor on the PIT.

The logic values presented by the mouse (on any of its outputs) may “bounce” when switching from one state to the next. That is, a 0 to 1 transition, for example, might make several transitions as shown in Figure 52. Any program using the mouse must deal with these bouncing signals in some way; a common method is to wait a few milliseconds for the signals to settle down.

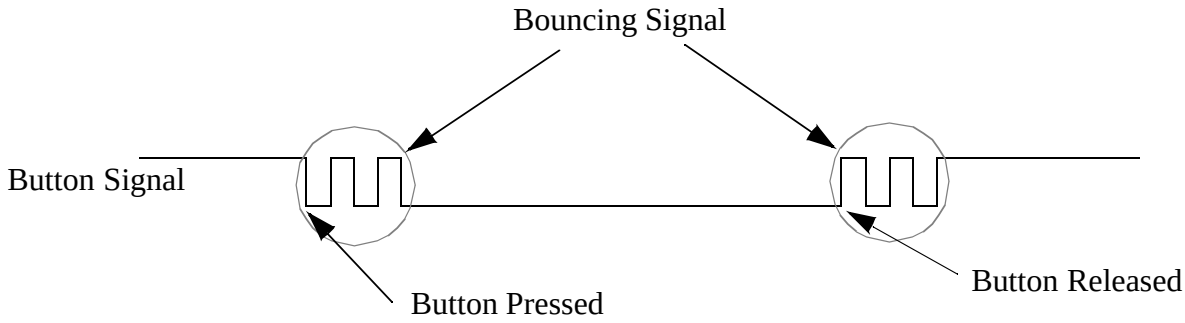

Figure 53 - *Bouncing Status Line*

Note that the mouse cannot generate interrupts unless it is interfaced to the M68000 bus through the SFPGA.

9.3 The Hex Keypad

The hex keypad has a set of 16 buttons labeled with the numbers 0 to 9 and letters A to F as shown in Figure 54. It connects to the Ultragizmo's PIT via a 40-pin ribbon cable.

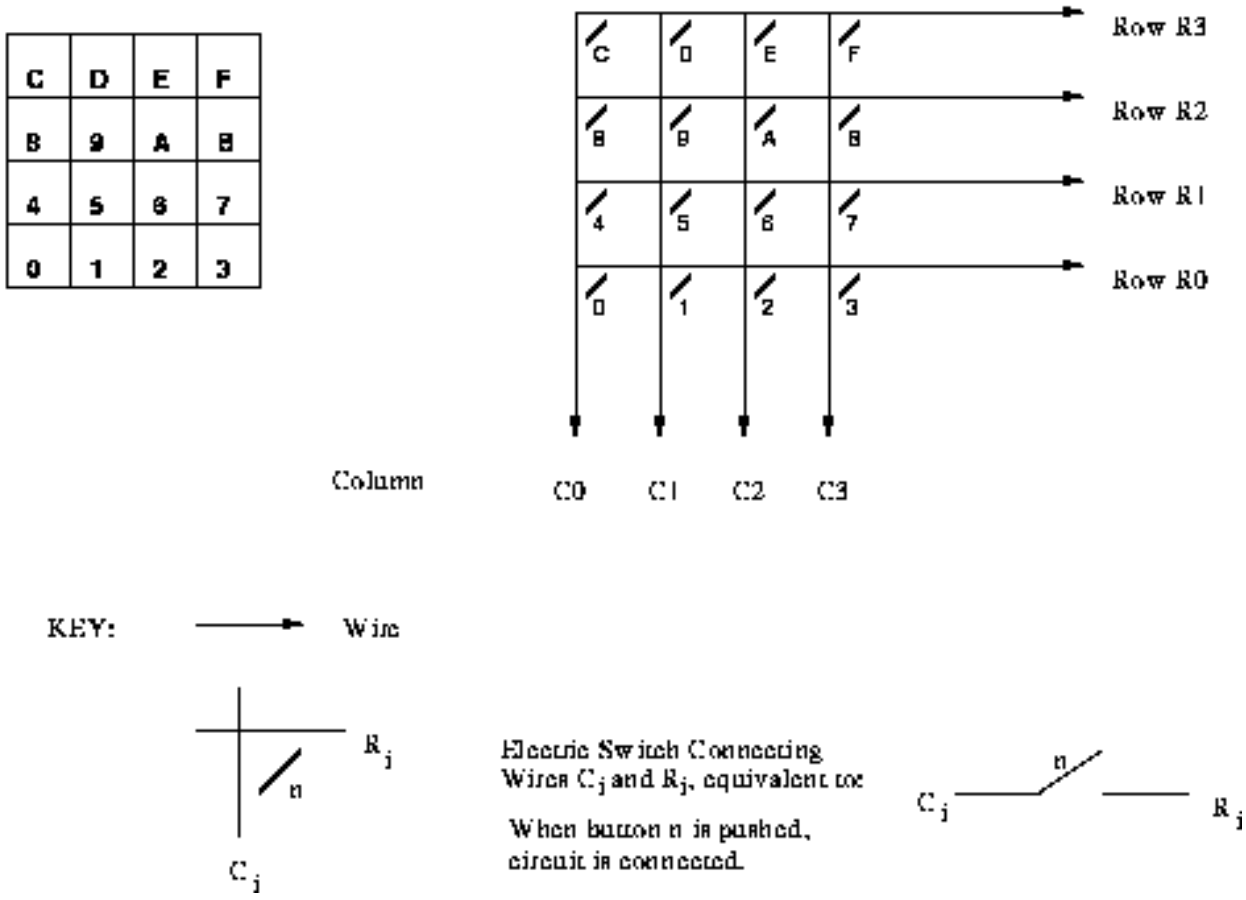

Figure 54 - *Hex Keypad Connections*

The hex keypad has eight wires emanating from it: four column wires (C0, C1, C2 and C3) and four row wires (R0, R1, R2 and R3). When any button is pressed, it electrically connects one of the column wires to one of the row wires as illustrated in Figure 54. For example, if button A is pressed then the wire labeled **column C2** is electrically connected to the wire labeled **row R2**. All the other wires remain unaffected.

Table 31 lists the pin connections on the hex keypad and its corresponding PIT connections.

When a key is pressed on the hex keypad, it bounces for a few milliseconds. The M68000 can determine, after debouncing for about 10 milliseconds, which key has been pressed by doing the following. The PIT pins corresponding to the row wires (PA0 -> PA3) should be set as inputs and the PIT pins corresponding to the column wires (PA4 -> PA7) should be set as outputs. By writing 0s on the outputs, exactly one of the inputs will be read as a 0. The others will be read as 1s since the PIT has pull-up resistors and the pins are electrically disconnected from anything else. The zero input corresponds to the row number of the button being pressed. Then the procedure is

reversed for rows and columns in order to determine a column number. Given the row and column number, the M68000 can tell which button is being pressed.

Hex keypad pin	PIT pin
R0	PA0
R1	PA1
R2	PA2
R3	PA3
C0	PA4
C1	PA5
C2	PA6
C3	PA7

Table 31 - *Hex Keypad to PIT Connections*

Note that the hex keypad cannot generate interrupts unless it is interfaced to the M68000 bus through the SFPGA.

9.4 Logic Analyzers

Three types of logic analyzers are available to test hardware in the labs. They are:

- Philips PM3580 Logic Analyzer
- HP 1664A Logic Analyzer
- HP 54645D Mixed Signal Oscilloscope

Please see Fred Aulich's website at *http://www.eecg.toronto.edu/~aulich* for instructions on how to use each one.

10 LEGO Control Board

10.1 Introduction

This Chapter outlines the use of the University of Toronto LEGO control board with the Ultragizmo board. Most of the work students have done with computers has led to something being displayed on the screen. But this is only a fraction of what computers are used for. Almost everything, from aircraft to office buildings, and from nuclear power plants to toys, is controlled by a computer. Unfortunately, real things are too expensive (and dangerous) to work with in an undergraduate lab.

With LEGO one can build a model of a real world mechanical system, and then use the computer to control the model. These models will demonstrate many real world problems such as gear slippage and sensor noise. The student must learn how to overcome these problems. This provides a better learning experience than can be obtained with a simulation.

The LEGO control board is designed to provide a simple interface between LEGO motors and sensors, and the Ultragizmo board. Section 10.2 describes the features of the board. Section 10.3 provides a step-by-step tutorial to explain the basic operation of the board. The rest of this chapter is intended to be a reference. Section 10.4 through Section 10.6 describe the motors, lights, and sensors respectively. Section 10.7 describes how to use the sensors to generate interrupts. Section 10.8 presents some tips on LEGO construction. Section 10.9 gives some ideas on using the LEGO board in the lab and Section 10.10 contains pointers to more information on LEGO and Section 10.11 lists several sample programs.

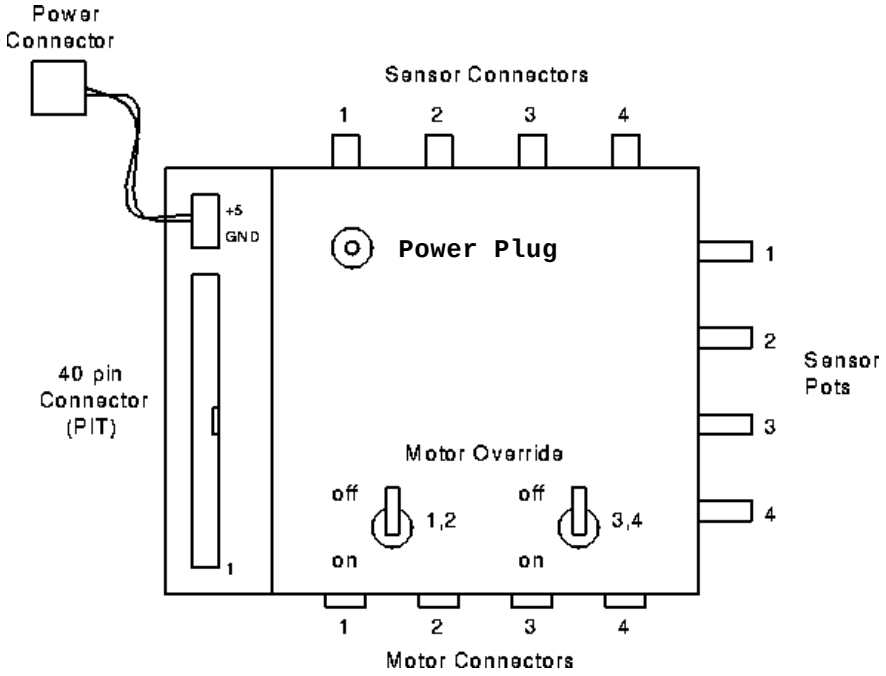

Figure 55 - *Diagram of the University of Toronto LEGO Controller Board.*

10.2 LEGO Control Board

Figure 55 shows the University of Toronto LEGO control board designed by Sunil Sebastian (9T5), Peter Pereira, and Fred Aulich. This board allows up to 4 motors and 4 sensors to be independently controlled through the parallel port (PIT) of the Ultragizmo board.

The motors use standard audio power connectors. These connectors provide a 5V DC signal of either polarity. Two motor override switches are provided (each controls two motors) to quickly stop the motors in case of a problem.

The sensors use standard audio signal connectors. Great care should be taken to ensure that the sensors are only connected to the sensor connectors, never elsewhere. Each sensor has a potentiometer to adjust sensitivity. When the potentiometer is fully clockwise the sensor is least sensitive (most likely to be read as a 0) and vice versa. The sensor potentiometers allow the sensors to be used reliably under many lighting conditions.

The power plug on top of the board provides 5V DC and is intended to be used to supply power to the lights. The audio signal connectors also connect to this port. Lights can also be connected to the motor connectors, allowing them to be turned on and off under computer control.

Pin	Purpose	PIT pin	Pin	Purpose	PIT pin
1	Motor 1 enable	PADOR0	2	Motor 1 direction	PADOR1
3	Motor 2 enable	PADOR2	4	Motor 2 direction	PADOR3
5	Motor 3 enable	PADOR4	6	Motor 3 direction	PADOR5
7	Motor 4 enable	PADOR6	8	Motor 4 direction	PADOR7
9	GND		10	Sensor 1	H1
11	GND		12	Sensor 2	H2
13	GND		14	Sensor 3	H3
15	GND		16	Sensor 4	H4
17	Sensor 1	PBDOR0	18	Sensor 2	PBDOR1
19	Sensor 3	PBDOR2	20	Sensor 4	PBDOR3
21	-		22	-	
23	-		24	-	
25	GND		26	-	
27	GND		28	-	
29	GND		30	-	
31	GND		32	-	
33	-		34	-	
35	-		36	-	
37	-		38	-	
39	-		40	-	

Table 32 - *Lego Connector Pinout*

The power connector can be attached directly to the power supply on the Ultragizmo board.

The 40 pin connector is designed to be attached directly to the PIT connector on the Ultragizmo board. It can also be connected to a protoboard or to the second PIT. The pinout for the connector is given in Table 32. The third column gives the corresponding pin on the PIT.

10.3 LEGO Tutorial

This tutorial will walk you step-by-step through an example in which the LEGO board is used to read a sensor and control a motor. For this tutorial you will need:

- Ultragizmo station
- LEGO control board
- 40 pin ribbon cable
- LEGO motor, sensor and light
- 3 LEGO cables, 1 each for a motor, a sensor, and a light

First, connect the power connector on the LEGO board to the power supply on the Ultragizmo board. The connector only goes in one way, and it doesn't matter if the Ultragizmo is already turned on. Next, connect the LEGO board to the PIT using the ribbon cable. Ensure that pin 1 on the PIT connector is attached to pin 1 on the LEGO board (this will happen automatically if you use a ribbon cable with notched connectors).

Ensure that both motor override switches are in the off position. Connect the LEGO motor to motor connector 1 using a grey cable. Connect the sensor to sensor connector 1 using a silver and gold cable. Connect the light to the power connector on top of the LEGO board with another cable. Be careful that these connections are done properly. Connecting a motor or light to a sensor connector may damage the LEGO board. Connecting a sensor to a motor connector or the power connector may damage the sensor.

Turn on the motor override switch for connectors 1 and 2. Since the reset state of the Ultragizmo PIT is $FF, the motor should begin spinning. Turn off the motor override switch.

We will now learn how to control the motor using block fill commands (**bf**). Initialize the PIT port A direction register (PADDR, address $c10002) to contain $ff. This sets all the bits in port A to be outputs.

You can now control the motor by changing the value in the port A data output register (PADOR) which has an address of $c10000. Fill the PADOR with $00 and turn on the override switch. The motor should be stopped. Bit 0 enables the motor and bit 1 determines the direction. Try filling the PADOR with the values $01, $02, and $03. Does the motor behave as expected?

Look at sample program 1 in Section 10.11 and make sure you understand how it works. Load it into the Ultragizmo and run it. This program displays the current value of the sensor (1 if lit, 0 if dark) and uses it to drive the motor. Pressing any key will exit this program. Notice that the sensor value changes (and the motor activates) depending on both the orientation and distance between the light and the sensor.

Move sensor potentiometer 1 fully clockwise. Place the light about 30 cm from the sensor pointing directly at it. The sensor should display a 0 (if not, move them farther apart). Slowly turn the potentiometer counter-clockwise to increase the sensitivity of the sensor until a 1 is displayed. At this point, any decrease in lighting should make the sensor read 0 again. Try passing objects

between the light and the sensor, and shading the sensor from room lighting. Getting the sensors to work reliably is one of the trickiest parts of LEGO design.

You have now used the LEGO board and a simple program to read a sensor and drive a motor. The next step is to build a model, and write a program to control it. Have fun!

10.4 Motors

The motors are controlled by port A of the PIT as shown in Table 33:

PADOR bit number	Purpose
0	Motor 1 Enable (1=enable, 0 =stop)
1	Motor 1 Direction
2	Motor 2 Enable
3	Motor 2 Direction
4	Motor 3 Enable
5	Motor 3 Direction
6	Motor 4 Enable
7	Motor 4 Direction

Table 33 - *LEGO Motor Control*

In order to control the motors, port A should be configured as outputs. The motors can then be controlled by writing data to the port A data output register PADOR. The following code fragment initializes the PIT and stops all motors:

```
move.b      #$ff,PADDR          ;Port A is all outputs (motors)
move.b      #$00,PBDDR          ;Port B is all inputs (sensors)
move.b      #$00,PADOR          ;Turn off all motors
```

The DC motors supplied by LEGO have two terminals. The speed of the motor depends on the voltage between the terminals (about 6,000 rpm at 5V), and the direction depends on the polarity. There is no absolute direction reference, if the motor seems to be working backwards, simply reverse the connector at the motor end. The LEGO board provides 5V DC to the motors. Speed control can be obtained by chopping (turning the motor on and off quickly). The percentage of time that the motor is on is called the duty cycle. The greater the duty cycle, the greater the speed. If this technique results in rough operation, try connecting a capacitor across the motor terminals.

The motors draw only a little current when turning (about 100 mA), but draw much more when starting or whenever there is an applied voltage but the motor is physically prevented from turning (up to 1.0 A). The latter situation should be avoided to prevent damage to the motor. One way to prevent this is to use a belt drive (see Section 10.8). This allows the motor to turn even if

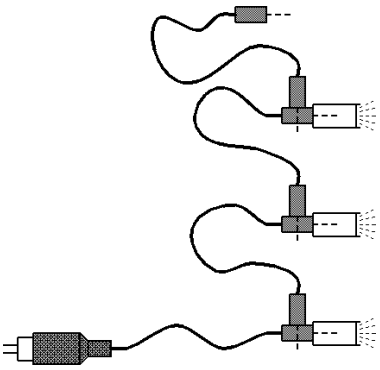

Figure 56 - *Lights (and Motors) Can Be Connected in Parallel by Daisy Chaining the LEGO Cables*

the mechanism becomes jammed. If for some reason a motor ever becomes jammed, turn off the override switch immediately.

Motors can be connected in parallel and controlled by a single connector on the LEGO board as illustrated in Figure 56. This technique draws a lot of current through one output and should be used sparingly.

10.5 Lights

The LEGO lights are designed to operate in conjunction with the sensors. They operate at 5V DC. They can be connected to the power connector on top of the LEGO board or to a motor connector. Connecting a light to a motor connector allows it to be turned on and off under program control. Multiple lights can be connected in parallel as shown in Figure 56.

10.6 Sensors

The LEGO sensors are light detectors with adjustable sensitivity. Figure 57 shows a sensor

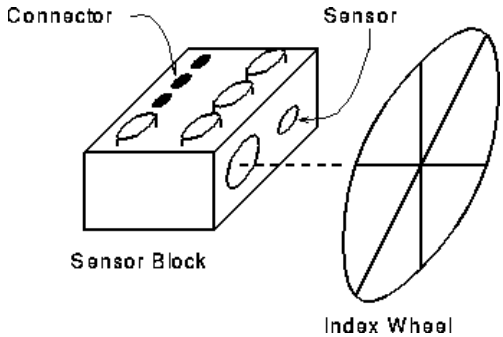

Figure 57 - *Location of the Sensor and Axle Hole Used to Mount an Index Wheel*

brick and the location of the actual sensor. The other hole is to facilitate the use of an index wheel, as will be described shortly.

The LEGO board converts the sensor output to a digital signal: 1 if light is detected and 0

otherwise. The sensor potentiometer controls the sensitivity (the amount of light needed to produce a 1). When fully clockwise the sensor is least sensitive (requires more light to produce a 1).

The sensors connect to bits 0-3 of the port B input buffer as shown in Table 34:

PBDI bit number	Purpose
0	Sensor 1 output (1=light, 0=dark)
1	Sensor 2 output
2	Sensor 3 output
3	Sensor 4 output

Table 34 - *LEGO Sensor Outputs*

In order to read the sensors port B should be configured as inputs. The sensors can then be read by reading data from the port B data input. The following code fragment initializes the PIT and moves the values from the sensors into d0.

```
move.b      #$ff,PADDR          ;Port A is all outputs (motors)
move.b      #$00,PBDDR          ;Port B is all inputs (sensors)
move.b      PBDI,d0             ;Read the sensor values into d0
```

The sensors can be used in a variety of mechanical configurations, only two of which will be described here. The first is as a position indicator that normally gives a 1, but gives a zero when an object breaks the light beam. This is relatively straight forward to implement.

The second configuration uses an index wheel to keep track of rotation. The index wheel is coloured with alternating black and white sectors. Light reflected from the disk will produce alternating 1's and 0's which can be counted in order to measure rotation. Great care must be taken to get this configuration to function properly. It is generally much easier, though less accurate, to use the 8-sector side of the index wheel instead of the 16-sector side.

First, the sensitivity must be calibrated to reliably detect all of the white and black areas. The program *Calibrate* in Section 10.11 can be used to do this:

1. Set-up the mechanical configuration so that the index wheel can be turned by hand. One possible configuration is shown in Figure 58. The actual positions of the light, sensor, and index wheel should match those used in your model. Note that the sensor and the light must be separated to avoid light 'leaking' into the sensor. Also, the separation between the index wheel and the sensor is critical. You might want to try putting a small pulley between the index wheel and the sensor. A reflective surface beneath the light also makes a big difference.
2. Load and run the program *Calibrate*. This program continuously displays the value

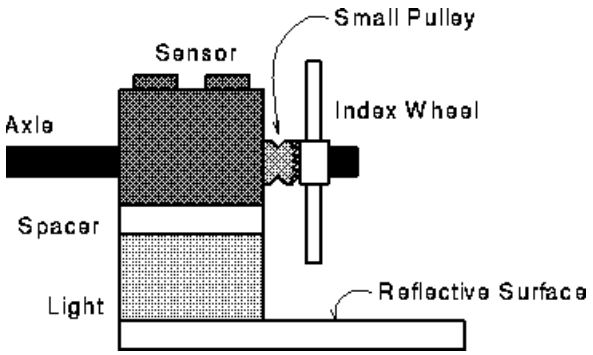

Figure 58 - *One Possible Configuration for Using a Sensor to Count the Rotations of an Index Wheel*

of the sensor and a count of the number of times it has changed.

3. Connect the light to the power connector and the sensor to sensor connector 1.
4. Turn sensor potentiometer 1 fully clockwise (least sensitive).
5. Turn the index wheel and ensure that all of the white and black areas are displayed as 1 and 0 respectively. If the sensor misses some white areas, turn the potentiometer counter-clockwise to increase its sensitivity. If the sensor misses some black areas, move the light farther from the index wheel, or move the wheel closer to the sensor, and re-calibrate.

Second, the sensor input must be debounced. As the edge between a white and black region passes in front of the sensor, it may produce a short burst of alternating ones and zeros as shown in Figure 59. The solution to this problem is to wait until the sensor output has settled before considering it valid. The following code fragment is intended to wait for a falling edge (a dark segment). After first detecting a 0, it waits for some time (delay) and then checks to make sure that the sensor output is still zero.

```
wait10   move.b    PBDI,d0          ;get the sensor outputs
         btst.b    sensor,d0        ; loop until the sensor gives a zero
         bne       wait10
         move.w    #DELAY,d1
deloop   dbra      d1,deloop        ;wait a while, DELAY equ $0010 works well
         move      PBDI,d0          ;check if the sensor is still zero
         btst.b    sensor,do
         bne       wait10           ;if not, wait for it to become zero
         ......    ......           ;if it is, carry on
```

Figure 59 - *An Example of Sensor Bounce*

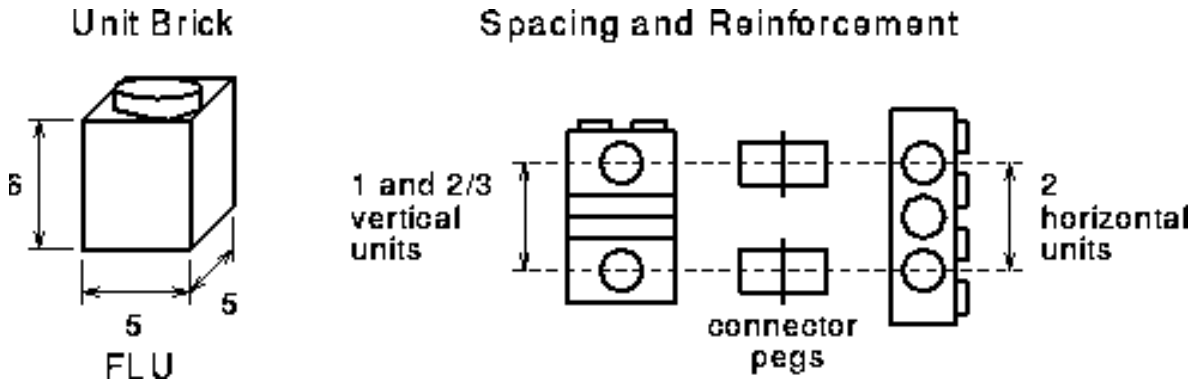

Figure 60 - *Fundamental LEGO Dimensions Showing How a Beam Can Be Used to Reinforce a Stack*

The sensors can also be set up to generate interrupts. This will be discussed in Section 10.7.

10.7 Sensor Interrupts

The LEGO control board and the PIT can be used to generate level 5 autovectored interrupts based on the transitions of the light sensor inputs.

As shown in Section 10.2, sensors 1 through 4 can be connected to the H1, H2, H3, and H4 pins of the PIT through a 40-pin connector. When level 5 interrupts are enabled, a 0 (not sensing light) to 1 (sensing light) transition or a 1 (sensing light) to 0 (not sensing light) transition on any of these four pins triggers a level 5 autovectored interrupt from the PIT (Section 8.2.2). See Section 8.2 for more details and an example on how to configure level 5 autovectored interrupts.

10.8 LEGO Construction

LEGO-Technics is great for constructing all sorts of interesting things. The best method is still the one kids use: have a general idea of what you want to build, and just simply start snapping pieces together. You will probably have to rebuild some parts of your model several times before you get it right. Don’t try and make your model compact, a larger model is easier to modify.

The instruction books provided with the kits give some ideas on how to put the pieces together. They have good diagrams on how to build gearboxes that step-down from high-speed low-torque motors to low-speed high-torque drives that can be used in your model. You might want to use one of the instruction books as a starting place, but don’t be afraid to modify the design, or try something completely different.

LEGO bricks aren’t square. A single unit brick has a length of one Fundamental LEGO Unit (FLU), and a length:width:height ratio of 5:5:6 as shown in Figure 60. This ratio, coupled with the existence of 1/3-height flat pieces, allows the creation of vertical spacings that perfectly match the horizontal ones. This allows vertical stacks of bricks to be reinforced with cross-beams and connector pegs, creating sturdy structures that won’t fall apart.

Axles should be supported by at least two beams (spaced as far apart as possible) to prevent them from twisting and possibly jamming. Axles, gears and pulleys should not be assembled too

tightly in order to reduce frictional losses. You should be able to move the parts easily by hand before trying to move them using the motors.

Pulley drives (elastic bands) tend to slip a lot. You can, however, use this to your advantage. If a motor seizes (is prevented from turning) it draws a lot of current which could damage the motor or the LEGO controller board. If your model is likely to cause a motor to seize, use a pulley connection. The slippage allows the motor to turn even when the model isn't moving. This technique is used in the claw on the robot arm.

Care must be taken when using the gears to ensure that they mesh properly. If the gears are too close together they will jam, and if they are too far apart they will slip. The instruction books have good examples of gear use.

10.9 Lab and Project Suggestions

The LEGO boards can be used in a number of ways. The obvious method is to connect the LEGO board directly to the Ultragizmo through the PIT. A LEGO lab then becomes an exercise in programming. One can also connect the LEGO board to a protoboard or to the SFPGA and then design the necessary hardware to interpret the sensor values and drive the motors. The most demanding configuration is to connect the LEGO board to the Ultragizmo with the SFPGA as an intermediary. For example, the software could send an instruction to turn on a motor for 12 pulses on a sensor. Hardware on the SFPGA would turn on the motor, count the pulses, and then turn off the motor. The SFPGA could also be used to generate level 6 autovectored or vectored interrupts based on LEGO board signals.

Most LEGO labs will take at least two weeks because of the time required to build a mechanical model. A simple one week lab could be based on the tutorial, Lab M10, or Lab M12. The object would be to drive motors based on sensor inputs, with little or no mechanical construction.

The LEGO board can also be combined with other hardware available in the lab such as the hex keypad, CODEC, SFPGA, and mouse.

10.10 For More Information

There is a lot of information available on the internet at *http://legowww.homepages.com/* and by *ftp* at *earthsea.stanford.edu* in the */pub/lego* directory. Some of the information in Section 10.8 was taken from Chapter 4 of the 6.270 Robot Builder's Guide by Fred Martin and Randy Sargent, Department of Elec. Eng. and Comp. Sci, MIT, 1992.

10.11 Sample Programs

All four of the following sample programs are available on-line on the *ugsparc* system in the directory */cad2/ultragizmo/MLabs*.

Sample1: Tutorial

```
* This program is for the tutorial in the LEGO board manual.
* It continually reads sensor 1 and displays the result on the
* terminal. When the sensor has a 1 (light detected) the motor
```

```
* is turned on.

* Declare addresses for PIT and the terminal

PADOR   equ         $c10000
PBDOR   equ         $c10001
PADDR   equ         $c10002
PBDDR   equ         $c10003
PADI    equ         $c10004
PBDI    equ         $c10005

SRB     equ         $fffff7f3
RBB     equ         $fffff7f7
TBB     equ         $fffff7f7

* To run the program, use go 20000

        org $20000

*Initialize PIT
        move.b      #$ff,PADDR          ;Port A is all outputs (motors)
        move.b      #$00,PBDDR          ;Port B is all inputs (sensors)
        move.b      #$00,PADOR          ;Turn off all motors

*Main loop to read the sensor, display it, and drive the motor
*Exits when any key is pressed.

main    move.b      PBDI,d0             ;read sensors into d0
        bsr         display             ;subroutine to display sensor value
        bsr         drive               ;subroutine to drive the motor
        btst.b      #0,SRB
        beq         main                ;if a key is pressed, clear the
        move.b      RBB,d0              ;input, and turn off the motors.
        move.b      #$00,PADOR
        trap        #15

* This subroutine displays the sensor value
display movea.l     #mesg,a0            ;make a0 point to a message string
        bsr         Print               ;subroutine to print a string
        move.b      #$30,value
        and.b       #$01,d0             ;d0 holds 0 or 1 depending on sensor
        add.b       d0,value            ;value contains ascii for 0 or 1
        movea.l     #value,a0
        bsr         Print
        rts

* Subroutine takes the sensor value (in d0) and decides
* whether or not to drive the motor
drive   btst.b      #0,d0
        beq         zero
        move.b      #$01,PADOR          ;turn on motor 1
        rts
zero    move.b      #$00,PADOR          ;turn off motor 1
        rts
```

```
* Subroutine prints the string pointed at by a0, ended with a 0
Print   btst.b      #2,SRB
        beq         Print
        move.b      (a0)+,TBB
        cmp.b       #0,(a0)
        bne         Print
        rts

* Declare data (strings)
mesg    dc.b        ‘ Sensor One (1=light, 0=dark) = ‘,0
value   dc.b        $30,$0d,0               ;sensor value followed by CR
```

Sample 2: Calibrate

```
* This program is very useful for testing a sensor with an index wheel.
* Connect the sensor to #1. Turn pot 1 fully clockwise (least sensitive)
* Run this program. Increase the sensitivity until all of the areas
* are detected reliably.

* Initialize addresses

PADOR   equ     $c10000
PBDOR   equ     $c10001
PADDR   equ     $c10002
PBDDR   equ     $c10003
PADI    equ     $c10004
PBDI    equ     $c10005

SRB     equ     $fffff7f3
RBB     equ     $fffff7f7
TBB     equ     $fffff7f7

* Debounce delay value

DELAY   equ     $0010           ;seems to work okay, might need to change this

* go 20000 to run program

 org $20000

*Initialize the PITs to keep the motors still (just in case)
        move.b  #$ff,PADDR      ;Output for motors
        move.b  #$00,PBDDR      ;Input for sensors
        move.b  #$00,PADOR      ;Turn motors 1-4 off

*Main loop to check the sensor and print an update
*when it changes. Hitting any key resets the counter.

        clr.l   d0              ;current state of sensor 1
        clr.l   d1              ;old state of sensor 1
        move.w  #-1,d2          ;current count of edges (updated to 0
*                                first time through loop)

main    btst.b  #0,SRB          ;check for keyboard, if none read sensors
        beq     sensors
        move.b  RBB,d2
        move.w  #-1,d2          ;there was a key input so clear the count
        clr.l   d1
        bra     main

sensors move.b  PBDI,d0         ;read the sensor, compare it to old
        cmp.b   d0,d1           ;value, branch to main if there is no change
        beq     main
        bsr     delay           ;wait for sensor to debounce
        move.b  PBDI,d0
        cmp.b   d0,d1           ;if the sensor isn't still different,
```

```
            beq         main            ;then go back to main (false alarm)

*           The sensor value has changed, so we need to update
*           the old value and the count, and display the results

            move.b      d0,d1           ;update old sensor value
            add.w       #1,d2           ;update count
            move.b      #$30,sens1      ;convert sensor output to ascii 1 or 0
            btst.b      #0,d0
            beq         skip
            move.b      #$31,sens1
skip        move.b      #$30,count      ;convert count to ascii, only works for 0-9
            add.b       d2,count
            movea.l     #mesg1,a0       ;print the sensor value and count
            jsr         Print
            movea.l      #sens1,a0
            jsr         Print
            movea.l     #mesg2,a0
            jsr         Print
            movea.l     #count,a0
            jsr         Print
            bra         main

* Prints the string pointed at by a0 and ended with a 0
Print       btst.b      #2,SRB
            beq         Print
            move.b      (a0)+,TBB
            cmp.b       #0,(a0)
            bne         Print
            rts

* Sits here for DELAY cycles
delay       move.l      DELAY,d4
deloop      dbra        d4,deloop
            rts


* Declare data
mesg1       dc.b        ‘ Sensor value =’,0
mesg2       dc.b        ‘ Count = ‘,0
sens1       dc.b        $30,0
count       dc.b        $30,$0d,0
```

Sample 3: Conveyor Belt

```
* Simple program to control a conveyor belt with a sensor at either
* end. If a sensor is blocked, belt goes until the other sensor is
* blocked. A key is pressed to reset and start the process over.

* Initialize addresses

PADOR   equ         $c10000
PBDOR   equ         $c10001
PADDR   equ         $c10002
PBDDR   equ         $c10003
PADI    equ         $c10004
PBDI    equ         $c10005

SRB     equ         $fffff7f3
RBB     equ         $fffff7f7
TBB     equ         $fffff7f7

* go 20000 to run program

        org $20000

*Initialize the pit to keep the motors still
        move.b      #$ff,PADDR          ;Output for motors
        move.b      #$00,PBDDR          ;Input for sensors
        move.b      #$00,PADOR          ;Turn motors 1-4 off

*Main loop to get inputs and drive motors
main    bsr         sensors
        cmp.b       #$30,s1dat          ;if sensor one is 0, move belt in direct 1
        beq         dir1
        cmp.b       #$30,s2dat          ;if sensor two is 0, move belt in direct 2
        beq         dir2
        bra         main

* keep motor going forward until sensor two is 0
dir1    bsr         sensors
        cmp.b       #$30,s2dat
        beq         stop
        move.b      #$01,PADOR
        bra         dir1

* keep motor going reverse until sensor one is 0
dir2    bsr         sensors
        cmp.b       #$30,s1dat
        beq         stop
        move.b      #$03,PADOR
        bra         dir2

* stop the motor and wait for keyboard input before continuing
stop    move        #$00,PADOR
        movea.l     #reset,a0           ;print a message to hit a key
        jsr         Print
```

```
wait    btst.b      #0,SRB          ;wait for a key
        beq         wait
        move.b      RBB,d0
        bra         main

* read and display current values of the two sensors
sensors move.b       PBDI,d2
        move.b      #$30,s1dat      ;initialize sensor values to ascii 0
        move.b      #$30,s2dat
        btst.b      #0,d2
        beq         skip1           ;if the sensor is a 1, change its value
        move.b      #$31,s1dat      ;to ascii 1
skip1   btst.b      #1,d2
        beq         skip2
        move.b      #$31,s2dat

* print the values
skip2   movea.l     #s1,a0
        jsr         Print
        movea.l     #s1dat,a0
        jsr         Print
        movea.l     #s2,a0
        jsr         Print
        movea.l     #s2dat,a0
        jsr         Print
        movea.l     #return,a0
        jsr         Print
        rts

* Prints to the screen. a0 points at the string. String ends with a 0.
Print   btst.b      #2,SRB
        beq         Print
        move.b      (a0)+,TBB
        cmp.b       #0,(a0)
        bne         Print
        rts

* declare data
s1      dc.b        ‘ Sensor 1 = ‘,0
s2      dc.b        ‘ Sensor 2 = ‘,0

s1dat   dc.b        $30,0
s2dat   dc.b        $30,0

return dc.b $0d,0
reset   dc.b        ‘Press any key to resume operation ‘,$0d,0
        0
```

11 Future Ultragizmo Upgrades

Several new components for the Ultragizmo board are being planned and designed. This chapter describes a few of them. Please check the */cad2/ultragizmo* directory on the *ugsparc* network, as well as Fred Aulich's web page at *http://www.eecg.toronto.edu/~aulich* for any updates.

11.1 Ultravideo VGA Board

The Ultravideo board will be a VGA graphics board which will interface to the Ultragizmo board via the PIT. An accompanying tester circuit will test the specifications of student video designs for suitability to monitors. When specifications are met, the VGA board can be hooked up to an actual monitor.

11.2 Magnetic Compass

This undergraduate design project is a compass which produces orientation values for the Ultragizmo board and can be used to help navigating LEGO cars.

11.3 New Laboratory Experiments

New labs are being created for the Ultragizmo board. One lab will use the SFPGA as a coprocessor for the MC68306. Three labs will initially be created for the Ultravideo board. One will use the mouse to move a pointer on a monitor. The second will require the student to download an image onto the Ultragizmo SRAM, possibly do some image manipulations, and output the result to the screen. The third will be a videoconferencing lab using both the Ultravideo board and the CODEC.

12 Data Sheets

Data sheets for Ultragizmo components are accessible via World Wide Web. The URL addresses are listed here. The URLs are up-to-date as of printing, but are of course subject to change.

Table 35 - *Data Sheet URLs for Ultragizmo Board Components*

Motorola MC68306 Intergrated EC000 Processor	http://www.mot-sps.com/products/microprocessors/32_bit/68k/mc68306.html
Altera Flex 10K70 FPGA	http://www.altera.com/html/literature/lf10k.html
Samsung KM6164002 SRAM	http://www.usa.samsungsemi.com/products/summary/speedsram/KM6164002A.htm
Cypress ICD2053B Programmable Clock Generator	http://www.cypress.com/cypress/prodgate/timi/icd2053b.html
Cirrus Logic Crystal CS4216 16-Bit Stereo Audio Codec	http://www.cirrus.com/products/overviews/cs4216.html